Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

А. Д. Лопатухина

Модель искусственного интеллекта игрового бота

Объектно-ориентированное программирование с явным выделением состояний

Проектная документация

Проект создан в рамках
«Движения за открытую проектную документацию»
http://is.ifmo.ru

Санкт-Петербург
2003

Введение

Цель проекта – создание игры, в которой принимает участие человек и бот, управляемый искусственным интеллектом (ИИ). Для реализации ИИ было решено использовать SWITCH-технологию, предложенную А. А. Шалыто [1] и развитую им совместно с Н. И. Туккелем (подробно ознакомиться с этой технологией можно на сайте [2]). Выбор этой технологии обуславливается тем, что она наиболее удобна для решения поставленной задачи.

1. Постановка задачи

Опишем подробно модель поведения «бота», приводя в скобках пояснения в терминах автоматов:

- в начальный момент времени монстр находится в состоянии ожидания *(состояние 0 автомата A0)*;
- из состояния ожидания монстр выходит при появлении противника *(событие e3)* или внешнего раздражителя. Раздражители бывают двух видов: обычный раздражитель *(e1)* и попадание *(e2)*;
- при появлении противника *(e3)* монстр переходит в атаку *(состояние 2 автомата A0)*;
- при появлении любого раздражителя монстр приступает к поиску противника *(состояние 1 автомата A0)*;
- если монстр, приступая к поиску, не был агрессивным, но во время поиска был ранен противником, он становится агрессивным *(петля в состоянии 1 автомата A0 по событию e2)*;
- время поиска ограничено сверху определенным значением;
- если за это время противник не найден *(e4)*, то монстр вновь возвращается в состояние ожидания;
- если в процессе поиска противник найден *(e3)*, монстр переходит в атаку (*состояние 2 автомата A0)*;
- попадание в монстра в процессе атаки приводит к тому, что он становится агрессивным, если он таковым еще не был *(петля в состоянии 2 автомата A0 по событию e2 с выполнением функции z03)*;
- смерть противника *(e6)* во время боя *(состояние 2 автомата A0)* приводит к тому, что монстр возвращается в состояние ожидания *(переход из состояния 2 в состояние 0 автомата A0 с выполнением функций z04, снимающей агрессию с монстра, и z05, завершающей атаку)*;
- потеря противника из виду *(e5)* во время боя приводит к тому, что монстр производит поиск *(переход в состояние 1 автомата A0 с выполнением функции z05, завершающей атаку)*. При этом уровень его агрессии не изменяется;
- смерть монстра *(состояние 3 автомата A0)* может произойти в любой момент.

Атака характеризуется следующими особенностями:

- она бывает трех типов: агрессивная *(состояние 3 автомата A1)*, преследование *(состояние 1 автомата A1)* и собственно атака *(состояние 2 автомата A1)*;
- если в момент начала атаки монстр неагрессивен *(x11)* и при этом расстояние до противника больше определенного значения *(x12)*, то он реализует преследование, особенность которого состоит в том, что при этом типе атаки монстр не стреляет, а лишь пытается догнать противника *(функция z12 не вызывается);*

- если монстр неагрессивен *(x11)* и при этом расстояние до противника не велико *(!x12)*, он реализует обычную атаку. При этом в каждый момент времени он перемещается в сторону противника *(z11)* и стреляет снарядами *(z12)*;
- если в момент начала атаки *(состояние 0 автомата A1)* монстр агрессивен *(!x11)*, он реализует агрессивную атаку, особенность которой состоит в том, что, реализуя эту атаку, монстр не может переключиться ни на какой другой тип атаки (*отсутствие переходов из состояния 3)*. Агрессивная атака может закончиться либо тем, что противник убежит от монстра, либо кто-нибудь из них умрет;
- если в процессе преследования монстр приближается к противнику на «пороговое» расстояние, то он переходит в атаку;
- если в процессе атаки противник сильно удаляется от монстра, то он переходит в преследование.

Для демонстрации работоспособности бота была создана упрощенная среда, обладающая всеми необходимыми характеристиками, в которой происходит поединок двух игроков. Правила поединка представлены ниже:

- сражение происходит на двумерном поле, на котором расставлены преграды, не проходимые как для игроков, так и для их снарядов;
- в сражении участвуют два игрока, один из которых управляется человеком с клавиатуры, а другой – ботом;
- управление человеком происходит с помощью четырех клавиш: стрелок вправо, влево и вверх и пробела;
- при нажатии пробела происходит выстрел в направлении, в котором игрок ориентирован в момент нажатия (количество снарядов не ограничено);
- при нажатии стрелки вправо игрок, управляемый человеком, поворачивается по часовой стрелке на определенных угол, при нажатии стрелки влево он поворачивается на такой же угол против часовой стрелки;
- при нажатии стрелки вперед игрок, управляемый человеком, продвигается на некоторое расстояние в том направлении, в котором он ориентирован в момент нажатия;
- если в процессе движения игрока он упирается в стену, то он перемещается только параллельно ей;
- если снаряд попадает в стену, то он уничтожается;
- если снаряд попадает в игрока, то снаряд уничтожается и жизнь игрока уменьшается на единицу;
- если два игрока сталкиваются, то жизни обоих уменьшаются на единицу;
- всего у игроков по четыре жизни, если в какой-то момент количество жизней некоего игрока становится равным нулю, то игрок погибает и восстанавливается в новом месте.

Пример окна работающей программы приведен на рис. 1. На этом рисунке фигуры красного цвета представляют собой преграды, игрок желтого цвета управляется человеком, а игрок зеленого цвета – бот. «Носик» игрока показывает, в каком направлении он ориентирован, а количество «лапок» определяется числом жизней (по одной «лапке» с каждой стороны на одну жизнь).

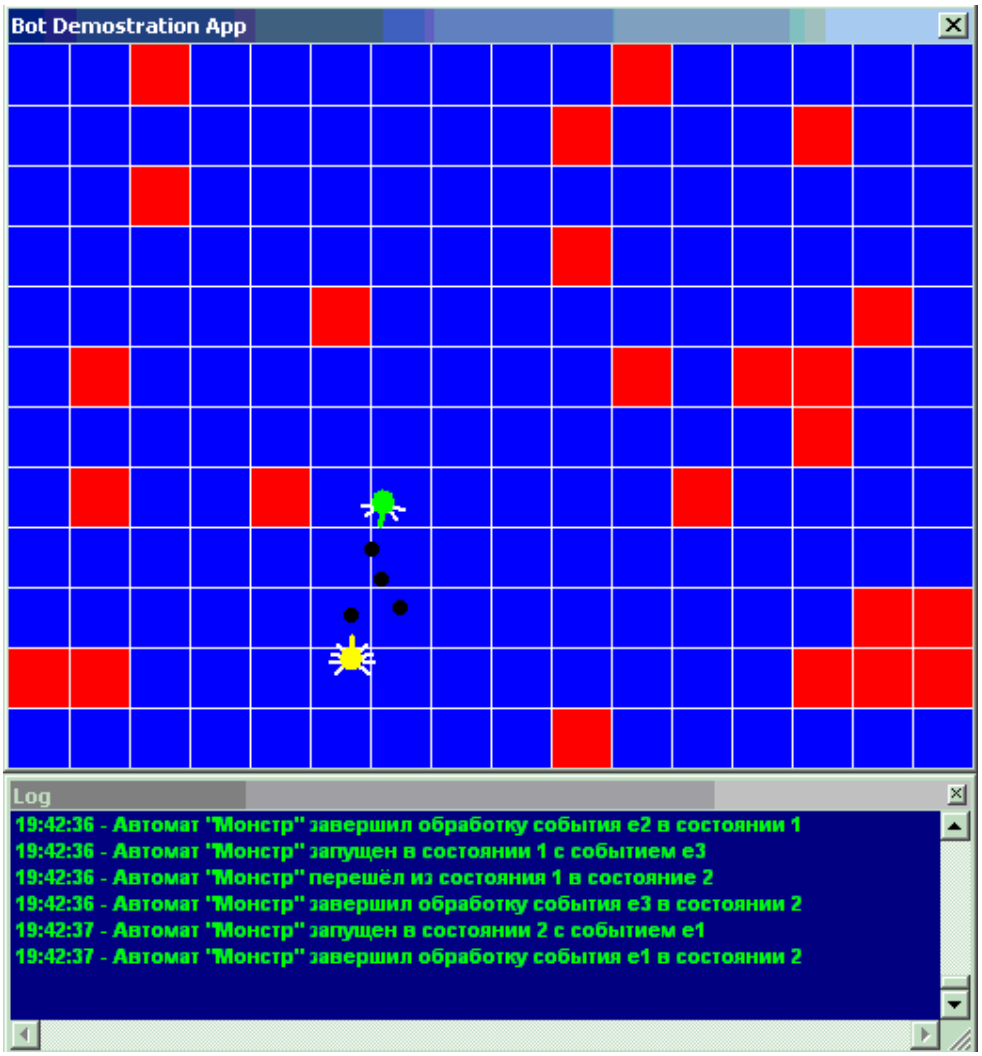

Рис. 1. Пример окна работающей программы

2. Диаграмма классов

Основные классы:

- класс «TGameObject» - базовый класс, который содержит поля и методы, общие для всех игровых объектов;
- класс «TMissle» - описывает снаряд;
- класс «TPlayer» - описывает игрока;
- класс «THumanPlayer» - описывает игрока, управляемого человеком;
- класс «TCPUPlayer» - описывает игрока, управляемого искусственным интеллектом;
- класс «TMonstr» - реализует автомат «Монстр» (A0);
- класс «TAttack» - реализует автомат «Атака» (A1).

Упрощенная диаграмма классов представлена на рис. 2.

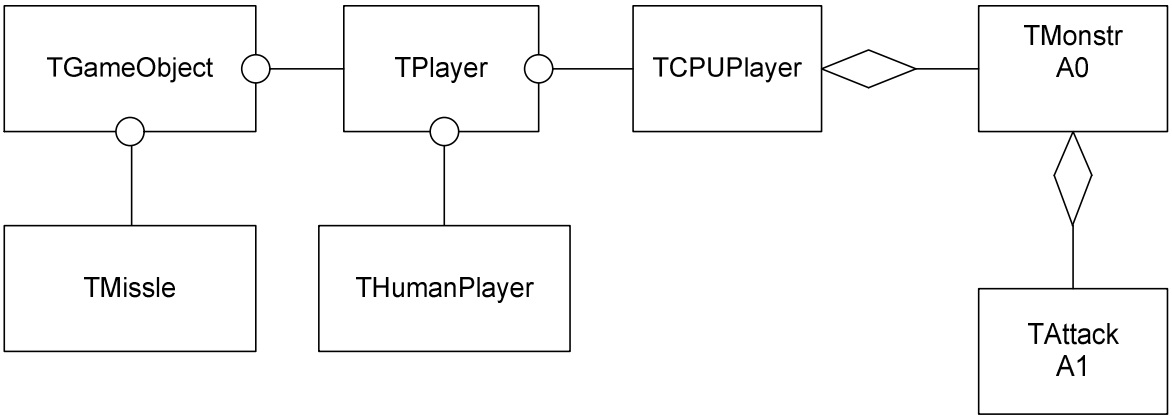

Рис. 2. Диаграмма классов

3. Класс «TGameObject»

3.1. Общее описание

Базовый класс, который содержит поля и методы, общие для всех игровых объектов, описываемых ниже.

3.2. Используемые методы

Перечислим используемые методы:

- Draw – производит отрисовку объекта;
- Update – производит обновление параметров;
- Name – возвращает имя объекта.

4. Класс «TMissle»

4.1. Общее описание

Класс реализует снаряд.

4.2. Используемые методы

Перечислим используемые методы:

- Draw – производит отрисовку снаряда;
- Update – производит обновление параметров;
- Name – возвращает имя снаряда.

5. Класс «TPlayer»

5.1. Общее описание

Класс содержит основные методы, необходимые для функционирования игрока.

5.2. Используемые методы

Перечислим используемые методы:

- Draw – производит отрисовку объекта;
- onKilled – реакция на уничтожение бота;
- onHit – реакция на попадание в бота;
- onOtherPlayerKilled – реакция на уничтожение другого игрока;
- onOtherPlayerFired – реакция на выстрел другого игрока;
- GeneratePosition –размещает игрока на корректной позиции;
- Restart – при необходимости производит перезапуск и уменьшает здоровье игрока;
- DoStep – делает шаг вперед;
- DoLeft – поворачивает налево;
- DoRight – поворачивает направо;
- DoFire – производит выстрел.

6. Класс «THumanPlayer»

6.1. Общее описание

Класс содержит основные методы, необходимые для функционирования игрока, управляемого человеком.

6.2. Используемые методы

Перечислим используемые методы:

- Update – производит обновление параметров;
- Name – возвращает имя игрока, управляемого человеком.

7. Класс «TCPUPlayer»

7.1. Общее описание

Класс содержит основные методы, необходимые для функционирования бота.

7.2. Используемые методы

Перечислим используемые методы:

- onKilled – реакция на уничтожение бота;
- onHit – реакция на попадание в бота;
- onOtherPlayerKilled – реакция на уничтожение другого игрока;
- onOtherPlayerFired – реакция на выстрел другого игрока;
- Update – производит обновление параметров;
- Name – возвращает имя объекта;
- OtherPlayerVisible – возвращает «истину», если другой игрок находится в пределах видимости;
- OtherPlayerAudible– возвращает «истину», если другой игрок издает звуки в пределах слышимости.

8. Класс «TMonstr»

8.1. Общее описание

Основу класса составляет интеллект монстра (автомат A0). Его описание приведено в следующем разделе.

8.2. Автомат «Монстр» (A0)

8.2.1. Общее описание

Автомат управления монстром. Он отвечает за общую стратегию ведения боя. Схема связей автомата приведена на рис. 3, а граф переходов – на рис. 4.

8.2.2. Схема связей

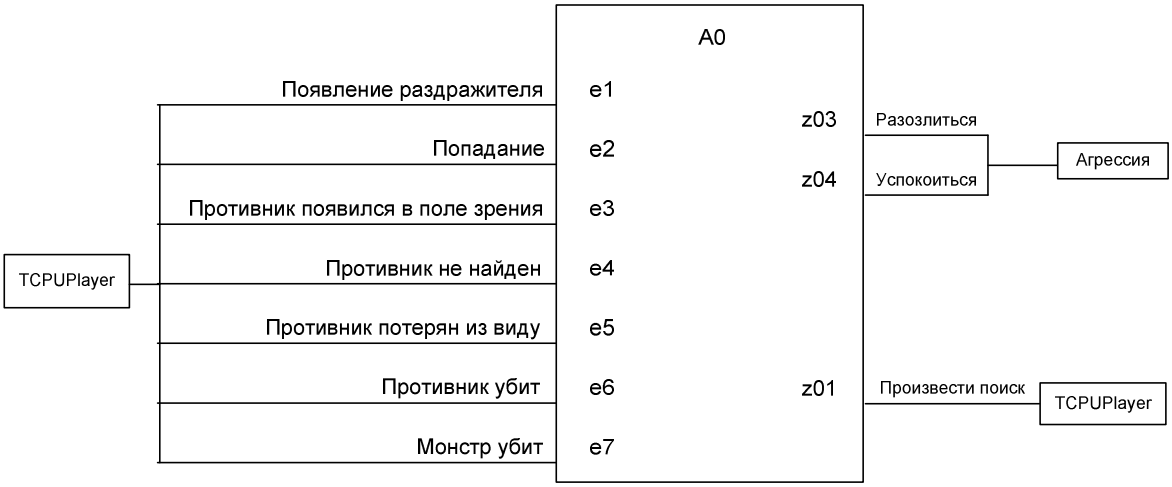

Рис. 3. Схема связей автомата «Монстр»

8.2.3.Граф переходов

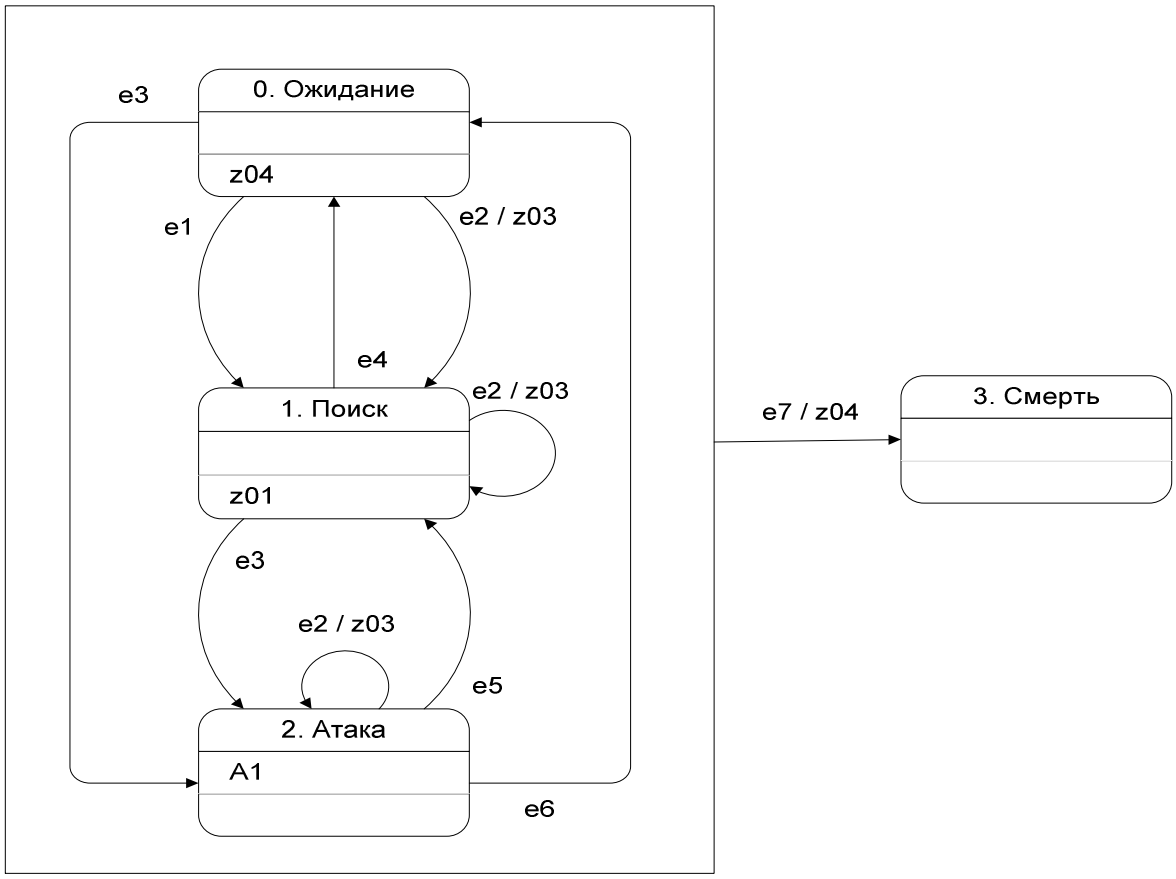

Рис. 4. Граф переходов автомата «Монстр»

9. Класс «TAttack»

9.1. Общее описание

Основу класса составляет автомат управления атакой монстра (A1). Его описание приведено ниже.

9.2. Автомат «Атака» (A1)

9.2.1. Общее описание

Автомат управления атакой монстра. Схема связей автомата приведена на рис. 5, граф перехода приведен на рис. 6.

9.2.2. Схема связей

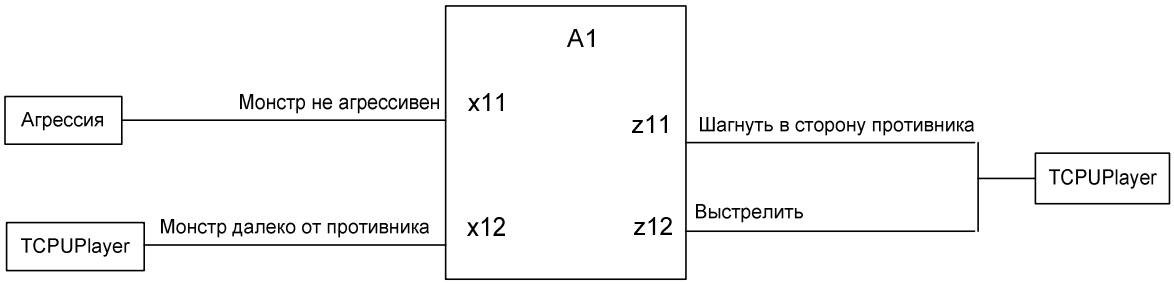

Рис. 5. Схема связей автомата «Атака»

9.2.3. Граф переходов

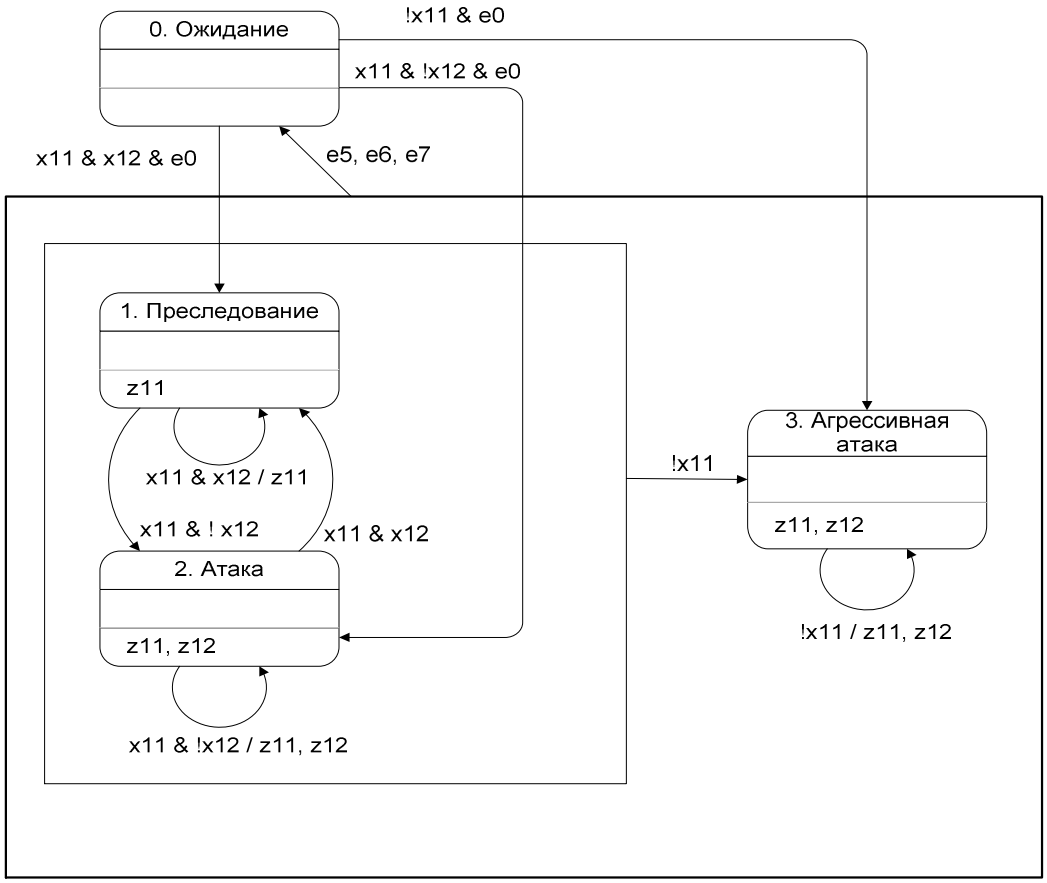

Рис. 6. Граф переходов автомата «Атака»

10. Пример протоколирования

Автоматически получен следующий протокол:

```
Автомат "Монстр" запущен в состоянии 0 с событием e3
Автомат "Монстр" перешёл из состояния 0 в состояние 2
Автомат "Атака" запущен в состоянии 0 с событием e0
Автомат "Атака" перешёл из состояния 0 в состояние 2
Автомат "Атака" завершил обработку события e0 в состоянии 2
Автомат "Монстр" завершил обработку события e3 в состоянии 2
Снаряд "Снаряд1" уничтожен: столкновение с объектом ("Игрок-
человек")
Игрок "Игрок-человек" был ранен
Автомат "Монстр" запущен в состоянии 2 с событием e2
Автомат "Атака" запущен в состоянии 2 с событием e2
Автомат "Атака" завершил обработку события e2 в состоянии 2
Автомат "Монстр" завершил обработку события e2 в состоянии 2
Игрок "Бот0" был ранен
Игрок "Игрок-человек" был ранен
Игрок "Игрок-человек" был ранен
Автомат "Монстр" запущен в состоянии 2 с событием e2
Автомат "Атака" запущен в состоянии 2 с событием e2
Автомат "Атака" перешёл из состояния 2 в состояние 3
Автомат "Атака" завершил обработку события e2 в состоянии 3
Автомат "Монстр" завершил обработку события e2 в состоянии 2
Игрок "Бот0" был ранен
Автомат "Монстр" запущен в состоянии 2 с событием e2
Автомат "Атака" запущен в состоянии 3 с событием e2
Автомат "Атака" завершил обработку события e2 в состоянии 3
Автомат "Монстр" завершил обработку события e2 в состоянии 2
Игрок "Бот0" был ранен
Игрок "Игрок-человек" был убит и восстановлен на новом месте
Автомат "Монстр" запущен в состоянии 2 с событием e6
Автомат "Атака" запущен в состоянии 3 с событием e6
Автомат "Атака" перешёл из состояния 3 в состояние 0
Автомат "Атака" завершил обработку события e6 в состоянии 0
Автомат "Монстр" перешёл из состояния 2 в состояние 0
Автомат "Монстр" завершил обработку события e6 в состоянии 0
Автомат "Монстр" запущен в состоянии 0 с событием e1
Автомат "Монстр" перешёл из состояния 0 в состояние 1
Автомат "Монстр" завершил обработку события e1 в состоянии 1
Автомат "Монстр" запущен в состоянии 1 с событием e3
Автомат "Монстр" перешёл из состояния 1 в состояние 2
Автомат "Атака" запущен в состоянии 0 с событием e0
Автомат "Атака" перешёл из состояния 0 в состояние 2
Автомат "Атака" завершил обработку события e0 в состоянии 2
Автомат "Монстр" завершил обработку события e3 в состоянии 2
Снаряд "Снаряд3" уничтожен: столкновение с объектом ("Снаряд2")
Автомат "Монстр" запущен в состоянии 2 с событием e1
Автомат "Атака" запущен в состоянии 2 с событием e1
Автомат "Атака" завершил обработку события e1 в состоянии 2
Автомат "Монстр" завершил обработку события e1 в состоянии 2
Снаряд "Снаряд6" уничтожен: столкновение с объектом ("Снаряд4")
Снаряд "Снаряд5" уничтожен: столкновение со стеной
Автомат "Монстр" запущен в состоянии 2 с событием e1
Автомат "Атака" запущен в состоянии 2 с событием e1
Автомат "Атака" завершил обработку события e1 в состоянии 2
Автомат "Монстр" завершил обработку события e1 в состоянии 2
Автомат "Монстр" запущен в состоянии 2 с событием e1
Автомат "Атака" запущен в состоянии 2 с событием e1
```

```
Автомат "Атака" завершил обработку события e1 в состоянии 2
Автомат "Монстр" завершил обработку события e1 в состоянии 2
Снаряд "Снаряд7" уничтожен: столкновение со стеной
Автомат "Монстр" запущен в состоянии 2 с событием e1
Автомат "Атака" запущен в состоянии 2 с событием e1
Автомат "Атака" завершил обработку события e1 в состоянии 2
Автомат "Монстр" завершил обработку события e1 в состоянии 2
Снаряд "Снаряд10" уничтожен: столкновение со стеной
Снаряд "Снаряд9" уничтожен: столкновение со стеной
Снаряд "Снаряд8" уничтожен: столкновение с объектом ("Снаряд12")
Снаряд "Снаряд11" уничтожен: столкновение со стеной
Снаряд "Снаряд13" уничтожен: столкновение со стеной
Автомат "Монстр" запущен в состоянии 2 с событием e1
Автомат "Атака" запущен в состоянии 2 с событием e1
Автомат "Атака" завершил обработку события e1 в состоянии 2
Автомат "Монстр" завершил обработку события e1 в состоянии 2
Автомат "Монстр" запущен в состоянии 2 с событием e1
Автомат "Атака" запущен в состоянии 2 с событием e1
Автомат "Атака" завершил обработку события e1 в состоянии 2
Автомат "Монстр" завершил обработку события e1 в состоянии 2
Снаряд "Снаряд14" уничтожен: столкновение с объектом
("Снаряд17")
Автомат "Монстр" запущен в состоянии 2 с событием e1
Автомат "Атака" запущен в состоянии 2 с событием e1
Автомат "Атака" завершил обработку события e1 в состоянии 2
Автомат "Монстр" завершил обработку события e1 в состоянии 2
Снаряд "Снаряд19" уничтожен: столкновение с объектом
("Снаряд16")
Снаряд "Снаряд18" уничтожен: столкновение с объектом ("Игрок-
человек")
Игрок "Игрок-человек" был ранен
Автомат "Монстр" запущен в состоянии 2 с событием e1
Автомат "Атака" запущен в состоянии 2 с событием e1
Автомат "Атака" завершил обработку события e1 в состоянии 2
Автомат "Монстр" завершил обработку события e1 в состоянии 2
Снаряд "Снаряд21" уничтожен: столкновение с объектом
("Снаряд20")
Снаряд "Снаряд22" уничтожен: столкновение с объектом ("Игрок-
человек")
Игрок "Игрок-человек" был ранен
Снаряд "Снаряд23" уничтожен: столкновение с объектом ("Игрок-
человек")
Игрок "Игрок-человек" был ранен
Автомат "Монстр" запущен в состоянии 2 с событием e7
Автомат "Атака" запущен в состоянии 2 с событием e7
Автомат "Атака" перешёл из состояния 2 в состояние 0
Автомат "Атака" завершил обработку события e7 в состоянии 0
Автомат "Монстр" перешёл из состояния 2 в состояние 0
Автомат "Монстр" завершил обработку события e7 в состоянии 0
Игрок "Бот0" был убит и восстановлен на новом месте
Игрок "Игрок-человек" был убит и восстановлен на новом месте
Автомат "Монстр" запущен в состоянии 0 с событием e6
Автомат "Монстр" завершил обработку события e6 в состоянии 0
```

11. Источники

1. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
2. http://is.ifmo.ru

Приложение. Исходные тексты программы

Файл «BotCPU.pas»

```
unit BotCPU;
interface
uses BotUnit, Graphics, SysUtils, Monstr;

var
  lastVisible  : boolean = false;
  lastAudible : boolean = false;

type


//Класс, описывающий бота
  TCPUPlayer = class(TPlayer)
  private
    ndx    : integer;
    Monstr : TMonstr;                                  //Автомат "Монстр"
  public
    procedure onKilled; override;                      //Реакция на уничтожение бота
    procedure onHit; override;                         //Реакция на попадание в бота
    procedure onOtherPlayerKilled(); override;         //Реакция на уничтожение игрока,
                                                       //управляемого человеком
    procedure onOtherPlayerFired(); override;          //Реакция на выстрел игрока,
                                                       //управляемого человеком
    constructor Create;                                //
    function Name(): string; override;                 //Возвращает имя объекта
    function OtherPlayerAudible: boolean;              //Проверяет, издает ли другой
                                                       //игрок звуки в пределах
                                                       //слышимости
    function OtherPlayerVisible: boolean;              //Проверяет, находится ли
                                                       //другой игрок в пределах
                                                       //видимости
    procedure Update(); override;                      //Производит обновление
                                                       //параметров бота
  end;

implementation
uses BotDemo;

constructor TCPUPlayer.Create;
var
  i: integer;
begin
  inherited;
  ndx:=objectCounter;
  inc(objectCounter);
  fRadius:=pWidth;
  fColor:=clLime;
  GeneratePosition();

  for i := 0 to high(Objects) do
    if Objects[i] <> nil then
      if Objects[i] is THumanPlayer then
      begin
        Monstr := TMonstr.Create(self, Objects[i]);
        break;
      end;

  search := 0;
end;

//Проверяет, издает ли другой игрок звуки в пределах слышимости
function TCPUPlayer.OtherPlayerAudible: boolean;
var
  i: integer;
```

```
begin
  result:=false;
  for i := 0 to high(Objects) do
    if (Objects[i] <> nil) and (Objects[i] is TPlayer) and (Objects[i] <> self) then
    begin
      result:=result or (sqrt(sqr(objects[i].xPosition-xPos)+sqr(objects[i].yPosition-
yPos)) < 60);
    end;
end;

//Проверяет, находится ли другой игрок в пределах видимости
function TCPUPlayer.OtherPlayerVisible: boolean;
var
  i: integer;
  rx,ry,a,b,c: double;
  ok1,ok2: boolean;
begin
  result:=false;
  for i := 0 to high(Objects) do
    if (Objects[i] <> nil) and (Objects[i] is TPlayer) and (Objects[i] <> self) then
    begin
      rx:=objects[i].xPosition;
      ry:=objects[i].yPosition;
      a:=cos((dir-8)*cor);
      b:=sin((dir-8)*cor);
      c:=-a*xPos-b*yPos;
      ok1:=a*rx+b*ry+c > 0;
      a:=cos((dir+8)*cor);
      b:=sin((dir+8)*cor);
      c:=-a*xPos-b*yPos;
      ok2:=a*rx+b*ry+c > 0;
      result:=result or (ok1 and ok2);
    end;
end;

//Возвращает имя объекта
function TCPUPlayer.Name(): string;
begin
  result:='Бот'+inttostr(ndx);
end;

//Реакция на уничтожение бота
procedure TCPUPlayer.onKilled;
begin
  Monstr.Run(7);
  lastVisible  := false;
  lastAudible  := false;
end;

//Реакция на попадание в бота
procedure TCPUPlayer.onHit;
begin
  Monstr.Run(2);
end;

//Реакция на уничтожение игрока, управляемого человеком
procedure TCPUPlayer.onOtherPlayerKilled();
begin
  Monstr.Run(6);
  lastVisible  := false;
  lastAudible  := false;
end;

//Реакция на выстрел игрока, управляемого человеком
procedure TCPUPlayer.onOtherPlayerFired();
begin
  Monstr.Run(1);
end;

//Производит обновление параметров бота
procedure TCPUPlayer.Update();
```

```
begin
  if Monstr.state = 2 then
    Monstr.Attack.Run(1024);

  if (search> 0) then
  begin
    if Monstr.attack.turnLeft then
      doLeft()
    else
      doRight();
    dec(search);
  end;

  inc(toFire);

  if OtherPlayerVisible and (not lastVisible) then
  begin
    search := 0;
    lastVisible := true;
    Monstr.Run(3);
  end
  else if (not OtherPlayerVisible) and lastVisible then
  begin
    lastVisible := false;
    Monstr.Run(5);
  end;

  if OtherPlayerAudible and (not OtherPlayerVisible) and (not lastAudible) then
    Monstr.Run(1);
  lastAudible := OtherPlayerAudible;

  CheckColission()
end;

end.
```

Файл «BotDemo.pas»

```
unit BotDemo;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, BotUnit, LogUnit, BotCPU;

type
  TMainForm = class(TForm)
    MainTimer: TTimer;
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure MainTimerTimer(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
    procedure FormKeyUp(Sender: TObject; var Key: Word; Shift: TShiftState);
  end;

var
  MainForm: TMainForm;
  DblBuffer: TBitmap;
  Maze: array [-1..12,-1..16] of boolean;
  Keys: array [0..255] of boolean;

  Objects: array of TGameObject;
```

```
procedure log(st: string);

implementation

{$R *.dfm}

//Производит обновление параметров системы
procedure UpdateGame();
var
  i: integer;
begin
  for i := 0 to high(Objects) do
    if Objects[i] <> nil then
      Objects[i].Update();
end;

//Готовит буфер для отрисовки
procedure PrepareBuffer();
var
  x,y,i: integer;
begin
  with DblBuffer do
  begin
    for y := 0 to 11 do
      for x:= 0 to 15 do
      begin
        if Maze[y,x] then Canvas.Brush.Color:=clRed
          else Canvas.Brush.Color:=clBlue;
        Canvas.FillRect(Rect(x*32,y*32,(x+1)*32,(y+1)*32));
      end;
    Canvas.Pen.Color:=clWhite;
    Canvas.Pen.Width:=1;
    for x := 1 to Width div 32 do
    begin
      Canvas.MoveTo(x*32,0);
      Canvas.LineTo(x*32,Height);
    end;
    for y := 1 to Height div 32 do
    begin
      Canvas.MoveTo(0,y*32);
      Canvas.LineTo(Width,y*32);
    end;
  end;
  for i:=0 to high(Objects) do
    if Objects[i] <> nil then
      Objects[i].Draw();
end;

//Создает главное окно
procedure TMainForm.FormCreate(Sender: TObject);
var
  i: integer;
begin
  Fillchar(keys,sizeof(keys),0);
  Randomize();
  ClientHeight:=128*3;
  ClientWidth:=128*4;
  Left:=64;
  Top:=64;
  DblBuffer:=TBitmap.Create;
  with DblBuffer do
  begin
    width:=ClientWidth;
    height:=ClientHeight;
  end;
  //Инициализация лабиринта
  fillchar(Maze,sizeof(Maze),0);
  for i := -1 to 16 do  //Создание границ (левой и правой)
  begin
    Maze[-1,i]:=true;
    Maze[12,i]:=true;
```

```
  end;
  for i := -1 to 12 do  //Создание границ (верхней и нижней)
  begin
    Maze[i,-1]:=true;
    Maze[i,16]:=true;
  end;
  for i := 1 to 16*12 div 8 do
    Maze[random(12),random(16)]:=true;
  //Создание игроков
  THumanPlayer.Create;
  TCPUPlayer.Create;
end;

//Закрывает главное окно
procedure TMainForm.FormClose(Sender: TObject; var Action: TCloseAction);
var
  i: integer;
begin
  DblBuffer.Free;
  for i := 0 to high(Objects) do
    Objects[i].Free;
end;

//Вызывает функции, производящие обновление всех параметров системы
procedure TMainForm.MainTimerTimer(Sender: TObject);
begin
  UpdateGame();
  PrepareBuffer();
  Canvas.Draw(0,0,DblBuffer);
end;

//Производит отрисовку
procedure TMainForm.FormPaint(Sender: TObject);
begin
  Canvas.Draw(0,0,DblBuffer);
end;

//Ведет протоколирование
procedure log(st: string);
var
  f: textfile;
begin
  if LogForm.LogMemo.Lines.Count > 128 then
    LogForm.LogMemo.Lines.Delete(0);
  LogForm.LogMemo.Lines.Add(TimeToStr(Now)+' - '+st);
  assignfile(f,'log.txt'); append(f);
  writeln(f,st);
  closefile(f);
end;

//Реакция на нажатие клавиши
procedure TMainForm.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  Keys[key]:=true;
  if key = VK_ESCAPE then close;
end;

//Реакция на отпускание клавиши
procedure TMainForm.FormKeyUp(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  Keys[Key]:=false;
end;

end.
```

Файл «BotUnit.pas»

```
unit BotUnit;
interface
uses Graphics, Classes, Windows, SysUtils;
const
  pWidth = 6;                                          //Ширина игрока
  pLength = 12;                                        //Длина пушки
  pFireTime = 8;                                       //Время выстрела
  pLife = 4;                                           //Число жизней игрока
  cor = -(2 * 3.14159265358979323846264338327795)/64;  //
  pSpeed = 2.1;                                        //Скорость игрока
  rSpeed = 3.7;                                        //Скорость снаряда
  rWidth = 3;                                          //Ширина снаряда

var
  objectCounter: integer = 0;

type

//Класс, описывающий игровой объект
  TGameObject = class(TObject)
  protected
    xPos, yPos: double;
    dir: integer;
    fRadius: integer;
  public
    property Radius: integer read fRadius;             //Радиус
    property xPosition: double read xPos;              //Координата x
    property yPosition: double read yPos;              //Координата y
    property direction: integer read dir;              //Ориентация в
                                                       //пространстве

    procedure Draw(); virtual; abstract;               //Производит отрисовку
    procedure Update(); virtual; abstract;             //Производит обновление
                                                       //параметров игрового
                                                       //объекта
    function Name(): string; virtual; abstract;        //Возвращает имя объекта

    constructor Create;
    destructor Destroy(); override;
  end;

//Класс, описывающий игрока
  TPlayer = class(TGameObject)
  protected
    fColor: TColor;
    toFire: integer;
    life: integer;
  public
    search: Integer;
    procedure onKilled; virtual;                       //Реакция на уничтожение
    procedure onHit; virtual;                          //Реакция на попадание
    procedure onOtherPlayerKilled(); virtual;          //Реакция на уничтожение
                                                       //другого игрока
    procedure onOtherPlayerFired(); virtual;           //Реакция на выстрел
                                                       //другого игрока
    procedure CheckColission();                        //Проверка коллизии
    procedure Draw(); override;                        //Производит отрисовку
    procedure GeneratePosition();                      //Размещает игрока на
                                                       //корректной позиции
    procedure Restart();                               //При необходимости
                                                       //производит перезапуск и
                                                       //уменьшает здоровье
                                                       //игрока
    procedure DoStep();                                //Шаг вперед
    procedure DoLeft();                                //Поворот налево
    procedure DoRight();                               //Поворот направо
    procedure DoFire();                                //Выстрел
```

```
  end;

//Класс, описывающий снаряд
  TMissle = class(TGameObject)
  private
    ndx: integer;
  public
    function Name(): string; override;                //Возвращает имя объекта
    procedure Draw(); override;                       //Производит отрисовку
    procedure Update(); override;                     //Производит обновление
                                                      //параметров снаряда
    constructor Create(sender: TPlayer);
    destructor Destroy(); override;
  end;

//Класс, описывающий игрока, управляемого человеком
  THumanPlayer = class(TPlayer)
  public
    constructor Create;
    function Name(): string; override;                //Возвращает имя объекта

    procedure Update(); override;                     //Производит обновление
                                                      //параметров игрока
  end;

implementation
uses BotDemo;

destructor TGameObject.Destroy();
var
  i: integer;
begin
  i:=0;
  while(i<length(Objects))do
  begin
    if(Objects[i]=self) then
      Objects[i]:=nil;
    inc(i);
  end;
  inherited;
end;

constructor TGameObject.Create;
var
  i, ndx: integer;
begin
  inherited;
  ndx:=-1;
  for i := 0 to high(Objects) do
    if Objects[i] = nil then ndx:=i;
  if ndx < 0 then
  begin
    ndx:=length(Objects);
    setLength(Objects,length(Objects)+1);
  end;
  Objects[ndx]:=self;
end;

//Реакция на уничтожение
procedure TPlayer.onKilled;
begin
end;

//Реакция на попадание
procedure TPlayer.onHit;
begin
end;

//Реакция на смерть другого игрока
procedure TPlayer.onOtherPlayerKilled();
begin
```

```
end;

//Реакция на выстрел другого игрока
procedure TPlayer.onOtherPlayerFired();
begin
end;

//Производит отрисовку
procedure TPlayer.Draw();
var
  i: integer;
begin
  with DblBuffer.Canvas do
  begin
    Pen.Width:=1;
    Pen.Color:=clWhite;
    Pen.Width:=2;
    for i := 1 to life do
    begin
      MoveTo(Round(xPos),Round(yPos));
      LineTo(Round(xPos+pLength*cos((dir-3-i*5)*cor)),Round(yPos+pLength*sin((dir-3-
i*5)*cor)));
      MoveTo(Round(xPos),Round(yPos));

LineTo(Round(xPos+pLength*cos((dir+3+i*5)*cor)),Round(yPos+pLength*sin((dir+3+i*5)*cor
)));
    end;
    Pen.Color:=fColor;
    Brush.Color:=fColor;
    Ellipse(Rect(round(xPos)-pWidth,round(yPos)-
pWidth,round(xPos)+pWidth,round(yPos)+pWidth));
    MoveTo(Round(xPos),Round(yPos));
    Pen.Width:=3;
    LineTo(Round(xPos+pLength*cos(dir*cor)),Round(yPos+pLength*sin(dir*cor)));
  end;
end;

//Проверка коллизии
procedure TPlayer.CheckColission();
var
  i: integer;
begin
  for i := 0 to high(Objects) do
    if (Objects[i] <> nil) and (Objects[i] is TPlayer) and (Objects[i] <> self) then
    begin
      if sqrt(sqr(xPos-Objects[i].xPos)+sqr(yPos-Objects[i].yPos)) <= self.fRadius*2
then
      begin
        self.Restart();
        (Objects[i] as TPlayer).Restart;
      end;
    end;
end;

//Размещает игрока на корректной позиции
procedure TPlayer.GeneratePosition();
var
  x,y: integer;
begin
  life:=pLife;
  repeat
    x:=random(4*128);
    y:=random(3*128);
    dir:=random(16);
  until not Maze[y shr 5, x shr 5];
  xPos:=x;
  yPos:=y;
end;
```

```
//Поворот налево
procedure TPlayer.DoLeft();
begin
  dir:=(dir+1) mod 64;
end;

//Поворот направо
procedure TPlayer.DoRight();
begin
  dir:=(dir+64-1) mod 64;
end;

//Шаг вперед
procedure TPlayer.DoStep();
var
  x,y: double;
begin
  x:=xPos+cos(dir*cor)*pSpeed;
  y:=yPos+sin(dir*cor)*pSpeed;
  if(x< 0) then x:=0.01;
  if(y< 0) then y:=0.01;
  if Maze[round(y) shr 5,round(x) shr 5] then
  begin
    log('Игрок "'+name()+'" столкнулся со стеной');
  end else
  begin
    xPos:=x;
    yPos:=y;
  end;
end;

//Выстрел
procedure TPlayer.DoFire();
var
  i: integer;
begin
  if toFire > pFireTime then
  begin
    TMissle.Create(self);
    toFire:=0;
    for i := 0 to high(Objects) do
      if (Objects[i] <> nil) and (Objects[i] is TPlayer) and (Objects[i] <> self) then
(Objects[i] as TPlayer).onOtherPlayerFired();
  end;
end;

//При необходимости производит перезапуск и уменьшает здоровье
procedure TPlayer.Restart();
var
  i: integer;
begin
  dec(Life);
  if Life <= 0 then
  begin
    onKilled();
    log('Игрок "'+name()+'" был убит и восстановлен на новом месте');
    for i := 0 to high(Objects) do
      if (Objects[i] <> nil) and (Objects[i] is TPlayer) and (Objects[i] <> self) then
(Objects[i] as TPlayer).onOtherPlayerKilled();
    GeneratePosition();
  end else
  begin
    onHit();
    log('Игрок "'+name()+'" был ранен');
  end;
end;

constructor THumanPlayer.Create;
begin
  inherited;
  fRadius:=pWidth;
```

```
  fColor:=clYellow;
  GeneratePosition();
end;

//Возвращает имя объекта
function THumanPlayer.Name(): string;
begin
  result:='Игрок-человек';
end;

//Производит обновление параметров
procedure THumanPlayer.Update();
begin
  inc(toFire);
  if Keys[VK_LEFT] then DoLeft();
  if Keys[VK_RIGHT] then DoRight();
  if Keys[VK_UP] then DoStep();
  if Keys[VK_SPACE] then DoFire();
  CheckColission();
end;

constructor TMissle.Create(sender: TPlayer);
begin
  inherited Create;
  fRadius:=rWidth;
  ndx:=objectCounter;
  inc(objectCounter);
  dir:=sender.dir;
  xPos:=sender.xPos+cos(dir*cor)*(rSpeed+pSpeed)*2;
  yPos:=sender.yPos+sin(dir*cor)*(rSpeed+pSpeed)*2;
end;

destructor TMissle.Destroy();
begin
  inherited;
end;

//Возвращает имя объекта
function TMissle.Name(): string;
begin
  result:='Снаряд'+inttostr(ndx);
end;

//Производит отрисовку
procedure TMissle.Draw();
begin
  with DblBuffer.Canvas do
  begin
    Pen.Color:=clBlack;
    Brush.Color:=clBlack;
    Ellipse(Rect(round(xPos)-rWidth,round(yPos)-
rWidth,round(xPos)+rWidth,round(yPos)+rWidth));
  end;
end;

//Производит обновление параметров
procedure TMissle.Update();
var
  i: integer;
begin
  xPos:=xPos+cos(dir*cor)*rSpeed;
  yPos:=yPos+sin(dir*cor)*rSpeed;
  if (xPos < 0 ) or (yPos < 0) then
  begin
    self.Free;
    exit;
  end;
  if Maze[round(yPos) shr 5,round(xPos) shr 5] then
  begin
  //Обработка столкновения со стеной
    log('Снаряд "'+name()+'" уничтожен: столкновение со стеной');
```

```
    self.Free;
  end else
  begin
  //Проверка столкновения с объектами
    for i := 0 to high(objects) do
      if (objects[i] <> nil) and (objects[i] <> self) then
        if sqrt(sqr(xPos-objects[i].xPos)+sqr(yPos-Objects[i].yPos)) <=
radius+Objects[i].Radius then
        begin
          log('Снаряд "'+name()+'" уничтожен: столкновение с объектом
("'+Objects[i].Name()+'")');
          if Objects[i] is TMissle then Objects[i].Free else
          begin
            (Objects[i] as TPlayer).Restart;
          end;
          self.Free;
          exit;
        end;
  end;
end;

end.
```

Файл «LogUnit.pas»

```
unit LogUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TLogForm = class(TForm)
    LogMemo: TMemo;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormCreate(Sender: TObject);
  end;

var
  LogForm: TLogForm;

implementation
uses BotDemo;
{$R *.dfm}

procedure TLogForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  MainForm.Close;
end;

procedure TLogForm.FormCreate(Sender: TObject);
begin
  Left:=MainForm.Left;
  Top:=MainForm.Top+MainForm.Height;
  Width:=MainForm.Width;
end;

end.
```

Файл «Monstr.pas»

```
unit Monstr;

interface
uses Attack, BotUnit;

type

//Класс, описывающий автомат "Монстр"
  TMonstr = class
  private
    name   : string;
    Player : TPlayer;
    human  : TGameObject;
  public
    state  : integer;                                          //Состояние
    attack : TAttack;                                          //Вложенный
                                                               //автомат "Атака"
    constructor Create(Parent : TPlayer; hplayer: TGameObject);
    procedure   Run (event : integer);                         //Запускает
                                                               //автомат

                                                               //Выходные
                                                               //воздействия
    procedure   z01 ();
    procedure   z03 ();
    procedure   z04 ();
  end;

implementation

uses BotDemo, SysUtils;

constructor TMonstr.Create;
begin
  attack := TAttack.Create(parent, hplayer);
  state  := 0;
  name   := '"Монстр"';
  player := parent;
  human  := hplayer;
end;

//Запускает автомат
procedure TMonstr.Run;
var
  newstate : integer;
begin
  newstate := -1;
  Log('Автомат ' + name + ' запущен в состоянии ' + IntToStr(state) +
      ' с событием e' + IntToStr(event));
  case state of
    0: begin
         case event of
           1:                                          newstate := 1;
           2: begin
                z03();                                 newstate := 1;
              end;
           3:                                          newstate := 2;
           7: begin
                z04();                                 newstate := 0;
              end;
         end;
       end;
    1: begin
         case event of
           2: z03();
           3:                                          newstate := 2;
           4: newstate := 0;
           7: begin
```

```
                z04();                                          newstate := 0;
              end;
          end;
        end;
    2: begin
         attack.Run(event);
         case event of
           2: z03();
           5: begin
                                                                newstate := 1;
              end;
           6: begin
                                                                newstate := 0;
              end;
           7: begin
                z04();
                                                                newstate := 0;
              end;
          end;
        end;
    3: begin
       end;
  end;
  if (newstate >= 0) and (state<>newstate) then
  begin
    Log('Автомат ' + name + ' перешёл из состояния ' + IntToStr(state) +
        ' в состояние ' + IntToStr(newstate));
    state := newstate;
    case state of
      0: begin
           z04();
         end;
      1: begin
           z01();
         end;
      2: begin
           attack.Run(0);
         end;
      3: begin
         end;
    end;
  end;
  Log('Автомат ' + name + ' завершил обработку события e' + IntToStr(event) +
      ' в состоянии ' + IntToStr(state));
end;

//Выходные воздействия
procedure TMonstr.z01;
begin
  log('Выходное воздействие z01 - поиск');
  player.search := 64;
end;

procedure TMonstr.z03;
begin
  log('Выходное воздействие z03 - увеличение агрессии');
  Agression := false;
end;

procedure TMonstr.z04;
begin
  log('Выходное воздействие z04 - уменьшение агрессии');
  Agression := true;
end;

end.
```

Файл «Attack.pas»

```
unit Attack;

interface

uses BotUnit;

var
  Agression : boolean = true;

type

//Класс, описывающий автомат "Атака"
  TAttack = class
  private
    state : integer;
    name  : string;
    Player : TPlayer;
    human  : TGameObject;
  public
    turnLeft : boolean;

    constructor Create(Parent: TPlayer;  HPlayer: TGameObject);
    procedure   Run (event : integer);                          //Запускает
                                                                //автомат

                                                                //Входные
                                                                //переменные

    function    x11() : boolean;
    function    x12() : boolean;

                                                                //Выходные
                                                                //воздействия

    procedure   z11 ();
    procedure   z12 ();
  end;

implementation

uses BotDemo, SysUtils, Math;

constructor TAttack.Create;
begin
  state  := 0;
  name   := '"Атака"';
  player := parent;
  human  := hplayer;
end;

//Запускает автомат
procedure TAttack.Run;
var
  newstate      : integer;
begin
  if event <> 1024 then
    Log('Автомат ' + name + ' запущен в состоянии ' + IntToStr(state) +
        ' с событием e' + IntToStr(event));
  newstate := -1;
  case state of
    0: begin
         if x11() and x12() and (event=0) then
                                                                newstate := 1;
         if x11() and not x12() and (event=0) then
                                                                newstate := 2;
         if not x11() and (event=0) then
                                                                newstate := 3;
```

```
       end;
    1: begin
         if x11() and x12() then
           z11();
         if x11() and not x12() then
                                                              newstate := 2;
         if not x11() then
                                                              newstate := 3;
         if (event=5) or (event=6) or (event=7) then
                                                              newstate := 0;
       end;
    2: begin
         if x11() and not x12() then
         begin
           z11();
           z12();
         end;
         if x11() and x12() then
                                                              newstate := 1;
         if not x11() then
                                                              newstate := 3;
         if (event=5) or (event=6) or (event=7) then
                                                              newstate := 0;
       end;
    3: begin
         if not x11() then
         begin
           z11();
           z12();
         end;
         if (event=5) or (event=6) or (event=7) then
                                                              newstate := 0;
       end;
  end;
  if (newstate >= 0) and (state<>newstate) then
  begin
    if event <> 1024 then
      Log('Автомат ' + name + ' перешёл из состояния ' + IntToStr(state) +
          ' в состояние ' + IntToStr(newstate));
    state := newstate;
    case state of
      0: ;
      1: z11();
      2: begin
           z11();
           z12();
         end;
      3: begin
           z11();
           z12();
         end;
    end;
  end;
  if event <> 1024 then
    Log('Автомат ' + name + ' завершил обработку события e' + IntToStr(event) +
        ' в состоянии ' + IntToStr(state));
end;

//Выходные воздействия
procedure TAttack.z11;
var
  angle : double;
begin
  log('Выходное воздействие z11 - шаг в сторону противника');
  if player.direction*cor < - Pi then
    angle := player.direction*cor + 2*Pi
  else
    angle := player.direction*cor;
  if arctan2(human.yPosition - player.yPosition, human.xPosition - player.xPosition) <
angle then
  begin
```

```
    player.DoLeft();
    turnLeft := true;
  end
  else
  begin
    player.DoRight();
    turnLeft := false;
  end;
  player.doStep();
end;

procedure TAttack.z12;
begin
  log('Выходное воздействие z12 - выстрел');
  player.DoFire();
end;

//Входные переменные
function TAttack.x11;
begin
  log('Произведен опрос входной переменной x11 (уровень агрессивности монстра)');
  result := Agression;
end;

function TAttack.x12;
begin
  log('Произведен опрос входной переменной x12 (величина расстояния до противника)');
  result := false;
end;

end.
```