Игровой тур

Оглавление

Глава 1. Code Delivery

1.1 Обзор

Мир Code Delivery — это мир машин, которые доставляют грузы. Мир является прямоугольником размера 1000 × 750. В этом мире есть станции погрузки и разгрузки, станции ремонта, препятствия и бонусы. Ваша задача — разработать стратегию для машины на языке Java (другие языки пока не поддерживаются). Ваша машина будет помещена в смоделированный мир вместе с машинами других участников, и в серии матчей будет выявлена наиболее успешная стратегия.

Одновременно в мире существуют три станции погрузки и три станции разгрузки. Каждая станция погрузки характеризуется временем существования, типом загружаемого груза и количеством груза, имеющегося в данный момент на станции. Станция разгрузки характеризуется временем существования; она может принимать любые грузы в любом количестве. По истечении времени существования станция исчезает. Новая станция появляется в случайном месте.

Тип груза определяет, какая часть общего объёма машины может быть занята этим грузом. Грузы бывают трех типов; может быть разрешено заполнение 50%, 30% и 20% от общего объёма машины. В соответствии с этим будем называть их грузами типа 50, типа 30 и типа 20. Одновременно в мире может существовать ровно одна станция, содержащая груз некоторого типа.

Машина может забирать груз с любой станции погрузки и отвозить груз на любую станцию разгрузки. Задача игрока — перевезти как можно больше груза. Начисление очков происходит при выгрузке груза на станции разгрузки.

Машина может быть одного из двух типов — легковая или грузовая. Тип машины определяется при её создании и не может быть изменен впоследствии. Каждая машина характеризуется количеством здоровья (HP, health points), расходом бензина (в единицу времени) и грузоподъёмностью.

Изначально машина имеет максимальное количество здоровья (100) и максимальное количество бензина (100), а ее скорость равна нулю. Количество бензина убывает равными долями каждую единицу игрового времени. Если бензобак машины опустел, она станет неактивной на некоторое время. По истечении этого времени машина возвращается в активное состояние с бензобаком, заполненным наполовину. Машина может увеличить количество бензина, если подберет бонус «Бензин». Однако превысить ёмкость бензобака нельзя.

Кроме бонуса типа «Бензин» существуют еще бонусы типа «Шип». Машина, подобравшая бонус «Шип», может выбросить его, когда (автор стратегии) сочтёт нужным. «Шип» делает неактивной на некоторое время машину, которая с ним столкнётся. Важно заметить, что если машина, уже имеющая на борту «Шип», подбирает бонус «Шип», то имеющийся «Шип» выбрасывается автоматически.

Бонусы появляются в произвольных местах в мире и могут после этого существовать сколь угодно долго. Когда машина подбирает бонус (типа «Бензин» или типа «Шип»), этот бонус исчезает из мира.

Выброшенный «Шип» также может существовать сколь угодно долго, пока какая-либо машина не наедет на него. После этого «Шип» исчезает из мира.

Когда машина сталкивается с препятствиями, стенами или другими машинами, она теряет часть здоровья и часть (10%) имеющегося на борту груза. Величина, на которую

уменьшится здоровье машины, зависит от скорости, с которой она двигалась. Восстановить здоровье машина может на станциях ремонта. Здоровье машины не может превышать 100.

Станций ремонта в мире две, они имеют фиксированное расположение. Каждое из препятствий также имеет фиксированное расположение и размеры.

Количество здоровья не может быть отрицательным. Наименьшее возможное значение количества здоровья — ноль; при этом значении машина становится неактивной либо до конца игры, либо до взаимодействия со станцией ремонта.

1.2 Игровой тур: общие положения

В начале Игрового тура участники получают доступ к рабочей станции, на которой установлен JDK 1.8, а также среды для разработки на Java — Eclipse, NetBeans, IntelliJ IDEA.

Также на рабочих станциях будет установлено специальное клиентское приложение Игрового тура, которое описано в соответствующей главе данного руководства.

В течение Игрового тура команда может неограниченное количество раз отправлять разработанный ею класс стратегии на сервер. В финальном тестировании — турнире стратегий — будет участвовать последний успешно скомпилированный код.

Все стратегии будут принимать участие как минимум в трёх матчах. В каждом матче принимают участие 4 стратегии. По истечении игрового времени они ранжируются по количеству набранных очков. В разделе «Правила» подробно описано, как именно будут проходить матчи и как будет определяться победитель игрового тура.

Использование на Игровом туре какого-либо заранее заготовленного кода запрещено. Попытка преднамеренного нарушения работоспособности сервера Игрового тура повлечёт за собой дисквалификацию участника.

1.3 Правила отбора

Турнир стратегий состоит из трёх раундов. Каждый раунд состоит из трёх серий матчей. В каждом матче участвуют четыре стратегии. Длительность матча — 4000 единиц времени. Перед первой серией матчей в каждом раунде разделение стратегий на группы по четыре происходит случайным образом. Перед второй и третьей сериями стратегии группируются в зависимости от суммарного количества очков, полученного стратегией к этому моменту в текущем раунде.

После первого раунда производится первый отсев. Стратегии будут отсортированы по суммарному количеству очков, набранному в первом раунде. К участию во втором раунде будет допущено количество стратегий, кратное четырём, и не превосходящее половины числа стратегий, участвовавших в первом раунде.

После второго раунда производится второй отсев. Стратегии будут отсортированы по суммарному количеству очков, набранному в втором раунде. К участию в третьем, финальном, раунде будет допущено четыре лучших стратегии.

Победителем объявляется стратегия, набравшая максимальное суммарное количество очков в третьем раунде. Также по итогам третьего раунда определяются стратегии, занявшие второе, третье и четвёртое места.

1.4 Совершение ходов в матче

В начале матча каждая стратегия инициализируется. После этого стратегии делают ходы. Игровое время дискретно; на один ход стратегии отводится одна единица времени.

В момент инициализации игровое время считается установленным в ноль. Далее, каждая из стратегий выполняет свой первый ход. После этого первая единица игрового времени считается истекшей. Затем стратегии выполняют свой второй ход, после чего истекает вторая единица игрового времени и т. д.

Каждая стратегия может потратить на инициализацию или совершение одного хода не более 0.25 с.

1.5 Описание игрового мира

Размер мира всегда составляет 1000 × 750 и не зависит от разрешения экрана и наоборот. Рис.1 иллюстрирует направление осей координат и положительное направление поворота.

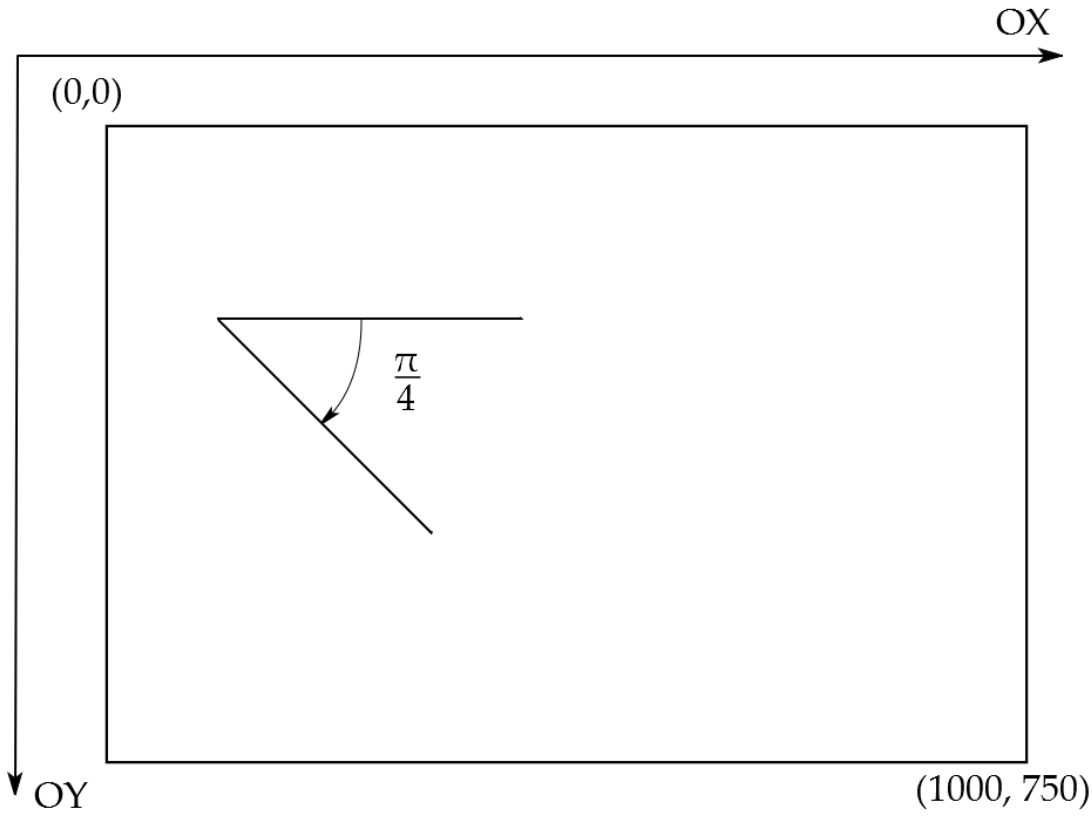

Рис. 1. Игровой мир

Машины появляются на заранее заданных позициях и имеют заранее заданную ориентацию. Машина не может покинуть мир.

Бонусы появляются в случайных позициях и имеют неограниченное время жизни. Бонус исчезает после взаимодействия с машиной (активной или неактивной).

Станции погрузки и разгрузки появляются в случайных позициях и имеют ограниченное (известное) время жизни. По истечении его станция исчезает. Минимально возможное время жизни станции — 500 единиц игрового времени, максимальное — 1000 единиц.

Объект «Шип» появляется как результат выбрасывания его машиной, ранее подобравшей бонус типа «Шип». Он существует неограниченное время, исчезает после взаимодействия с машиной.

Любой объект игрового мира может рассматриваться как круг некоторого радиуса. Радиусы объектов приведены в таблице ниже.

Объект	Радиус
Легковая машина	30
Грузовая машина	40
Станция	40
Бонус	20
Шип	15

Препятствия также могут быть рассмотрены как круги. Размеры и местоположение каждого препятствия фиксированы и одинаковы в каждом матче.

1.6 Характеристики машин

Каждую единицу времени машина расходует бензин. Бензин расходуется равными долями (см. таблицу ниже). Если бензин у машины закончился, машина становится неактивной на 500 единиц игрового времени. Затем машина восстанавливает активность, её бензобак при этом будет заполнен наполовину.

Также машина может стать неактивной, если ее здоровье стало равным нулю.

При перемещении по миру машина может изменять направление движения и скорость. Ограничений на величину скорости не существует. Ускорение, а также угол поворота ограничены (см. таблицу).

Характеристика	Легковая машина	Грузовая машина
Максимальная загрузка	100	150
Расход бензина (в единицу времени)	1/9	1/6
Максимальное ускорение (1 / (единица времени)^2)	0.5	0.4
Максимальный угол поворота (радиан / единица времени)	0.08	0.06

Максимальное ускорение указано по абсолютной величине. Машина может задействовать тормоз (торможение — это фактически движение с максимально возможным ускорением, направленным противоположно скорости). Между машиной и дорогой существует трение.

Машина может столкнуться с другой машиной, с препятствием, со стенами мира. Все эти столкновения считаются абсолютно упругими. Массы легковой и грузовой машин

считаются равными. Взаимодействие машин с объектами игрового мира описано в следующем разделе.

Машина может выбросить «Шип» (если ранее она подобрала бонус типа «Шип»). «Шип» появляется на расстоянии 80 единиц от центра машины, в направлении, обратном направлению движения машины.

1.7 Взаимодействие машин с другими объектами

Считается, что два объекта взаимодействуют, если расстояние между их центрами не превосходит суммы их радиусов. В таблице ниже представлены результаты взаимодействия активной машины (как легковой, так и грузовой) с объектами игрового мира.

Объект	Результат взаимодействия машины с объектом
Машина (активная)  	Если машины одинаковых типов (обе легковые или обе грузовые), каждая из них теряет часть здоровья. Величина, на которую уменьшится здоровье, зависит от скоростей машин. Если сталкиваются легковая и грузовая машины, то каждая из них также теряет часть здоровья. Однако при этом вычисляется потеря здоровья для легковой машины, а потеря здоровья для грузовой машины составляет половину от потери легковой машины. В столкновении машин любых типов каждая из них утрачивает 10% груза, имеющегося у неё на борту. Уменьшение груза происходит следующим образом. Сначала вычисляется количество груза, которое будет утрачено. После этого происходит уменьшение на это количество груза типа 50. Если груза этого типа меньше, чем утрачиваемое количество груза, то его количество полностью обнуляется, а недостающее количество груза набирается за счет груза типа 30. Если и груза типа 30 не хватает, то утрачивается и часть груза типа 20.
Машина (неактивная)  	При столкновении машин активная машина теряет часть здоровья и 10% груза, имеющегося у неё на борту. Величина, на которую уменьшится здоровье, вычисляется, как если бы активными были обе машины. Но здоровье изменяется только у активной машины, неактивная машина не теряет здоровье. Также неактивная машина не утрачивает никакую часть своего груза.
Бонус «Бензин» 	Машина может добавить к имеющемуся у неё количеству бензина до 40 единиц. Если в бензобаке 60 и более единиц, подобрав бонус «Бензин», машина полностью заполнит бензобак (100).

Бонус «Шип» 	Подобрав бонус «Шип», машина может впоследствии в любой момент игры выбросить его (как правило, с целью нанести урон другим машинам). Если машина ранее уже подобрала бонус «Шип» и не выJ,бросила его, то ранее подобранный «Шип» выбрасывается автоматически при подборе нового.
Объект «Шип» 	Если машина наехала на объект «Шип», то она становится неактивной на 100 единиц игрового времени.
Станция погрузки 	Если машина находится на станции погрузки, то за одну единицу времени на неё загружается одна единица груза. Этот процесс прекращается либо по достижении машиной максимальной загрузки грузом данного типа, либо по завершении груза на станции. Машина может покинуть станцию погрузки в любой момент времени.
Станция разгрузки 	Если машина находится на станции разгрузки, за одну единицу времени с неё выгружается одна единица груза. При этом сначала полностью выгружается груз типа 50, потом груз типа 30, и после этого уже груз типа 20. Станция разгрузки принимает грузы любого типа в неограниченном количестве. Машина может покинуть станцию разгрузки в любой момент времени.
Станция ремонта 	Машина увеличивает на станции ремонта свое здоровье: 2 единицы здоровья за одну единицу времени. Здоровье не может стать больше 100.
Препятствие или стена (см. файл с анимацией)	Машина теряет часть здоровья; уменьшение здоровья зависит от её скорости. Также машина теряет 10% груза, имеющегося у неё на борту. Уничтожение груза происходит в том же порядке, который описан при столкновении с машиной.

Неактивная машина не может совершать каких-либо действий самостоятельно. Однако она может взаимодействовать с другими машинами, а также со станцией ремонта, если будет перемещена к ней в результате взаимодействия с другой машиной. В период неактивности ни груз, ни здоровье машины не изменяются.

Если неактивная машина взаимодействует с бонусом, бонус исчезает из игрового мира, но машина этот бонус не приобретает.

Если расстояние между активной машиной и неактивной машиной меньше, чем (сумма их радиусов + 15), то активная машина может отобрать у неактивной 10 единиц груза. Тип этих 10 единиц груза считается универсальным, а активная машина пополнит сначала свои запасы груза типа 20, потом (если груз типа 20 более не может быть помещен на активную машину) — запасы груза типа 30; если же и груз типа 30 более не может быть помещен на активную машину, пополняются запасы груза типа 50. Превысить грузоподъёмность машина не может.

Глава 2. Программирование машины

2.1 Обзор

На диске W: рабочей станции, которая будет предоставлена команде, будет содержаться папка Code Delivery (путь к папке будет объявлен дополнительно). В ней, в частности, Вы сможете найти документ README.txt, описывающий технические детали, связанные с созданием проектов и запуском клиентского приложения.

Также вам будут доступны Javadoc-файлы (они содержатся в последней главе настоящего руководства). Javadoc-файлы, доступные с рабочей станции, могут содержать более новую информацию, нежели печатное руководство. Настоятельно рекомендуем обращаться именно к последней версии Javadoc-файлов.

Ваша задача — дополнить реализацию класса *MyStrategy*. Вы можете ограничиться реализацией методов *init()* и *move()*. Вместе с тем вы можете добавлять в класс MyStrategy другие методы и поля.

Важное замечание. Не переименовывайте класс *MyStrategy* и не помещайте его в пакет (используйте пакет по умолчанию).

2.2 Инициализация

Когда ваша машина помещается в мир, вызывается метод *init()* класса *MyStrategy*. Вы можете поместить в данный метод любой код инициализации, который должен быть вызван. Однако этот код должен выполняться не более 0.25 секунды. Если ваш инициализационный код не успел завершиться вовремя или же его выполнение привело к ошибке (исключению — exception или ошибке времени выполнения — runtime error), ваша стратегия не будет участвовать в текущем матче. В момент инициализации игровой мир ещё не существует, поэтому вы не должны вызывать метод *getWorld()* в методе *init()*.

По умолчанию вашей стратегии присваивается имя — название Вашей команды. Чтобы присвоить стратегии другое имя, нужно во время инициализации машины использовать метод *getSelfControl().setName()*, передав ему желаемое имя в качестве параметра. Текущее ограничение на максимальную длину имени — 13 символов. Рекомендуем также обратиться к документации данного метода.

По умолчанию машина создаётся как легковая машина. Изменить тип машины при инициализации можно с помощью метода *getSefControl().setVehicleType()*.

2.3 Движение машины

После инициализации всех машин происходит вызов метода *move()* каждой машины в порядке очереди. Код в методе *move()* описывает действия, которые может совершать ваша стратегия.

В частности, машина может находить объекты в мире с помощью метода *getWorld()*, задавать скорость машины, используя возвращаемый методом *getSelfControl()* экземпляр интерфейса *SelfControl*. Более подробно об этих и других методах Вы можете прочесть в JavaDoc-файлах.

Каждую единицу игрового времени симулятор сначала опрашивает все машины, после чего симулирует заявленные машинами в их методах *move()* действия. При этом применяются все ограничения, введённые правилами. Например, если машина неактивна, то все заявленные ею действия будут проигнорированы.

Также отметим, что все заявленные машиной действия, должны быть выполнены не более чем за 0.25 секунды. Если же код не успевает завершиться вовремя или же его выполнение приводит к ошибке (исключению — exception или ошибке времени выполнения — runtime error), ваша стратегия прекращает участие в текущем матче.

2.4 Пример простого кода (Go to Center player)

Приведём пример реализации методов *init()* и *move()*.

```
import deliveryGame.interfaces.Enemy;
import deliveryGame.interfaces.Player;
import deliveryGame.interfaces.World;

public class MyStrategy extends Player {
    @Override
    public void init() {
        getSelfControl().setName("GoToCenter");
        getSelfControl().setVehicleType(Enemy.VehicleType._HEAVY_);
    }

    @Override
    public void move() {
        World world = getWorld();
        getSelfControl().move(1);
        getSelfControl().turnTo(world.getWidth()/2, world.getHeight()/2);
    }
}
```

2.5 Клиентское приложение

Информацию о запуске клиентского приложения (далее — Клиент) вы сможете найти документе README.txt в папке на рабочей станции. Для авторизации в Клиенте используйте выданные вам логин и пароль.

Клиент даёт вам возможность тестировать вашу стратегию локально и отправлять код стратегии на сервер. Для этого вы можете использовать вкладки «Локальный запуск» или «Глобальный запуск» Клиента.

Для локального тестирования вы можете использовать стратегии, написанные вами, а также несколько простых стратегий, код которых доступен с рабочей станции. Для этого на вкладке «Локальный запуск» выберите стратегии и запустите игру.

Также вы можете протестировать текущую версию своей стратегии на сервере. Когда вы отправляете с помощью вкладки «Глобальный запуск» вашу стратегию на сервер, Клиент загружает информацию обо всех последних версиях отправленных на сервер стратегий других команд. Вы можете запустить игру с любыми стратегиями (в том числе с вашей собственной).

Обратите внимание, что локальное тестирование позволяет вам сопоставлять несколько версий вашей стратегии, в то время как при глобальном тестировании вы можете тестировать только последние отправленные на сервер версии стратегий.

Результаты запущенного вами локального и глобального тестирования будут доступны только вам. Эти результаты не оказывают никакого влияния на положение вашей стратегии в турнире стратегий.

При глобальном тестировании компиляция стратегии происходит на сервере. Спустя небольшое время после отправки стратегии вы получите сообщение со статусом компиляции.

Напомним, что отправлять нужно только файл, содержащий класс MyStrategy. Весь код, который составляет вашу стратегию, должен находиться в одном файле. Также ваша стратегия не должна использовать какие-либо операции ввода / вывода (если вы ведёте лог, не забудьте перед отправкой на сервер закомментировать код, отвечающий за это).

В режиме локального тестирования вам также доступно **Ручное управление**. Вы можете выбрать цвет машины, за которую будете играть. В этом случае стратегия будет управляться не кодом, а ожидать команд с клавиатуры. Все ограничения, описанные в правилах, в этом случае также действуют.

Список клавиш для управления машиной приведён в таблице ниже.

Действие	Клавиша
Ехать вперёд	W
Ехать назад	S
Поворот налево	A
Поворот направо	D
Тормозить	Space
Бросить шип	I
Отобрать груз	O

Глава 3. JavaDoc

Classes

Class Player

```
java.lang.Object

      framework.interfaces.FrameworkPlayer

            deliveryGame.interfaces.Player
```

```
public abstract class Player
       extends framework.interfaces.FrameworkPlayer
```

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

Constructors

Player

```
public Player()
```

Methods

getSelfControl

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

Доступ к машине (игроку)

Returns:

- SelfControl объект, предоставляющий информацию о машине

getWorld

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

Доступ к информации об игровом мире и его составляющих.

Returns:

объект World

init

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

Метод инициализации машины (игрока).Задает значения полей selfControl и world, которые необходимы для управления машиной и получении информации об игровом мире. Участники не должны вызывать этот метод.

Parameters:

selfControl - объект, предоставляющий информацию о машине

world - объект предоставляющий информацию об игровом мире.

init

public abstract void **init**()

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 (0.25с)

move

public abstract void **move**()

Абстрактный метод перемещения. Метод должен быть переопределен в классе-наследнике. Вызывается, чтобы сделать следующий ход.

Чтобы управлять машиной в этом методе, используйте объект selfControl

Чтобы получить информацию об игровом мире, используйте объект world

Ограничения:

Время выполнения ограничено (0.25с)

Class BonusType

public enum **BonusType** implements Bonus.BaseBonusType

Типы бонусов в игровом мире:

бензин

шипы

Enum Constant Detail

public static final BonusType GASOLINE

public static final BonusType SPIKES

Class EffectType

public enum **EffectType** implements Effect.BaseEffectType

Типы эффектов которые могут быть применены к машине:

не способна двигаться, поскольку закончился бензин

не способна двигаться, поскольку наехала на шип

не способна двигаться, поскольку закончилось здоровье

Enum Constant Detail

public static final EffectType DISABLED

public static final EffectType SPIKES_DAMAGE

public static final EffectType BROKEN

Class Action

public enum **Action** implements FrameworkSelfControl.BaseAction

Типы действий, которые могут быть совершены игроком:
бросить шип
отобрать груз

Enum Constant Detail

public static final Action FIRE_SPIKES

public static final Action STOLE

Class Enemy.VehicleType

public static enum **Enemy.VehicleType**
implements FrameworkEnemy.BaseEnemyType

Возможные типы машин:
легковая
грузовая

Enum Constant Detail

public static final Enemy.VehicleType LIGHT

public static final Enemy.VehicleType HEAVY

Class WorkStation.Type

public static enum **WorkStation.Type** implements Station.BaseStationType

Возможные типы станций:
станция погрузки
станция разгрузки
станция ремонта

Enum Constant Detail

public static final WorkStation.Type PICKUP

public static final WorkStation.Type DROP

public static final WorkStation.Type REPAIR

Class PlayerType

public enum **PlayerType**

Возможные цвета игроков

Enum Constant Detail

public static final PlayerType RED

public static final PlayerType GREEN

public static final PlayerType YELLOW

public static final PlayerType BLUE

Interfaces

Interface Bonus

public interface Bonus extends Unit

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

Methods

getBonusType

Bonus.BaseBonusType **getBonusType**()

Метод возвращает тип бонуса. Существует два различных типов бонуса.

GASOLINE - Добавляет машине 40 пунктов бензина.
SPIKES - Машина получает шип.

Returns:

тип бонуса

Interface Effect

public interface **Effect**

Интерфейс описывает эффекты, демонстрирующие состояние машины.

Methods

getRemainingDuration

int **getRemainingDuration**()

Возвращает оставшееся время действия эффекта.

Returns:

- время действия в единицах игрового времени

getType

Effect.BaseEffectType **getType**()

Возвращает тип эффекта. Существует 3 типа эффектов.

эффект DISABLED - машина не может двигаться

Если эффект DISABLED применяется отдельно, без каких-либо других эффектов, то у машины кончился бензин.

Если эффект DISABLED применяется совместно с эффектом SPIKES_DAMAGE, то машина наехала на шип

Если эффект DISABLED применяется совместно с эффектом BROKEN, то у машины кончилось здоровье

Returns:

тип эффекта.

Interface Enemy

public interface **Enemy** extends FrameworkEnemy

Интерфейс предоставляющий доступ к информации о машине-противнике.

Methods

getName

java.lang.String **getName**()

Returns:

Имя машины-противника

getPlayerType

PlayerType **getPlayerType**()

Returns:

PlayerType машины-противника.

getEffects

java.util.List<Effect> **getEffects**()

Returns:

список эффектов примененных к машине-противнику.

hasEffectOfType

boolean **hasEffectOfType**(Effect.BaseEffectType **effectType**)

Проверяет есть ли некоторый эффект среди эффектов, примененных к машине-противнику.

Parameters:
effectType - проверяемый эффект.

Returns:

true если эффект применяется в настоящее время к машине-противнику, иначе false.

getCargoWeight

int **getCargoWeight**()

Returns:

Количество груза, на борту машины-противника.

getHealth

int **getHealth**()

Returns:

Текущее здоровье (количество HP) машины-противника

getVehicleType

Enemy.VehicleType **getVehicleType**()

Returns:

VehicleType машины-противника.

Interface WorkStation

public interface **WorkStation** extends Unit

Интерфейс предоставляющий информацию о станциях.

Methods

getAvailableCargo

int **getAvailableCargo**()

Returns:

количество груза на станции, доступного для погрузки

getRemainingTimeLife

int **getRemainingTimeLife**()

Returns:

оставшееся время жизни станции в единицах игрового времени

getFillingPercentage

int **getFillingPercentage**()

Returns:

часть (в процентах) от максимальной грузоподъёмности машины, которая может быть заполнена на станции (определяется типом груза на станции)

getStationType

Returns:

Тип станции.

Interface Shell

public interface **Shell** extends MovableUnit

Интерфейс описывающий снаряд Шип. Наследник MovableUnit

Methods

getType

Shell.BaseShellType **getType**()

Метод, возвращающий тип снаряда. В текущей реализации существует только один тип снаряда - шип (Spikes)

Returns:

возвращает тип снаряда

Interface Obstacle

public interface **Obstacle** extends Unit

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

Interface MovableUnit

public interface **MovableUnit** extends Unit

Интерфейс, описывающий объект, способный перемещаться в игровом мире. Объект характеризуется вектором скорости. В мире существует трение, поэтому скорость со временем снижается.

Methods

getHorizontalSpeed

float **getHorizontalSpeed**()

Возвращает горизонтальную составляющую скорости.

Returns:

проекция вектора скорости на ось Ox

getVerticalSpeed

float **getVerticalSpeed**()

Возвращает вертикальную составляющую скорости.

Returns:

проекция вектора скорости на ось Oy

getSpeed

float **getSpeed**()

Возвращает абсолютную величину скорости объекта

Returns:

длина вектора скорости.

getSpeedAngle

float getSpeedAngle()

Возвращает угол между направлением вектора скорости и осью Ox

Returns:

угол в радианах

Interface SelfControl

public interface **SelfControl** extends FrameworkSelfControl, Enemy

Интерфейс, позволяющий получить информацию о машине и управлять ею. Назначение:

В момент инициализации можно задать имя машины

Изменить ускорение машины

Изменить угол поворота машины

Тормозить

Выбросить шип

Methods

move

void **move**(double speed)

Задаёт ускорение машины.

Передаваемый в метод параметр может принимать значения от -1.0 до 1.0

В случае передачи параметра, превосходящего 1 по абсолютному значению, будет выполнена замена speed на Math.sign(speed)

Положительное значение ускорения приводит к росту скорости

Отрицательное значение ускорения приводит к уменьшению скорости. Применение отрицательного ускорения может привести к изменению направления скорости.

Нулевое значение не влечёт изменения скорости машины.

Ограничения: методы move() и brake() взаимоисключающие. При вызове обоих методов применяться будет тот, который вызван последним

Parameters:

speed - double значение ускорения .

setName

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

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

Parameters:
name - имя машины.

turn

void **turn**(float angle)

Задаёт угол поворота машины (от её текущего направления движения)

Parameters:
angle - double значение от *-pi* до *pi*.

turnTo

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

Задаёт угол поворота машины

Устанавливает угол поворота машины как угол между двумя векторами: вектором, направленным из центра машины вдоль направления её движения, и вектором, направленным из центра машины в точку с координатами (x, y)

Parameters:
x - координата x точки

y - координата y точки

turnTo

void **turnTo**(Unit unit)

Задаёт угол поворота машины

Устанавливает угол поворота машины как угол между двумя векторами: вектором, направленным из центра машины вдоль направления её движения, и вектором направленным из центра машины в центр unit.

Parameters:
unit - unit в направлении которого хотим повернуться.

getScore

int **getScore**()

Returns:

Количество очков, набранное игроком на момент запроса

getActionCoolDown

int **getActionCoolDown**(FrameworkSelfControl.BaseAction action)

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.

Returns:
Количество единиц игрового времени

fireSpikes

void **fireSpikes**()

Выбрасывает шип (если возможно)

stole

void **stole**()

Отбирает груз у другой машины (если возможно).

brake

void **brake**()

Выполняет торможение

Ограничение:

не может применяться одновременно с move(). Если вызываются методы move() и brake(), выполняться будет вызванный последним

hasSpike

boolean **hasSpike**()

Проверяет, имеется ли у игрока шип

Returns:

true при наличии шипа, иначе false

setVehicleType

void **setVehicleType**(Enemy.VehicleType vehicleType)

Устанавливает тип машины.

Parameters:

vehicleType - выбранный тип.

get50Cargo

int **get50Cargo**()

Returns:

Количество груза типа 50, уже находящегося на машине

get30Cargo

int **get30Cargo**()

Returns:

Количество груза типа 30, уже находящегося на машине

get20Cargo

int **get20Cargo**()

Returns:

Количество груза типа 20, уже находящегося на машине

getGasolineRemain

float **getGasolineRemain**()

Returns:

остаток бензина у машины

Interface Unit

public interface **Unit**

Базовый интерфейс для всех объектов игрового мира. Каждый объект характеризуется позицией и радиусом (все объекты мира 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.

Methods

getAngle

float **getAngle**()

Возвращает угол между исходным положением объекта и его текущим положением.

Исходное положение объекта задаётся вектором, направленным из центра объекта в его переднюю точку Текущее положение объекта задаётся вектором, направленным из центра объекта в его переднюю точку.

Returns:

угол в радианах от -PI до PI

getAngleTo

float **getAngleTo**(float x, float y)

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

Returns:

угол в радианах от -PI до PI

getAngleTo

float **getAngleTo**(Unit obj)

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.

Returns:

угол в радианах от -PI до PI

getDistanceTo

float **getDistanceTo**(float x,float y)

Возвращает расстояние от центра объекта до точки (x,y)

Parameters:

x - координата x точки

y - координата y точки

Returns:

расстояние до точки

getDistanceTo

float **getDistanceTo**(Unit obj)

Возвращает расстояние между центрами текущего объекта и объекта obj

Parameters:

obj - объект

Returns:

расстояние до объекта

getRadius

float **getRadius**()

Возвращает радиус объекта.

Returns:

радиус

getX

float **getX**()

Возвращает координату x объекта

Returns:

координата x

getY

float **getY**()

Возвращает координату y объекта

Returns:

координата y

Interface World

public interface **World** extends FrameworkWorld

Интерфейс, предоставляющий доступ к свойствам игрового мира.

Мир представляет собой прямоугольник размера getWidth() x getHeight()

Methods

getWidth

float **getWidth**()

Returns:

ширина прямоугольника, представляющего мир

getHeight

float **getHeight**()
FrameworkWorld

Returns:

высота прямоугольника, представляющего мир

getTick

int **getTick**()

Returns:

время, прошедшее от создания игрового мира

getBonuses

java.util.List<Bonus> **getBonuses**()

Returns:

Список бонусов, присутствующих в данный момент в игровом мире.

getShells

java.util.List<Shell> **getShells**()

Returns:

список снарядов (шипов), выпущенных игроками и присутствующих в данный момент в игровом мире

getEnemies

java.util.List<Enemy> **getEnemies**()

Returns:

Список противников, присутствующих в игровом мире.

getWorkStations

java.util.List<WorkStation> **getWorkStations**()

Returns:

Список станций, присутствующих в игровом мире.

getObstacles

java.util.List<Obstacle> **getObstacles**()

Returns:

Список препятствий в игровом мире.