С авторами можно связаться по адресу: mail@avrorasystems.com, "для Шалыто".

Туккель Н.И., Шалыто А.А.

Система управления танком для игры RoboCode

Объектно-ориентированное программирование с явным выделением состояний

Программная документация

Санкт-Петербург
2002

Содержание

Введение

Для алгоритмизации и программирования задач логического управления была предложена SWITCH-технология, которая в дальнейшем была развита применительно к разработке программного обеспечения событийных систем (www.softcraft.ru).

До последнего времени в SWITCH-технологии объединялись два стиля программирования — процедурный и автоматный. При этом авторов интересовал вопрос совмещения автоматного и объектно-ориентированного стилей.

Первый шаг в этом направлении был выполнен авторами при программировании в объектно-ориентированной среде быстрой разработки "Flora" ("Использование SWITCH-технологии при разработке программ в среде Flora/C+. Модель технологического процесса в цехе холодной прокатки. Программная документация", www.softcraft.ru). При этом автоматы, описывающие основные алгоритмы функционирования системы, были выделены как отдельные объекты среды разработки.

Настоящая работа представляет дальнейшее развитие SWITCH-технологии, заключающееся в совместном использовании объектно-ориентированного и автоматного стилей программирования. Разработанный при этом подход может быть назван "объектно-ориентированное программирование с явным выделением состоянием". В рамках этого подхода программирование выполняется на объектно-ориентированном языке. При этом необходимо отметить, что основное внимание в настоящей работе уделяется проектированию рассматриваемого класса программ с применением схем и диаграмм, отличающихся от используемых в универсальном языке объектно-ориентированного моделирования UML.

Предлагаемая технология иллюстрируется на примере игры для программистов, целью которой является создание виртуальных роботов, обладающих средствами обнаружения и уничтожения противника. Каждая из подобных игр имеет специфику, определяемую средой исполнения, правилами проведения боев и используемым языком программирования.

Исходя из целей настоящей работы, нас интересовала такая игра, которая в качестве языка программирования использовала бы не специализированный язык, а один из широко распространенных объектно-ориентированных языков. Единственной известной авторам игрой, отвечающей этому требованию, является недавно (в 2001 году) разработанная компанией "Alphaworks" игра "Robocode" (http://robocode.alphaworks.ibm.com).

Эта игра позиционируется фирмой IBM как средство для обучения программированию на языке Java на примере создания программы, являющейся системой управления танком, предоставляемым средой исполнения.

Настоящая работа содержит документацию, созданную при проектировании программы управления танком.

История создания танка состоит из двух частей. Сначала был создан **нестреляющий** танк (counterwallrobot.Cynical_1), который в течении нескольких дней (с 28.09.01 до 02.10.01) был лучшим на открытом турнире, практически ежедневно проводимом создателем сайта http://robocode.isbeautiful.org. Начиная с 15.10.01, он стал стреляющим (counterwallrobot.Cynical_2), и занимал 5-6 место в мире. Разработка следующей версии, рассматриваемой в данной работе (counterwallrobot.Cynical_3), была приостановлена в связи с

написанием данной работы, в рамках которой создаваемый танк выполнил свое предназначение.

1. Постановка задачи

Целью настоящей работы является создание программы управления танком, отвечающей описанным в документации на игру требованиям среды исполнения, внешний вид которой в процессе боя двух танков приведен на рис. 1.

Рис. 1. Фрагмент боя

2. Описание подхода

После завершения создания программы, предлагаемый подход может быть сформулирован (по крайней мере для полного ее документирования) как "идеальная" технология, фиксирующая принятые решения.

1. На основе анализа предметной области выделяются классы и строится диаграмма классов, отражающая, в основном, наследование и агрегирование (например, вложенность).

2. Для каждого класса разрабатывается словесное описание, по крайней мере, в форме перечня решаемых задач.

3. Для каждого класса создается его структурная схема, несколько напоминающая карту CRC (Class—Responsibility—Collaboration, Класс—Ответственность—Кооперация). Нотация, используемая при построении структурных схем классов, приведена на рис. 2. Отметим, что в общем случае классу нет необходимости знать о вызывающих его объектах, а все вызываемые объекты не обязательно должны быть указаны, как, впрочем, и стрелки на концах линий.

Интерфейс класса образуют открытые (public) атрибуты и методы, к которым обращаются другие объекты. Эти методы, в свою очередь, вызывают закрытые (private) методы рассматриваемого класса.

Как открытые, так и закрытые методы, в общем случае могут передавать сообщения другим объектам.

Отметим, что открытые и закрытые атрибуты в тексте программы для удобства внесения изменений объявляются вместе, несмотря на то, что в нотации на рис. 2. они изображены отдельно.

Закрытые методы могут быть разделены на две части: автоматные и остальные. Автоматные методы в общем случае делятся на три разновидности: методы, реализующие автоматы; методы, реализующие

входные переменные автоматов; методы, реализующие выходные воздействия автоматов.

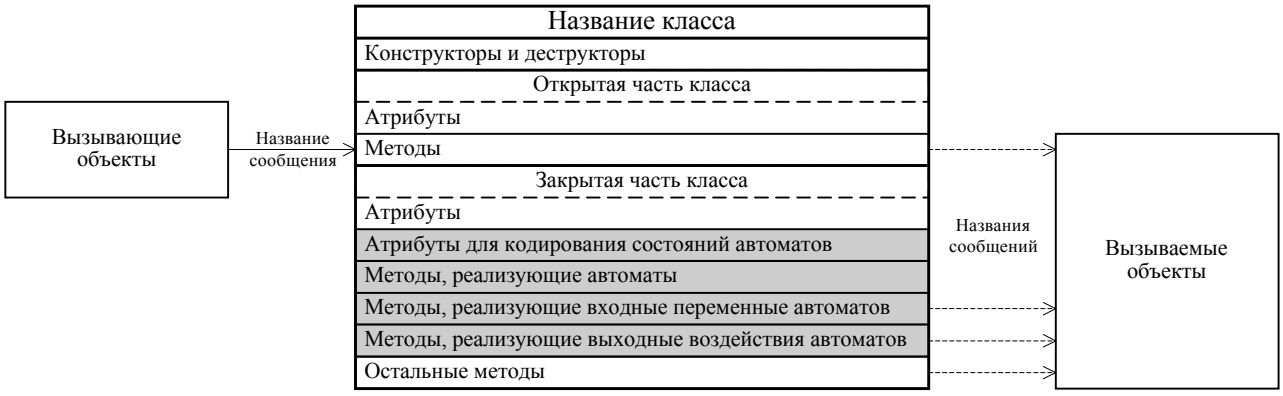

Рис. 2. Нотация, используемая при построении структурных схем классов

4. При наличии в классе нескольких автоматов строится схема их взаимодействия.

5. Для каждого класса составляются перечни событий, входных переменных и выходных воздействий.

6. Для каждого автомата разрабатывается словесное описание.

7. Для каждого автомата строится схема связей, определяющая его интерфейс, входные воздействия которого состоят из входных переменных, предикатов, проверяющих номера состояний и событий. В этой схеме в качестве событий (наряду с другими) могут быть указаны сообщения, получаемые объектом и приведенные в структурной схеме класса.

В схеме связей показываются все события, с которыми автомат запускается (вызывается), вне зависимости от того, используются ли они в пометках дуг и петель графа переходов.

В отличие от структурной схемы класса, на входах и выходах в схеме связей автомата указываются не названия сообщений, а названия соответствующих входных и выходных воздействий.

8. Для каждого автомата строится граф переходов.

9. Для каждого класса разрабатывается соответствующая ему часть программы в целом. Ее структура должна быть изоморфна структурной схеме класса, а методы, соответствующие автоматам, реализуются по шаблону. При этом методы, соответствующие входным переменным и выходным воздействиям автомата, располагаются отдельно от метода, реализующего автомат, из которого они только вызываются.

Благодаря такой реализации, методы, соответствующие входным переменным и выходным воздействиям автомата, могут быть абстрактными или полиморфными, и переопределяться в порождаемых классах. Таким образом, имеется возможность создать базовый класс для управления некоторой группой устройств, в котором реализуются только автоматы, а используемые ими входные переменные и выходные воздействия переопределяются на уровне порождаемых классов, управляющих конкретными устройствами.

10. Для изучения поведения программы, определяемого, в том числе, и взаимодействием объектов, а в ходе разработки — для ее отладки, **автоматически** строятся протоколы, описывающие работу всех автоматов в терминах состояний, переходов, входных переменных и выходных воздействий с указанием соответствующих объектов. Это обеспечивается за счет включения функций протоколирования в

соответствующие методы. При необходимости могут протоколироваться также и аналоговые параметры. Тот факт, что входные и выходные воздействия "привязаны" к объектам, автоматам и состояниям упрощает понимание таких протоколов по сравнению с протоколами, которые строятся традиционно. Из рассмотрения протоколов следует, что автоматы (в отличие от классов) абстракциями не являются. При этом можно утверждать, что автоматы структурируют поведение, которое с их помощью описывается весьма компактно и строго.

11. Выпускается созданная в ходе проекта документация.

Как отмечалось выше, предложенная технология является "идеальной", поэтому при создании реальных проектов, некоторые из указанных документов могут быть упрощены или исключены вовсе.

3. Диаграмма классов

Первым разрабатываемым документом является диаграмма классов, в верхней части которой расположены предоставляемые разработчику классы (в том числе из стандартной библиотеки) и среда исполнения, а в нижней части — классы, созданные при проектировании программы (рис. 2).

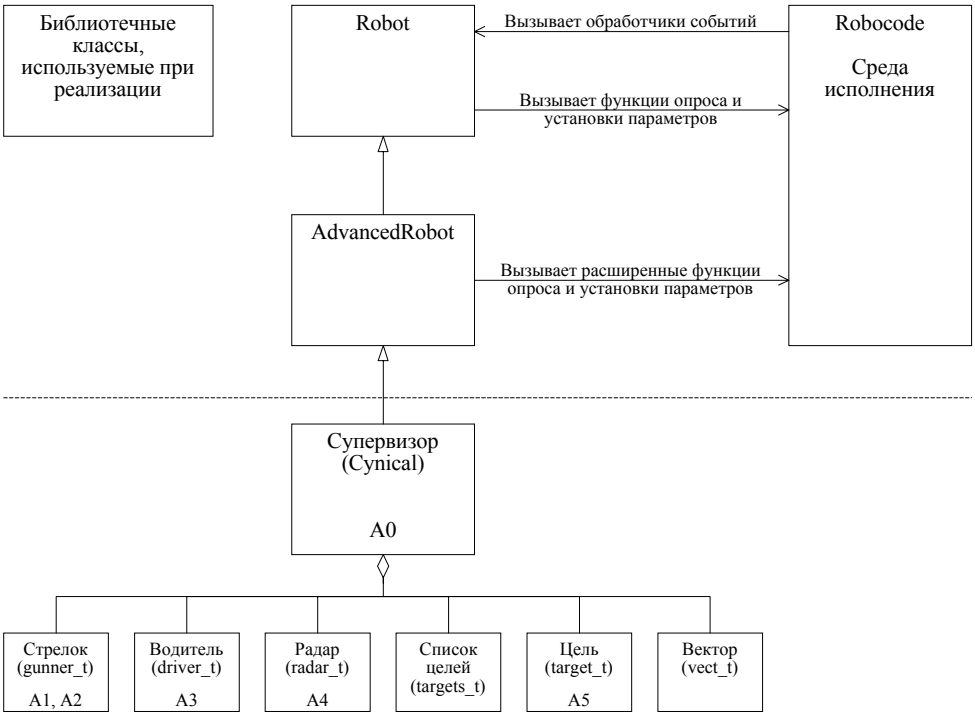

Рис. 3. Диаграмма классов

Среда формирует определенные правилами игры события и вызывает их обработчики, объявленные в классе "Robot". Эти обработчики наследуются в классе "AdvancedRobot". Указанные классы реализуют вызываемые из разрабатываемой системы управления методы, обеспечивающие непосредственное управление танком за счет опроса и установки его параметров в среде исполнения. К такой разновидности методов относится, например, метод turnRight(), который обеспечивает поворот танка направо.

В соответствии с требованиями программного интерфейса среды исполнения к структуре программы, она должна представлять собой один класс, порожденный от класса "Robot" или, как в данном случае, от класса "AdvancedRobot". Название содержащего программу

пакета "counterwallrobot" вместе с названием головного класса образуют название танка. Головной класс "Супервизор" в программе имеет имя "Cynical", и содержит шесть внутренних (вложенных) классов, сформированных на основе анализа задачи, которые носят следующие названия: "Стрелок", "Водитель", "Радар", "Список целей", "Цель" и "Вектор". Класс "Стрелок" является системой управления стрельбой, класс "Водитель" — системой управления маневрированием, а класс "Радар" — системой управления радаром.

Все эти шесть классов реализованы как внутренние для того, чтобы они не воспринимались средой исполнения как самостоятельные системы управления танками.

На диаграмме классы, содержащие автоматы, имеют соответствующие пометки.

4. Класс "Супервизор"

4.1. Словесное описание

Класс "Супервизор" является головным в приведенной иерархии классов. Основной задачей класса "Супервизор" является обработка событий среды исполнения. При этом все происходящие в течение шага события запоминаются в очереди и обрабатываются в начале каждого шага. Кроме этого, класс выполняет специальную обработку событий начала и конца раунда и начала и конца шага.

Также класс содержит набор вспомогательных вычислительных методов, используемых другими классами программы.

4.2. Структурная схема класса

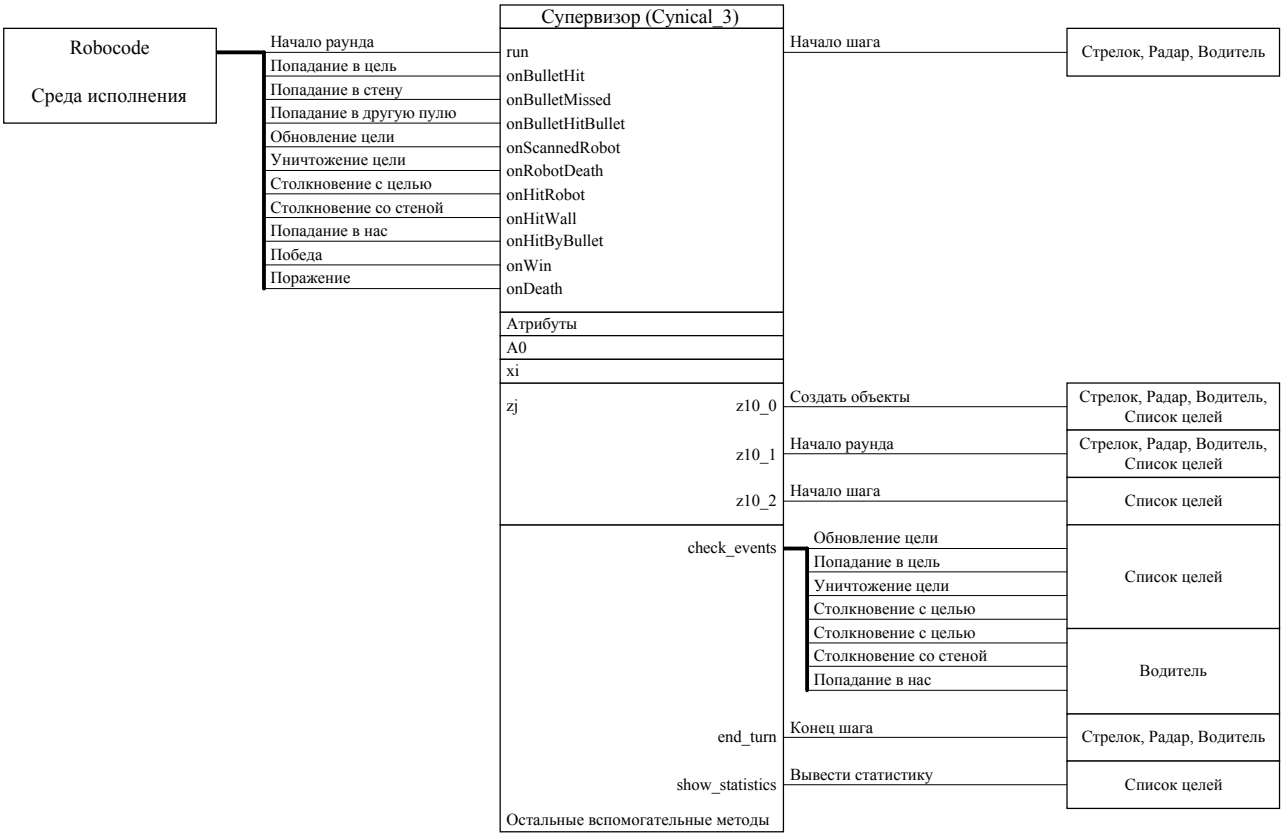

Рис. 4. Структурная схема класса "Супервизор"

4.3. Автомат опроса входных параметров

4.3.1. Словесное описание

Автомат отслеживает начало и завершение раунда, выполняет инициализацию в начале раунда, опрос параметров в начале каждого шага и вывод статистики раунда в конце каждого из них.

4.3.2. Схема связей

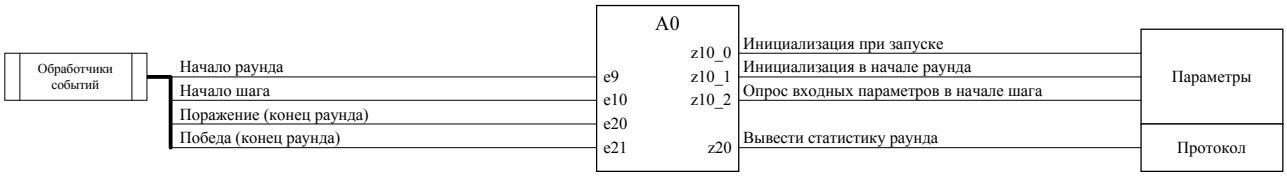

Рис. 5. Схема связей автомата опроса входных параметров

4.3.3. Граф переходов

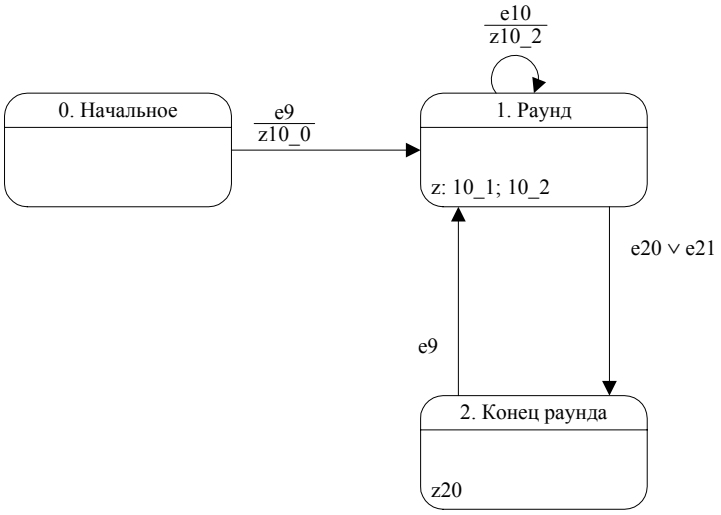

Рис. 6. Граф переходов автомата
опроса входных параметров

5. Класс "Стрелок"

5.1. Словесное описание

Класс предназначен для управления стрельбой и решает следующие задачи:

– управление пушкой, включая наведение и определение момента выстрела;

– определение скорости охлаждения пушки.

Класс не содержит вспомогательных методов, обеспечивающих решение вычислительных задач. Решение таких задач выполняется в других классах, методы которых рассматриваемый класс вызывает.

5.2. Структурная схема класса

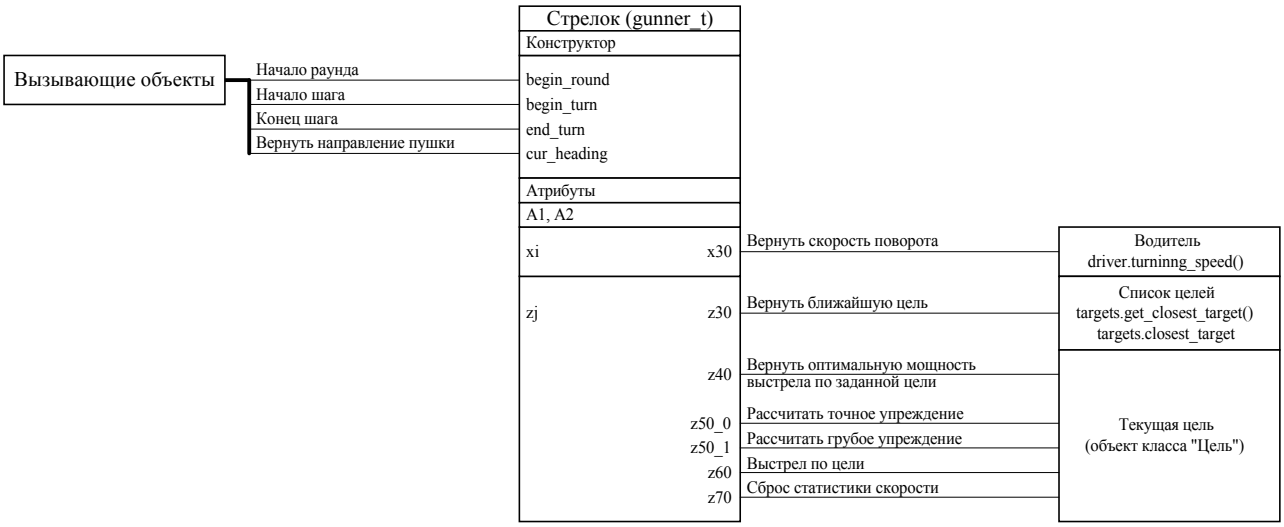

Рис. 7. Структурная схема класса "Стрелок"

5.3. Схема взаимодействия автоматов

Рис. 8. Схема взаимодействия автоматов класса "Стрелок"

5.4. Автомат управления пушкой

5.4.1. Словесное описание

Автомат выполняет выбор цели, ее сопровождение, наведение пушки и определение момента выстрела. Кроме этого, автомат определяет какой из созданных алгоритмов расчета упреждения следует использовать на различных этапах сопровождения цели.

Отметим, что по условиям игры пушка не может стрелять, если ее температура больше нуля. Проверку температуры пушки выполняют входные переменные x20, x21 и x22.

5.4.2. Схема связей

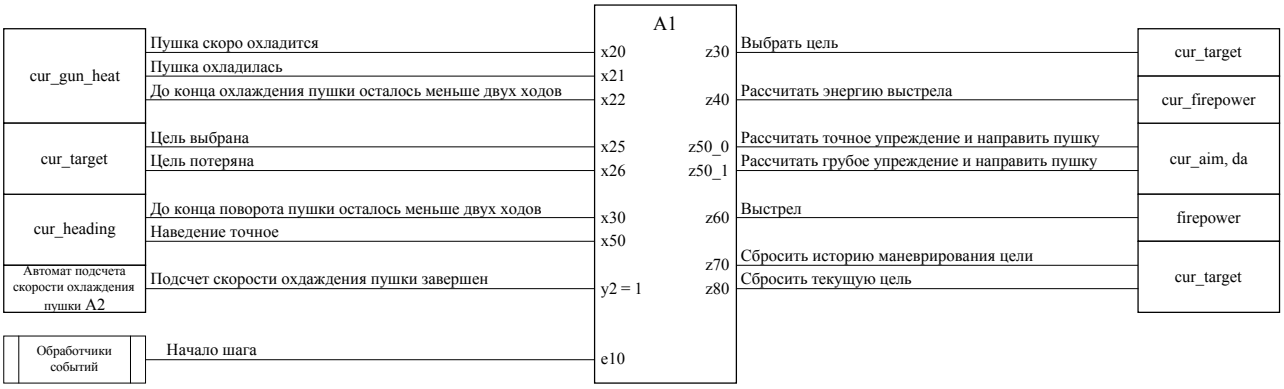

Рис. 9. Схема связей автомата управления пушкой

5.4.3. Граф переходов

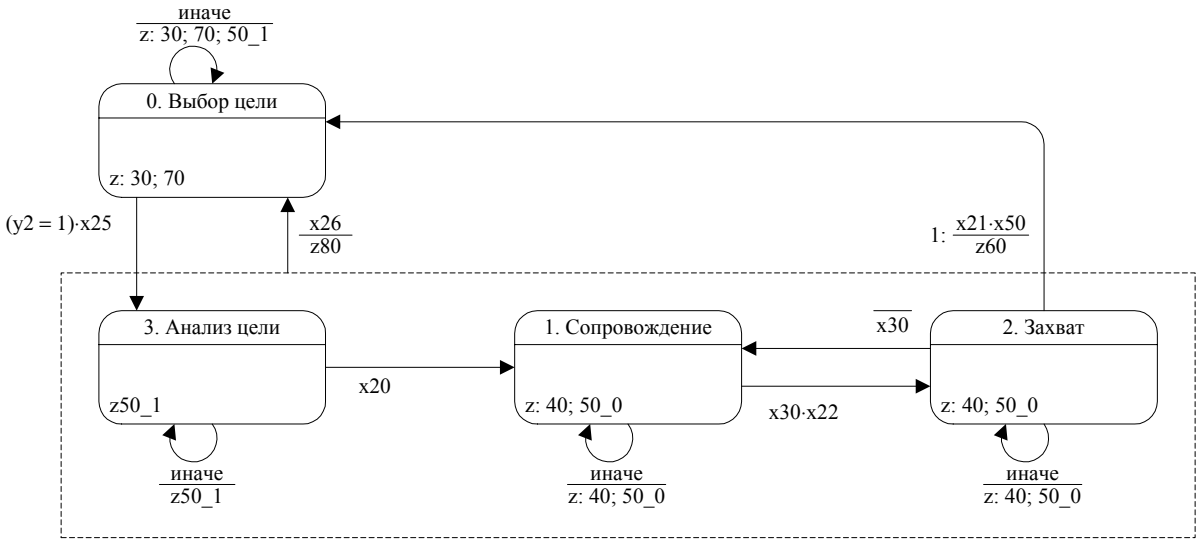

Рис. 10. Граф переходов автомата управления пушкой

5.5. Автомат определения скорости охлаждения пушки

5.5.1. Словесное описание

Рассматриваемая программа создавалась для одной из ранних версий исполнительной среды Robocode, не содержащей функции определения скорости охлаждения пушки.

Расчет скорости охлаждения выполняется под управлением рассматриваемого автомата путем сравнения температуры пушки на соседних шагах в начале раунда.

Обратим внимание, что начальная вершина автомата недостижима, так как расчет скорости охлаждения пушки выполняется однократно в начале игры.

5.5.2. Схема связей

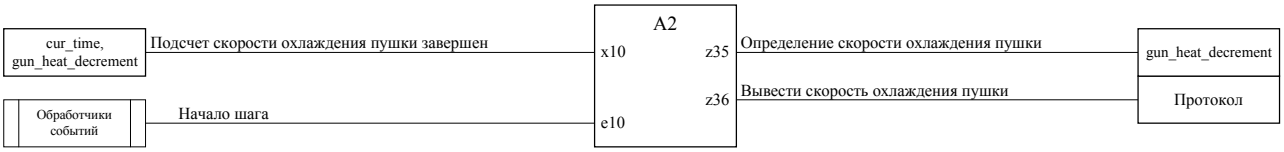

Рис. 11. Схема связей автомата определения скорости охлаждения пушки

5.5.3. Граф переходов

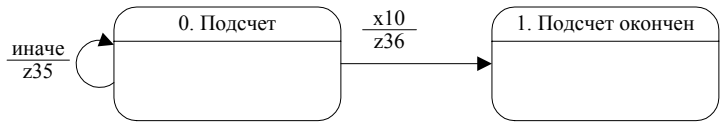

Рис. 12. Граф переходов автомата определения скорости охлаждения пушки

6. Класс "Водитель"

6.1. Словесное описание

Класс "Водитель" реализует разработанные эвристически алгоритмы маневрирования с движением по следующим траекториям: "маятник", "дуга", "уклонение" и "останов". Кроме этого, с использованием вычислительного алгоритма, реализованного в классе "Список целей", выполняется расчет направления движения для обеспечения удаления от стен и целей.

6.2. Структурная схема класса

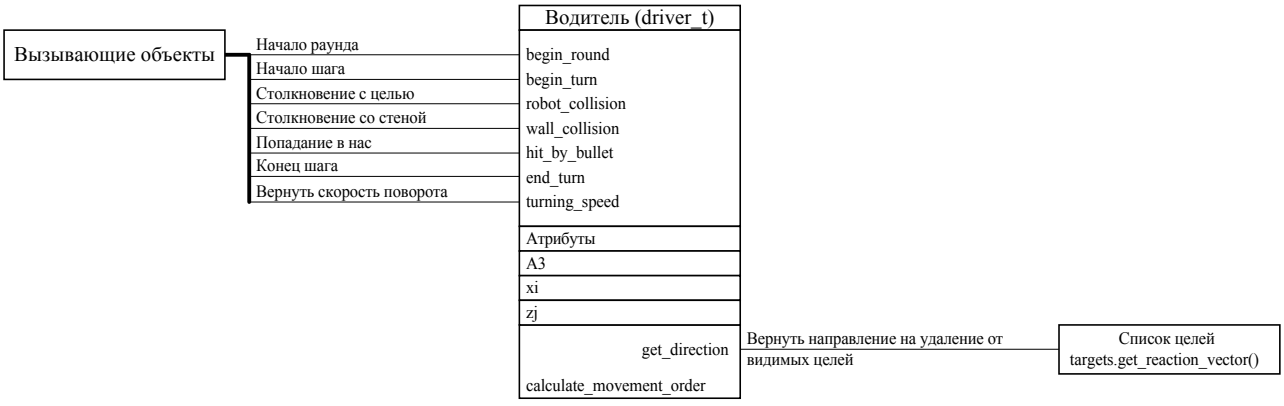

Рис. 13. Структурная схема класса "Водитель"

6.3. Автомат управления маневрированием

6.3.1. Словесное описание

Автомат выполняет выбор одного из реализованных алгоритмов маневрирования.

6.3.2. Схема связей

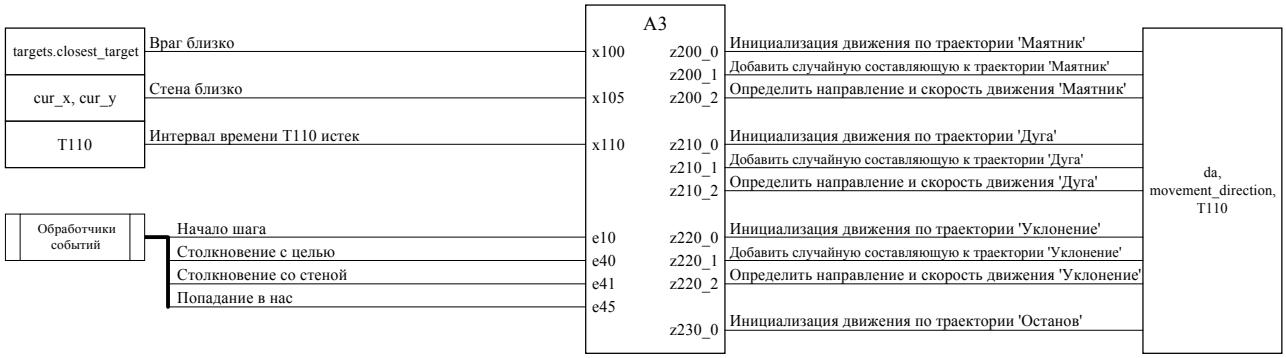

Рис. 14. Схема связей автомата управления маневрированием

6.3.3. Граф переходов

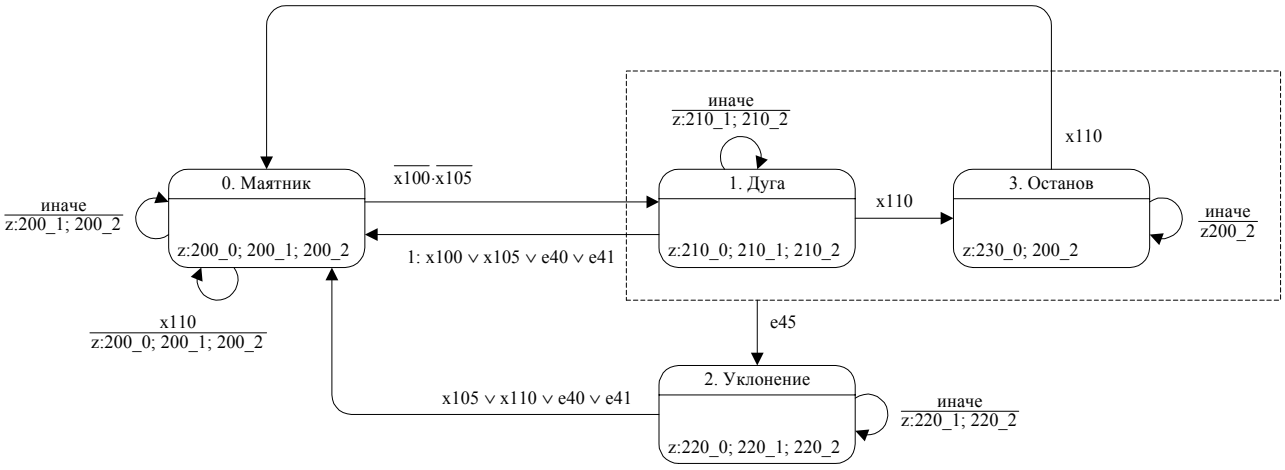

Рис. 15. Граф переходов автомата управления маневрированием

7. Класс "Радар"

7.1. Словесное описание

Класс осуществляет управление радаром, обеспечивающее минимальное время сканирования всех целей.

7.2. Структурная схема класса

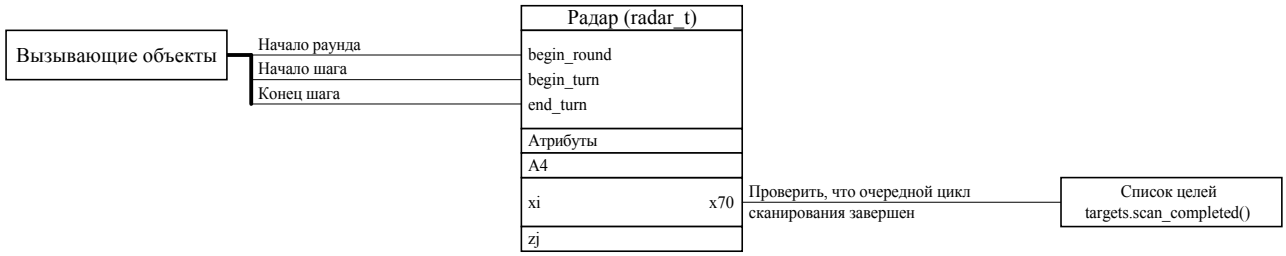

Рис. 16. Структурная схема класса "Радар"

7.3. Автомат управления радаром

7.3.1. Словесное описание

Автомат управляет радаром так, что сканирование всех известных целей осуществляется за минимально возможное время. При этом радар должен быть повернут на минимально необходимый угол. Например, если все цели находятся с одной стороны от нашего танка, то радар не будет сканировать пространство с противоположной стороны от танка.

7.3.2. Схема связей

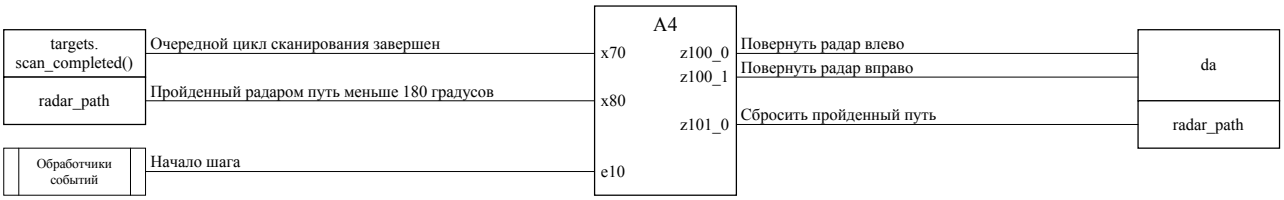

Рис. 17. Схема связей автомата управления радаром

7.3.3. Граф переходов

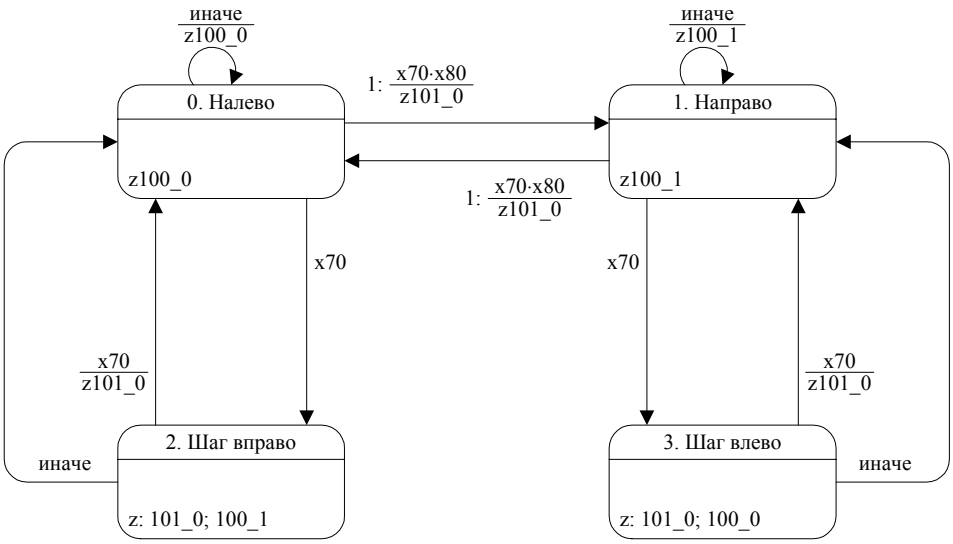

Рис. 18. Граф переходов автомата управления радаром

8. Класс "Список целей"

8.1. Словесное описание

Класс реализует список целей и обрабатывает все события, из которых может быть извлечена информация о целях. Метод `scan_completed()` определяет все ли цели были отсканированы с момента завершения предыдущего сканирования. Метод `get_reaction_vector()` определяет направление, удаляющее танк от всех известных целей.

Класс не содержит автоматных методов.

8.2. Структурная схема класса

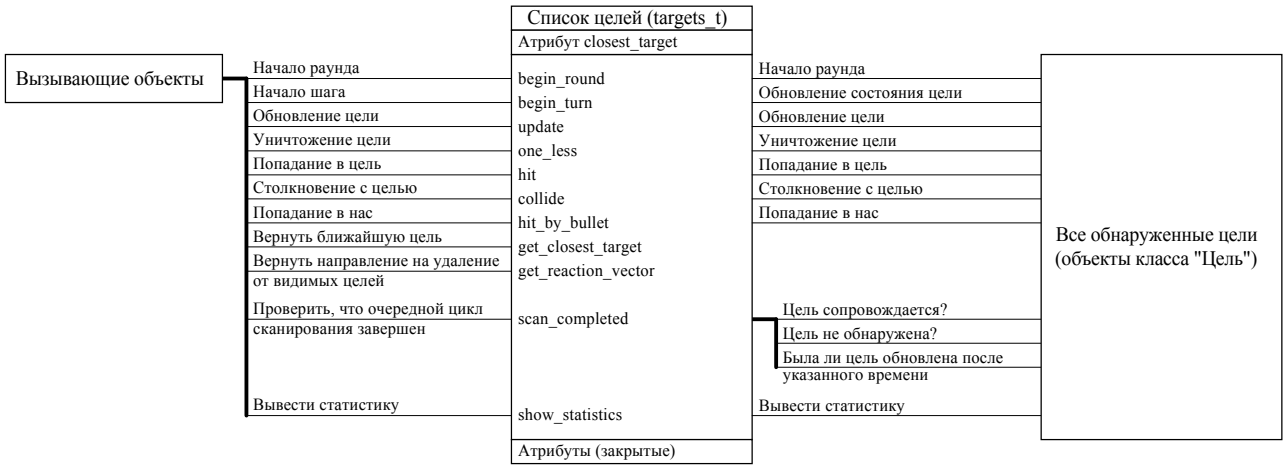

Рис. 19. Структурная схема класса "Список целей"

9. Класс "Цель"

9.1. Словесное описание

Класс выполняет хранение и обработку информации о цели. В рассматриваемом классе реализуются наиболее сложные вычислительные алгоритмы, имеющиеся в программе:

- ведение статистики попаданий в цель;
- определение мощности выстрела по цели исходя из расстояния до нее, ее энергии, статистически определенной вероятности попадания, собственной энергии;
- различные по точности и быстродействию способы расчета упреждения при стрельбе.

9.2. Структурная схема класса

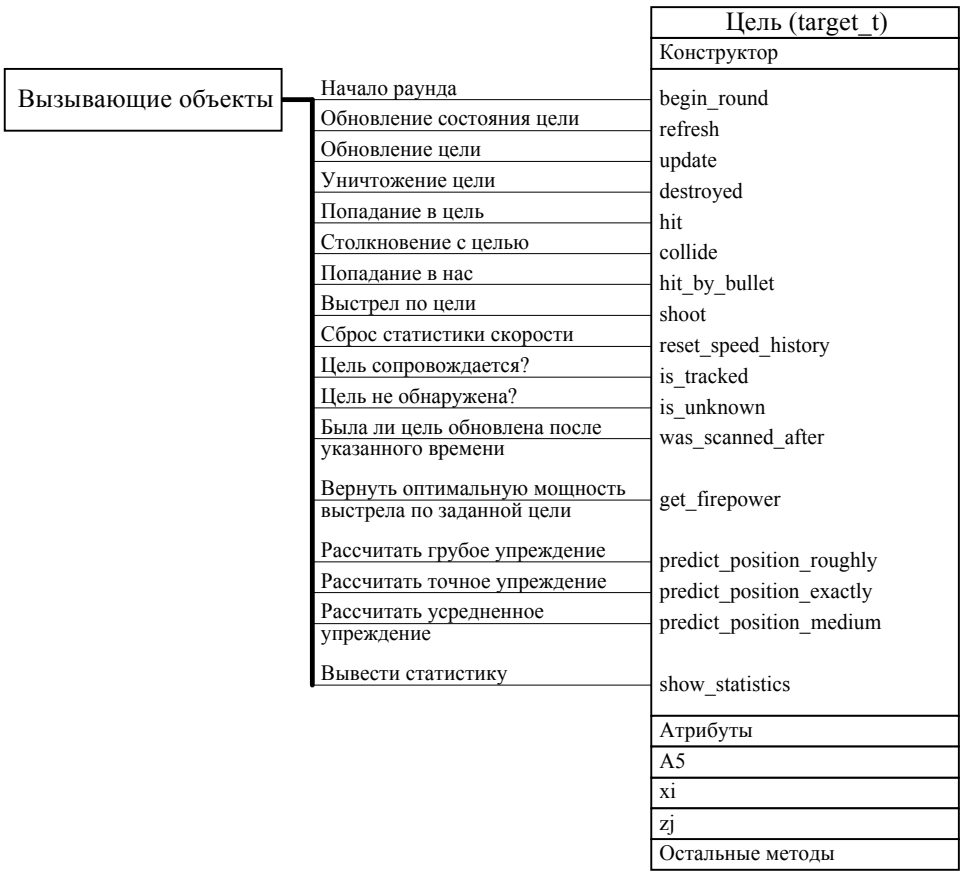

Рис. 20. Структурная схема класса "Цель"

9.3. Автомат определения состояния цели

9.3.1. Словесное описание

Автомат определяет состояние цели. В начале раунда все цели считаются не обнаруженными. При приходе события, содержащего информацию о цели, она считается сопровождаемой. Если цель уничтожена или информация о ней устарела, то цель считается потерянной.

9.3.2. Схема связей

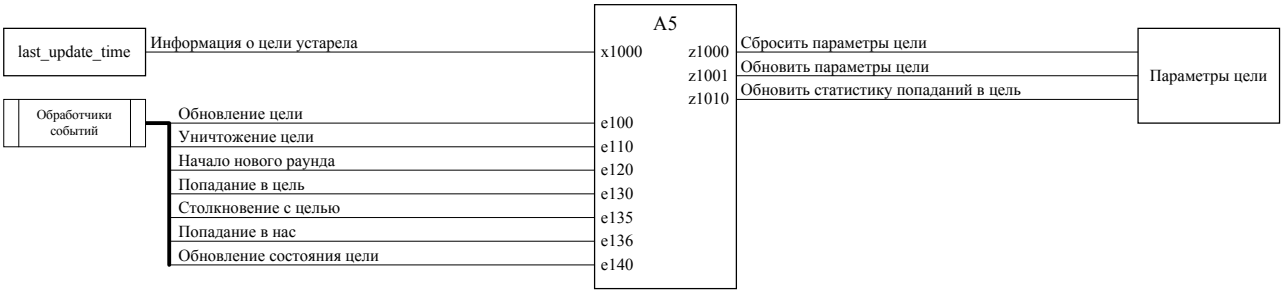

Рис. 21. Схема связей автомата определения состояния цели

9.3.3. Граф переходов

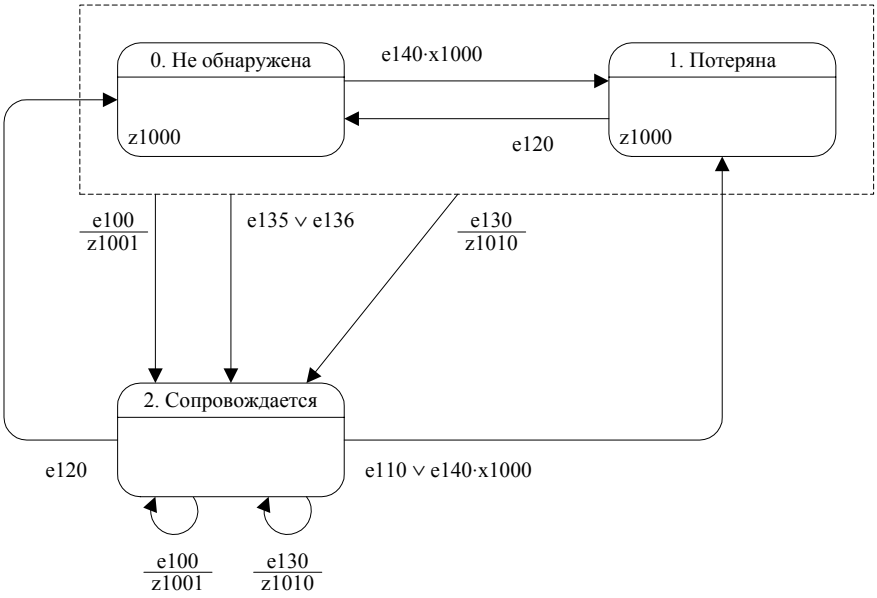

Рис. 22. Граф переходов автомата определения состояния цели

10. Класс "Вектор"

10.1. Словесное описание

Класс реализует двумерной вектор с хранением координат как в декартовой, так и в радиальной системе. Основное назначение класса — реализация операции сложения векторов.

10.2. Структурная схема класса

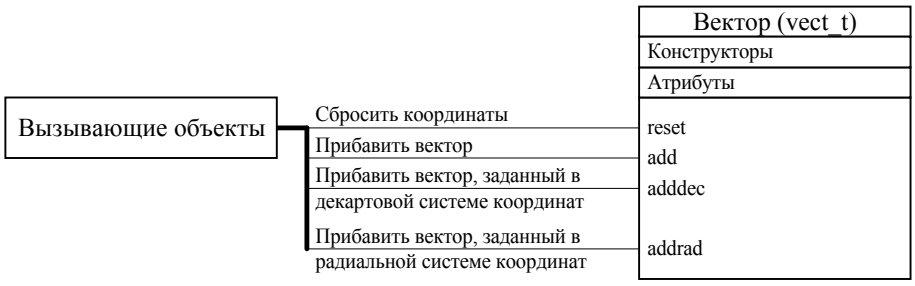

Рис. 23. Структурная схема класса "Вектор"

11. Текст программы

```
package counterwallrobot ;
import robocode.* ;
import java.util.* ;
import java.io.* ;

//======
// Класс "Супервизор".
//
public class Cynical_3 extends AdvancedRobot
{
  //*** Атрибуты.
  private final boolean ANY_DEBUG =         true ;
  private final boolean TO_LOG_FILE =       true && ANY_DEBUG ;
  private final boolean TO_CONSOLE =        false && ANY_DEBUG ;
  private final boolean SWITCH_DEBUG =      true ;
  private final boolean INPUTS_LOGGING =    true && SWITCH_DEBUG && ANY_DEBUG ;
  private final boolean OUTPUTS_LOGGING =   true && SWITCH_DEBUG && ANY_DEBUG ;
  private final boolean OBJECTS_LOGGING =   true && ANY_DEBUG ;
  private final boolean A0_LOGGING =        true && SWITCH_DEBUG && ANY_DEBUG ;
  private final boolean A0_BEGIN_LOGGING =  true && A0_LOGGING && ANY_DEBUG ;
  private final boolean A0_END_LOGGING =    true && A0_LOGGING && ANY_DEBUG ;
  private final boolean A0_ERROR_LOGGING =  true && A0_LOGGING && ANY_DEBUG ;
  private final boolean A0_TRANS_LOGGING =  true && A0_LOGGING && ANY_DEBUG ;

  private final double  precision = 1e-06 ;
  private final double  PI = Math.PI ;
  private final double  dPI = 2 * PI ;
  private final double  PI2 = PI / 2 ;
                        // Максимальная скорость поворота пушки.
  private final double  gun_rotation_speed = Math.toRadians(20) ;
                        // Максимальная линейная скорость.
  private final double  speed_max = 8 ;
                        // Максимальная скорость поворота.
  private final double  turning_speed_max = Math.toRadians(10) ;
                        // Максимальная энергия выстрела.
  private final double  max_firepower = 3 ;
                        // Базовая энергия выстрела.
  private final double  base_firepower = max_firepower/2.0 ;
                        // Сколько времени можно не стрелять.
  private final double  fire_delay_critical = 400 ;
  private final double  life_ok = 50 ;
  private final double  life_warning = 30 ;
  private final double  life_critical = 15 ;

  private double        battlefield_width, battlefield_height ;    // Размеры поля.
  private double        robot_size,         // Размер робота.
                        robot_size2 ;       // Половинный размер робота.
  private double        collision_delta ;
  private int           cur_robots_count ;  // Текущее количество других роботов.
  private int           total_robots_count ;// Общее количество роботов.
  private long          cur_time ;          // Номер текущего шага.
  private Random        random = new Random() ;
  private double        cur_life ;          // Текущая жизнь.
  private long          last_fire_time ;    // Момент последнего выстрела.
  private long          hits, misses, hitted_by_bullet ; // Статистика.
  private long          walls_collisions ;
  private radar_t       radar ;
  private driver_t      driver ;
  private gunner_t      gunner ;
  private targets_t     targets ;
  private Vector        events ;

  private int           y0 = 0 ;

  // Начало раунда.
  public void
  run()
  {
    A0(9) ; // Вызвать автомат A0 с событием "Начало раунда".

    while(true)
    {
      cur_time = getTime() ;
```

```
    log( "--------- " + cur_time + " ---------" ) ;

    check_events() ;      // Обработать очередь событий.

    A0(10) ;              // Вызвать автомат A0 с событием "Начало шага".
    gunner.begin_turn() ; // Передать событие "Начало шага" объекту "Стрелок".
    radar.begin_turn() ;  // Передать событие "Начало шага" объекту "Радар".
    driver.begin_turn() ; // Передать событие "Начало шага" объекту "Водитель".

    end_turn() ;          // Обработать событие "Конец шага".
  }
}

// Попадание в цель.
public void
onBulletHit( BulletHitEvent first_e )
{
  //log( "Получили событие попадания в цель." ) ;
  events.add( first_e ) ;
}

// Попадание в стену.
public void
onBulletMissed( BulletMissedEvent first_e )
{
  //log( "Получили событие попадания в стену." ) ;
  events.add( first_e ) ;
}

// Попадание в другую пулю.
public void
onBulletHitBullet( BulletHitBulletEvent first_e )
{
  //log( "Получили событие попадания в другую пулю." ) ;
  events.add( first_e ) ;
}

// Обновление цели.
public void
onScannedRobot( ScannedRobotEvent first_e )
{
  //log( getTime() + ": Получили событие обновления цели." ) ;
  events.add( first_e ) ;
}

// Уничтожение цели.
public void
onRobotDeath( RobotDeathEvent first_e )
{
  //log( "Получили событие уничтожения цели." ) ;
  events.add( first_e ) ;
}

// Столкновение с целью.
public void
onHitRobot( HitRobotEvent first_e )
{
  //log( "Получили событие столкновения с целью." ) ;
  events.add( first_e ) ;
}

// Столкновение со стеной.
public void
onHitWall( HitWallEvent first_e )
{
  //log( "Получили событие столкновения со стеной." ) ;
  events.add( first_e ) ;
}

// Попадание в нас.
public void
onHitByBullet( HitByBulletEvent first_e )
{
  //log( "Получили событие попадания в нас." ) ;
  events.add( first_e ) ;
}
```

```
// Победа.
public void
onWin( WinEvent first_e )
{
  //log( "Получили событие победы." ) ;
  A0( 21 ) ;
}

// Поражение.
public void
onDeath( DeathEvent first_e )
{
  //log( "Получили событие поражения." ) ;
  A0( 20 ) ;
}

// Реализация автомата A0.
private void
A0( int e )
{
  int   y_old = y0 ;

  if( OBJECTS_LOGGING )
    log( "Для объекта 'Супервизор':" ) ;

  if( A0_BEGIN_LOGGING )
    log_begin( "A0", y0, e ) ;

  switch( y0 )
  {
    case 0:
      if( e == 9 ) { z10_0() ;                    y0 = 1 ; }
    break ;

    case 1:
      if( e == 20 || e == 21 )                    y0 = 2 ;
      else
      if( e == 10 ) { z10_2() ; }
    break ;

    case 2:
      if( e == 9 )                                y0 = 1 ;
    break ;

    default:
      if( A0_ERROR_LOGGING )
        log_error( "A0", y0 ) ;
  }

  if( y0 != y_old )
  {

  if( A0_TRANS_LOGGING )
    log_trans( "A0", y0, y_old ) ;

  switch( y0 )
  {
    case 1:
      z10_1() ; z10_2() ;
    break ;

    case 2:
      z20() ;
    break ;

    default:
      if( A0_ERROR_LOGGING )
        log_error( "A0", y0 ) ;
  }
  }

  if( A0_END_LOGGING )
    log_end( "A0", y0 ) ;
}
```

```
//*** Реализация выходных воздействий.
private void
z10_0()
{
  if( OUTPUTS_LOGGING )
    log_output( "z10_0", "Инициализация при запуске" ) ;

  // Установить приоритет событий.
  setEventPriority("RobotDeathEvent", 17 );
  setEventPriority("ScannedRobotEvent", 16 );
  setEventPriority("HitRobotEvent", 15 );
  setEventPriority("HitWallEvent", 14 );
  setEventPriority("BulletHitEvent", 13 );
  setEventPriority("HitByBulletEvent", 12 );
  setEventPriority("BulletMissedEvent", 11 );

  // Инициализация параметров.
  battlefield_width = getBattleFieldWidth() ;
  battlefield_height = getBattleFieldHeight() ;
  robot_size = (getWidth() + getHeight()) / 2.0 ;
  robot_size2 = robot_size / 2.0 ;
  collision_delta = robot_size2 - 5 ;
  hits = 0 ; misses = 0 ; hitted_by_bullet = 0 ;
  walls_collisions = 0 ;
  total_robots_count = getOthers() ;

  // Создание объектов.
  targets = new targets_t() ;
  radar = new radar_t() ;
  driver = new driver_t() ;
  gunner = new gunner_t() ;
  events = new Vector() ;
}

private void
z10_1()
{
  if( OUTPUTS_LOGGING )
    log_output( "z10_1", "Инициализация в начале раунда" ) ;

  log( "" ) ;
  log( "====================================" ) ;
  log( "***" ) ;
  log( "*** Раунд " + (getRoundNum() + 1) ) ;
  log( "***" ) ;
  log( "====================================" ) ;
  log( "" ) ;

  clearAllEvents() ;

  setAdjustGunForRobotTurn(true);
  setAdjustRadarForGunTurn(true);

  cur_time = 0 ;

  // Передать событие "Начало раунда" остальным объектам.
  targets.begin_round() ;
  driver.begin_round() ;
  radar.begin_round() ;
  gunner.begin_round() ;

  if( events != null ) events.clear() ;
}

private void
z10_2()
{
  if( OUTPUTS_LOGGING )
    log_output( "z10_2", "Инициализация в начале шага" ) ;

  cur_robots_count = getOthers() ;
  cur_life = getLife() ;

  // Передать событие "Начало шага" объекту "Список целей".
  targets.begin_turn() ;
}
```

```
private void
z20()
{
  if( OUTPUTS_LOGGING )
    log_output( "z20", "Вывести статистику раунда" ) ;

  show_statistics() ;
}

//*** Вспомогательные методы.

// Считать все события из очереди.
private void
check_events()
{
  Iterator      events_iter = events.iterator() ;
  Event         e ;

  while( events_iter.hasNext() )
  {
    e = (Event)events_iter.next() ;

    if( e instanceof ScannedRobotEvent )
    {
      // Обновление цели.
      targets.update( (ScannedRobotEvent)e ) ;
    }
    else
    if( e instanceof BulletHitEvent )
    {
      // Попадание в цель.
      targets.hit( (BulletHitEvent)e ) ;
      hits++ ;
    }
    else
    if( e instanceof RobotDeathEvent )
    {
      // Уничтожение цели.
      targets.one_less( (RobotDeathEvent)e ) ;
    }
    else
    if( e instanceof HitRobotEvent )
    {
      // Столкновение с целью.
      driver.robot_collision() ;
      targets.collide( (HitRobotEvent)e ) ;
    }
    else
    if( e instanceof HitWallEvent )
    {
      // Столкновение со стеной.
      driver.wall_collision() ;
      walls_collisions++ ;
    }
    else
    if( e instanceof BulletMissedEvent )
    {
      // Попадание в стену.
      misses++ ;
    }
    else
    if( e instanceof HitByBulletEvent )
    {
      // Попадание в нас.
      driver.hit_by_bullet() ;
      targets.hit_by_bullet((HitByBulletEvent)e) ;
      hitted_by_bullet++ ;
    }
  }

  events.clear() ;
}
```

```
// Конец шага.
private void
end_turn()
{
  //log( "da = " + da*180/PI + ", da_gun = " + da_gun*180/PI
  //       + ", da_radar = " + da_radar*180/PI + ", speed = " + speed ) ;

  driver.end_turn() ; // Передать событие "Начало шага" объекту "Водитель".
  radar.end_turn() ;  // Передать событие "Начало шага" объекту "Радар".
  gunner.end_turn() ; // Передать событие "Начало шага" объекту "Стрелок".
}

// Вывести статистику.
private void
show_statistics()
{
  long shots = hits+misses ;

  targets.show_statistics() ;
  log( "Выстрелов: " + shots + ", попаданий: " + hits + ", промахов: " + misses ) ;
  log( "Меткость: " + (double)hits/shots ) ;
  log( "Попали в нас: " + hitted_by_bullet ) ;
  log( "Столкновений со стенами: " + walls_collisions ) ;
}

// Определить по заданным параметрам элементарный путь, пройденный танком,
// предполагая скорости фиксированными.
private vect_t
get_path( double v, double w, double heading, double T )
{
  vect_t  path ;

  if( Math.abs(w) > precision )
  {
    // Движение по дуге.
    double  R = Math.abs(v/w) ;
    double  to_circle_center, from_circle_center ;

    to_circle_center = normalize_angle(
                         w*v >= 0 ?
                           heading + PI2 :
                           heading - PI2 ) ;
    from_circle_center = normalize_angle( to_circle_center + PI + w * T ) ;

    path = new vect_t( to_circle_center, R ) ;
    path.addrad( from_circle_center, R ) ;
  }
  else
  {
    // Движение по прямой.
    path = new vect_t( heading, v * T ) ;
  }

  return path ;
}

// Расчитать скорость выстрела заданной мощности.
private double
get_bullet_speed( double firepower )
{
  return ( 20 - 3*firepower ) ;
}

// Вычислить максимальную скорость поворота
// по заданной линейной скорости.
private double
get_turning_speed( double speed )
{
  return Math.min
  (
    turning_speed_max,
    turning_speed_max * (0.4 + 0.6 * (1 - speed / speed_max))
  ) ;
}
```

```
// Приведение угла в диапазон от 0 до 2PI.
private double
normalize_angle( double a )
{
  a = dPI + (a % dPI) ;
  a %= dPI ;
  return a ;
}

// Определить минимальную разницу между
// двумя углами с учетом перехода через ноль.
private double
get_angle_diff( double from, double to )
{
  double  diff = to - from ;

  if( Math.abs(diff) <= PI ) return diff ;

  if( diff < 0 )
    diff += dPI ;
  else
  if( diff > 0 )
    diff -= dPI ;

  return diff % dPI ;
}

// Если заданный угол поворота больше 180,
// повернуть в противоположном направлении.
private double
shortest_turn( double da )
{
  if( da > PI )
    da -= dPI ;
  else
  if( da < -PI )
    da += dPI ;

  return da ;
}

// Вычислить угловую координату вектора.
private double
get_angle( double x, double y )
{
  double a ;

  if( y == 0 )
  {
    return x > 0 ? PI2 : 3*PI2 ;
  }

  a = Math.atan( x/y ) ;

  if( y < 0 ) a += PI ;

  return a ;
}

private void
log( String str )
{
  if( TO_CONSOLE )
    out.println( str ) ;
  if( TO_LOG_FILE )
    System.out.println( str ) ;
}

private void
log( String str, String symbol )
{
  String  out_str = symbol + " " + str ;
  if( TO_CONSOLE )
    out.println( out_str ) ;
  if( TO_LOG_FILE )
    System.out.println( out_str ) ;
}
```

```
private void
log_begin( String a_name, int state, int event )
{
  log( a_name +  ": Автомат " + a_name
       + " запущен в состоянии " + state
       + " с событием e" + event, "{" ) ;
}

private void
log_end( String a_name, int state )
{
  log( a_name +  ": Автомат " + a_name
       + " завершил свою работу в состоянии " + state, "}" ) ;
}

private void
log_error( String a_name, int state )
{
  log( a_name + ": Неизвестное состояние " + state + "!", "!" ) ;
}

private void
log_trans( String a_name, int state_to, int state_from )
{
  log( a_name +  ": Автомат " + a_name
       + " перешел из состояния " + state_from
       + " в состояние " + state_to, " T" ) ;
}

private void
log_input( String x_name, String comment, boolean result )
{
  String res_str = result ? "ДА" : "НЕТ" ;
  log( x_name + ": " + comment + "? - " + res_str, " i" ) ;
}

private void
log_output( String z_name, String comment )
{
  log( z_name + ": " + comment, " *" ) ;
}

//======
// Класс "Стрелок".
//
public class gunner_t extends Object
{
  //*** Атрибуты.
  private final boolean SWITCH_DEBUG =      true ;
  private final boolean INPUTS_LOGGING =    true && SWITCH_DEBUG && ANY_DEBUG ;
  private final boolean OUTPUTS_LOGGING =   true && SWITCH_DEBUG && ANY_DEBUG ;
  private final boolean A1_LOGGING =        true && SWITCH_DEBUG && ANY_DEBUG ;
  private final boolean A1_BEGIN_LOGGING =  true && A1_LOGGING && ANY_DEBUG ;
  private final boolean A1_END_LOGGING =    true && A1_LOGGING && ANY_DEBUG ;
  private final boolean A1_ERROR_LOGGING =  true && A1_LOGGING && ANY_DEBUG ;
  private final boolean A1_TRANS_LOGGING =  true && A1_LOGGING && ANY_DEBUG ;
  private final boolean A2_LOGGING =        true && SWITCH_DEBUG && ANY_DEBUG ;
  private final boolean A2_BEGIN_LOGGING =  true && A2_LOGGING && ANY_DEBUG ;
  private final boolean A2_END_LOGGING =    true && A2_LOGGING && ANY_DEBUG ;
  private final boolean A2_ERROR_LOGGING =  true && A2_LOGGING && ANY_DEBUG ;
  private final boolean A2_TRANS_LOGGING =  true && A2_LOGGING && ANY_DEBUG ;

  private double        old_heading, cur_heading ;  // Направление пушки.
  private double        old_gun_heat, cur_gun_heat, // Температура пушки.
                        gun_heat_decrement ;        // Скорость охлаждения пушки.
  private target_t      cur_target ;                // Текущая цель.
  private vect_t        cur_aim ;                   // Текущий прицел.
  private double        cur_firepower ;             // Текущая мощность выстрела.

  private double        da ;  // На сколько надо повернуть пушку на данном шаге.
  private double        firepower ;                 // Мощность производимого выстрела.

  private int           y1 = 0 ;
  private int           y2 = 0 ;
```

```
// Конструктор.
gunner_t()
{
  gun_heat_decrement = 0 ;
  cur_aim = new vect_t() ;
}

// Начало раунда.
public void
begin_round()
{
  old_heading = cur_heading = getGunHeadingRadians() ;
  old_gun_heat = cur_gun_heat = getGunHeat() ;
  cur_target = null ;
  cur_aim.reset() ;
  last_fire_time = 0 ;
}

// Начало шага.
public void
begin_turn()
{
  old_heading = cur_heading ;
  cur_heading = getGunHeadingRadians() ;
  old_gun_heat = cur_gun_heat ;
  cur_gun_heat = getGunHeat() ;

  da = 0 ;
  firepower = 0 ;

  A1(10) ;
  A2(10) ;
}

// Конец шага.
public void
end_turn()
{
  setTurnGunRightRadians( da ) ;

  //log( "firepower = " + firepower ) ;
  if( firepower >= 0.1 )
  {
    //log( "Выстрел firepower = " + firepower ) ;
    //log( "Выстрел на gun = " + getGunHeadingRadians()*180/PI ) ;
    Bullet bullet = fireBullet( firepower ) ;
    if( bullet != null )
    {
      last_fire_time = cur_time ;
      if( cur_target != null )
        cur_target.shoot( firepower ) ;
    }
    //log( "Выстрел на gun = " + getGunHeadingRadians()*180/PI ) ;
  }
  else
  {
    scan() ;
  }
}

// Вернуть направление пушки.
public double
cur_heading()
{ return cur_heading ; }

// Реализация автомата A1.
private void
A1( int e )
{
  int   y_old = y1 ;

  if( OBJECTS_LOGGING )
    log( "Для объекта 'Стрелок':" ) ;

  if( A1_BEGIN_LOGGING )
    log_begin( "A1", y1, e ) ;
```

```
  switch( y1 )
  {
    case 0:
      if( (y2 == 1) && x25() )                          y1 = 3 ;
      else
      { z30() ; z70() ; z50_1() ; }
    break ;

    case 1:
      if( x26() )          { z80() ;                    y1 = 0 ; }
      else
      if( x30() && x22() )                              y1 = 2 ;
      else
      { z40() ; z50_0() ; }
    break ;

    case 2:
      if( x26() )          { z80() ;                    y1 = 0 ; }
      else
      if( x21() && x50() ) { z60() ;                    y1 = 0 ; }
      else
      if( !x30() )                                      y1 = 1 ;
      else
      { z40() ; z50_0() ; }
    break ;

    case 3:
      if( x26() )          { z80() ;                    y1 = 0 ; }
      else
      if( x20() )                                       y1 = 1 ;
      else
      { z50_1() ; }
    break ;

    default :
      if( A1_ERROR_LOGGING )
        log_error( "A1", y1 ) ;
  }

  if( y1 != y_old )
  {

  if( A1_TRANS_LOGGING )
    log_trans( "A1", y1, y_old ) ;

  switch( y1 )
  {
    case 0:
      z30() ; z70() ;
    break ;

    case 1:
      z40() ; z50_0() ;
    break ;

    case 2:
      z40() ; z50_0() ;
    break ;

    case 3:
      z50_1() ;
    break ;
  }

  }

  if( A1_END_LOGGING )
    log_end( "A1", y1 ) ;
}
```

```
// Реализация автомата A2.
private void
A2( int e )
{
  int   y_old = y2 ;

  if( A2_BEGIN_LOGGING )
    log_begin( "A2", y2, e ) ;

  switch( y2 )
  {
    case 0:
      if( x10() )  { z36() ;     y2 = 1 ; }
      else
      { z35() ; }
    break ;

    case 1:
    break ;

    default :
      if( A2_ERROR_LOGGING )
        log_error( "A2", y2 ) ;
  }

  if( y2 != y_old )
  {

  if( A2_TRANS_LOGGING )
    log_trans( "A2", y2, y_old ) ;
  }

  if( A2_END_LOGGING )
    log_end( "A2", y2 ) ;
}


//*** Реализация входных переменных.
private boolean
x10()
{
  boolean result = (cur_time > 3) && (gun_heat_decrement > 0) ;

  if( INPUTS_LOGGING )
    log_input( "x10", "Подсчет скорости охлаждения пушки завершен", result ) ;
  return result ;
}

private boolean
x20()
{
  boolean result = cur_gun_heat/gun_heat_decrement <= 3 ;

  if( INPUTS_LOGGING )
    log_input( "x20", "Пушка скоро охладится", result ) ;
  return result ;
}

private boolean
x21()
{
  boolean result = cur_gun_heat <= 0 ;

  if( INPUTS_LOGGING )
    log_input( "x21", "Пушка охладилась", result ) ;
  return result ;
}

private boolean
x22()
{
  boolean result = cur_gun_heat/gun_heat_decrement <= 1 ;

  if( INPUTS_LOGGING )
    log_input( "x22", "До конца охлаждения пушки меньше двух ходов", result ) ;
  return result ;
}
```

```
private boolean
x25()
{
  boolean result = cur_target != null ;

  if( INPUTS_LOGGING )
    log_input( "x25", "Цель выбрана", result ) ;
  return result ;
}

private boolean
x26()
{
  boolean result = true ;

  if( cur_target != null )
    result = !cur_target.is_tracked() ;

  if( INPUTS_LOGGING )
    log_input( "x26", "Цель потеряна", result ) ;
  return result ;
}

private boolean
x30()
{
  boolean result = true ;

  double  gun_to_go = get_angle_diff( cur_heading, cur_aim.a ) ;
  double  turn_direction = gun_to_go >= 0 ? 1 : -1 ;
                           // Обращение к объекту класса "Водитель".
  double  gun_turning_speed = turn_direction * gun_rotation_speed
                            + driver.turning_speed() ;

  result = Math.abs( gun_to_go / gun_turning_speed ) <= 1 ;

  if( INPUTS_LOGGING )
    log_input( "x30", "До конца поворота пушки меньше двух ходов", result ) ;
  return result ;
}

private boolean
x50()
{
  boolean result = true ;

  double  gun_to_go = get_angle_diff( cur_heading, cur_aim.a ) ;
                      //shortest_turn( cur_aim.a - cur_heading ) ;
  result = Math.abs(gun_to_go) < precision ;

  if( INPUTS_LOGGING )
    log_input( "x50", "Наводка правильная", result ) ;
  return result ;
}

//*** Реализация выходных воздействий.
private void
z30()
{
  if( OUTPUTS_LOGGING )
    log_output( "z30", "Выбрать цель" ) ;

  cur_target = targets.get_closest_target( 8 ) ;
  if( cur_target == null )
    cur_target = targets.closest_target ;
}

private void
z35()
{
  if( OUTPUTS_LOGGING )
    log_output( "z35", "Подсчет скорости охлаждения пушки" ) ;

  gun_heat_decrement = old_gun_heat - cur_gun_heat ;
}
```

```
private void
z36()
{
  if( OUTPUTS_LOGGING )
    log_output( "z36", "Вывести скорость охлаждения пушки" ) ;
  log( "gun_heat_decrement = " + gun_heat_decrement ) ;
}

private void
z40()
{
  if( OUTPUTS_LOGGING )
    log_output( "z40", "Рассчитать мощность выстрела" ) ;

  double  P = 0.25 ;
  if( cur_life >= life_ok )
  {
    P = 0.2 ;
    if( cur_robots_count > 2 )
    {
      P = 0.4 ;
    }
  }
  else
  if( cur_life <= life_warning )
  {
    P = 0.25 ;
    if( cur_robots_count > 2 )
    {
      P = 0.5 ;
    }
  }
  else
  if( cur_life <= life_critical )
  {
    P = 0.6 ;
  }

  cur_firepower = cur_target.get_firepower(P) ;
}

private void
z50_0()
{
  if( OUTPUTS_LOGGING )
    log_output( "z50_0", "Рассчитать точное упреждение и направить пушку" ) ;

  if( cur_target != null  )
  {
    vect_t predicted_pos ;

    predicted_pos = cur_target.predict_position_medium(
                      get_bullet_speed(cur_firepower), 2 ) ;

    cur_aim.reset( predicted_pos.a, predicted_pos.R ) ;
    da = get_angle_diff( cur_heading, predicted_pos.a ) ;
  }
}

private void
z50_1()
{
  if( OUTPUTS_LOGGING )
    log_output( "z50_1",
                "Рассчитать приблизительное упреждение и направить пушку" ) ;

  if( cur_target != null  )
  {
    vect_t predicted_pos ;

    predicted_pos = cur_target.predict_position_roughly(
                      get_bullet_speed(base_firepower), 2 ) ;

    cur_aim.reset( predicted_pos.a, predicted_pos.R ) ;
    da = get_angle_diff( cur_heading, predicted_pos.a ) ;
  }
}
```

```
  private void
  z60()
  {
    if( OUTPUTS_LOGGING )
      log_output( "z60", "Выстрел" ) ;

    firepower = cur_firepower ;

    //log( "Выстрел на gun = " + getGunHeadingRadians()*180/PI ) ;
  }

  private void
  z70()
  {
    if( OUTPUTS_LOGGING )
      log_output( "z70", "Сбросить историю маневрирования цели" ) ;

    if( cur_target != null )
      cur_target.reset_speed_history() ;
  }

  private void
  z80()
  {
    if( OUTPUTS_LOGGING )
      log_output( "z80", "Сбросить текущую цель" ) ;

    cur_target = null ;
  }
} // gunner_t

//======
// Класс "Водитель".
//
public class driver_t extends Object
{
  //*** Атрибуты.
  private final boolean SWITCH_DEBUG =      true ;
  private final boolean INPUTS_LOGGING =    true && SWITCH_DEBUG && ANY_DEBUG ;
  private final boolean OUTPUTS_LOGGING =   true && SWITCH_DEBUG && ANY_DEBUG ;
  private final boolean A3_LOGGING =        true && SWITCH_DEBUG && ANY_DEBUG ;
  private final boolean A3_BEGIN_LOGGING =  true && A3_LOGGING && ANY_DEBUG ;
  private final boolean A3_END_LOGGING =    true && A3_LOGGING && ANY_DEBUG ;
  private final boolean A3_ERROR_LOGGING =  true && A3_LOGGING && ANY_DEBUG ;
  private final boolean A3_TRANS_LOGGING =  true && A3_LOGGING && ANY_DEBUG ;

  private final double  walls_k = 100 ;        // Коэффициент "отталкивания" от стен.
  private final double  heading_delta = Math.toRadians(30) ;

  private long          T110 ;                 // Момент срабатывания таймера T110.
  private double        old_heading, cur_heading ;   // Направление движения.
  private double        old_x, old_y, cur_x, cur_y ; // Координаты.
  private double        turning_speed ;              // Текущая скорость поворота.
  private double        old_speed, cur_speed ;       // Линейная скорость.

  private double        da ;    // На сколько надо повернуть на данном шаге.
  private double        speed ; // С какой скоростью надо двигаться на данном шаге.
  private vect_t        direction = new vect_t() ;   // Направление движения.

  private int           y3 = 0 ;

  // Начало раунда.
  public void
  begin_round()
  {
    old_heading = cur_heading = getHeadingRadians() ;
    old_speed = cur_speed = 0 ;
    old_x = cur_x = getX() ;
    old_y = cur_y = getY() ;
    turning_speed = 0 ;
    T110 = 0 ;
    direction.reset() ;

    da = 0 ;
    speed = speed_max ;
  }
```

```
// Начало шага.
public void
begin_turn()
{
  old_heading = cur_heading ;
  cur_heading = getHeadingRadians() ;
  old_x = cur_x ; cur_x = getX() ;
  old_y = cur_y ; cur_y = getY() ;
  old_speed = cur_speed ;
  cur_speed = Math.sqrt( Math.pow( cur_x-old_x, 2 ) + Math.pow( cur_y-old_y, 2 )) ;
  cur_speed = cur_speed * ( speed >= 0 ? 1 : -1 ) ;
  turning_speed = get_angle_diff( old_heading, cur_heading ) ;

  da = 0 ;

  A3(10) ;
}

// Столкновение с целью.
public void
robot_collision()
{
  A3(40) ;
}

// Столкновение со стеной.
public void
wall_collision()
{
  A3(41) ;
}

// Попадание в нас.
public void
hit_by_bullet()
{
  A3(45) ;
}

// Конец шага.
public void
end_turn()
{
  setTurnRightRadians( da ) ;
  setAhead( speed*100 ) ;
}

// Вернуть скорость поворота.
public double
turning_speed()
{ return turning_speed ; }

// Реализация автомата A3.
private void
A3( int e )
{
  int   y_old = y3 ;

  if( OBJECTS_LOGGING )
    log( "Для объекта 'Водитель':" ) ;

  if( A3_BEGIN_LOGGING )
    log_begin( "A3", y3, e ) ;

  switch( y3 )
  {
    case 0:
      if( !x100() && !x105() )                                  y3 = 1 ;
      else
      if( x110() )
      { z200_0() ; z200_1() ; z200_2() ; }
      else
      { z200_1() ; z200_2() ; }
    break ;
```

```
    case 1:
      if( e == 45 )                                          y3 = 2 ;
      else
      if( x100() || x105() || e == 40 || e == 41 )           y3 = 0 ;
      else
      if( x110() )                                           y3 = 3 ;
      else
      { z210_1() ; z210_2() ; }
    break ;

    case 2:
      if( x105() || x110() || e == 40 || e == 41 )           y3 = 0 ;
      else
      { z220_1() ; z220_2() ; }
    break ;

    case 3:
      if( e == 45 )                                          y3 = 2 ;
      else
      if( x110() )                                           y3 = 0 ;
      else
      { z200_2() ; }
    break ;

    default :
      if( A3_ERROR_LOGGING )
        log_error( "A3", y3 ) ;
  }

  if( y3 != y_old )
  {

  if( A3_TRANS_LOGGING )
    log_trans( "A3", y3, y_old ) ;

  switch( y3 )
  {
    case 0:
      z200_0() ; z200_1() ; z200_2() ;
    break ;

    case 1:
      z210_0() ; z210_1() ; z210_2() ;
    break ;

    case 2:
      z220_0() ; z220_1() ; z220_2() ;
    break ;

    case 3:
      z230_0() ; z200_2() ;
    break ;
  }

  }

  if( A3_END_LOGGING )
    log_end( "A3", y3 ) ;
}

//*** Реализация входных переменных.
private boolean
x100()
{
  boolean   result = true ;

  if( targets.closest_target != null )
    result = targets.closest_target.R < 300 ;

  if( INPUTS_LOGGING )
    log_input( "x100", "Враг близко", result ) ;
  return result ;
}
```

```
private boolean
x105()
{
  double    collision_delta = robot_size2 + 40 ;

  boolean   result =
       cur_x < 0 + collision_delta
    || cur_x > battlefield_width - collision_delta
    || cur_y < 0 + collision_delta
    || cur_y > battlefield_height - collision_delta ;

  if( INPUTS_LOGGING )
    log_input( "x105", "Стена близко", result ) ;
  return result ;
}

private boolean
x110()
{
  boolean   result = cur_time >= T110 ;

  if( INPUTS_LOGGING )
    log_input( "x110", "Сработал таймер T110", result ) ;
  return result ;
}

//*** Реализация выходных воздействий.
private void
z200_0()
{
  if( OUTPUTS_LOGGING )
    log_output( "z200_0", "Инициализация движения по траектории 'Маятник'" ) ;

  setMaxVelocity(10) ;
  get_direction() ;
}

private void
z200_1()
{
  if( OUTPUTS_LOGGING )
    log_output( "z200_1",
                "Добавить случайную составляющую к траектории 'Маятник'" ) ;

  // Добавить случайную составляющую.
  direction.reset( direction.a, 1 ) ;

  double angle_diff = normalize_angle( direction.a +
                                       (random.nextBoolean() ? PI2 : -PI2) ) ;
  direction.addrad( angle_diff, 0.4 ) ;
}

private void
z200_2()
{
  if( OUTPUTS_LOGGING )
    log_output( "z200_2", "Определить направление и скорость движения 'Маятник'" ) ;

  calculate_movement_order( true ) ;
}

private void
z210_0()
{
  if( OUTPUTS_LOGGING )
    log_output( "z210_0", "Инициализация движения по траектории 'Дуга'" ) ;

  double TTT = targets.closest_target != null
               ? targets.closest_target.R / 20.0 : 10 ;

  TTT = TTT * ( random.nextDouble() * 0.5 + 0.8 ) ;
  T110 = cur_time + Math.round(TTT) ;

  get_direction() ;
  setMaxVelocity(10) ;
```

```
  if( targets.closest_target != null )
  {
    double delta1 =
      Math.abs(get_angle_diff( direction.a, targets.closest_target.a )) ;
    double delta2 =
      Math.abs(get_angle_diff( direction.a,
                               normalize_angle(targets.closest_target.a + PI) )) ;
    if( delta1 > heading_delta || delta2 > heading_delta )
    {
      direction.a = normalize_angle( targets.closest_target.a + PI
                                     + (random.nextBoolean() ? PI2/2 : -PI2/2) ) ;
    }
  }
}

private void
z210_1()
{
  if( OUTPUTS_LOGGING )
    log_output( "z210_1", "Добавить случайную составляющую к траектории 'Дуга'" ) ;
}

private void
z210_2()
{
  if( OUTPUTS_LOGGING )
    log_output( "z210_2", "Определить направление и скорость движения 'Дуга'" ) ;

  calculate_movement_order( false ) ;
  da = 0 ;
  //da = movement_direction ;
}


private void
z220_0()
{
  if( OUTPUTS_LOGGING )
    log_output( "z220_0", "Инициализация движения по траектории 'Уклонение'" ) ;

  T110 = cur_time + 17 ;
  get_direction() ;
  direction.a = - PI2 * (direction.a/Math.abs(direction.a)) ;
  setMaxVelocity(5) ;
}

private void
z220_1()
{
  if( OUTPUTS_LOGGING )
    log_output( "z220_1",
                "Добавить случайную составляющую к траектории 'Уклонение'" ) ;
}

private void
z220_2()
{
  if( OUTPUTS_LOGGING )
    log_output( "z220_2", "Определить направление движения 'Уклонение'" ) ;

  calculate_movement_order( false ) ;
}

private void
z230_0()
{
  if( OUTPUTS_LOGGING )
    log_output( "z230_0", "Инициализация движения по траектории 'Останов'" ) ;

  double TTT = targets.closest_target != null
               ? targets.closest_target.R / 20.0 : 10 ;

  TTT = TTT * ( random.nextDouble() * 0.3 + 0.3 ) ;

  T110 = cur_time + Math.round(TTT) ;
  setMaxVelocity( 0 ) ;
}
```

```
  //*** Вспомогательные методы.
  // Получить направление движения на удаление от целей и стен.
  private void
  get_direction()
  {
    direction.reset() ;

    // Учесть расстояние до стен.
    direction.adddec( walls_k / cur_x, 0 ) ;
    direction.adddec( -walls_k / (battlefield_width - cur_x), 0 ) ;
    direction.adddec( 0, walls_k / cur_y ) ;
    direction.adddec( 0, -walls_k / (battlefield_height - cur_y) ) ;

    // Учесть расстояние до всех видимых целей.
                  // Обращение к объекту класса "Список целей".
    direction.add( targets.get_reaction_vector() ) ;
  }

  // Вычислить требуемый угол поворота.
  private void
  calculate_movement_order( boolean reverse_possible )
  {
    double  da1,  // Угол, на который надо повернуться при движении вперед.
            da2 ; // Угол, на который надо повернуться при движении назад.

    da1 = get_angle_diff( cur_heading, direction.a ) ;
    da2 = get_angle_diff( cur_heading, normalize_angle( direction.a - PI )) ;

    if( reverse_possible )
    {
      if( Math.abs(da1) < Math.abs(da2) )
      {
        da = da1 ;
        speed = speed_max ;
      }
      else
      {
        da = da2 ;
        speed = -speed_max ;
      }
    }
    else
    {
      if( speed > 0 )
        da = da1 ;
      else
        da = da2 ;
    }
  }
} // driver_t

//======
// Класс "Радар".
//
public class radar_t extends Object
{
  //*** Атрибуты.
  private final boolean SWITCH_DEBUG =      true ;
  private final boolean INPUTS_LOGGING =    true && SWITCH_DEBUG && ANY_DEBUG ;
  private final boolean OUTPUTS_LOGGING =   true && SWITCH_DEBUG && ANY_DEBUG ;
  private final boolean A4_LOGGING =        true && SWITCH_DEBUG && ANY_DEBUG ;
  private final boolean A4_BEGIN_LOGGING =  true && A4_LOGGING && ANY_DEBUG ;
  private final boolean A4_END_LOGGING =    true && A4_LOGGING && ANY_DEBUG ;
  private final boolean A4_ERROR_LOGGING =  true && A4_LOGGING && ANY_DEBUG ;
  private final boolean A4_TRANS_LOGGING =  true && A4_LOGGING && ANY_DEBUG ;

  private double        old_heading, cur_heading ; // Направление радара.
                        // Угол, на который надо повернуть радар на следующем шаге.
  private double        da ;
  private double        radar_path ;               // Пройденный радаром путь.


  private int           y4 = 0 ;
```

```
// Начало раунда.
public void
begin_round()
{
  old_heading = cur_heading = getRadarHeadingRadians() ;
  da = 0 ;
  radar_path = 0 ;
}

// Начало шага.
public void
begin_turn()
{
  old_heading = cur_heading ;
  cur_heading = getRadarHeadingRadians() ;
  radar_path = radar_path + get_angle_diff( old_heading, cur_heading ) ;

  A4(10) ;
}

// Конец шага.
public void
end_turn()
{
  setTurnRadarRightRadians( da ) ;
}

// Реализация автомата A4.
private void
A4( int e )
{
  int   y_old = y4 ;

  if( OBJECTS_LOGGING )
    log( "Для объекта 'Радар':" ) ;

  if( A4_BEGIN_LOGGING )
    log_begin( "A4", y4, e ) ;

  switch( y4 )
  {
    case 0:
    {
      boolean   X70 = x70() ;

      if( X70 && x80() ) { z101_0() ;                    y4 = 1 ; }
      else
      if( X70 )                                          y4 = 2 ;
      else
      { z100_0() ; }
    }
    break ;

    case 1:
    {
      boolean   X70 = x70() ;

      if( X70 && x80() ) { z101_0() ;                    y4 = 0 ; }
      else
      if( X70 )                                          y4 = 3 ;
      else
      { z100_1() ; }
    }
    break ;

    case 2:
      if( x70() ) { z101_0() ;                           y4 = 0 ; }
      else
                                                         y4 = 0 ;
    break ;

    case 3:
      if( x70() ) { z101_0() ;                           y4 = 1 ; }
      else
                                                         y4 = 1 ;
    break ;
```

```
    default :
      if( A4_ERROR_LOGGING )
        log_error( "A4", y4 ) ;
  }

  if( y4 != y_old )
  {

  if( A4_TRANS_LOGGING )
    log_trans( "A4", y4, y_old ) ;

  switch( y4 )
  {
    case 0:
      z100_0() ;
    break ;

    case 1:
      z100_1() ;
    break ;

    case 2:
      z101_0() ; z100_1() ;
    break ;

    case 3:
      z101_0() ; z100_0() ;
    break ;
  }

  }

  if( A4_END_LOGGING )
    log_end( "A4", y4 ) ;
}

//*** Реализация входных переменных.
private boolean
x70()
{
                    // Обращение к объекту класса "Список целей".
  boolean result = targets.scan_completed() ;

  if( INPUTS_LOGGING )
    log_input( "x70", "Цикл сканирования завершен", result ) ;
  return result ;
}

private boolean
x80()
{
  boolean result = Math.abs(radar_path) < PI ;

  if( INPUTS_LOGGING )
    log_input( "x80", "Пройденный радаром путь меньше 180 градусов", result ) ;
  return result ;
}

//*** Реализация выходных воздействий.
private void
z100_0()
{
  if( OUTPUTS_LOGGING )
    log_output( "z100_0", "Повернуть радар влево" ) ;

  da = -1000 ;
}

private void
z100_1()
{
  if( OUTPUTS_LOGGING )
    log_output( "z100_1", "Повернуть радар вправо" ) ;

  da = 1000 ;
}
```

```
  private void
  z101_0()
  {
    if( OUTPUTS_LOGGING )
      log_output( "z101_0", "Сбросить память пройденного радаром пути" ) ;

    radar_path = 0 ;
  }
} // radar_t

//======
// Класс "Список целей".
//
public class targets_t extends Object
{
  //*** Атрибуты.
  public target_t       closest_target ;       // Ближайшая цель.
  private final double  targets_k_delta = 13 ; // Изменение коэффициента
                                               // "отталкивания" от целей в
                                               // зависимости от количества целей.

  private double        targets_k ;            // Коэффициент "отталкивания" от целей.
  private Hashtable     targets_table = new Hashtable(1) ;
  private long          last_scan_completion_time = 0 ;

  // Начало раунда.
  public void
  begin_round()
  {
    Enumeration targets_list = targets_table.elements() ;
    target_t    t ;

    // Коэффициент "отталкивания" от целей зависит от количества целей.
    targets_k = 270 - getOthers() * targets_k_delta ;
    if( targets_k <= 0 ) targets_k = 10 ;
    closest_target = null ;

    while( targets_list.hasMoreElements() )
    {
      t = (target_t)targets_list.nextElement() ;
      t.begin_round() ;
      //log( t.name ) ;
    }

    last_scan_completion_time = 0 ;
  }

  // Начало шага.
  public void
  begin_turn()
  {
    Enumeration targets_list = targets_table.elements() ;
    target_t    t ;

    while( targets_list.hasMoreElements() )
    {
      t = (target_t)targets_list.nextElement() ;
      t.refresh() ;
    }

    closest_target = get_closest_target( 100 ) ;
  }

  // Обновление цели.
  public void
  update( ScannedRobotEvent e )
  {
    target_t    t ;
    String      robot_name = e.getName() ;

    t = (target_t)targets_table.get( robot_name ) ;
    if( t != null )
    {
      // Цель существует. Обновить информацию.

      //log( getTime() + ": Обновить информацию о " + robot_name ) ;
```

```
    t.update(e) ;
  }
  else
  {
    // Создать запись о новой обнаруженной цели.
    //log( "Новая цель (" + robot_name + ")" ) ;
    t = new target_t(e) ;
    targets_table.put( robot_name, t ) ;
  }
}

// Уничтожение цели.
public void
one_less( RobotDeathEvent e )
{
  target_t    t ;
  String      robot_name = e.getName() ;

  t = (target_t)targets_table.get( robot_name ) ;
  targets_k += targets_k_delta ;
  if( t != null )
    t.destroyed() ;
}

// Попадание в цель.
public void
hit( BulletHitEvent e )
{
  target_t    t ;
  String      robot_name = e.getName() ;

  //log( "Targets hit for " + robot_name ) ;

  t = (target_t)targets_table.get( robot_name ) ;
  if( t != null )
    t.hit() ;
}

// Столкновение с целью.
public void
collide( HitRobotEvent e )
{
  target_t    t ;
  String      robot_name = e.getName() ;

  //log( "Targets hit for " + robot_name ) ;

  t = (target_t)targets_table.get( robot_name ) ;
  if( t != null )
    t.collide() ;
}

// Попадание в нас.
public void
hit_by_bullet( HitByBulletEvent e )
{
  target_t    t ;
  String      robot_name = e.getName() ;

  t = (target_t)targets_table.get( robot_name ) ;
  if( t != null )
    t.hit_by_bullet() ;
}

// Вернуть ближайшую цель.
public target_t
get_closest_target( double time_on_aiming )
{
  Enumeration targets_list = targets_table.elements() ;
  target_t    chosen_target = null, t = null ;

  // Среди целей, на которые орудие успеет навестить быстрее,
  // чем за заданное время, выбрать ближайшую.
  while( targets_list.hasMoreElements() )
  {
    t = (target_t)targets_list.nextElement() ;
```

```
    if( t.is_tracked() )
    {
      //log( "Проверили " + t.name + " на " + t.a *180/PI ) ;
      //if( Math.abs(shortest_turn( t.a - gunner.cur_heading() ))
      if( Math.abs(get_angle_diff( gunner.cur_heading(), t.a ))
          < time_on_aiming * gun_rotation_speed )
      {
        // Цель в зоне досягаемости орудия.
        //log( t.name + " в зоне." ) ;

        if( chosen_target == null )
          chosen_target = t ;
        else
        if( t.R < chosen_target.R )
          chosen_target = t ;
      }
    }
  }

  return chosen_target ;
}

// Вернуть направление на удаление от видимых целей.
public vect_t
get_reaction_vector()
{
  Enumeration targets_list = targets_table.elements() ;
  target_t    t ;
  vect_t      reaction = new vect_t() ;

  while( targets_list.hasMoreElements() )
  {
    t = (target_t)targets_list.nextElement() ;
    if( t.is_tracked() )
    {
      reaction.addrad( normalize_angle(t.a+PI), targets_k / t.R ) ;
    }
  }
  return reaction ;
}

// Проверить, что очередной цикл сканирования завершен.
public boolean
scan_completed()
{
  Enumeration targets_list = targets_table.elements() ;
  target_t    t ;
  int         tracked_targets = 0 ;

  //log( "Вызов scan_completed" ) ;

  if( !targets_list.hasMoreElements() ) return false ;
  if( cur_time < 20 ) return false ;

  // Проверить, все ли сопровождаемые цели отсканированы.
  while( targets_list.hasMoreElements() )
  {
    t = (target_t)targets_list.nextElement() ;
    if( t.is_tracked() )
    {
      // Цель сопровождается.

      if( !t.was_scanned_after(last_scan_completion_time) )
      {
        // Цель не была сканирована в новом проходе.
        // следовательно новый проход сканирования еще не завершен.
        return false ;
      }
      tracked_targets++ ;
    }
  }

  // Здесь оказались, если:
  //  - список целей не пуст;
  //  - не начало раунда;
  //  - все сопровождаемые цели (если они есть) отсканированы.
  // Значит если есть сопровождаемые цели, то цикл сканирования завершен.
```

```
    if( tracked_targets > 0 )
    {
      last_scan_completion_time = cur_time ;
      return true ;
    }
    else
      return false ;
  }

  //======
  //
  //
  public void
  show_statistics()
  {
    Enumeration targets_list = targets_table.elements() ;
    target_t    t ;

    while( targets_list.hasMoreElements() )
    {
      t = (target_t)targets_list.nextElement() ;
      t.show_statistics() ;
    }
  }
} // targets_t

//======
// Класс "Цель".
//
public class target_t extends Object
{
  //*** Аттрибуты.
  private final boolean SWITCH_DEBUG =      true ;
  private final boolean INPUTS_LOGGING =    true && SWITCH_DEBUG && ANY_DEBUG ;
  private final boolean OUTPUTS_LOGGING =   true && SWITCH_DEBUG && ANY_DEBUG ;
  private final boolean A5_LOGGING =       true && SWITCH_DEBUG && ANY_DEBUG ;
  private final boolean A5_BEGIN_LOGGING = true && A5_LOGGING && ANY_DEBUG ;
  private final boolean A5_END_LOGGING =   true && A5_LOGGING && ANY_DEBUG ;
  private final boolean A5_ERROR_LOGGING = true && A5_LOGGING && ANY_DEBUG ;
  private final boolean A5_TRANS_LOGGING = true && A5_LOGGING && ANY_DEBUG ;
  private final double  long_range = 650 ;
  private final double  close_range = 150 ;
  private final double  point_blank_range = 70 ;
  private final long    old_info = 20 ; // Количество раундов, через которое
                                        // информация считается устаревшей.
  private String       name ;           // Имя цели.
  private double       a,               // Пеленг.
                       R,               // Дистанция.
                       v, old_v,        // Линейная скорость.
                       med_v,
                       dv,              // Изменение скорости.
                       w, old_w,        // Скорость поворота.
                       med_w ;
  private boolean      collision_detected ;
                       // Направление движения цели при последнем измерении.
  private double       cur_target_heading,
                       // Направление движения цели при предпоследнем измерении.
                       old_target_heading ;

                       // Время последнего измерения параметров цели.
  private long         last_update_time ;
                       // На какое время рассчитывается упреждение.
  private double       prediction_interval ;
                       // Упреждающий вектор от текущего положения нашего робота.
  private vect_t       predicted_vector = new vect_t() ;
  private double       cur_target_life ;
  private double       hits, shots ;
  private double       T_base, P_base, P0 ;
  private double       T_medium ;

  private int          y5 = 0 ;
```

```
// Конструктор.
target_t( ScannedRobotEvent e )
{
  name = e.getName() ;
  a = normalize_angle( Cynical_3.this.getHeadingRadians()
                     + e.getBearingRadians()) ;
  R = e.getDistance() ;
  med_v = old_v = v = e.getVelocity() ;
  dv = 0 ;
  med_w = old_w = w = 0 ;
  old_target_heading = cur_target_heading = e.getHeadingRadians() ;
  last_update_time = cur_time ;

  hits = 0 ; shots = 0 ;
  T_medium = T_base = 3 ;
  P_base = 1 ; P0 = 1.2 ;

  A5(100,e) ;
}

// Начало раунда.
public void
begin_round()
{
  A5(120, null) ;
}

// Обновление состояния цели.
public void
refresh()
{
  A5(140, null) ;
}

// Обновление цели.
public void
update( ScannedRobotEvent e )
{
  //log( "Updating target " + name ) ;

  A5(100, e) ;
}

// Уничтожение цели.
public void
destroyed()
{
  A5(110, null) ;
}

// Попадание в цель.
public void
hit()
{
  A5(130, null) ;
}

// Столкновение с целью.
public void
collide()
{
  A5(135, null) ;
}

// Попадание в нас.
public void
hit_by_bullet()
{
  A5(136, null) ;
}
```

```
// Выстрел по цели.
public void
shoot( double firepower )
{
  double T = predicted_vector.R / get_bullet_speed(firepower) ;

  T = T < 1 ? 1 : T ;

  double P = P0 - T * ( P0 - P_base ) / T_base  ;

  P = P > P0 ? P0 : P ;
  P = P < 0 ? 0 : P ;

  //log( "Выстрел, P = " + P + ", F = " + firepower ) ;

  P = P / 1.2 ;

  P_base = P0 - T_base * ( P0 - P ) / T ;
  P_base = P_base > P0 ? P0 : P_base ;
  P_base = P_base < 0 ? 0 : P_base ;

  T_medium = ( T_medium + T ) / 2 ;

  //log( "После, P = " + P + ", Pb = " + P_base + ", Tm = " + T_medium ) ;

  shots++ ;
}

// Сброс статистики скорости.
public void
reset_speed_history()
{
  med_w = w ;
  med_v = v ;
}

// Цель сопровождается.
public boolean
is_tracked()
{
  return y5 == 2 ;
}

// Цель не обнаружена.
public boolean
is_unknown()
{
  return y5 == 0 ;
}

// Была ли цель обновлена после указанного времени.
public boolean
was_scanned_after( long scan_time )
{
  return last_update_time > scan_time ;
}

// Вернуть оптимальную мощность выстрела по заданной цели.
public double
get_firepower( double P )
{
  double  R = predicted_vector.R ;
  double  S = ( P0 - P_base ) * R / ( ( P0 - P )*T_base ) ;
  double  F = ( 20 - S ) / 3.0 ;

  //log( "F = " + F ) ;
  F = F > max_firepower ? max_firepower : F ;
  double  damage = F * 3
                   + ( F > 1 ? 2 * (F-1) : 0 ) ;

  if( damage > cur_target_life )
  {
    if( cur_target_life < 3 )
    {
      F = cur_target_life / 3.0 ;
      F = F < 0.1 ? 0.1 : F ;
    }
```

```
    else
    {
      F = (cur_target_life + 2.0) / 5.0 ;
    }
  }

  if( F < 0.1 )
  {
    if( (R < long_range) && (cur_life > life_warning) )
    {
      F = 0.1 ;
    }
    else
    if(     ( cur_target_life > cur_life )
        && ( (cur_time - last_fire_time) > fire_delay_critical ))
    {
      F = 0.1 ;
    }
  }
  else
  {
    if( cur_life < life_critical )
    {
      F = F > cur_life / 5 ? cur_life / 5 : F ;
      F = F < 0.1 ? 0.1 : F ;

      if( cur_life < 0.2 )
        F = 0 ;
    }
  }

  //log( "get_firepower: F = " + F + ", R = " + R + ", Pb = " + P_base
  //       + ", P0 = " + P0 + ", Tb = " + T_base + ", P = " + P ) ;

  return F ;
}

// Рассчитать грубое упреждение.
public vect_t
predict_position_roughly( double bullet_speed, double prediction_gap )
{
  double      cur_bullet_time ;

  cur_bullet_time = prediction_gap + R / bullet_speed ;

  predicted_vector.reset( a, R ) ;

  if( R > close_range )
  {
    if( Math.abs(v) > precision )
    {
      predicted_vector.add( get_path( med_v, med_w,
                                      cur_target_heading, cur_bullet_time )) ;
    }
  }

  return predicted_vector ;
}

// Рассчитать точное упреждение.
public vect_t
predict_position_exactly( double bullet_speed, double prediction_gap )
{
  predict_position( v, w, bullet_speed, prediction_gap ) ;

  return predicted_vector ;
}

// Рассчитать усредненное упреждение.
public vect_t
predict_position_medium( double bullet_speed, double prediction_gap )
{
  predict_position( med_v, med_w, bullet_speed, prediction_gap ) ;

  return predicted_vector ;
}
```

```
// Вывести статистику.
public void
show_statistics()
{
  log( "---- Статистика для " + name + " ----" ) ;
  log( "Выстрелов: " + shots + ", попаданий: " + hits ) ;
  log( "Вероятность: " + hits/shots + ", базовая: " + P_base ) ;
  log( "---------" ) ;
}

// Реализация автомата A5.
private void
A5( int e, Event robocode_event )
{
  int   y_old = y5 ;

  if( OBJECTS_LOGGING )
    log( "Для объекта 'Цель' (" + name + "):" ) ;

  if( A5_BEGIN_LOGGING )
    log_begin( "A5", y5, e ) ;

  switch( y5 )
  {
    case 0:
      if( e == 100 ) { z1001(robocode_event) ;   y5 = 2 ; }
      else
      if( e == 130 ) { z1010(robocode_event) ;   y5 = 2 ; }
      else
      if( e == 135 || e == 136 )                 y5 = 2 ;
      else
      if( e == 140 && x1000() )                  y5 = 1 ;
    break ;

    case 1:
      if( e == 100 ) { z1001(robocode_event) ;   y5 = 2 ; }
      else
      if( e == 130 ) { z1010(robocode_event) ;   y5 = 2 ; }
      else
      if( e == 135 || e == 136 )                 y5 = 2 ;
      else
      if( e == 120 )                             y5 = 0 ;
    break ;

    case 2:
      if( e == 110 || e == 140 && x1000() )      y5 = 1 ;
      else
      if( e == 120 )                             y5 = 0 ;
      else
      if( e == 100 )
      { z1001(robocode_event) ; }
      else
      if( e == 130 )
      { z1010(robocode_event) ; }
    break ;

    default :
      if( A5_ERROR_LOGGING )
        log_error( "A5", y5 ) ;
  }

  if( y5 != y_old )
  {
  if( A5_TRANS_LOGGING )
    log_trans( "A5", y5, y_old ) ;

  switch( y5 )
  {
    case 0:
      z1000() ;
    break ;

    case 1:
      z1000() ;
    break ;
  }
  }
```

```
  if( A5_END_LOGGING )
    log_end( "A5", y5 ) ;
}

//*** Реализация входных переменных.
private boolean
x1000()
{
  boolean result = (cur_time - last_update_time) > old_info ;

  if( INPUTS_LOGGING )
    log_input( "x1000", "Информация о цели устарела", result ) ;
  return result ;
}

//*** Реализация выходных воздействий.
private void
z1000()
{
  if( OUTPUTS_LOGGING )
    log_output( "z1000", "Сбросить параметры цели" ) ;

  med_w = old_w = w = 0 ;
  dv = 0 ;
  med_v = old_v = v = 0 ;
  last_update_time = 0 ;
  predicted_vector.reset() ;
}

private void
z1001( Event event )
{
  if( OUTPUTS_LOGGING )
    log_output( "z1001", "Обновить параметры цели" ) ;

  if( last_update_time != cur_time )
  {
    ScannedRobotEvent   e = (ScannedRobotEvent) event ;

    //log( "Updating target " + name ) ;

    cur_target_life = e.getLife() ;
    R = Math.abs(e.getDistance()) ;
    a = normalize_angle( getHeadingRadians() + e.getBearingRadians()) ;

    old_v = v ;
    v = e.getVelocity() ;
    med_v = (v + med_v) / 2.0 ;
    dv = (v - old_v)/( cur_time - last_update_time ) ;

    old_target_heading = cur_target_heading ;
    cur_target_heading = normalize_angle(e.getHeadingRadians()) ;
    old_w = w ;
    w = get_angle_diff(old_target_heading, cur_target_heading)
          /(cur_time - last_update_time) ;
    med_w = (w + med_w) / 2.0 ;

    //log( "v = " + v + ", dv = " + dv + ", w = " + w + ", accelerating = "
    //       + accelerating + ", braking = " + braking ) ;
    //log( "w = " + w ) ;

    last_update_time = cur_time ;

    //log( "Update (" + name + "): a = " + a*180/PI + ", R = " + R ) ;
  }
}

private void
z1010( Event e )
{
  if( OUTPUTS_LOGGING )
    log_output( "z1010", "Обновить статистику попаданий в цель" ) ;

  double P = P0 - T_medium * ( P0 - P_base ) / T_base  ;

  P = P * 1.4 ;
```

```
  P = P > P0 ? P0 : P ;
  P = P < 0 ? 0 : P ;

  P_base = P0 - T_base * ( P0 - P ) / T_medium ;
  P_base = P_base > P0 ? P0 : P_base ;
  P_base = P_base < 0 ? 0 : P_base ;

  hits++ ;
}

//*** Вспомогательные методы.

// Рассчитать упреждение с учетом:
// - отличия во времени подлета пули;
// - того, что выстрел будет с задержкой на prediction_gap.
private vect_t
predict_position( double v, double w,
                  double bullet_speed, double prediction_gap )
{
  final int   maximum_iterations = 5 ;
  double      old_bullet_time, cur_bullet_time ;
  int         i = 0 ;
  vect_t      cur_prediction = new vect_t() ;

  if( R < point_blank_range )
  {
    cur_prediction.reset( a, R ) ;
  }
  else
  if( Math.abs(v) > precision )
  {
    old_bullet_time = cur_bullet_time = prediction_gap + R / bullet_speed ;

    collision_detected = false ;
    cur_prediction = get_predicted_point( v, w, cur_bullet_time, true ) ;

    if( !collision_detected )
      for( i = 0 ; i < maximum_iterations ; i++ )
      {
        old_bullet_time = cur_bullet_time ;
        cur_bullet_time = prediction_gap + cur_prediction.R / bullet_speed ;
        if( Math.abs( cur_bullet_time - old_bullet_time ) < 0.5 ) break ;
        cur_prediction = get_predicted_point( v, w, cur_bullet_time, false ) ;
      }

    //log( "Количество итераций: " + i++ ) ;

  }
  else
  {
    cur_prediction.reset( a, R ) ;
  }

  predicted_vector.reset( cur_prediction.a, cur_prediction.R ) ;

  return predicted_vector ;
}

// Расчитать участок траектории с неизменной скоростью.
private vect_t
get_predicted_point( double v, double w,
                     double t, boolean check_collisions )
{
  vect_t    cur_prediction = new vect_t( a, R ) ;
            // Считаем от нашего танка, так как надо получить
            // абсолютные координаты для проверки вылета.
  double    time_discret = 5 ;

  if( check_collisions )
  {
    // Убогий (по реккурентной формуле) способ подсчета точки
    // столкновения цели со стеной.
    // Разбиваем весь участок траектории на куски и
    // для каждого куска проверяем выход за границу.
    // Если выход за границу произошел - определяем
    // точный момент выхода и считаем заново путь,
    // проделанный от исходной точки до рассчитанного момента выхода.
```

```
      for( double i = 0 ; i*time_discret < t ; i++ )
      {
        double  time_delta = Math.min( time_discret,
                                       t - i*time_discret ) ;

        cur_prediction.add( get_path( v, w, cur_target_heading, time_delta )) ;

        if( vector_out_of_field( cur_prediction ) )
        {
          // Цель вылетела.
          // Момент времени, в который это произошло,
          // лежит в интервале [i*time_discret ; i*time_discret + time_delta].
          // Дальше считать по шагам.

          collision_detected = true ;
          cur_prediction.reset( a, R ) ;
          cur_prediction.add( get_path( v, w, cur_target_heading, i*time_discret )) ;

          for( int j = 1 ; j <= time_delta ; j++ )
          {
            cur_prediction.add( get_path( v, w, cur_target_heading, 1 )) ;
            if( vector_out_of_field( cur_prediction ) )
            {
              // Определили точный момент выхода цели за границу поля.
              // Он равен i*time_discret + j.
              // Таким образом, надо сосчитать путь за это время.
              cur_prediction.reset( a, R ) ;
              cur_prediction.add(
                get_path( v, w, cur_target_heading, i*time_discret + j )) ;
              return cur_prediction ;
            }
          }

          // Если оказались в этой точке программы, то цель почему-то
          // не доехала до края за время, которое рассчитывали по шагам.
          cur_prediction.reset( a, R ) ;
          cur_prediction.add(
            get_path( v, w, cur_target_heading, i*time_discret + time_delta )) ;
          return cur_prediction ;
        }
      }  // for под дискретам времени.
    } // if( check_collisions )

    // Если оказались в этой точке программы, то цель вообще
    // не выходила за пределы поля.
    cur_prediction.reset( a, R ) ;
    cur_prediction.add(
      get_path( v, w, cur_target_heading, t )) ;

    return cur_prediction ;
  }

  // Определить, выходит ли вектор, направленный из
  // текущего положения танка за границу поля.
  private boolean
  vector_out_of_field( vect_t vector )
  {
    double  x, y ;
    x = getX() + vector.x ;
    y = getY() + vector.y ;

    return (  x < 0 + collision_delta
           || x > battlefield_width - collision_delta
           || y < 0 + collision_delta
           || y > battlefield_height - collision_delta ) ;
  }
}   // target_t class.
```

```
  //======
  // Класс "Вектор".
  //
  public class vect_t extends Object
  {
    //*** Атрибуты.
    public double a, R, x, y ;

    //*** Конструкторы.
    vect_t()
    { reset() ; } ;

    vect_t( double a1, double R1 )
    { reset(a1,R1) ; }

    // Сбросить координаты.
    public void reset()
    { a = 0 ; R = 0 ; x = 0 ; y = 0 ; }

    // Сбросить координаты.
    public void reset( double a1, double R1 )
    { a = a1 ; R = R1 ; x = R1 * Math.sin(a1) ; y = R1 * Math.cos(a1) ; } ;

    // Прибавить вектор.
    public void add( vect_t vect2 )
    {
      adddec( vect2.x, vect2.y ) ;
    }

    // Прибавить вектор, заданный в декартовой системе координат.
    public void adddec( double x2, double y2 )
    {
      double a1 = a, R1 = R, x1 = x, y1 = y ;
      double x3 = 0, y3 = 0 ;
      double a3 = 0, R3 = 0 ;

      x3 = x1 + x2 ;
      y3 = y1 + y2 ;
      R3 = x3 * x3 + y3 * y3 ;
      x = x3 ; y = y3 ;
      if( R3 < 0 ) return ;
      R3 = Math.sqrt( R3 ) ;
      a3 = get_angle( x3, y3 ) ;
      a = normalize_angle(a3) ; R = R3 ;
    }

    // Прибавить вектор, заданный в радиальной системе координат.
    public void addrad( double a2, double R2 )
    {
      double a1 = a, R1 = R, x1 = x, y1 = y ;
      double x2 = 0, y2 = 0 ;
      double x3 = 0, y3 = 0 ;
      double a3 = 0, R3 = 0 ;

      x2 = R2 * Math.sin(a2) ; y2 = R2 * Math.cos(a2) ;
      x3 = x1 + x2 ;
      y3 = y1 + y2 ;
      R3 = x3 * x3 + y3 * y3 ;
      x = x3 ; y = y3 ;

      if( R3 < 0 ) return ;
      R3 = Math.sqrt( R3 ) ;
      a3 = get_angle( x3, y3 ) ;

      //log( "x1 = " + x1 + ", y1 = " + y1 + ", a1 = " + a1*180/PI ) ;
      //log( "x2 = " + x2 + ", y2 = " + y2 + ", a2 = " + a2*180/PI ) ;
      //log( "x3 = " + x3 + ", y3 = " + y3 + ", a3 = " + a3*180/PI ) ;
      //log( "---" ) ;
      a = normalize_angle(a3) ; R = R3 ;

      //log( "(" + a1 + "," + R1 + ") + (" + a2 + "," + R2 + ") = ("
      //       + a3 + "," + R3 + ")" ) ;
    }
  }  // vect_t class.
} // Класс "Супервизор".
```

12. Фрагмент протокола боя двух танков

```
Для объекта 'Супервизор':
{ A0: Автомат A0 запущен в состоянии 0 с событием e9
 * z10_0: Инициализация при запуске
 T A0: Автомат A0 перешел из состояния 0 в состояние 1
 * z10_1: Инициализация в начале раунда

*** Раунд 1

 * z10_2: Инициализация в начале шага
} A0: Автомат A0 завершил свою работу в состоянии 1
--------- 0 --------- Начальный шаг (e9)
Для объекта 'Супервизор':
{ A0: Автомат A0 запущен в состоянии 1 с событием e10
 * z10_2: Инициализация в начале шага
} A0: Автомат A0 завершил свою работу в состоянии 1
Для объекта 'Стрелок':
{ A1: Автомат A1 запущен в состоянии 0 с событием e10
 * z30: Выбрать цель
 * z70: Сбросить историю маневрирования цели
 * z50_1: Рассчитать грубое упреждение и направить пушку
} A1: Автомат A1 завершил свою работу в состоянии 0
{ A2: Автомат A2 запущен в состоянии 0 с событием e10
 i x10: Подсчет скорости охлаждения пушки завершен? - НЕТ
 * z35: Подсчет скорости охлаждения пушки
} A2: Автомат A2 завершил свою работу в состоянии 0
Для объекта 'Радар':
{ A4: Автомат A4 запущен в состоянии 0 с событием e10
 i x70: Цикл сканирования завершен? - НЕТ
 * z100_0: Повернуть радар влево
} A4: Автомат A4 завершил свою работу в состоянии 0
Для объекта 'Водитель':
{ A3: Автомат A3 запущен в состоянии 0 с событием e10
 i x100: Враг близко? - ДА
 i x110: Сработал таймер T110? - ДА
 * z200_0: Инициализация движения по траектории 'Маятник'
 * z200_1: Добавить случайную составляющую к траектории 'Маятник'
 * z200_2: Определить направление и скорость движения 'Маятник'
} A3: Автомат A3 завершил свою работу в состоянии 0

--------- 30 --------- Выстрел по цели
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e100
 * z1001: Обновить параметры цели
} A5: Автомат A5 завершил свою работу в состоянии 2
Для объекта 'Супервизор':
{ A0: Автомат A0 запущен в состоянии 1 с событием e10
 * z10_2: Инициализация в начале шага
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e140
 i x1000: Информация о цели устарела? - НЕТ
} A5: Автомат A5 завершил свою работу в состоянии 2
} A0: Автомат A0 завершил свою работу в состоянии 1
Для объекта 'Стрелок':
{ A1: Автомат A1 запущен в состоянии 3 с событием e10
 i x26: Цель потеряна? - НЕТ
 i x21: Пушка охладилась? - ДА
 i x50: Наведение точное? - ДА
 * z60: Выстрел
 T A1: Автомат A1 перешел из состояния 3 в состояние 0
 * z30: Выбрать цель
 * z70: Сбросить историю маневрирования цели
} A1: Автомат A1 завершил свою работу в состоянии 0
{ A2: Автомат A2 запущен в состоянии 1 с событием e10
} A2: Автомат A2 завершил свою работу в состоянии 1
Для объекта 'Радар':
{ A4: Автомат A4 запущен в состоянии 0 с событием e10
 i x70: Цикл сканирования завершен? - ДА
 i x80: Пройденный радаром путь меньше 180 градусов? - ДА
 * z101_0: Сбросить память пройденного радаром пути
 T A4: Автомат A4 перешел из состояния 0 в состояние 1
 * z100_1: Повернуть радар вправо
} A4: Автомат A4 завершил свою работу в состоянии 1
Для объекта 'Водитель':
{ A3: Автомат A3 запущен в состоянии 0 с событием e10
```

```
 i x100: Враг близко? - ДА
 i x110: Сработал таймер T110? - ДА
 * z200_0: Инициализация движения по траектории 'Маятник'
 * z200_1: Добавить случайную составляющую к траектории 'Маятник'
 * z200_2: Определить направление и скорость движения 'Маятник'
} A3: Автомат A3 завершил свою работу в состоянии 0

--------- 42 --------- Попадание в цель (e130)
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e100
 * z1001: Обновить параметры цели
} A5: Автомат A5 завершил свою работу в состоянии 2
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e130
 * z1010: Обновить статистику попаданий в цель
} A5: Автомат A5 завершил свою работу в состоянии 2
Для объекта 'Супервизор':
{ A0: Автомат A0 запущен в состоянии 1 с событием e10
 * z10_2: Инициализация в начале шага
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e140
 i x1000: Информация о цели устарела? - НЕТ
} A5: Автомат A5 завершил свою работу в состоянии 2
} A0: Автомат A0 завершил свою работу в состоянии 1
Для объекта 'Стрелок':
{ A1: Автомат A1 запущен в состоянии 1 с событием e10
 i x26: Цель потеряна? - НЕТ
 i x20: Пушка скоро охладится? - НЕТ
 * z50_1: Рассчитать грубое упреждение и направить пушку
} A1: Автомат A1 завершил свою работу в состоянии 1
{ A2: Автомат A2 запущен в состоянии 1 с событием e10
} A2: Автомат A2 завершил свою работу в состоянии 1
Для объекта 'Радар':
{ A4: Автомат A4 запущен в состоянии 1 с событием e10
 i x70: Цикл сканирования завершен? - ДА
 i x80: Пройденный радаром путь меньше 180 градусов? - ДА
 * z101_0: Сбросить память пройденного радаром пути
 T A4: Автомат A4 перешел из состояния 1 в состояние 0
 * z100_0: Повернуть радар влево
} A4: Автомат A4 завершил свою работу в состоянии 0
Для объекта 'Водитель':
{ A3: Автомат A3 запущен в состоянии 0 с событием e10
 i x100: Враг близко? - ДА
 i x110: Сработал таймер T110? - ДА
 * z200_0: Инициализация движения по траектории 'Маятник'
 * z200_1: Добавить случайную составляющую к траектории 'Маятник'
 * z200_2: Определить направление и скорость движения 'Маятник'
} A3: Автомат A3 завершил свою работу в состоянии 0

--------- 59 --------- Попадание в нас (e45)
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e100
 * z1001: Обновить параметры цели
} A5: Автомат A5 завершил свою работу в состоянии 2
Для объекта 'Водитель':
{ A3: Автомат A3 запущен в состоянии 0 с событием e45
 i x100: Враг близко? - ДА
 i x110: Сработал таймер T110? - ДА
 * z200_0: Инициализация движения по траектории 'Маятник'
 * z200_1: Добавить случайную составляющую к траектории 'Маятник'
 * z200_2: Определить направление и скорость движения 'Маятник'
} A3: Автомат A3 завершил свою работу в состоянии 0
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e136
} A5: Автомат A5 завершил свою работу в состоянии 2
Для объекта 'Супервизор':
{ A0: Автомат A0 запущен в состоянии 1 с событием e10
 * z10_2: Инициализация в начале шага
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e140
 i x1000: Информация о цели устарела? - НЕТ
} A5: Автомат A5 завершил свою работу в состоянии 2
} A0: Автомат A0 завершил свою работу в состоянии 1
Для объекта 'Стрелок':
{ A1: Автомат A1 запущен в состоянии 2 с событием e10
 i x26: Цель потеряна? - НЕТ
 i x30: До конца поворота пушки меньше двух ходов? - ДА
```

```
 i x22: До конца охлаждения пушки меньше двух ходов? - ДА
 T A1: Автомат A1 перешел из состояния 2 в состояние 3
 * z40: Рассчитать мощность выстрела
 * z50_0: Рассчитать точное упреждение и направить пушку
} A1: Автомат A1 завершил свою работу в состоянии 3
{ A2: Автомат A2 запущен в состоянии 1 с событием e10
} A2: Автомат A2 завершил свою работу в состоянии 1
Для объекта 'Радар':
{ A4: Автомат A4 запущен в состоянии 0 с событием e10
 i x70: Цикл сканирования завершен? - ДА
 i x80: Пройденный радаром путь меньше 180 градусов? - ДА
 * z101_0: Сбросить память пройденного радаром пути
 T A4: Автомат A4 перешел из состояния 0 в состояние 1
 * z100_1: Повернуть радар вправо
} A4: Автомат A4 завершил свою работу в состоянии 1
Для объекта 'Водитель':
{ A3: Автомат A3 запущен в состоянии 0 с событием e10
 i x100: Враг близко? - ДА
 i x110: Сработал таймер T110? - ДА
 * z200_0: Инициализация движения по траектории 'Маятник'
 * z200_1: Добавить случайную составляющую к траектории 'Маятник'
 * z200_2: Определить направление и скорость движения 'Маятник'
} A3: Автомат A3 завершил свою работу в состоянии 0

--------- 332 --------- Конец раунда (e20)
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e100
 * z1001: Обновить параметры цели
} A5: Автомат A5 завершил свою работу в состоянии 2
Для объекта 'Супервизор':
{ A0: Автомат A0 запущен в состоянии 1 с событием e10
 * z10_2: Инициализация в начале шага
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e140
 i x1000: Информация о цели устарела? - НЕТ
} A5: Автомат A5 завершил свою работу в состоянии 2
} A0: Автомат A0 завершил свою работу в состоянии 1
Для объекта 'Стрелок':
{ A1: Автомат A1 запущен в состоянии 0 с событием e10
 i x25: Цель выбрана? - ДА
 T A1: Автомат A1 перешел из состояния 0 в состояние 1
 * z50_1: Рассчитать грубое упреждение и направить пушку
} A1: Автомат A1 завершил свою работу в состоянии 1
{ A2: Автомат A2 запущен в состоянии 1 с событием e10
} A2: Автомат A2 завершил свою работу в состоянии 1
Для объекта 'Радар':
{ A4: Автомат A4 запущен в состоянии 1 с событием e10
 i x70: Цикл сканирования завершен? - ДА
 i x80: Пройденный радаром путь меньше 180 градусов? - ДА
 * z101_0: Сбросить память пройденного радаром пути
 T A4: Автомат A4 перешел из состояния 1 в состояние 0
 * z100_0: Повернуть радар влево
} A4: Автомат A4 завершил свою работу в состоянии 0
Для объекта 'Водитель':
{ A3: Автомат A3 запущен в состоянии 0 с событием e10
 i x100: Враг близко? - ДА
 i x110: Сработал таймер T110? - ДА
 * z200_0: Инициализация движения по траектории 'Маятник'
 * z200_1: Добавить случайную составляющую к траектории 'Маятник'
 * z200_2: Определить направление и скорость движения 'Маятник'
} A3: Автомат A3 завершил свою работу в состоянии 0
Для объекта 'Супервизор':
{ A0: Автомат A0 запущен в состоянии 1 с событием e20
 T A0: Автомат A0 перешел из состояния 1 в состояние 2
 * z20: Вывести статистику раунда
---- Статистика для gg.Tarsier ----
Выстрелов: 23, попаданий: 5
Вероятность: 0.217, базовая: 0.943
---------
Выстрелов: 23, попаданий: 5, промахов: 18
Меткость: 0.227
Попали в нас: 7
Столкновений со стенами: 0
} A0: Автомат A0 завершил свою работу в состоянии 2
```