Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

М.И. Гуисов, А.А. Шалыто

Задача Д. Майхилла «Синхронизация цепи стрелков»

Вариант 1

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

Содержание

1. Постановка задачи **3**
2. Диаграмма классов **4**
3. Класс «ЦЕПЬ» **5**
3.1. Словесное описание **5**
3.2. Структурная схема класса **5**
4. Класс «ОФИЦЕР» **5**
4.1. Словесное описание **5**
4.2. Структурная схема класса **5**
4.3. Автомат управления действиями офицера **5**
4.3.1. Словесное описание **5**
4.3.2. Схема связей **6**
4.3.3. Граф переходов **6**
5. Класс «СТРЕЛОК» **6**
5.1. Словесное описание **6**
5.2. Структурная схема класса **6**
5.3. Автомат управления действиями стрелка **7**
5.3.1. Словесное описание................................ **7**
5.3.2. Схема связей................................ **7**
5.3.3. Граф переходов **7**
6. Текст программы **8**
7. Протокол работы программы **13**
Источники **14**

1. Постановка задачи

В настоящей работе для задачи с параллельными процессами демонстрируется эффективность применения объектно-ориентированного программирования с явным выделением состояний [1], в котором совместно используются объектно-ориентированный и автоматный стили программирования. В качестве примера выбрана задача о синхронизации цепи стрелков, предложенная Д.Майхиллом в 1957 г. [2].

Сформулируем эту задачу. Предположим, что стрелкам, выстроенным в цепь, разрешена следующая процедура общения между собой.

1. Каждый стрелок может общаться непосредственно только со своими ближайшими соседями слева и справа. При этом левофланговый также может общаться с офицером.
2. Сеансы общения могут происходить по одному разу в каждую единицу времени. При этом считается, что офицер и каждый из стрелков имеют часы, которые синхронизированы между собой. Синхронизации часов недостаточно для синхронности выстрела, так как число стрелков в цепи априори не известно.
3. Сеанс общения стрелка состоит в том, что он передает своим соседям по одному условному знаку и принимает к сведению те условные знаки, которые ему сообщают соседи.

В некоторый момент левофланговый получает от офицера приказ **ОГОНЬ**, который адресован всем стрелкам и должен быть выполнен ими одновременно. Это обеспечивается передачей сигнала по цепи до ее конца и возвратом этого сигнала к левофланговому стрелку.

В зависимости от того, могут ли стрелки «считать до N» (номер стрелка) и какие знаки могут передаваться в цепи, задача имеет разные решения, отличающиеся по сложности.

Если счет запрещен и в качестве условных знаков нельзя передавать номер стрелка в цепи, может быть применен алгоритм В.И.Левенштейна. Его модификация описана в работе [3]. В этом алгоритме каждый стрелок реализуется автоматом с восемью состояниями. Основная идея этого алгоритма состоит в «делении отрезка цепи пополам». По цепи распространяется множество сигналов с задержками 2^n-1, где n – индекс сигнала. При встрече некоторого сигнала с первым происходит деление цепи. От места встречи распространяются новые сигналы с теми же индексами. Автоматы, находящиеся в месте встречи, переходят в состояние готовности, предшествующее «синхронному» состоянию. Идея Левенштейна позволила распространять любое число сигналов по цепи автоматов с конечным, и как отмечено выше, небольшим числом состояний.

В настоящей работе, как и в работах [4,5], разрешен «счет до N», а общение стрелков допускает передачу номера стрелка.

Каждый стрелок в цепи должен руководствоваться следующими правилами.

1. Левофланговый, получив приказ, запоминает число один (свой порядковый номер) и через единицу времени сообщает его соседу справа.
2. Если сосед слева сообщил стрелку число **K-1**, стрелок запоминает свой порядковый номер (**K**) и через единицу времени передает его соседу справа, или, если стрелок – правофланговый, отвечает соседу слева о своей готовности и приступает к обратному счету (**K**, **K-1**, ...).
3. Если порядковый номер стрелка равен **K** и сосед справа сообщил ему о своей готовности, через минуту стрелок передает сигнал о готовности налево и приступает к обратному счету (**K**, **K-1**, ...). Левофланговый сообщает о своей готовности офицеру.
4. Досчитав до нуля, стрелок открывает огонь, нажимая курок один раз за единицу времени.

В результате применения этих правил первый синхронный выстрел (залп) произойдет через **2N** единиц времени после получения левофланговым приказа офицера.

2. Диаграмма классов

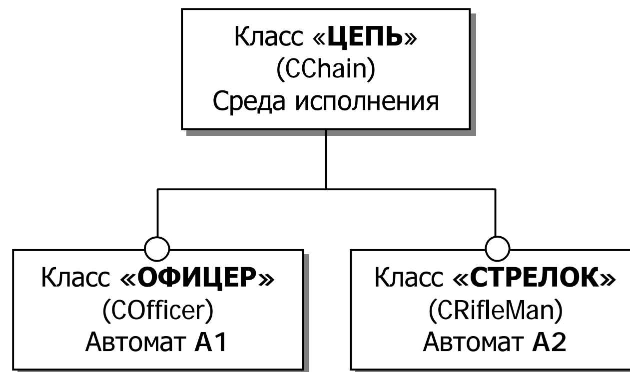

Рис. 1

Обратим внимание, что в программе N экземпляров класса «СТРЕЛОК» и по одному экземпляру остальных классов.

3. Класс «ЦЕПЬ»

3.1. Словесное описание

Этот класс является главным в приведенной выше иерархии классов. Основной его задачей является связь действий офицера и солдат в шеренге. Он отвечает также за протоколирование работы программы, вызывая соответствующие процедуры классов «ОФИЦЕР» и «СТРЕЛОК».

3.2. Структурная схема класса

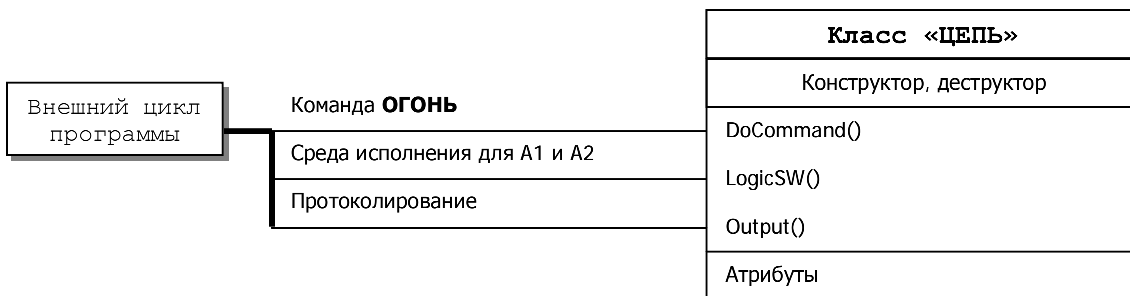

Рис. 2

4. Класс «ОФИЦЕР»

4.1. Словесное описание

Этот класс реализует офицера, отдающего приказы солдату, стоящему на левом фланге. Кроме того, класс протоколирует состояния автомата **A1**, управляющего действиями офицера.

4.2. Структурная схема класса

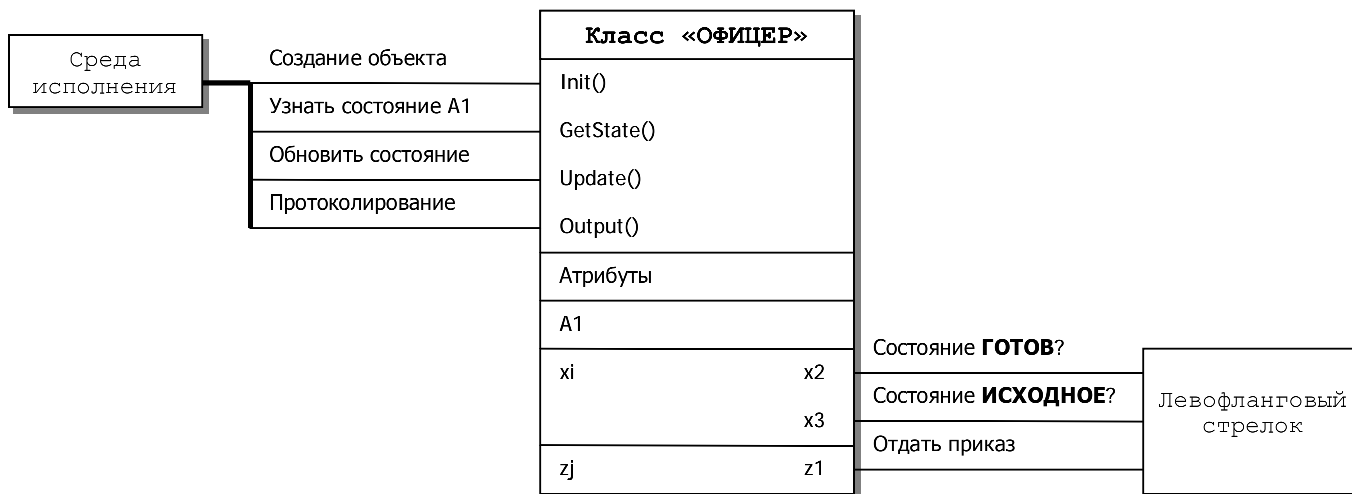

Рис. 3

4.3. Автомат управления действиями офицера

4.3.1. Словесное описание

Обычно офицер находится в **ИСХОДНОМ** состоянии. При необходимости провести залп он отдает команду левофланговому стрелку. При этом офицер переходит в состояние **ПРИКАЗ**. В этом состоянии он ждет готовности левофлангового стрелка (это значит, что вся цепь уже готова к стрельбе). После этого офицер переходит в состояние **ОБЗОР**. Переход в **ИСХОДНОЕ** состояние происходит после возвращения левофлангового солдата в **ИСХОДНОЕ** состояние. Обратим внимание, что, так как в данном случае применяется объектный подход, вместо непосредственного доступа к значению внутренней переменной Y2, кодирующей состояния автомата A2 («Стрелок»), для нее создан интерфейс - введены функции X2 и X3, общающиеся с переменной Y2 через функцию GetState().

4.3.2. Схема связей

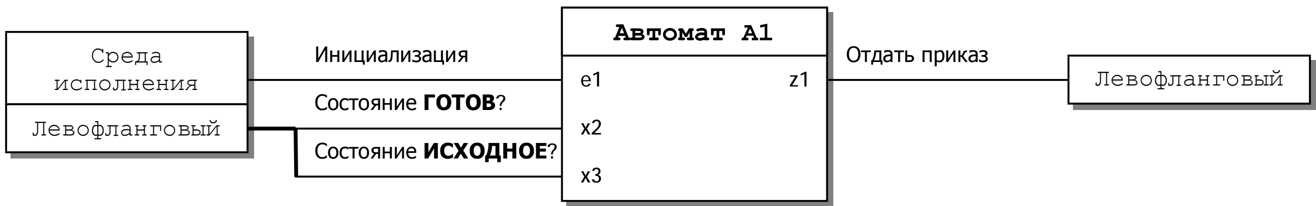

Рис. 4

4.3.3. Граф переходов

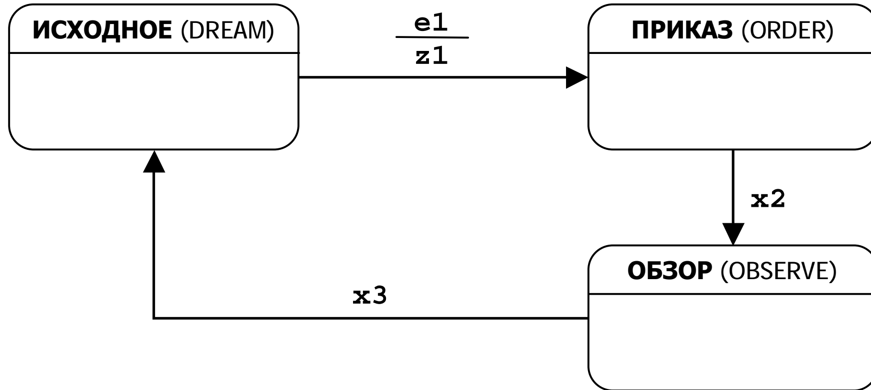

Рис. 5

5. Класс «СТРЕЛОК»

5.1. Словесное описание

Этот класс реализует стрелка, выполняющего приказ, обеспечивает связь его с ближайшими соседями и протоколирует состояния вложенного автомата **A2**.

5.2. Структурная схема класса

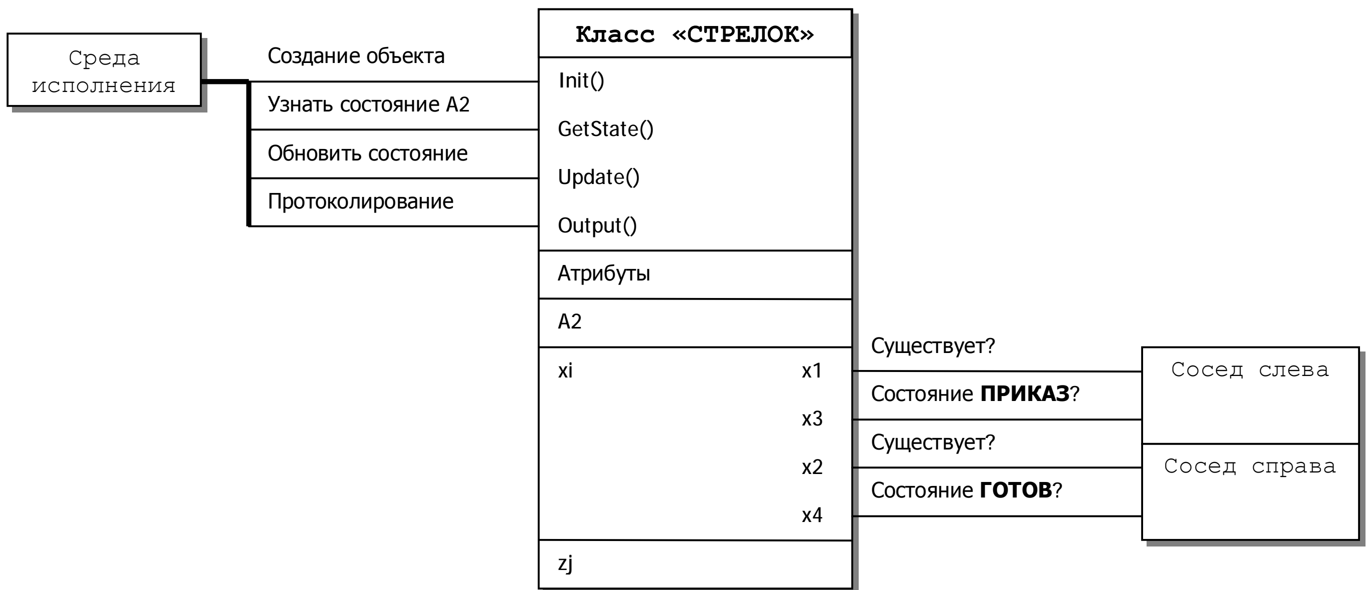

Рис. 6

5.3. Автомат управления действиями стрелка

5.3.1. Словесное описание

Стрелок, как и офицер, обычно находится в **ИСХОДНОМ** состоянии. Получив приказ от соседа (левофланговый от офицера), стрелок запоминает свой порядковый номер, передает приказ направо и переходит в состояние **ПРИКАЗ.** В этом состоянии стрелок находится до тех пор, пока сосед справа не сообщит ему о своей готовности. После этого стрелок переходит в состояние **ГОТОВ** (правофланговый переходит в состояние **ГОТОВ** без ожидания). В состоянии **ГОТОВ** стрелок для синхронизации считает от своего порядкового номера до нуля. Досчитав, переходит в состояние **ОГОНЬ**, в котором находится, пока не кончатся патроны в магазине, делая один выстрел в единицу времени. После этого переходит в **ИСХОДНОЕ** состояние.

5.3.2. Схема связей

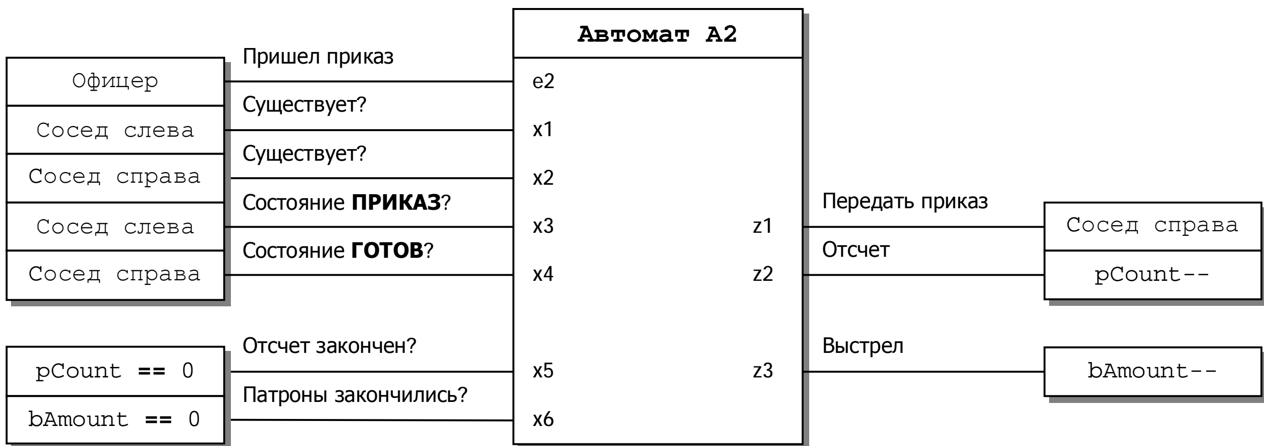

Рис. 7

5.3.3. Граф переходов

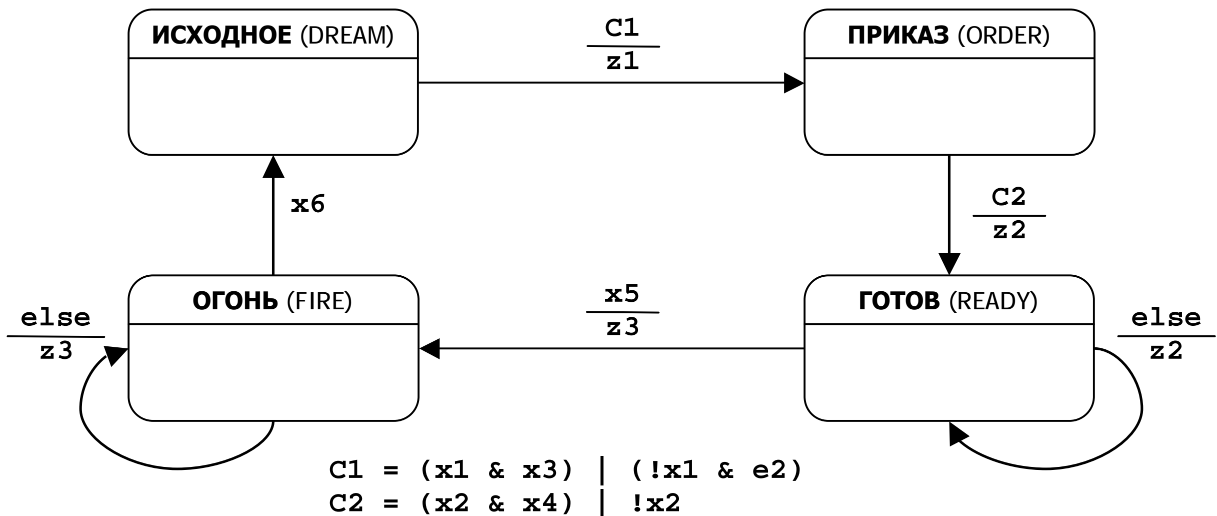

Рис. 8

6. Текст программы

6.1. Класс «ЦЕПЬ». Заголовочный файл "Chain.h"

```
#pragma once
#include "officer.h"
#define rmNumber 4                              // Количество стрелков

class CChain {
public:
        CChain();                               // Конструктор
        ~CChain();                              // Деструктор

        void DoCommand();                       // Приказ офицеру
        void LogicSW();                         // Логика класса
        void Output(FILE *FDat, int step);      // Протоколирование
private:
        CSQueue *Queue;                         // Очередь сообщений
        COfficer Officer;                       // Офицер
        CRifleMan *rmArray;                     // Массив стрелков
};
```

6.2. Реализация класса «ЦЕПЬ». Файл "Chain.cpp"

```
#include "stdafx.h"
#include "chain.h"

// Конструктор
CChain::CChain() {
        Queue = new CSQueue();
        rmArray = new CRifleMan [rmNumber];
        Officer.Init(&rmArray[0], Queue);
        for (int i=0; i<rmNumber; i++) {
        rmArray[i].Init((i > 0 ? &rmArray[i-1] : NULL),      // Сосед слева
               (i < rmNumber-1 ? &rmArray[i+1] : NULL));     // Сосед справа
        }
}

// Деструктор
CChain::~CChain() {
        delete Queue;
        delete rmArray;
}

// Передать приказ офицеру
void CChain::DoCommand() {
        Queue->push_back(1);
}

// Функция протоколирования
void CChain::Output(FILE *FDat, int step) {
        fprintf(FDat, "\n%.2i. logic entrance\n", step);
        Officer.Output(FDat);
        for (int i=0; i<rmNumber; i++) {
                fprintf(FDat, "rifleman %i ", i+1);
                rmArray[i].Output(FDat);
        }
}

// Логика класса
void CChain::LogicSW() {
        int e;
        Officer.Update();
        for (int i=0; i<rmNumber; i++) rmArray[i].Update();
        Queue->push_back(-1);              // Пустое сообщение (внешнее событие)
        while (!Queue->IsEmpty()) {
                e = Queue->back();
                Queue->pop_back();
                Officer.A(e);
                for (i=0; i<rmNumber; i++) rmArray[i].A(e);
        }
        Officer.S();
        for (i=0; i<rmNumber; i++) rmArray[i].S();
}
```

6.3. Класс «Офицер». Заголовочный файл “Officer.h”

```
#pragma once
#include "rifleman.h"
#define st_OBSERVE 2                    // Состояние ОБЗОР

class COfficer {
public:
      void Init(CRifleMan *_mostleft, CSQueue *pQ);

      void A(CSMessage &msg);           // Процедура выбора перехода автомата A1
      void S();                         // Процедура, осуществляющая переход
      void Output(FILE *FDat);          // Вывод результатов (в файл)
      void Update();                    // Обновление переменных автомата
      int  GetState();                  // Возвращает состояние автомата

private:
      int y_o, y_n;                     // Переменные состояния автомата
      CRifleMan *mLeft;                 // Ближайший стрелок
      CSQueue *Queue;                   // Ссылка на очередь (для создания сообщений)

      bool x2();                        // Левофланговый готов?
      bool x3();                        // Левофланговый в исходном состоянии?
      void z1();                        // Отдать приказ левофланговому
};
```

6.4. Реализация класса «Офицер». Файл “Officer.cpp”

```
#include "stdafx.h"
#include "officer.h"

// Первичная инициализация класса "Офицер"
void COfficer::Init(CRifleMan *_mostleft, CSQueue *pQ) {
	mLeft  = _mostleft;
	y_n    = st_DREAM;
	Queue = pQ;
}

// Процедура выбора перехода автомата A1
void COfficer::A(int e) {
	switch (y_o) {
	case st_DREAM:
		if (e == 1)		{ y_n = st_ORDER;}
		break;

	case st_ORDER:
		if (x2())		{ y_n = st_OBSERVE; }
		break;

	case st_OBSERVE:
		if (x3())		{ y_n = st_DREAM; }
	}
}

// Процедура, осуществляющая переход
void COfficer::S() {
	switch (y_o) {
	case st_DREAM:
		if (y_n == st_ORDER) z1();
	}
}

// Функция обновления переменных автомата
void COfficer::Update() {
    y_o = y_n;
}

// Возвращает состояние автомата
int COfficer::GetState() {
    return y_o;
}

// Протоколирование
void COfficer::Output(FILE *FDat) {
	char *tmp;
	switch (y_o) {
		case st_DREAM:   tmp = "DREAM";   break;
		case st_ORDER:   tmp = "ORDER";   break;
		case st_OBSERVE: tmp = "OBSERVE";
	}
	fprintf(FDat, "officer    state %s \n", tmp);
}

// Левофланговый готов?
bool COfficer::x2() {
	return mLeft->GetState() == st_READY;
}

// Левофланговый спит?
bool COfficer::x3() {
	return mLeft->GetState() == st_DREAM;
}

// Отдать приказ левофланговому
void COfficer::z1() {
	Queue->push_back(2);
}
```

6.5. Класс «Стрелок». Заголовочный файл "RifleMan.h"

```
#pragma once
#define st_DREAM    0                    // Состояние СОН
#define st_ORDER    1                    // Состояние ПРИКАЗ
#define st_READY    2                    // Состояние ГОТОВ
#define st_FIRE     3                    // Состояние ОГОНЬ
#define mSize       3                    // Размер магазина
typedef std::list<int> CSQueue;

class CRifleMan {
public:
      int pNumber;                       // Личный номер
      void Init(CRifleMan *_left, CRifleMan *_right);

      void A(int e);                     // Процедура выбора перехода автомата A2
      void S();                          // Процедура осуществления перехода A2
      void Output(FILE *FDat);           // Вывод результатов (в файл)
      void Update();                     // Обновление в начале шага
      int  GetState();                   // Возвращает состояние автомата

private:
      int y_o, y_n;                      // Переменные состояния автомата
      CRifleMan *left, *right;           // Указатели на соседей
      int bAmount_o, bAmount_n;          // Количество оставшихся в магазине пуль

      bool x1();                         // Есть сосед-стрелок слева?
      bool x2();                         // Есть сосед-стрелок справа?
      bool x3();                         // Состояние соседа слева ПРИКАЗ?
      bool x4();                         // Состояние соседа справа ГОТОВ?
      bool x5();                         // Отсчет закончен?
      bool x6();                         // Кончились патроны?

      void z1();                         // Запомнить свой номер, зарядить магазин
      void z2();                         // Уменьшить номер на 1
      void z3();                         // Сделать выстрел
};
```

6.6. Реализация класса «Стрелок». Файл "RifleMan.cpp"

```
#include "stdafx.h"
#include "rifleman.h"


// Первичная инициализация класса "Стрелок"
void CRifleMan::Init(CRifleMan *_left, CRifleMan *_right) {
      left        = _left;
      right       = _right;
      y_n               = st_DREAM;
      bAmount_n   = 0;
}

// Процедура выбора перехода автомата A2
void CRifleMan::A(int e) {
      switch (y_o) {
      case st_DREAM:
            if (x1() ? x3() : (e == 2))       { y_n = st_ORDER;   }
            break;

      case st_ORDER:
            if (x2() ? x4() : true)           { y_n = st_READY;   }
            break;

      case st_READY:
            if (x5())                         { y_n = st_FIRE;    }
            break;

      case st_FIRE:
            if (x6())                         { y_n = st_DREAM;   }
      }
}
```

```
// Процедура, осуществляющая переход
void CRifleMan::S() {
	switch (y_o) {
	case st_DREAM:
		if (y_n == st_ORDER)		z1();
		break;

	case st_ORDER:
		if (y_n == st_READY)		z2();	// Начало отсчета
		break;

	case st_READY:
		if (y_n == st_FIRE)		z3();	// Открытие огня
		else				z2();	// Петля в состоянии ГОТОВ
		break;

	case st_FIRE:
		if (y_n == st_FIRE)		z3();	// Петля в состоянии ОГОНЬ
	}
}

// Обновление в начале шага
void CRifleMan::Update() {
	bAmount_o = bAmount_n;
	y_o = y_n;
}

// Возвращает состояние автомата
int CRifleMan::GetState() { return y_o; }

// Протоколирование
void CRifleMan::Output(FILE *FDat) {
	char *tmp;
	switch (y_o) {
		case st_DREAM:	tmp = "DREAM";		break;
		case st_ORDER:	tmp = "ORDER";		break;
		case st_READY:	tmp = "READY";		break;
		case st_FIRE:	tmp = "FIRE";
	}
	fprintf(FDat, "state %s \t bullets %i/%i\n",
			tmp, bAmount_o, mSize);
}

bool CRifleMan::x1() { return (left != NULL); }		// Есть сосед-стрелок слева?

bool CRifleMan::x2() { return (right != NULL); }	// Есть сосед-стрелок справа?

bool CRifleMan::x3() {				// Состояние соседа слева "ПРИКАЗ"?
	return left->GetState() == st_ORDER;
}

bool CRifleMan::x4() {				// Состояние соседа справа "ГОТОВ"?
	return right->GetState() == st_READY;
}

bool CRifleMan::x5() {				// Отсчет закончен?
	return (pNumber == 0);
}

bool CRifleMan::x6() {				// Кончились патроны?
	return (bAmount_n == 0);
}

void CRifleMan::z1() {				// Вспомнить свой номер, зарядить магазин
	if (!left)	pNumber = 1;
	else		pNumber = left->pNumber + 1;
	bAmount_n = mSize;
}

void CRifleMan::z2() { pNumber--; }		// Уменьшить номер на 1

void CRifleMan::z3() { bAmount_n--; }		// Сделать выстрел
```

7. Протокол работы программы

Данный протокол: 4 стрелка, 3 патрона в магазине у каждого.

```
01. logic entrance
officer    state DREAM
rifleman 1 state DREAM     bullets 0/3
rifleman 2 state DREAM     bullets 0/3
rifleman 3 state DREAM     bullets 0/3
rifleman 4 state DREAM     bullets 0/3

02. logic entrance
officer    state DREAM
rifleman 1 state DREAM     bullets 0/3
rifleman 2 state DREAM     bullets 0/3
rifleman 3 state DREAM     bullets 0/3
rifleman 4 state DREAM     bullets 0/3

03. logic entrance
officer    state ORDER
rifleman 1 state DREAM     bullets 0/3
rifleman 2 state DREAM     bullets 0/3
rifleman 3 state DREAM     bullets 0/3
rifleman 4 state DREAM     bullets 0/3

04. logic entrance
officer    state ORDER
rifleman 1 state ORDER     bullets 3/3
rifleman 2 state DREAM     bullets 0/3
rifleman 3 state DREAM     bullets 0/3
rifleman 4 state DREAM     bullets 0/3

05. logic entrance
officer    state ORDER
rifleman 1 state ORDER     bullets 3/3
rifleman 2 state ORDER     bullets 3/3
rifleman 3 state DREAM     bullets 0/3
rifleman 4 state DREAM     bullets 0/3

06. logic entrance
officer    state ORDER
rifleman 1 state ORDER     bullets 3/3
rifleman 2 state ORDER     bullets 3/3
rifleman 3 state ORDER     bullets 3/3
rifleman 4 state DREAM     bullets 0/3

07. logic entrance
officer    state ORDER
rifleman 1 state ORDER     bullets 3/3
rifleman 2 state ORDER     bullets 3/3
rifleman 3 state ORDER     bullets 3/3
rifleman 4 state ORDER     bullets 3/3

08. logic entrance
officer    state ORDER
rifleman 1 state ORDER     bullets 3/3
rifleman 2 state ORDER     bullets 3/3
rifleman 3 state ORDER     bullets 3/3
rifleman 4 state READY     bullets 3/3

09. logic entrance
officer    state ORDER
rifleman 1 state ORDER     bullets 3/3
rifleman 2 state ORDER     bullets 3/3
rifleman 3 state READY     bullets 3/3
rifleman 4 state READY     bullets 3/3

10. logic entrance
officer    state ORDER
rifleman 1 state ORDER     bullets 3/3
rifleman 2 state READY     bullets 3/3
rifleman 3 state READY     bullets 3/3
rifleman 4 state READY     bullets 3/3

11. logic entrance
officer    state ORDER
rifleman 1 state READY     bullets 3/3
rifleman 2 state READY     bullets 3/3
rifleman 3 state READY     bullets 3/3
rifleman 4 state READY     bullets 3/3

12. logic entrance
officer    state OBSERVE
rifleman 1 state FIRE      bullets 2/3
rifleman 2 state FIRE      bullets 2/3
rifleman 3 state FIRE      bullets 2/3
rifleman 4 state FIRE      bullets 2/3

13. logic entrance
officer    state OBSERVE
rifleman 1 state FIRE      bullets 1/3
rifleman 2 state FIRE      bullets 1/3
rifleman 3 state FIRE      bullets 1/3
rifleman 4 state FIRE      bullets 1/3

14. logic entrance
officer    state OBSERVE
rifleman 1 state FIRE      bullets 0/3
rifleman 2 state FIRE      bullets 0/3
rifleman 3 state FIRE      bullets 0/3
rifleman 4 state FIRE      bullets 0/3

15. logic entrance
officer    state OBSERVE
rifleman 1 state DREAM     bullets 0/3
rifleman 2 state DREAM     bullets 0/3
rifleman 3 state DREAM     bullets 0/3
rifleman 4 state DREAM     bullets 0/3

16. logic entrance
officer    state DREAM
rifleman 1 state DREAM     bullets 0/3
rifleman 2 state DREAM     bullets 0/3
rifleman 3 state DREAM     bullets 0/3
rifleman 4 state DREAM     bullets 0/3
```

Из протокола следует, что с начиная с третьего шага все четыре стрелка переходят из исходного состояния в состояние **ГОТОВ**, затем синхронно выпускают по три пули (шаги 12-14) и «засыпают» (переходят в исходное состояние). Офицер «засыпает» последним.

Источники

1. *Туккель Н.И.*, *Шалыто А.А.* Система управления танком для игры Robocode. Объектно-ориентированное программирование с явным выделением состояний. Проектная документация. http://is.ifmo.ru раздел «Проекты».

2. *Goto E.* A Minimum Time Solution of the Firing Squad Problem //Dittoed Course Notes for Applied Mathematics. Harvard Univ., 1962.

3. *Варшавский В.И., Мараховский В.Б., Песчанский В.А., Розенблюм Л.Я.* Однородные структуры. М.: Энергия, 1973.

4. *Трахтенброт Б.А.* Алгоритмы и вычислительные автоматы. М.: Советское радио, 1974.

5. *Любченко В.С.* «Батарея, огонь!» или задача Майхилла для Microsoft Visual C++. О синхронизации процессов в среде Windows // Мир ПК. 2000. N2.

6. *Шалыто А.А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.

7. *Буч Г.* Обьектно-ориентированный анализ и проектирование. М.: Бином, 1998.