Содержание

1 Игра

1.1 Общее описание

У каждого игрока есть в распоряжении автоматические заводы и боевые роботы. Заводы могут исключительно производить роботов. Роботы могут передвигаться и атаковать роботов противника. Также роботы могут захватывать нейтральные и вражеские заводы.

Игрок управляет только своими заводами. Он определяет, производить ли роботов на каждом из них, а также какую управляющую программу заложить в каждого робота. После того, как завершается производство очередного робота, он начинает действовать автономно. При этом действия робота определяются заложенной в него управляющей программой до тех пор, пока он не будет уничтожен (или до конца игры). Игроку в любой момент времени доступна внешняя информация о происходящем на поле, а каждому роботу — аналогичная информация о состоянии поля в некоторой окрестности.

1.2 Правила игры

Изначально каждый игрок контролирует один завод. Остальные заводы, находящиеся на карте, никому не принадлежат (нейтральны) в начале игры. Расположение заводов одинаково во всех играх.

Моделирование игры происходит по тактам. Длительность игры ограничена фиксированным количеством тактов. Если в какой-то момент игрок лишается всех своих роботов и заводов, то игра завершается досрочно. Он проигрывает и получает ноль очков, а его противник одерживает полную победу и получает 1 000 000 очков.

Когда заканчивается отведённое количество тактов, игра принудительно завершается. В этом случае каждый игрок получает количество очков, пропорциональное сумме стоимостей всех своих роботов и заводов к этому моменту. Коэффициент пропорциональности определяется таким образом, чтобы в сумме оба игрока получили 1 000 000 очков.

1.3 Игровая механика

Каждый завод и каждый робот имеют форму круга. Круги, соответствующие роботам, не могут пересекаться друг с другом. Завод может пересекаться с любым роботом (будем считать, что заводы находятся под землёй).

Каждый такт робот может сдвинуться на расстояние, не превосходящее его максимально допустимую скорость. При возникновении коллизий между роботами все виновники останавливаются в месте столкновения. Роботы не могут выходить за пределы квадратного игрового поля. На игровом поле введены координаты x и y, лежащие в диапазоне от 0 до `FIELD_SIZE`.

Робот может выстрелить из своего орудия в любую точку поля. Между двумя последовательными выстрелами должно пройти как минимум определённое количество тактов. Вылетевший из орудия снаряд летит равномерно и прямолинейно в заданную точку. Снаряд летит «навесом», поэтому ни с чем не сталкивается в пути (в частности, пролетает через роботов насквозь). Снаряд взрывается либо когда он долетает до цели, либо по прошествии фиксированного количества тактов.

У каждого робота есть силовой щит. Когда новый робот выходит с завода, его силовой щит полностью заряжен. Попаданием называется взрыв снаряда внутри круга, соответствующего роботу. От каждого попадания количество энергии уменьшается на фиксированное число — урон снаряда. Когда количество энергии становится неположительным, силовой щит отключается и робот уничтожается.

Если на территории завода находятся исключительно роботы игрока, которому этот завод не принадлежит, то постепенно осуществляется захват завода. На захват завода уходит значительное время. Скорость захвата до определённого предела зависит от количества роботов на территории завода. Если на территории завода находятся роботы нескольких игроков, то захват приостанавливается. Если же на территории нет никаких роботов, или есть только роботы хозяина, то прогресс по захвату завода постепенно теряется. При этом скорость потери также зависит от количества роботов хозяина.

На производство одного робота уходит фиксированное время. Если на территории завода есть вражеские роботы, то производство на нём приостанавливается. Когда производство робота завершается, он появляется на территории завода.

1.4 Управление заводами

Участник сдаёт программу управления заводами на одном из обычных языков программирования (`C++`, `Java`, `Pascal`). В этой программе должны быть реализованы две функции: `Strategy` и `Program`.

Функция `Strategy` вызывается каждый такт. В этой функции игрок может использовать информацию о текущем состоянии поля. В результате работы функции игрок должен установить для каждого завода, нужно ли строить на нём робота. Подсказка: скорее всего, нужно!

Функция `Program` вызывается каждый раз, когда на заводе игрока достраивается робот. В функцию передаётся индекс робота, которого требуется запрограммировать. В этой функции также доступна информация о состоянии поля. Игрок должен вернуть из функции информацию для инициализации робота. Эта информация состоит из двух строк: строки данных и инициализирующего скрипта. Любая из этих строк (или обе строки) может быть пустой. Суммарный размер передаваемой информации ограничен.

1.5 Управление роботами

Участник сдаёт программу управления роботами на языке `LUA`. Эта программа должна определять глобальную функцию `Think`.

Несмотря на то, что код управляющей программы одинаков для всех роботов, каждый робот имеет своё индивидуальное состояние интерпретатора LUA, то есть отдельный `lua_State`. Поэтому программа робота может делать всё, что позволяет ей язык `LUA`, никак не влияя на поведение программ других роботов. В частности, каждый робот может спокойно хранить свои индивидуальные данные в глобальных переменных.

Жизненный цикл робота выглядит так:

```
--initialization
data = {строка данных}
pcall ({инициализирующий скрипт})
pcall ({код программы управления роботами})
--running
while true do
```

```
  pcall (Think)
end
```

На фазе инициализации робота происходит следующее. Сначала в глобальную переменную записывается строка данных (результат функции `Program`). Затем выполняется инициализирующий скрипт (также результат `Program`). В последнюю очередь запускается общий код программы управления роботами.

Глобальная функция `Think` должна быть установлена в течение инициализации. Предполагается, что участник будет определять эту функцию внутри кода программы управления роботами. Функция `Think` вызывается для каждого робота каждый такт. Внутри неё робот может получать информацию об окружающем мире, используя Robot API. Также робот может устанавливать команды по перемещению и стрельбе. Замечание: на фазе инициализации Robot API недоступен.

2 Ограничения

Ограничения делятся на «мягкие» и «жёсткие».

Если игрок нарушил жёсткое ограничение, то происходит одно из двух:

1. Terminate — игра завершается полной победой противника, нарушитель получает ноль очков.
2. Disconnect — игра продолжается, но программа управления заводами нарушителя отключается.

На объём памяти, используемый программой управления заводами каждого игрока, установлено жёсткое ограничение в 256 мегабайт. За нарушение — Disconnect. На суммарный объём памяти всех роботов одного игрока также установлено жёсткое ограничение в 256 мегабайт. За нарушение — Terminate.

Мягкие ограничения бывают только по времени. Если игрок нарушил мягкое ограничение, то в последующий такт или такты какие-то из его функций не будут вызваны. Нарушение мягких ограничений является нормальным явлением, особенно для функции `Think` роботов.

Каждое мягкое ограничение задаётся параметрами T и S. В k-ый такт функция вызывается, если суммарное время всех её запусков к этому моменту меньше $S + Tk$. Таким образом, параметр T ограничивает среднее время работы на один такт, а S задаёт некоторый начальный запас. Ниже указаны ограничения по времени для различных функций.

Функция	Жёсткий TL	Действие при нарушении	Мягкий TL: T ($+S$)
Strategy	3 сек	Disconnect	1 мс (+ 500 мс)
Program	3 сек	Disconnect	1 мс (+ 500 мс)
Init (роботы)	3 сек	Terminate	1 мс
Think (роботы)	3 сек	Terminate	5 мс

Функция `Think` каждый такт запускается для роботов в случайном порядке. Если в какой-то момент определяется нарушение мягкого ограничения по времени для игрока, то для остальных его роботов `Think` не запускается. Таким образом, если происходит превышение мягкого ограничения по времени, то каждый такт думает лишь некоторое случайно выбранное подмножество роботов игрока.

Если в программе управления заводами происходит ошибка времени исполнения, то игра продолжается. Это эквивалентно действию Disconnect для игрока.

3 Система тестирования

Решением участника считаются два файла: первый содержит код управления заводами (на `C++`/`Pascal`/`Java`), а второй — код управления роботами на `LUA`.

При отправке решения в систему тестирования сдавать нужно оба файла. Каждый из них проходит проверку на компилируемость. Учтите, что в языке `LUA` использование неопределённой переменной или функции не является ошибкой компиляции, а регистр букв важен при именовании.

В решениях на `Java` основной класс должен называться `SolutionJava`, и он должен реализовывать интерфейс `SolutionInterface`. Вы можете называть класс по-другому локально, однако в системе тестирования решение с другим названием класса не будет компилироваться.

3.1 Дисквалификация

В решении запрещается:

1. **Создавать дополнительные нити исполнения.**
2. **Читать из файлов и писать в файлы.**

За нарушение этих правил вы можете быть дисквалифицированы.

Если вам нужен отладочный вывод, пишите его в файл `PlayerLog_%d.txt` в текущей директории. В имени файла вместо `%d` должен стоять номер вашего игрока (`playerIndex` = 0 или 1). Запись в файл с этим именем — единственное исключение из правил.

3.2 Регулярная посылка

Когда вы посылаете решение в систему тестирования, оно тестируется как регулярная посылка. При этом запускается игра вашего решения против некоторого решения жюри. Номер игрока для вашего решения выбирается случайным образом при каждой посылке. Результаты регулярной посылки показываются на вкладке «Результаты».

Для каждой посылки можно посмотреть исходный код обеих программ управления. Результат тестирования может быть либо `Compile error`, либо `Tested`. Поле `Score` показывает счёт игры в виде двух чисел. Одно из них показано жирным шрифтом — это очки вашего решения. Другое число — это очки программы жюри. В поле `Log` вы можете скачать логи игры, а также полную историю игры.

Обратите внимание, что при возникновении различных проблем (ошибка времени исполнения, зависание и т.д. и т.п.) результат вашей посылки будет всё равно `Tested`. Подробно о произошедшем можно узнать, просмотрев текстовые логи игрового сервера, а также клиента с вашим решением. Настоятельно рекомендуется просматривать эти логи после посылки.

На вкладке «Рейтинг» отображается результат вашей последней регулярной посылки. Нетрудно понять, что этот рейтинг не имеет большой значимости в плане определения силы вашего решения.

3.3 Промежуточный турнир

После одного, двух, трёх и четырёх часов от начала соревнования будет запускаться промежуточный турнир. О проведении каждого турнира будет предупреждение (минут за пять). В промежуточный турнир от каждого участника будет выбираться последнее успешно скомпилированное решение к моменту начала турнира.

Промежуточный турнир тестируется в течение примерно 50 минут. Турнир проводится по Швейцарской системе и состоит из туров. В каждом туре решения некоторым образом разбиваются на пары и играют друг с другом. Каждое решение играет ровно две игры в туре: по одной игре на каждое значение `playerIndex`. Количество туров в турнире не определено (на сколько времени хватит).

После завершения каждого тура обновляется турнирный рейтинг решений. В этом рейтинге команды ранжируются по суммарному количеству очков, набранных в прошедших турах турнира. Ссылка на турнирный рейтинг появится в новостях в системе тестирования после начала турнира.

Логи и истории игр промежуточного турнира можно скачать. Для этого нужно нажать на ссылку, которая будет выложена в новостях рядом со ссылкой на рейтинг турнира. По ссылке вы попадёте в директорию, в которой находятся логи всех ваших игр этого промежуточного турнира. При этом скорее всего понадобится ввести логин/пароль, совпадающий с вашим логином/паролем в системе тестирования `nsuts`.

3.4 Финальный турнир

Финальный турнир проходит после конца соревнования. В него забирается последнее скомпилировавшееся решение каждого участника.

Финальный турнир проводится аналогично промежуточному. По возможности мы проведём полный круговой турнир, то есть каждое решение сыграет с каждым. Турнирный рейтинг строится так же: команды ранжируются по сумме своих очков во всех играх финального турнира. Турнирный рейтинг финального турнира является итоговым рейтингом соревнования, то есть рейтингом первого дня олимпиады.

4 Архив материалов

В новостях в системе тестирования доступна ссылка для скачивания материалов соревнования (`materials.exe`). Пароль к архиву:

armvscore

А данном соревновании рекомендуется задавать вопросы. Жюри постарается ответить на все вопросы, на которые оно в состоянии ответить.

Предполагается, что работа с материалами будет вестить в командной строке. Должны быть прописаны пути к компилятору `Visual C++`. Самый простой способ — работать в `Far Manager`. На рабочем столе есть иконка запуска `Far`, внутри которого все необходимые пути уже прописаны. Вы можете редактировать ваше решение в любом редакторе или `IDE` на ваш вкус. Однако процесс сборки и запуска решений довольно нетривиален, и жюри предоставляет только набор `.bat`-скриптов для этого.

4.1 Содержимое архива

После распаковки архива рекомендуется собрать игровой сервер:

```
c_all.bat
```

Далее рекомендуется собрать пример решения на том языке, на котором вы хотите решать задачу:

```
c_sol.bat SolutionXXX.XXX
```

Затем запустить игру этого решения с самим собой вместе с glut-визуализатором:

```
run_glut.bat SolutionXXX.exe SolutionXXX.exe
```

И наконец, проиграть записанную историю этой игры на том же glut-визуализаторе:

```
run_history.bat
```

В архиве присутствуют:

файл или директория	описание содержимого
`c_sol.bat`	Скрипт сборки клиента-решения. Подробная инструкция по использованию ниже.
`run_console.bat`	Скрипт запуска консольного игрового сервера.
`run_glut.bat`	Скрипт запуска игрового сервера с включенным glut-визуализатором.
`run_history.bat`	Скрипт запуска glut-визуализатора в режиме проигрыша истории.
`c_all.bat`	Скрипт для сборки игрового сервера.
`w_all.bat`, `w_logs.bat`	Скрипты для очистки. Пользоваться необязательно.
`factories.pdf`	Это условие задачи.
`SolutionCpp.cpp`, `SolutionJava.java`, `SolutionPas.pas`	Примеры решений на трёх языках. Это файлы программ управления заводами.
`SolutionCpp.lua`, `SolutionJava.lua`, `SolutionPas.lua`	Код программ управления роботами для примеров решений. Содержимое этих трёх файлов совпадает.
`Client\...`	Код для сборки клиентов-решений.
`Client\Cpp\...`	Код для сборки решений на `C++`. При сборке ваше решение копируется в эту директорию и компилируется здесь. Здесь находится `DataTypes.h`, который используется в вашем решении.
`Client\Java\...`	Код для сборки решений на `Java`. При сборке ваше решение копируется в эту директорию. Здесь компилируются все классы, потом они копируются куда следует. `exe`-файл клиента настроен загружать класс с определённым именем.
`Client\Pas\...`	Код для сборки решений на `Pascal`. При сборке ваше решение копируется в эту директорию и собирается в `DLL`. `exe`-файл клиента настроен загружать `DLL` с определённым именем.
`Common\...`	Общий код для игрового сервера и клиентов-решений.
`LUA\...`	Полный исходный код `LUA 5.1.5`. Единственное отличие от официального дистрибутива — включена проверка `LUA-C API`.
`GameLogic\...`	Код игровой логики. Вы можете смотреть и изменять локально этот код, хотя делать это не рекомендуется.
`Server\...`	Код консольного сервера. Именно этот сервер работает в системе тестирования.

SimpleVis\...	Код простого визуализатора на GLUT. Этот визуализатор может либо запускать игру и показывать состояние в реальном времени, либо проигрывать историю игры.

4.2 Описание .bat-скриптов

c_sol.bat — скрипт сборки клиента-решения.
Принимает один параметр: имя файла с исходным кодом решения (программы управления заводами).
Например: c_sol.bat SolutionJava.java

Язык определяется автоматически по расширению файла (.cpp / .java / .pas). Собранный exe-файл клиента появляется в текущей директории с тем же именем, что и исходный файл. В случае языков Pascal и Java также появляются файлы .dll и .class. Эти файлы также имеют то же имя, что и файл решения.

run_console.bat, run_glut.bat — скрипты запуска игрового сервера. Каждый из этих скриптов принимает два параметра: имена exe-файлов двух собранных клиентов-решений.
Например: c_sol.bat SolutionJava.exe SolutionCpp.exe

При запуске каждое решение загружает LUA-код программы управления роботами из файла с тем же именем, но с расширением .lua. В примере первому решению будет соответствовать код SolutionJava.lua, а второму — SolutionCpp.lua.

Можно установить дополнительный параметр -d, чтобы включить отладочный режим. Этот параметр должен быть последним в командной строке. В этом режиме не проверяются ограничения по времени, памяти, размеру истории, а в LUA-коде роботов можно свободно пользоваться всеми стандартными библиотеками без ограничений.

run_history.bat — скрипт запуска glut-визуализатора в режиме проигрыша истории. Принимает один параметр: имя .bin-файла с историей игры. Если параметр отсутствует, то история загружается из файла History.bin, расположенного в текущей директории.

4.3 Документация

Отдельно доступна некоторая документация по языку LUA. Вы можете скачать её по ссылке в новостях системы тестирования (docs.exe). Если вы незнакомы с языком, рекомендуется просмотреть LuaIn15Minutes.txt. Также доступен Reference Manual и второе издание книги “Programming in LUA”, но, к сожалению, только на английском.

5 Strategy API

В программе управления заводами требуется реализовать функции Strategy и Program. В каждой из них есть полный доступ на чтение данных игрового мира WorldData. Подробное описание структуры WorldData приведено ниже. Замечание: не все первоначально предполагаемые возможности вошли в окончательную версию игры, поэтому может казаться, что некоторые поля не имеют смысла.

Функция Strategy возвращает пары <идентификатор завода, класс производимого робота>. Каждая такая пара воспринимается как приказ заводу вести себя в дальнейшем заданным образом. Если нужно строить робота, то класс должен быть равен нулю. Если не нужно, то минус единице.

Функция Program принимает индекс робота, которого требуется запрограммировать. Ниже описано, что такое индекс. Функция должна возвращать две строки. Первая строка (строка данных) будет записана в глобальную переменную data в состоянии робота. Вторая строка (инициализирующий скрипт) будет выполнена как LUA-код. Суммарный размер обеих строк не должен превышать 1024 байта.

5.1 Языки

В приведённой ниже таблице кратко описаны различия в API на разных языках.

Элемент API	Описание в языках

Сигнатура функции `Strategy`	`std::map<int, int> Strategy();`
	`Map<Integer, Integer> strategy();`

	`procedure Strategy();`
	команды заводов устанавливаются при помощи:
	`SetBuildAt(id, classIdx : longint);`

Сигнатура функции `Program`	`std::pair<std::string, std::string> Program(int robotIndex);`
	`ProgramResult program(int robotIndex);`

	`procedure ProgramRobot(index : longint;`
	`var data : pchar; var dataLen : longint;`
	`var prog : pchar; var progLen : longint);`

Номер игрока (0 или 1)	`extern int playerIndex;` (Solution.h)
	`SolutionInterface.playerIndex` (SolutionInterface.java)
	`var playerIndex : longint;` (DataTypes.pas)
Данные `WorldData`	`extern const WorldData *pWorld;` (Solution.h)
	`public class WorldData` (класс со статическими методами)
	`var world : ptrWorldData;` (DataTypes.pas)
Описание `WorldData`	находится в DataTypes.h
	разбросано по файлам: WorldData.java, Constants.java, CoordsXY.java, Unit.java, Factory.java, Robot.java, Projectile.java, FactoryClass.java, RobotClass.java, ProjectileClass.java
	находится в DataTypes.pas
Пример использования API	`const Robot &robot = pWorld->robots[i];` `double x = robot.position.x;`
	`int robot = WorldData.robot(i);` `double x = CoordsXY.x(Unit.position(robot));`
	`x := world^.robots[i].position.x;`

5.2 WorldData

В первую очередь нужно ввести некоторые термины, используемые в `WorldData`.

Юнит — это единица игрового мира. Юнит может быть одного из трёх типов: завод, робот или снаряд. У каждого юнита есть идентификатор (ID) — неотрицательное целое число. Идентификатор не меняется на протяжении всей жизни юнита и уникален на протяжении всей игры.

Все юниты одного типа лежат в одном массиве (factories, robots и projectiles соответственно). Номер юнита в этом массиве называется индексом. Индекс юнита не меняется на протяжении всей его жизни. В каждом массиве есть «мёртвые» элементы, которые не соответствуют никакому реальному юниту. Они отмечены особым образом: у них идентификатор юнита равен -1.

Класс — это общие характеристики всех схожих юнитов одного типа. Изначально предполагалось, что можно будет строить роботов разных видов, но потом от этого отказались. Поэтому в рамках текущего соревнования все юниты одного типа имеют одинаковый класс. Номер класса всегда равен нулю. Информация о данных класса содержится в factoryClasses, robotClasses, projectileClasses соответственно. Всё это массивы единичного размера.

имя поля	значение
`Unit`	Набор данных, которые есть у каждого юнита.
`Unit::id`	Идентификатор юнита. Сервер выдаёт юнитам идентификаторы в порядке их создания, начиная с нуля. Таким образом, идентификатор юнита не меняется на протяжении жизни и уникален на протяжении игры. Выделенное значение $id = -1$ означает, что данный юнит «мёртв», то есть реально не существует.
`Unit::owner`	Номер игрока-хозяина юнита. Игроки имеют номера 0 и 1. Для нейтрального завода номер хозяина равен -1.
`Unit::type`	Тип юнита. Равен 1 для завода, 2 для робота и 3 для снаряда.
`Unit::position`	Текущее положение юнита. Состоит из компонент X и Y, каждая из которых лежит в диапазоне $[0; \texttt{FIELD\_SIZE}]$.

`Unit::radius`	Радиус юнита. У роботов этот радиус используется при определении коллизий и попаданий снарядов. Для заводов это радиус их территории. У снарядов радиус всегда равен нулю.
`Unit::classIdx`	Неиспользуемое значение, всегда равно нулю.
`Factory`	Набор данных завода (включает данные юнита).
`Factory::robotClassIdx`	Что строит завод. Равно 0, если завод производит робота, и -1 — если не производит.
`Factory::buildTimeRemaining`	Сколько времени осталось до завершения текущего производства. Если заводу не мешают вражеские юниты, то новый робот появится через кол-во тактов, равное этому числу.
`Factory::capturingPlayer`	Номер игрока, захватывающего завод. Если завод сейчас никто не захватывает, то равен номеру хозяина.
`Factory::captureProgressTime`	Текущий прогресс по захвату завода. Если это значение достигнет `timeToCapture`, то завод перейдёт в игроку-захватчику.
`FactoryClass`	Набор данных, одинаковых для всех заводов.
`FactoryClass::radius = 50.0`	Радиус любого завода.
`FactoryClass::timeToCapture = 5000`	Сколько тактов нужно для захвата завода одному роботу. Точнее сколько робото-тактов нужно для захвата завода.
`FactoryClass::maxCapturingRobots = 10`	Сколько максимум роботов могут одновременно участвовать в захвате завода. На территории может находится больше роботов противника, но «лишние» роботы не увеличивают скорость захвата.
`FactoryClass::score = 100`	Стоимость завода. При завершении игры за каждый завод, которым владеет игрок, ему к очкам добавляется это число.
`Robot`	Набор данных робота (включает данные юнита).
`Robot::health`	Текущий заряд силового щита робота. Заряд уменьшается от попаданий снарядов. Робот уничтожается, когда это число становится неположительным.
`Robot::shootTimeRemaining`	Сколько ещё тактов будет перезаряжаться орудие робота. В течение этого количества тактов робот не может стрелять.
`Robot::ammoRemaining`	Неиспользуемое значение, всегда равно -1.
`RobotClass`	Набор данных, одинаковый для всех роботов.
`RobotClass::radius = 5.0`	Радиус робота.
`RobotClass::visibilityRadius = 50.0`	Радиус видимости робота. В `Robot API` доступны только те юниты, расстояние до которых меньше этого числа.
`RobotClass::maxHealth = 10`	Максимальный заряд силового щита робота. У любого вновь построенного робота заряд силового щита равен этому числу.
`RobotClass::maxSpeed = 1.0`	Максимальная скорость робота. Любой робот в течение одного такта может переместиться на расстояние, не превосходящее этого числа.
`RobotClass::projectileClassIdx = 0`	Неиспользуемое значение.
`RobotClass::shootTimeCooldown = 5`	Минимальное количество тактов между двумя последовательными выстрелами робота.
`RobotClass::totalAmmo = -1`	Неиспользуемое значение.
`RobotClass::timeToBuild = 50`	Количество тактов, необходимое для того, чтобы построить робота на заводе.
`RobotClass::score = 1`	Стоимость робота. При завершении игры за каждого робота, которым владеет игрок, ему к очкам добавляется это число.
`Projectile`	Набор данных снаряда (включает данные юнита).
`Projectile::velocity`	Вектор скорости снаряда. Каждый такт снаряд смещается на этот вектор. Вектор скорости постоянен на протяжении жизни снаряда, его норма равна `ProjectileClass::speed`.
`Projectile::homingTargetId`	Неиспользуемое значение.

`Projectile::lifeTimeRemaining`	Сколько тактов осталось лететь снаряду. Это число может быть дробным, в таком случае снаряд упадёт посреди такта. Такая ситуация обрабатывается, хотя неточно.
`ProjectileClass`	Набор данных, одинаковый для всех снарядов.
`ProjectileClass::damage = 1`	Урон от попадания снаряда. В результате попадания в робота у него вычитается это число из заряда силового щита.
`ProjectileClass::isHoming = false`	Неиспользуемое значение.
`ProjectileClass::speed = 4.0`	Скорость полёта любого снаряда.
`ProjectileClass::maxLifeTime = 15.0`	Максимальное время полёта снаряда. Фактически, дальность полёта снаряда не превышает `maxLifeTime * speed`.

`WorldData::currentTime`	Сколько прошло тактов с начала игры.
`WorldData::maximalMatchTime`	Сколько тактов длится игра. Можно одержать досрочно полную победу, захватив все заводы противника и уничтожив всех его роботов.
`PLAYERS_COUNT`	Количество игроков в игре, всегда равно 2.
`FIELD_SIZE`	Размер игрового поля.
`FACTORY_CLASS_COUNT`, `ROBOT_CLASS_COUNT`, `PROJECTILE_CLASS_COUNT`	Количество классов для каждого типа юнитов всегда равно 1.
`MAX_FACTORY_COUNT`	Ограничение сверху на количество заводов. Множество заводов не изменяется на протяжении игры, могут лишь меняться хозяева.
`MAX_ROBOT_COUNT`	Ограничение сверху на количество роботов. Каждый игрок может иметь `MAX_ROBOT_COUNT`/2 роботов максимум. Когда этот лимит достигнут, новые роботы не выходят с заводов игрока, пока будет уничтожен какой-нибудь его робот.
`MAX_PROJECTILE_COUNT`	Ограничение сверху на количество снарядов. Просьба не пытаться его достичь: может возникнуть проблема с неожиданно большим размером истории игры.
`MAX_UNITS_COUNT`	Ограничение сверху на количество юнитов.

6 Robot API

У каждого робота есть глобальный объект `api`. Этот объект доступен только при вызове функции `Think`. Используя методы этого объекта, можно получать информацию о юнитах в видимой области и устанавливать команды.

При каждом запуске `Think` все юниты, видимые роботом, занумерованы от 1 до n. Эта нумерация выбирается произвольно и может изменяться от вызова к вызову. Сам робот имеет выделенный номер 0.

метод Robot API	эквивалент WorldData	значение
`api:count()`	*(нет)*	количество видимых юнитов исключая себя (то есть n)
`api:time()`	`currentTime`	текущий номер такта
`api.field_size`	`FIELD_SIZE`	размер игрового поля (поле квадратное)
`api:move_by_delta(dx, dy)`	*(нет)*	*(команда робота)*: сдвигаться на вектор (dx, dy) в такт; норма вектора автоматически уменьшается до скорости робота, если нужно
`api:shoot_stop()`	*(нет)*	*(команда робота)*: не стрелять
`api:shoot_at_pos(x, y)`	*(нет)*	*(команда робота)*: стрелять по точке с координатами (x, y)
`api:id(k)`	`id`	идентификатор юнита (здесь и далее k — номер видимого юнита)
`api:owner(k)`	`owner`	номер игрока-хозяина
`api:type(k)`	`type`	тип юнита (завод/робот/снаряд)

api:position(k)	position	координаты юнита; возвращает два числа; пример использования: myX, myY = api:position(0)
api:radius(k)	radius	радиус юнита
api:health(k)	health	заряд силового щита робота
api:shoot_time_remaining(0)	shootTimeRemaining	сколько ещё тактов нельзя стрелять; работает только для себя (то есть вызывающего робота)
api:max_health(k)	maxHealth	максимальный заряд силового щита робота
api:visibility_radius(k)	visibilityRadius	радиус видимости робота
api:shoot_time_cooldown(k)	shootTimeCooldown	минимальное количество тактов между последовательными выстрелами робота
api:max_speed(k)	maxSpeed	максимальная скорость робота
api:projectile_speed(k)	speed	скорость полёта снарядов, которыми стреляет заданный робот
api:projectile_damage(k)	damage	урон от снарядов, которыми стреляет заданный робот
api:velocity(k)	velocity	вектор скорости снаряда; возвращает два числа аналогично api:position
api:damage(k)	damage	урон от снаряда
api:max_life_time(k)	maxLifeTime	максимальное время полёта снаряда
api:speed(k)	speed	скорость полёта снаряда

Команды передвижения и стрельбы сохраняются в памяти робота. Далее робот выполняет их каждый такт до тех пор, пока ему не дадут другие команды. В некоторых ситуациях команды сбрасываются, например при превышении ограничения по времени или при попадании вражеского снаряда. Поэтому рекомендуется в конце каждого вызова Think устанавливать актуальные приказы по перемещению и стрельбе.

7 Логи

В результате игры образуются некоторые файлы в текущей директории.

создаваемый файл	Содержимое файла.
score.txt	Два целых числа: количество очков, которое набрал каждый игрок. Сначала записано кол-во очков нулевого игрока, затем первого.
ServerLog.txt	Текстовый лог игрового сервера. В этот файл записываются все проблемы, связанные с программами роботов. Также здесь отражаются сообщения о превышении ограничений.
ClientLog_%d.txt	Текстовый лог клиента-решения. В этот файл записываются все проблемы, возникшие на стороны клиента. В основном это относится к неправильному поведению программы управления заводами.
JavaExceptions_%d.txt	Файл с распечаткой исключений программы на Java. При нормальном поведении программы управления заводами этот файл должен быть пустым.
log.txt	Дополнительный текстовый лог клиента. Этот файл возникает только в том случае, когда клиент не может узнать имя игрока у сервера. Скорее всего, вы никогда не увидите этот файл.
History.bin	Бинарная история игры. Вы можете проиграть её при помощи run_history.bat. Этот файл может быть довольно большим.

Установлено ограничение на размер бинарной истории: 10 килобайт на кадр в среднем. Если на поле много юнитов и это ограничение превышается, то в истории пропускаются кадры. При проигрывании истории это может выглядеть как подтормаживание. Если во время соревнования возникнут проблемы со скачиванием историй, лимит может быть уменьшен.

7.1 Расшифровка сообщений

Ниже описаны сообщения, которые вы можете увидеть в текстовых логах клиента и сервера.

Too many messages of this type. They will be suppressed in future.

Некоторые проблемы могут возникать очень много раз. Поэтому сообщения каждого типа перестают

выдаваться после 10000 штук.

`Cannot read main robot program from file \"%s\"!`

Не удалось прочитать код программы управления роботами из указанного файла. Проверьте, что файл с заданным именем существует в той же директории, в которой лежит `.exe`-файл клиента-решения.

`Error loading script \"%s\" (owner = %d, id = %d):`

Не удалось скомпилировать инициализирующий скрипт или программу управления роботами. Указан идентификатор и номер хозяина проблемного робота. В следующей строке лога указано сообщение об ошибке от компилятора `LUA`.

`Error running script \"%s\" (owner = %d, id = %d):`

Возникла ошибка при выполнении инициализирующего скрипта или программы управления роботами. В следующей строке лога указано сообщение об ошибке от интерпретатора `LUA`.

`Function Think is not set (owner = %d, id = %d)`

После инициализации робота не установлена глобальная переменная `Think`. Проверьте, что в коде программы управления роботами объявлена эта функция без префикса `local`.

`Error on Think (owner = %d, id = %d):`

Возникла ошибка при выполнении функции `Think`. В следующей строке лога указано сообщение об ошибке от интерпретатора `LUA`.

`Player tries to set command for a wrong unit (id = %d)!`

Программа управления заводами неверно выставила параметр «идентификатор завода» в функции `Strategy`. Неправильный идентификатор указан в сообщении. Этот идентификатор должен совпадать с индексом завода, которым владеет игрок.

`Player tries to build an unknown robot (class = %d)!`

Программа управления заводами неверно выставила параметр «номер класса робота» в функции `Strategy`. Этот параметр должен быть равен 0, если строить робота нужно, и -1 — если не нужно.

`Duplicate factory command ignored (id = %d)`

Программа управления заводами указала несколько команд для одного завода. Все кроме одной из них проигнорированы. Идентификатор завода указан в сообщении.

`Program for robot (id = %d) is too large: %d + %d = %d !`

Программа управления заводами указала данные для инициализации робота, размер которых превосходит ограничение. Указан идентификатор инициализируемого робота, а также размеры строки данных, инициализирующего скрипта (и суммарный размер). Суммарный размер данных при инициализации не должен превосходить 1024 байта.

`Player %d TLE: Strategy request skipped`

Функция `Strategy` указанного игрока не была вызвана из-за превышения мягкого ограничения по времени.

`Player %d TLE: Program request skipped`

Функция `Program` указанного игрока не была вызвана из-за превышения мягкого ограничения по времени.

`Player %d TLE: Init finished only for %d of %d robots`

При инициализации роботов было превышено мягкое ограничение по времени. Указан номер игрока, количество успешно проинициализированных роботов и общее количество роботов, нуждавшихся в инициализации.

`Player %d TLE: Think finished only for %d of %d robots`

Превышено мягкое ограничение по времени при запуске функции `Think` роботов. Указан номер игрока, который превысил ограничение, а также количество успешно подумавших роботов (в этот такт) и общее количество роботов игрока. Каждый такт роботы игрока случайным образом переупорядочиваются, поэтому даже в ситуации превышения ограничения по времени каждый робот будет думать время от времени.

`Hard time limit exceeded on Strategy (player = %d)!`

Программа управления заводами превысила жёсткое ограничение по времени в функции `Strategy`. Номер виновного игрока указан. Вероятнее всего, программа просто зависла.

`Hard time limit exceeded on Program (player = %d)!`

Программа управления заводами превысила жёсткое ограничение по времени в функции `Program`. Номер виновного игрока указан. Вероятнее всего, программа просто зависла.

`Hard time limit exceeded on Init:%s (owner = %d, id = %d)!`

Превышено жёсткое ограничение по времени при запуске инициализирующего скрипта или программы управления роботами. Указан номер виновного игрока, а также идентификатор робота. Вероятнее всего, инициализация робота просто зависла.

`Hard time limit exceeded on Think (owner = %d, id = %d)!`

Превышено жёсткое ограничение по времени при запуске функции `Think` робота. Вероятнее всего, робота призадумался и завис.

`Player %d: Memory limit exceeded (client)!`

Программа управления заводами указанного игрока превысила ограничение по объёму памяти.

`Player %d: Memory limit exceeded (robots)!`

Программа управления роботами указанного игрока превысила ограничение по объёму памяти.

`Cannot write %d bytes to pipe (%s)!`, `Cannot read %d bytes from pipe (%s)!`

Если это сообщение в логе сервера, значит один из клиентов-решений внезапно завершился. Скорее всего, в решении произошла ошибка времени исполнения. Если это сообщение в логе клиента, значит внезапно завершился игровой сервер. Может быть, вы его закрыли?...

`Server terminated prematurely!`, `Client terminated prematurely!`

Это сообщение говорит о том, что игровой сервер или решение-клиент завершился ненормальным образом. Нормальное завершение — это когда заканчивается время игры или кто-нибудь одерживает полную победу.

`Client (%d; %u) disconnected!`

Решение-клиент принудительно завершён. Первое число — номер виновного игрока, второе — номер убитого процесса. Обычно такое отсоединение происходит при превышении какого-либо жёсткого ограничения.

`Player %d is disqualified!`

Игрок с указанным номером получает ноль очков за эту игру. Его противник, соответственно, автоматически одерживает полную победу. Это происходит при превышении некоторых жёстких ограничений. Поскольку из-за превышения этих жёстких ограничений дальнейшее моделирование игры не представляется возможным, то сервер принудительно завершается, а виновный получает ноль очков.