Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

Д. С. Абдрашитов

Система управления персонажем в многопользовательской ролевой игре

Объектно-ориентированное программирование с явным выделением состояний

Проектная документация

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

Санкт-Петербург
2006

Оглавление

Введение

В данной работе рассматривается применение технологии автоматного программирования [1] для создания искусственного интеллекта на примере игры *Аладон* [2]. Эффективность применения автоматного подхода при проектировании систем с искусственным интеллектом показана в работах [3 – 5].

Цель настоящей работы состоит в разработке бота для текстовой многопользовательской игры *Аладон*, способного без участия человека проходить «неинтересные» уровни игры.

Применение SWITCH-технологии, предложенной *А. А. Шалыто* [6], позволило разработать простую для понимания логику искусственного интеллекта. Проектная документация подробно описывает созданную систему, что позволит при необходимости модифицировать ее.

1. Постановка задачи

Текстовые многопользовательские ролевые игры являются достаточно популярными в Интернете на сегодняшний день. Только в российской части Интернета существует более десятка серверов таких игр.

Для участия в игре пользователи подключаются к серверу с помощью специального клиентского программного обеспечения или стандартного текстового терминала *telnet*. В игре пользователь управляет персонажем. Сервер присылает текстовое описание ситуации, в которой находится персонаж. Пользователь может послать серверу текстовые команды, описывающие, что следует сделать персонажу. Например, от сервера может придти сообщение, что персонаж устал, в ответ на это пользователь может отослать команду «спать», и персонаж уснет для восстановления сил. Пример такой ситуации изображен на рис. 1.

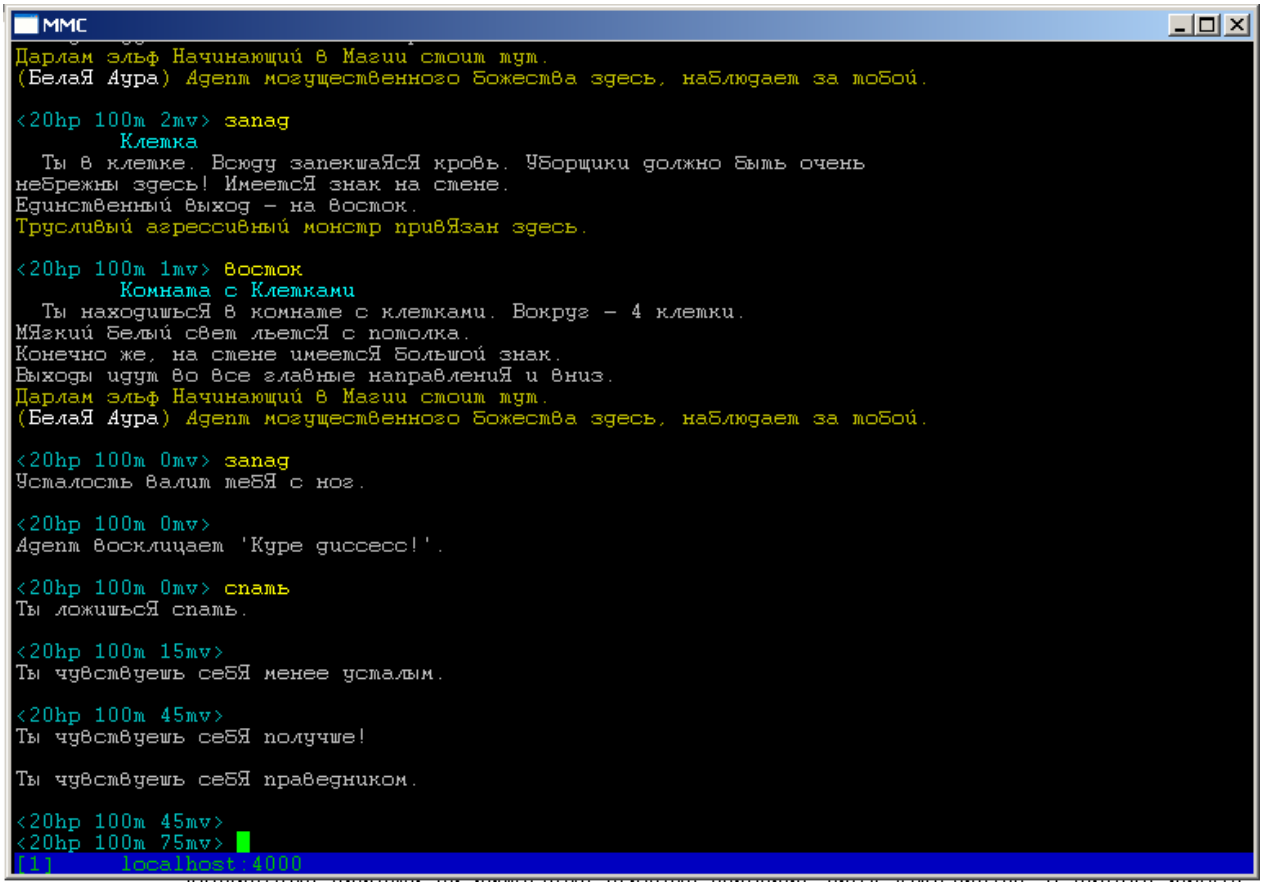

Рис. 1. Пример игровой ситуации

За победу над противниками в битвах персонажу начисляются очки. Чем больше у персонажа очков, тем больше у него возможностей действия. Из-за этого опытным игрокам скучно управлять персонажем, имеющим небольшое число очков опыта. Решением этой проблемы может стать создание автоматического бота, способного играть вместо пользователя до тех пор, пока персонаж не наберет требуемое для интересной игры число очков опыта.

Несмотря на то, что все текстовые ролевые игры основаны на общих идеях, конкретные правила на различных игровых серверах зачастую отличаются. В данной работе рассматриваются правила, изложенные на сервере *Аладон*, который является на сегодняшний день самым популярным в русскоязычной части сети Интернет.

Кратко опишем ту часть правил игры, которая была использована при создании бота.

Игровое пространство делится на комнаты, между которыми персонаж может перемещаться. Пользователю в текстовом виде описывается обстановка комнаты, в которой находится персонаж. В комнате помимо персонажа могут находиться предметы, другие персонажи и управляемые сервером монстры. Предметы персонаж может брать и использовать. С монстрами персонажу следует сражаться для получения очков. За каждую победу в битве начисляются очки. За поражение в битве персонаж лишается всей экипировки и части очков. Существуют комнаты, в которых запрещено вести бой. Они предназначены для безопасного отдыха.

Персонаж обладает единицами жизни, магии и перемещения. Все эти параметры восстанавливаются с течением времени. Восстановление происходит быстрее, если персонаж отдыхает. Если он спит, то восстановление происходит максимально быстро. Восстановление

замедляется, если персонаж голоден или испытывает жажду. Для утоления голода или жажды используется запас провизии. Пополняется запас провизии в магазине.

В битве противники поочередно наносят друг другу удары. Удар уменьшает число единиц жизни того, кому он был нанесен. Тот, кто теряет все единицы жизни, считается проигравшим в битве. Для нанесения дополнительных повреждений применяются боевые заклинания. Персонаж может воспользоваться боевым заклинанием, если у него достаточно единиц магии. Каждое применение заклинания уменьшает число единиц магии. Пример битвы персонажа с монстром приведен на рис. 2.

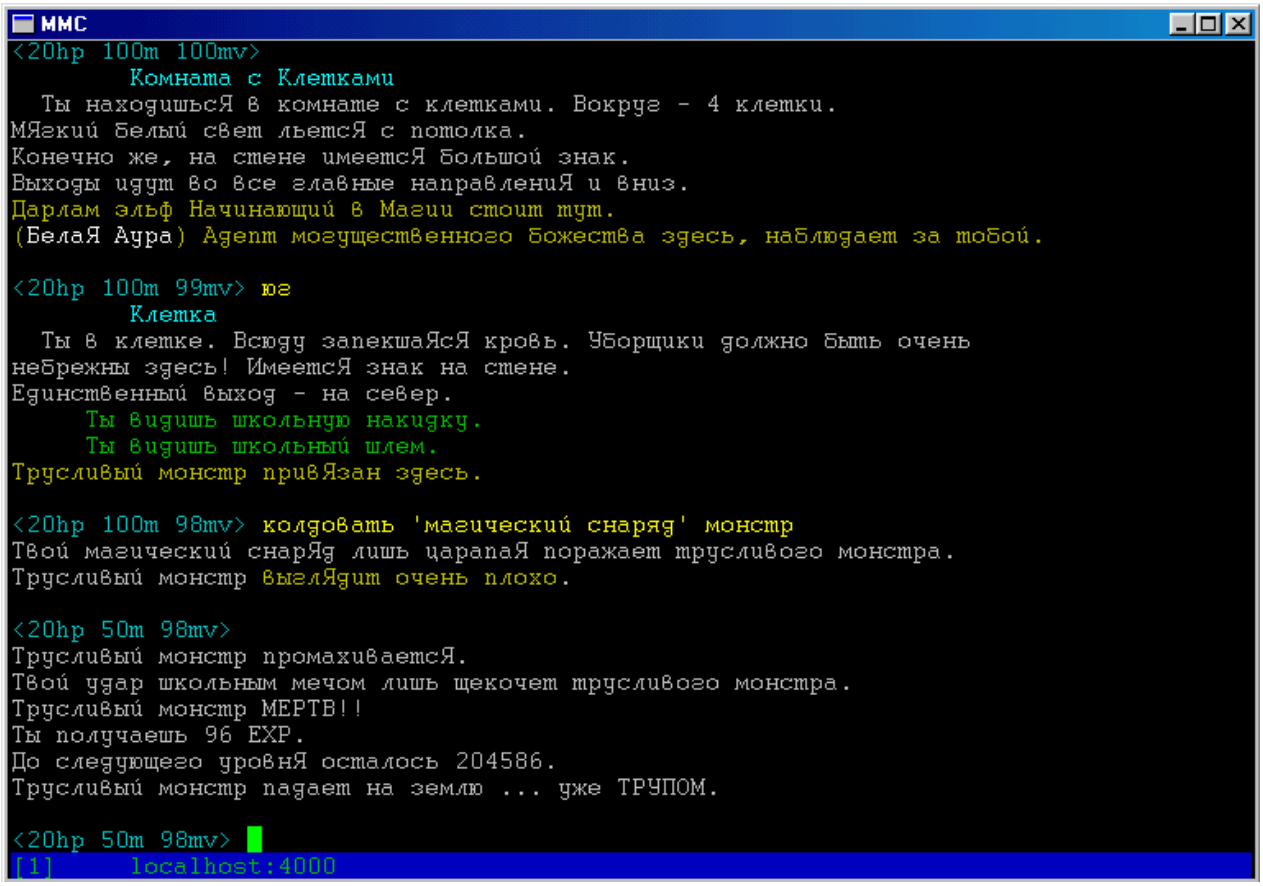

Рис. 2. Пример битвы

После победы над монстром пользователю сообщается, сколько очков получил персонаж. Получение очков является основной задачей бота. Сообщение о получении очков будет являться индикатором выполнения ботом поставленной задачи. Формат сообщения: «Ты получаешь … EXP».

Единицы перемещения уменьшаются при переходе персонажа из комнаты в комнату. Персонаж становится временно обездвиженным, когда у него кончаются единицы перемещения.

Игра организована в многопользовательском режиме. На сервере в одном игровом пространстве одновременно существуют несколько персонажей, управляемых различными пользователями. Сервер находится в сети Интернет и доступен круглосуточно. Существует локальная версия игры. Она предназначена для тестирования и отладки клиентского программного обеспечения. В локальной версии игры сервер запускается на компьютере пользователя. Для подключения к серверу не используется сеть. Игра происходит в

однопользовательском режиме – одновременно в игровом мире существует только один персонаж, управляемый пользователем.

При разработке бота используется локальная версия игры. Это обусловлено удобством отладки и тем, что локальной версии не требуется сеть. Однако разработанный бот может быть использован в многопользовательской игре. В последней главе описывается применение бота в реальной игре на сервере *Аладон*.

2. Диаграмма классов

Малая связность задач, решаемых искусственным интеллектом, в игре обуславливает создание отдельных автоматов для решения каждой из них. Отдельно выделен автомат, решающий задачу управления соединением с игровым сервером. Автоматы работают параллельно в отдельных потоках и взаимодействуют только через переменные состояний.

Для анализа текстовой информации, приходящей от сервера, применяется адаптер `MAdapter`. Он преобразует входящий поток текста во входные переменные и порождает события для автоматов. Автоматы обращаются к адаптеру для преобразования выходных воздействий в текст, который отсылается на игровой сервер.

Анализ текстовой информации, поступающей от сервера, состоит в нахождении строк удовлетворяющих заданным шаблонам. Например, строка «Ты хочешь пить» сигнализирует о том, что персонаж испытывает жажду. Индикатором участия в битве, является получение строки, заканчивающейся фразами «в прекрасном состоянии», «легко ранен», «ранен», «серьезно ранен», «выглядит очень плохо» и «в ужасном состоянии».

Диаграмма классов приведена на рис. 3.

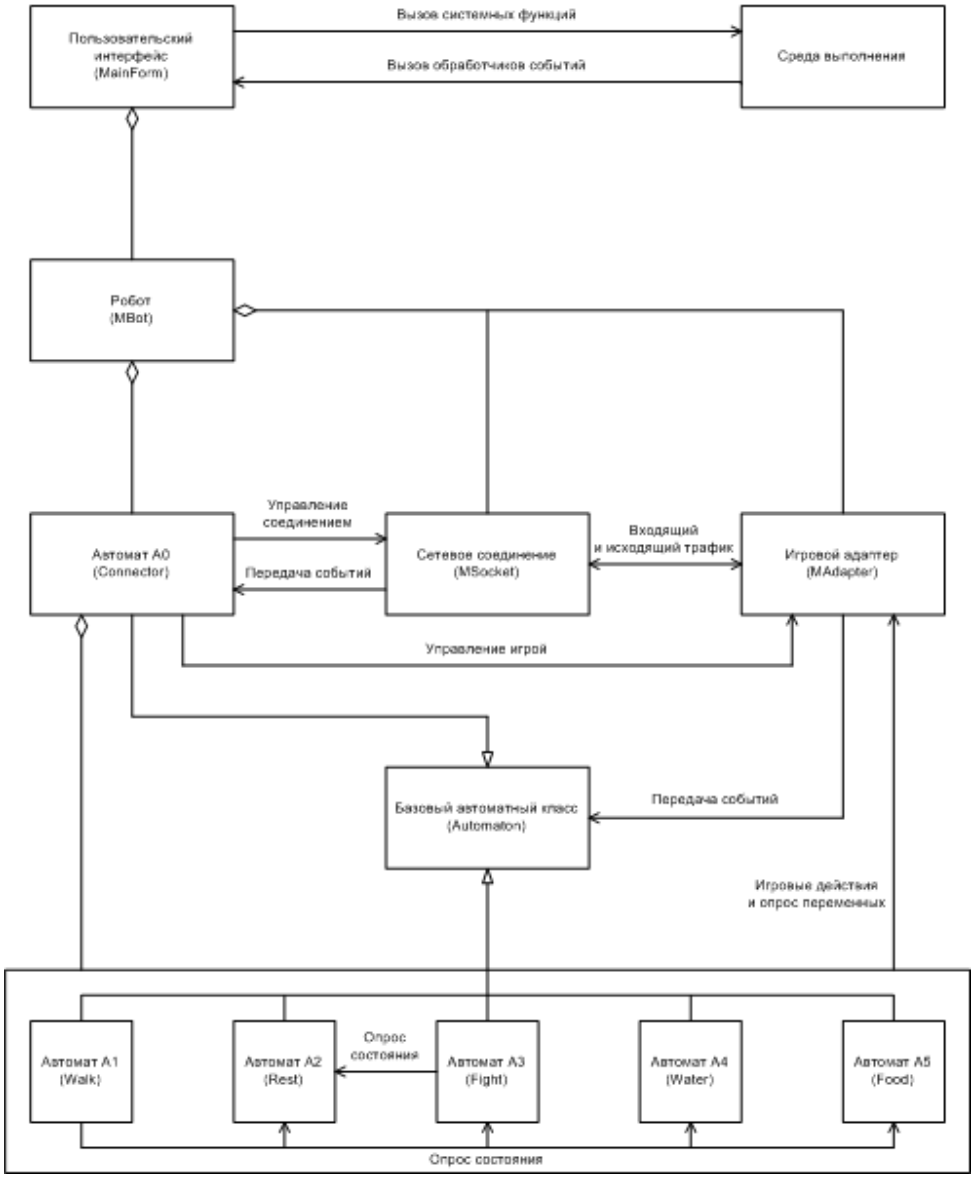

Рис. 3. Диаграмма классов

Пользовательский интерфейс реализован в классе `MainForm`. Класс `MainFrom` содержит в себе ссылку на класс бота `MBot`. Класс бота используется для инициализации игрового адаптера `MAdapter`, сетевого соединения `MSocket` и автомата управления соединением `Connector`. Класс сетевого соединения `MSocket` порождает события для автомата управления соединением `Connector` и выступает посредником во взаимодействии игрового адаптера с сервером. Автоматный класс управления соединением `Connector` создает и уничтожает автоматы игровой логики `A1-A5`. Все автоматные классы унаследованы от базового класса `Automaton`. В классе Automaton реализовано создание отдельного потока для работы автомата, определены общие для всех автоматных классов методы запуска, останова и ведения протокола.

3. Автомат «Управление соединением» (A0)

3.1. Словесное описание

Автомат предназначен для управления соединением с игровым сервером. Он выполняет подключение к серверу, вход в игру и контроль ошибок сетевого подключения. После успешного входа в игру этот автомат создает и запускает остальные автоматы управления ботом.

Схема связей автомата A0 приведена на рис. 4, а граф переходов – на рис. 5.

3.2. Схема связей

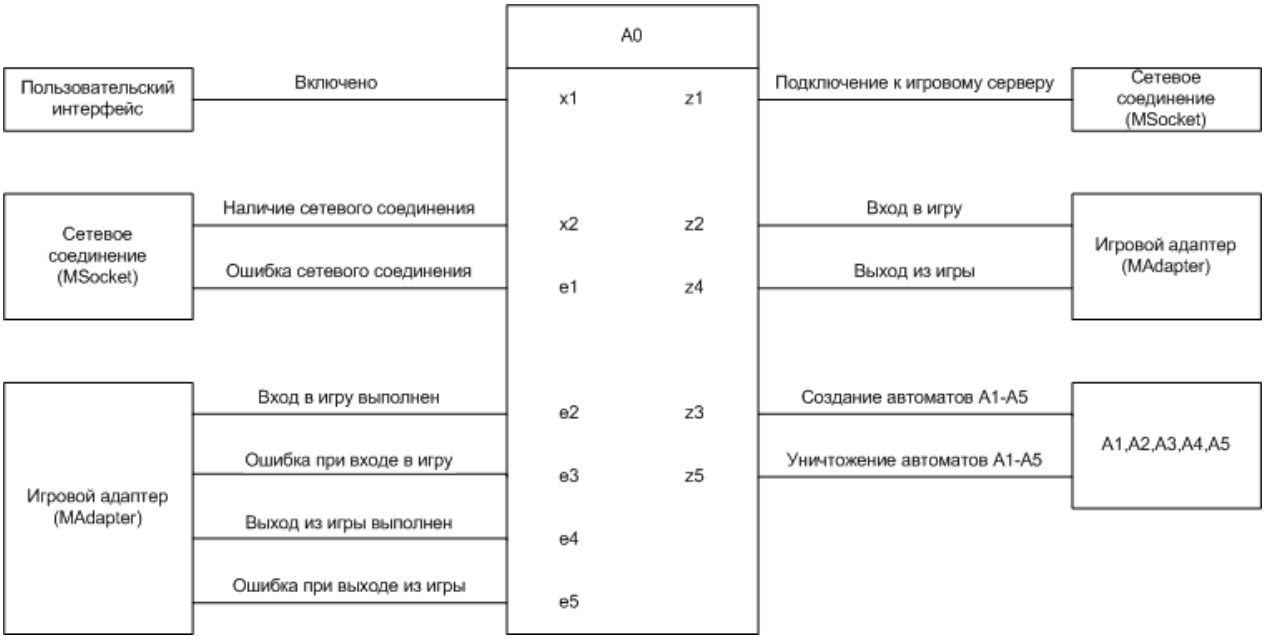

Рис. 4. Схема связей автомата A0

3.3. Граф переходов

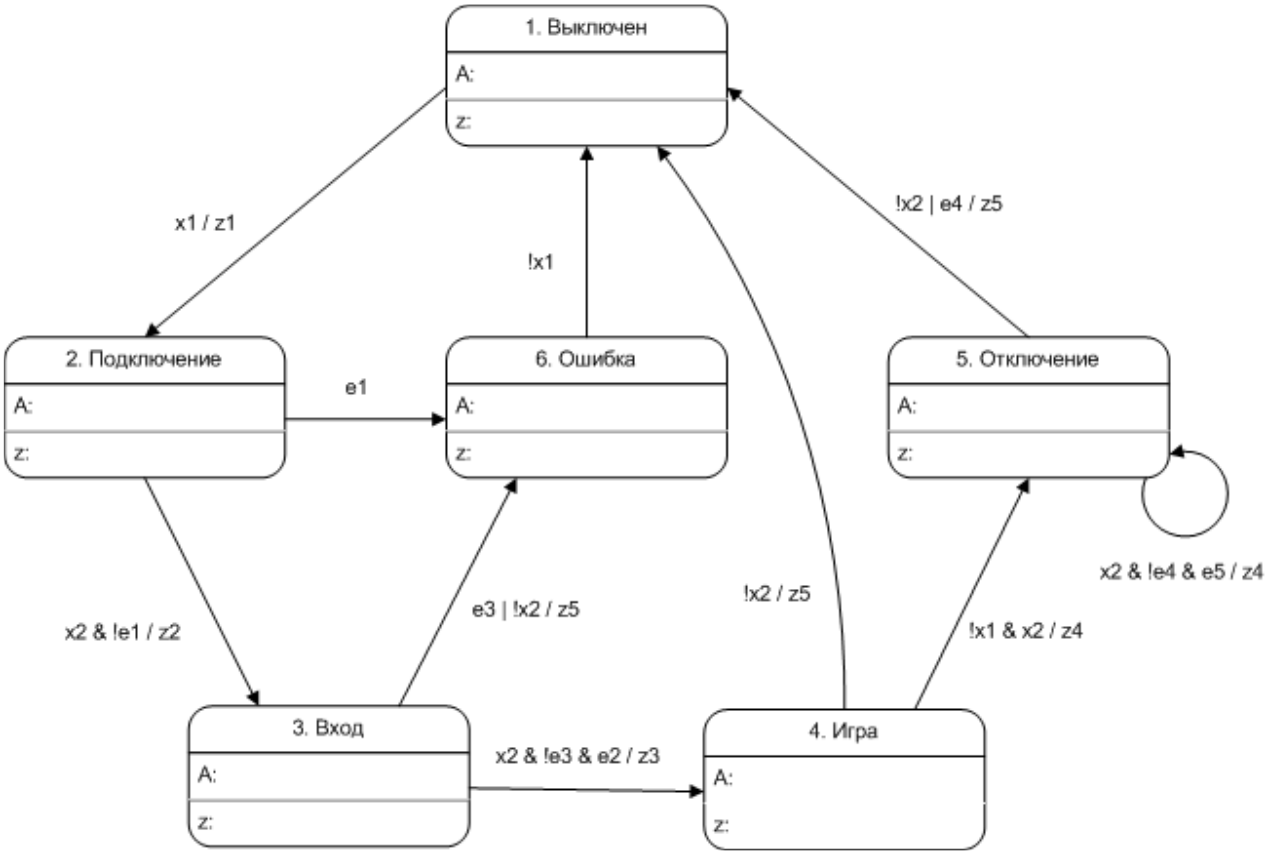

Рис. 5. Граф переходов автомата A0

4. Автомат «Перемещение» (A1)

4.1. Словесное описание

Автомат предназначен для управления перемещением персонажа между игровыми комнатами. В зависимости от состояний других автоматов устанавливается одна из четырех возможных целей перемещения: комната с противником, комната отдыха, магазин еды или магазин воды. После выбора направления осуществляется перемещение по выбранному направлению.

Схема связей автомата A1 приведена на рис. 6, а граф переходов – на рис. 7.

4.2. Схема связей

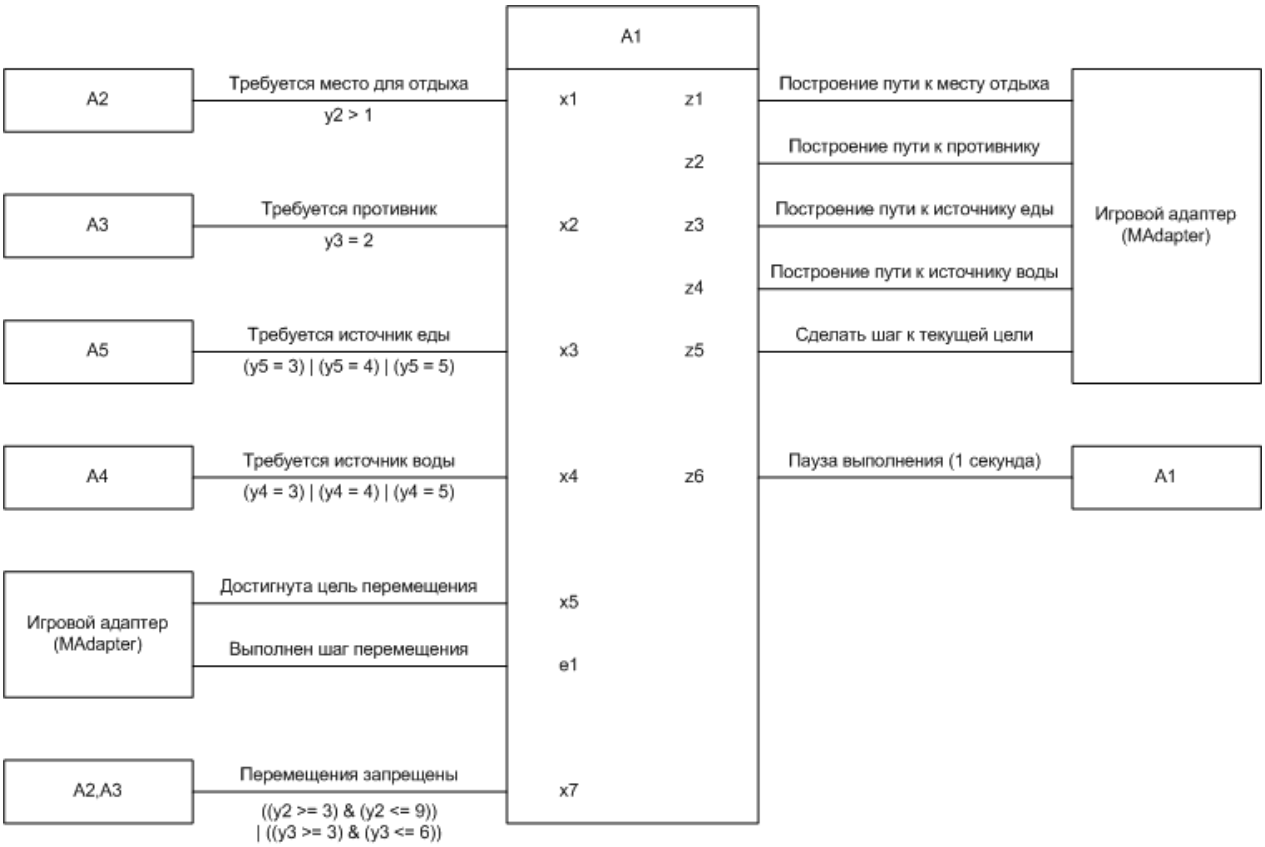

Рис. 6. Схема связей автомата A1

4.3. Граф переходов

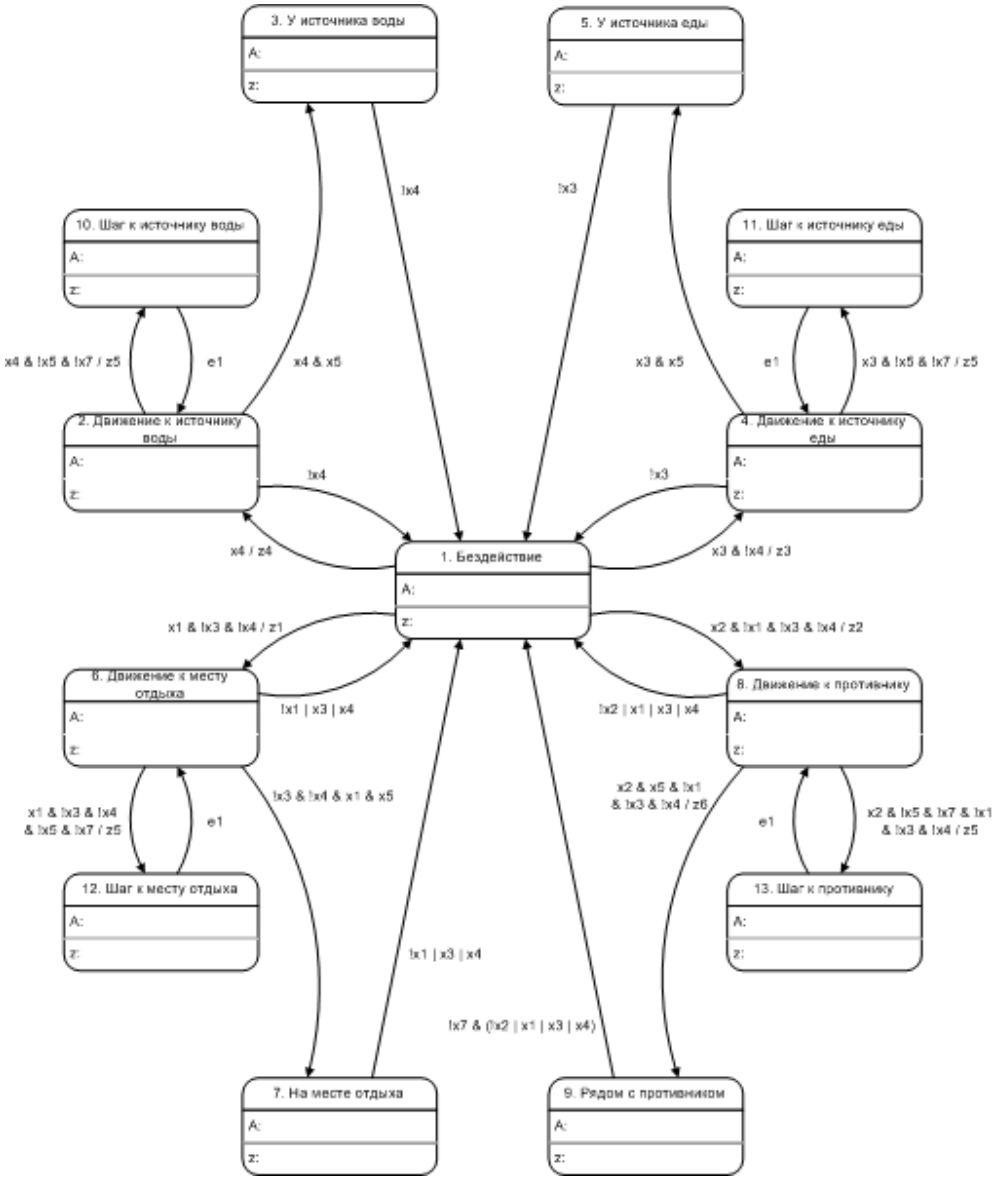

Рис. 7. Граф переходов автомата A1

5. Автомат «Управление отдыхом» (A2)

5.1. Словесное описание

Автомат предназначен для контроля над количеством единиц жизни, магии и перемещения. Когда уровень одного из этих жизненно важных параметров становится слишком низким, персонажу следует перемеситься в безопасную комнату и лечь спать. Некоторое время после битвы персонаж слишком взволнован и не может спать. В этом

случае ему необходимо сесть отдохнуть. Отдых необходим также в случае обездвиживания персонажа из-за отсутствия единиц перемещения.

Схема связей автомата A2 приведена на рис. 8, а граф переходов – на рис. 9.

5.2. Схема связей

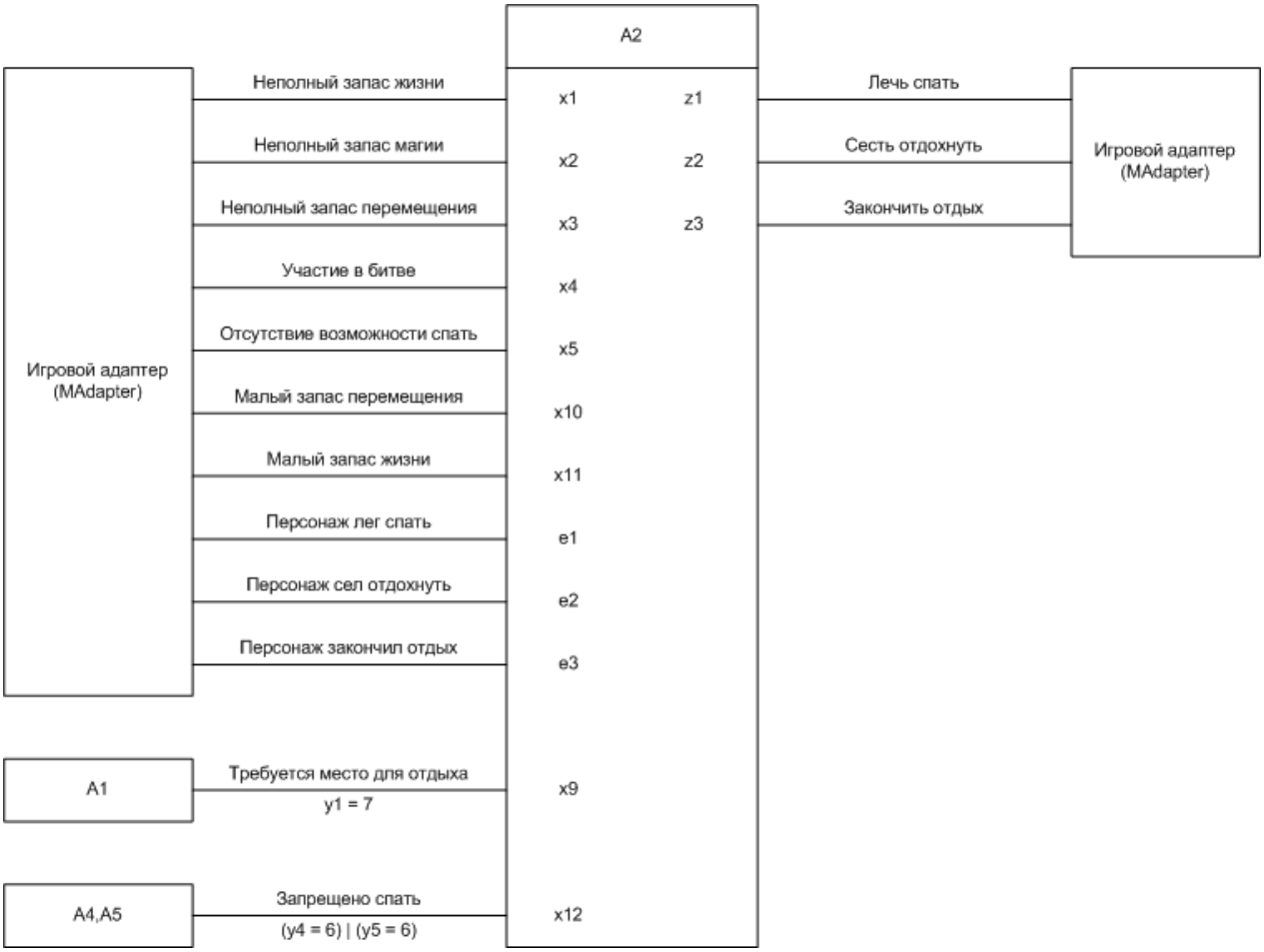

Рис. 8. Схема связей автомата A2

5.3. Граф переходов

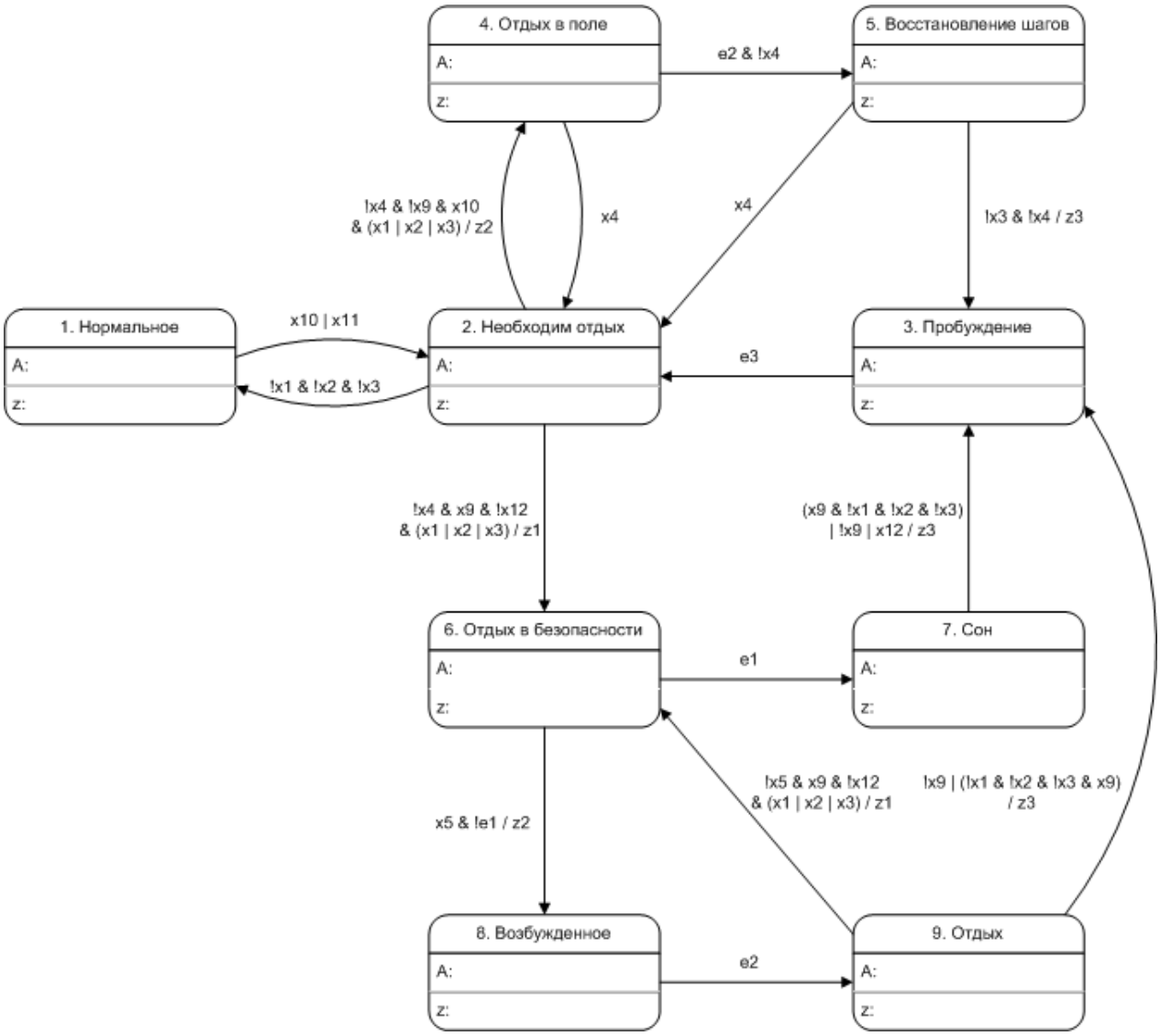

Рис. 9. Граф переходов автомата A2

6. Автомат «Управление битвой» (A3)

6.1. Словесное описание

Автомат осуществляет выбор противника и управление ведением битвы. Нападение на противника осуществляется только в том случае, если у персонажа достаточно для этого единиц жизни. При наличии достаточного количества единиц магии применяются боевые заклинания.

Схема связей автомата A3 приведена на рис. 10, а граф переходов – на рис. 11.

6.2. Схема связей

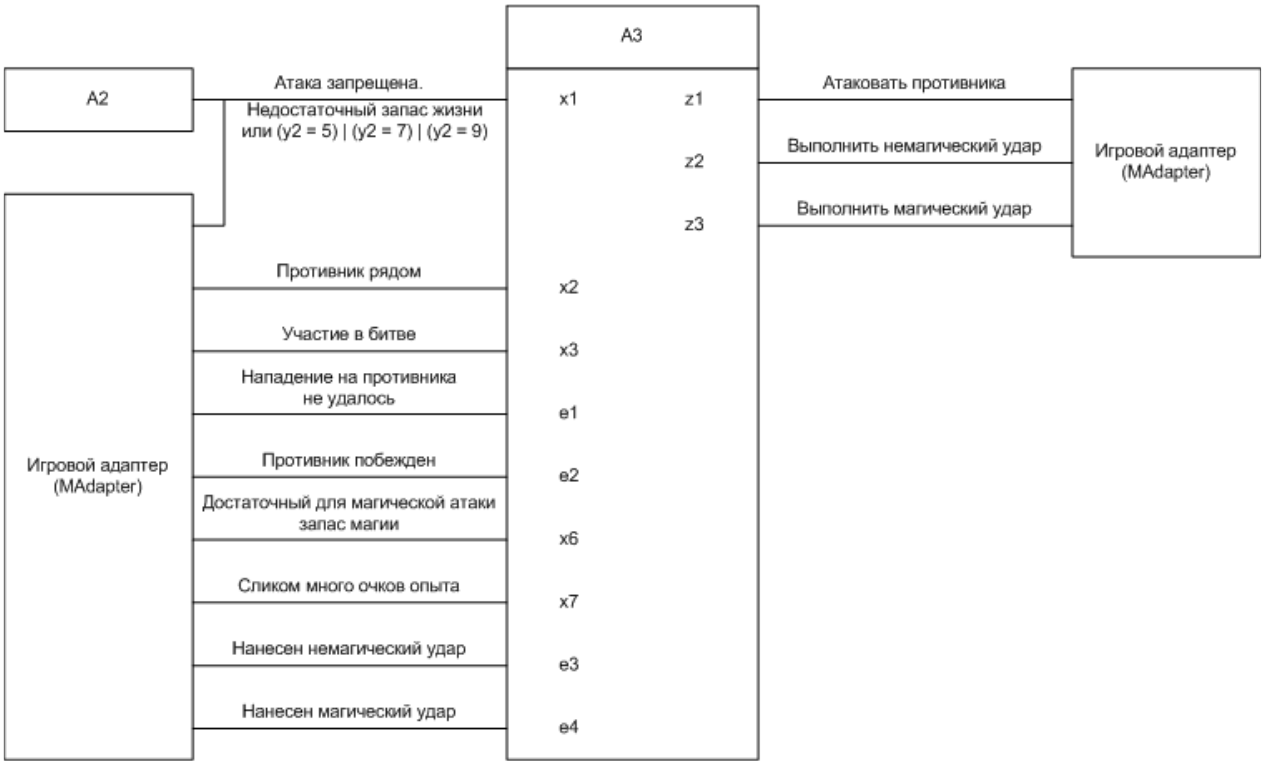

Рис. 10. Схема связей автомата A3

6.3. Граф переходов

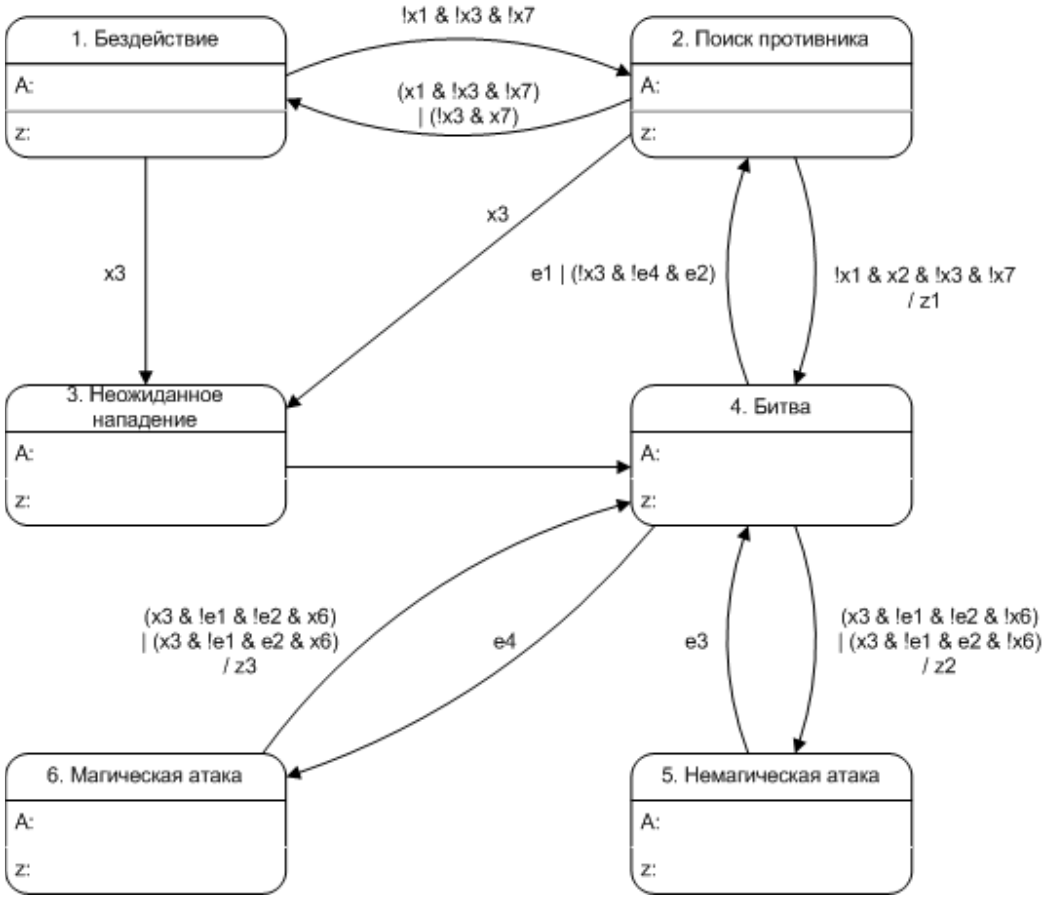

Рис. 11. Граф переходов автомата A3

7. Автомат «Вода» (A4)

7.1. Словесное описание

Автомат управляет утолением жажды персонажа. Жажда утоляется запасами воды, которые персонаж носит с собой. Когда запас воды истощен, персонажу следует сходить в магазин и купить емкость с водой. Если у персонажа кончаются деньги, автомат переходит в состояние ошибки, поскольку отсутствует возможность приобрести воду.

Схема связей автомата A4 приведена на рис. 12, а граф переходов – на рис. 13.

7.2. Схема связей

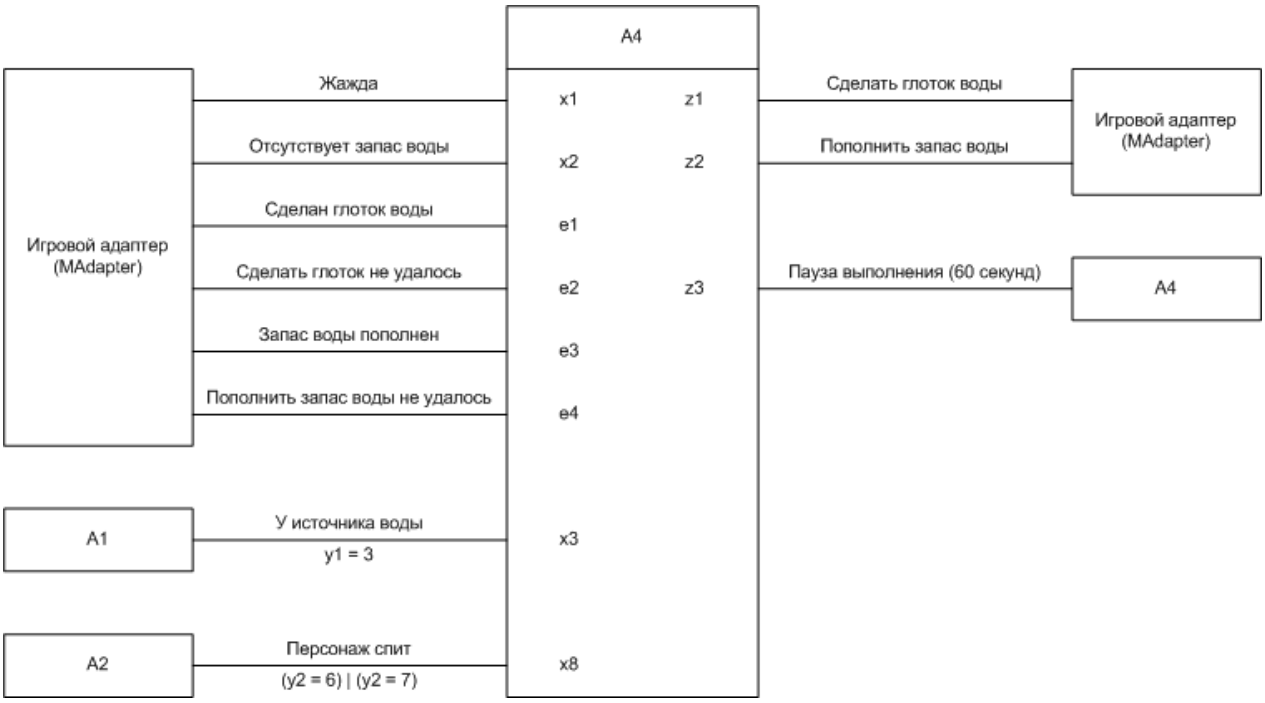

Рис. 12. Схема связей автомата A4

7.3. Граф переходов

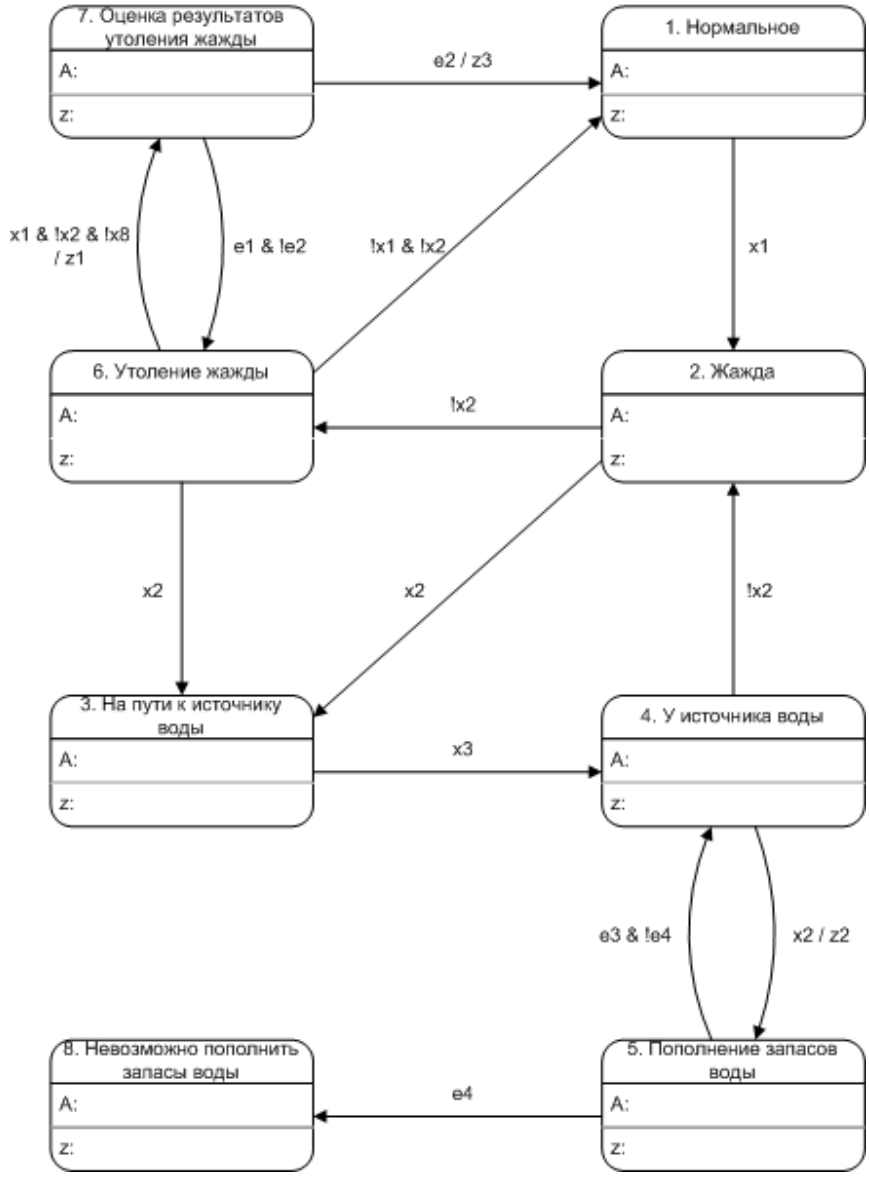

Рис. 13. Граф переходов автомата A4

8. Автомат «Еда» (A5)

8.1. Словесное описание

Автомат управляет утолением голода персонажа. Задача контроля над питанием персонажа похожа на задачу управления утолением жажды персонажа. По этой причине устройство автомата «Еда» (A5) аналогично устройству автомата «Вода» (A4).

Схема связей автомата A5 приведена на рис. 14, а граф переходов – на рис. 15.

8.2. Схема связей

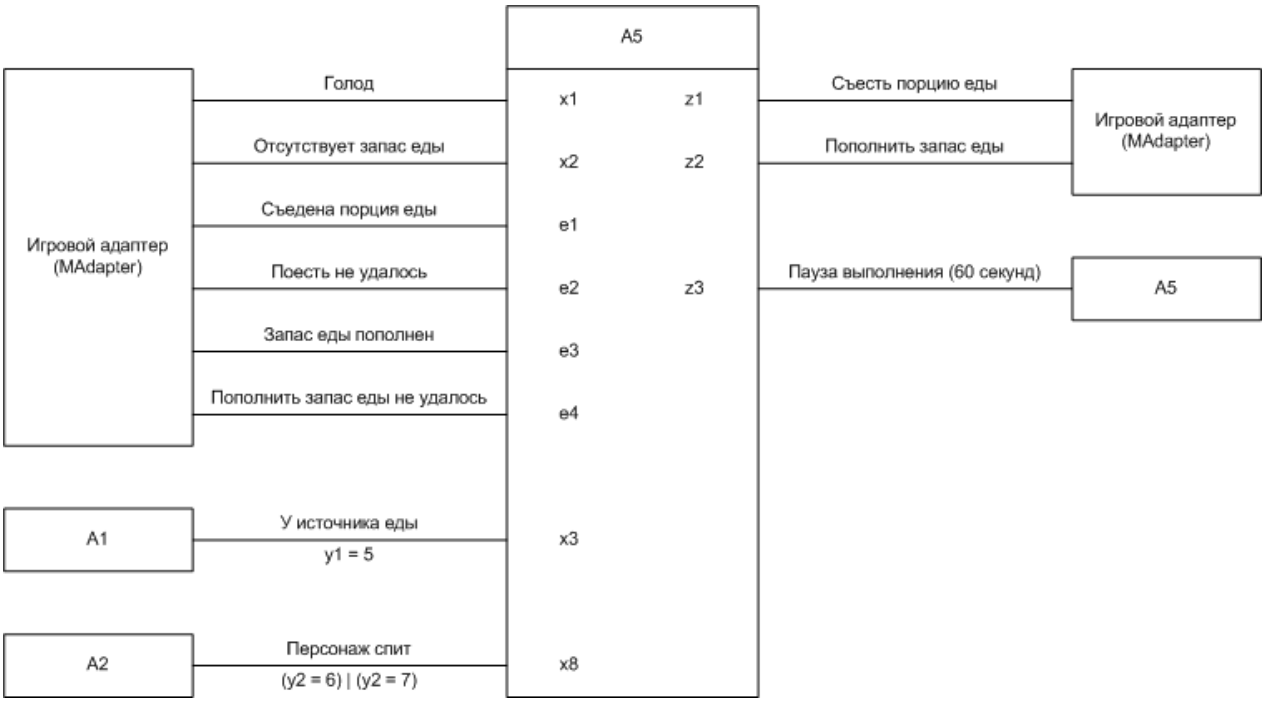

Рис. 14. Схема связей автомата A5

8.3. Граф переходов

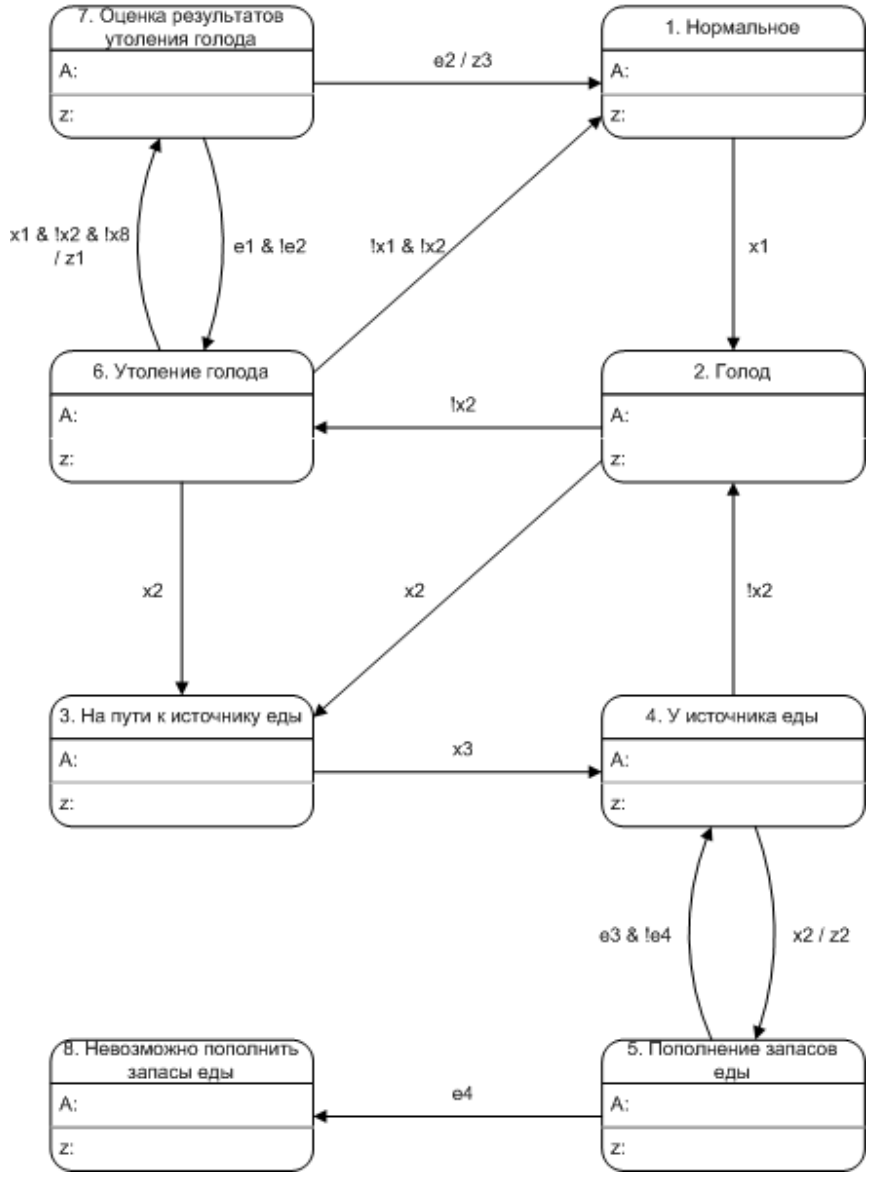

Рис. 15. Граф переходов автомата A5

9. Пример отладочного протокола

Ведение отладочного протокола позволяет в случае возникновения неполадок быстро их обнаружить и выяснить их причину. В протоколе отражаются переходы автоматов из одного состояния в другое, полученные от сервера строки и посланные на сервер команды.

Ниже приводится фрагмент отладочного протокола, содержащий бой с монстром в локальной версии игры.

```
Состояние изменилось A2: 6 -> 7 [x1=1 x2=1 x3=1 x4=0 x5=0 x9=1
x10=1 x11=1 x12=0 e1 e3]
Послана строка: "встать"
```

```
Состояние изменилось A2: 7 -> 3 [x1=0 x2=0 x3=0 x4=0 x5=0 x9=1
x10=0 x11=0 x12=0 e3]
Состояние изменилось A2: 3 -> 2 [x1=0 x2=0 x3=0 x4=0 x5=0 x9=1
x10=0 x11=0 x12=0 e3]
Состояние изменилось A2: 2 -> 1 [x1=0 x2=0 x3=0 x4=0 x5=0 x9=1
x10=0 x11=0 x12=0]
Состояние изменилось A3: 1 -> 2 [x1=0 x2=0 x3=0 x6=1 x7=0 e1 e3]
Состояние изменилось A1: 7 -> 1 [x1=0 x2=1 x3=0 x4=0 x5=1 x7=0]
Состояние изменилось A1: 1 -> 8 [x1=0 x2=1 x3=0 x4=0 x5=1 x7=0]
Послана строка: "запад"
Состояние изменилось A1: 8 -> 13 [x1=0 x2=1 x3=0 x4=0 x5=0 x7=0]
Получена строка: "<20hp 20mhp 100m 100mm 100mv 100mmv 64487exp
63513mexp> Ты просыпаешьсЯ и встаешь."
Получена строка: "  Комната с Клетками"
Получена строка: "  Ты находишьсЯ в комнате с клетками. Вокруг - 4
клетки."
Получена строка: "МЯгкий белый свет льетсЯ с потолка."
Получена строка: "Конечно же, на стене имеетсЯ большой знак."
Получена строка: "Выходы идут во все главные направлениЯ и вниз."
Получена строка: "[Видимые выходы: север восток юг запад низ]"
Получена строка: "(БелаЯ Аура) Адепт могущественного божества
здесь, наблюдает за тобой."
Состояние изменилось A1: 13 -> 8 [x1=0 x2=1 x3=0 x4=0 x5=0 x7=0 e1]
Послана строка: "запад"
Состояние изменилось A1: 8 -> 13 [x1=0 x2=1 x3=0 x4=0 x5=0 x7=0]
Получена строка: "<20hp 20mhp 100m 100mm 100mv 100mmv 64487exp
63513mexp>     Клетка"
Получена строка: "  Ты в клетке. Всюду запекшаЯсЯ кровь. Уборщики
должно быть очень"
Получена строка: "небрежны здесь! ИмеетсЯ знак на стене."
Получена строка: "Единственный выход - на восток."
Получена строка: "[Видимые выходы: восток]"
Получена строка: "Трусливый агрессивный монстр привЯзан здесь."
Состояние изменилось A1: 13 -> 8 [x1=0 x2=1 x3=0 x4=0 x5=1 x7=0 e1]
Послана строка: "убить монстр"
Состояние изменилось A3: 2 -> 4 [x1=0 x2=1 x3=0 x6=1 x7=0 e1 e3]
Состояние изменилось A3: 4 -> 2 [x1=0 x2=1 x3=0 x6=1 x7=0 e1 e3]
Послана строка: "убить монстр"
Состояние изменилось A3: 2 -> 4 [x1=0 x2=1 x3=0 x6=1 x7=0 e3]
Состояние изменилось A1: 8 -> 9 [x1=0 x2=1 x3=0 x4=0 x5=1 x7=0]
Получена строка: "<20hp 20mhp 100m 100mm 99mv 100mmv 64487exp
63513mexp> Все, туда не пройти."
Получена строка: "<20hp 20mhp 100m 100mm 99mv 100mmv 64487exp
63513mexp> Ты промахиваешьсЯ."
Получена строка: "Трусливый агрессивный монстр в прекрасном
состоЯнии. "
Послана строка: "кол ма"
Состояние изменилось A3: 4 -> 6 [x1=0 x2=0 x3=1 x6=1 x7=0 e3]
```

```
Получена строка: "<20hp 20mhp 100m 100mm 99mv 100mmv 64487exp
63513mexp> Ты это и делаешь!"
Получена строка: "Трусливый агрессивный монстр в прекрасном
состоЯнии. "
Получена строка: "<20hp 20mhp 100m 100mm 99mv 100mmv 64487exp
63513mexp> Ты терЯешь концентрацию."
Получена строка: "Трусливый агрессивный монстр в прекрасном
состоЯнии. "
Состояние изменилось A3: 6 -> 4 [x1=0 x2=0 x3=1 x6=1 x7=0 e3 e4]
Послана строка: "кол ма"
Состояние изменилось A3: 4 -> 6 [x1=0 x2=0 x3=1 x6=1 x7=0 e3]
Получена строка: "<20hp 20mhp 75m 100mm 99mv 100mmv 64487exp
63513mexp> "
Получена строка: "Твой рубЯщий удар проходит мимо трусливого
агрессивного монстра."
Получена строка: "Когти трусливого агрессивного монстра проносЯтсЯ
мимо тебЯ."
Получена строка: "Трусливый агрессивный монстр в прекрасном
состоЯнии. "
Получена строка: "<20hp 20mhp 75m 100mm 99mv 100mmv 64487exp
63513mexp> Твой магический снарЯд лишь царапаЯ поражает трусливого
агрессивного монстра."
Получена строка: "Трусливый агрессивный монстр ранен. "
Состояние изменилось A3: 6 -> 4 [x1=0 x2=0 x3=1 x6=1 x7=0 e3 e4]
Послана строка: "кол ма"
Состояние изменилось A3: 4 -> 6 [x1=0 x2=0 x3=1 x6=1 x7=0 e3]
Получена строка: "<20hp 20mhp 25m 100mm 99mv 100mmv 64487exp
63513mexp> "
Получена строка: "Твой удар школьным мечом лишь щекочет трусливого
агрессивного монстра."
Получена строка: "Когти трусливого агрессивного монстра проносЯтсЯ
мимо тебЯ."
Получена строка: "Трусливый агрессивный монстр выглЯдит очень
плохо. "
Получена строка: "<20hp 20mhp 25m 100mm 99mv 100mmv 64487exp
63513mexp> У тебЯ не хватает энергии."
Получена строка: "Трусливый агрессивный монстр выглЯдит очень
плохо. "
Состояние изменилось A3: 6 -> 4 [x1=0 x2=0 x3=1 x6=0 x7=0 e3 e4]
Состояние изменилось A3: 4 -> 5 [x1=0 x2=0 x3=1 x6=0 x7=0 e3]
Состояние изменилось A3: 5 -> 4 [x1=0 x2=0 x3=1 x6=0 x7=0 e3]
Состояние изменилось A3: 4 -> 5 [x1=0 x2=0 x3=1 x6=0 x7=0]
Получена строка: "<20hp 20mhp 25m 100mm 99mv 100mmv 64487exp
63513mexp> "
Получена строка: "Ты промахиваешьсЯ."
Получена строка: "Трусливый агрессивный монстр промахиваетсЯ."
Получена строка: "Трусливый агрессивный монстр выглЯдит очень
плохо. "
Состояние изменилось A3: 5 -> 4 [x1=0 x2=0 x3=1 x6=0 x7=0 e3]
```

```
Состояние изменилось A3: 4 -> 5 [x1=0 x2=0 x3=1 x6=0 x7=0]
Получена строка: "<20hp 20mhp 25m 100mm 99mv 100mmv 64487exp
63513mexp> "
Получена строка: "Ты промахиваешьсЯ."
Получена строка: "Трусливый агрессивный монстр промахиваетсЯ."
Получена строка: "Трусливый агрессивный монстр выглЯдит очень
плохо. "
Состояние изменилось A3: 5 -> 4 [x1=0 x2=0 x3=1 x6=0 x7=0 e3]
Состояние изменилось A3: 4 -> 5 [x1=0 x2=0 x3=1 x6=0 x7=0]
Получена строка: "<20hp 20mhp 25m 100mm 99mv 100mmv 64487exp
63513mexp> "
Получена строка: "Твой рубЯщий удар проходит мимо трусливого
агрессивного монстра."
Получена строка: "Трусливый агрессивный монстр едва дотрагиваЯсь
терзает тебЯ ."
Получена строка: "Трусливый агрессивный монстр выглЯдит очень
плохо. "
Состояние изменилось A3: 5 -> 4 [x1=0 x2=0 x3=1 x6=0 x7=0 e3]
Состояние изменилось A3: 4 -> 5 [x1=0 x2=0 x3=1 x6=0 x7=0]
Получена строка: "<19hp 20mhp 25m 100mm 99mv 100mmv 64487exp
63513mexp> "
Получена строка: "Твой удар школьным мечом лишь щекочет трусливого
агрессивного монстра."
Получена строка: "Трусливый агрессивный монстр МЕРТВ!!"
Послана строка: "улыб"
Получена строка: "Ты получаешь 112 EXP."
Получена строка: "До следующего уровнЯ осталось 63401."
Состояние изменилось A3: 5 -> 4 [x1=0 x2=0 x3=1 x6=0 x7=0 e2 e3]
Получена строка: "Голова трусливого агрессивного монстра
разлетаетсЯ вдребезги и его мозги брызгают на тебЯ."
Получена строка: "Ты берешь 3 серебрЯных монет из трупа трусливого
агрессивного монстра."
Получена строка: "<19hp 20mhp 25m 100mm 99mv 100mmv 64599exp
63401mexp> Ты счастливо улыбаешьсЯ."
Состояние изменилось A3: 4 -> 5 [x1=0 x2=0 x3=1 x6=0 x7=0 e2]
Состояние изменилось A3: 5 -> 4 [x1=0 x2=0 x3=0 x6=0 x7=0 e2 e3]
Состояние изменилось A3: 4 -> 2 [x1=0 x2=0 x3=0 x6=0 x7=0 e2]
Состояние изменилось A1: 9 -> 8 [x1=0 x2=1 x3=0 x4=0 x5=1 x7=0]
Послана строка: "восток"
```

10. Работа в реальных условиях

Для запуска робота в реальных условиях многопользовательской игры необходимо сначала создать персонажа на сервере *Аладон*. Это можно сделать, подключившись при помощи стандартного текстового терминала *telnet* к серверу, расположенному по адресу www.aladon.ru:9000. После подключения требуется, следуя указаниям сервера, создать новый персонаж. Желательно чтобы персонаж имел магическую профессию. Используя команды

«север», «юг», «запад», «восток», требуется переместить персонажа в комнату, которая называется «учебный класс Зампа» и научить персонажа боевому заклинанию. Для обучения персонажа необходимо ввести команду «практиковать магический снаряд». Затем персонажа требуется переместить в комнату, которая называется «комната с клетками». Перед запуском робота требуется убедиться, что для персонажа отключены цвета и включены режимы «автовыходы», «автозолото», «автожертва». Для включения и выключения цветов применяется команда «цвет». Для включения и выключения режимов используются команды «автовыходы», «автозолото», «автожертва».

После создания персонажа необходимо внести информацию о нем в файл настройки бота. В файле настройки указывается адрес и порт игрового сервера, название адаптера, имя и пароль персонажа. В этой версии бота используется адаптер *RealAdapter*. Ниже приведен фрагмент отладочного протокола для игры в многопользовательском режиме:

```
Получена строка: "Эта версиЯ создана Jul 30 2005, 15:48:15"
Получена строка: "Добро пожаловать в мир АЛАДОН."
Получена строка: "Последний вход - Tue Jun 13 18:40:44 2006 с
адреса 194.85.161.34"
Получена строка: "  Учебный боевой зал"
Получена строка: "  Запах пота и стали заполнЯет эту комнату.
Вокруг - четыре клетки."
Получена строка: "Неровный свет от факелов отбрасывает дрожащие
тени."
Получена строка: "Конечно же, на стене имеетсЯ большой знак."
Получена строка: "Выходы идут во все главные направлениЯ и вниз."
Получена строка: "[Видимые выходы: север восток юг запад низ]"
Получена строка: "(БелаЯ Аура) ГигантскаЯ фигура воина возвышаетсЯ
над тобой."
Получена строка: "ТебЯ ожидают 130 новостей."
Получена строка: "ТебЯ ждет 36 измененииий."
Получена строка: "Тебе нужно прочесть 42 письма."
Получена строка: "29 идей, посетивших светлые головы, ждет тебЯ."
Получена строка: "<20hp 20mhp 100m 100mm 100mv 100mmv 89600exp
89600mexp>     Учебный боевой зал"
Получена строка: "  Запах пота и стали заполнЯет эту комнату.
Вокруг - четыре клетки."
Получена строка: "Неровный свет от факелов отбрасывает дрожащие
тени."
Получена строка: "Конечно же, на стене имеетсЯ большой знак."
Получена строка: "Выходы идут во все главные направлениЯ и вниз."
Получена строка: "[Видимые выходы: север восток юг запад низ]"
Получена строка: "(БелаЯ Аура) ГигантскаЯ фигура воина возвышаетсЯ
над тобой."
Получена строка: "<20hp 20mhp 100m 100mm 100mv 100mmv 89600exp
89600mexp> ТебЯ теперь нельзЯ призвать."
Получена строка: "<20hp 20mhp 100m 100mm 100mv 100mmv 89600exp
89600mexp> Ты уже стоишь."
Получена строка: "<20hp 20mhp 100m 100mm 100mv 100mmv 89600exp
89600mexp> Приглашение установлено в <%hhp %Hmhp %mm %Mmm %vmv
%Vmmv %xexp %Xmexp> "
```

```
Состояние изменилось A1: 1 -> 8 [x1=0 x2=1 x3=0 x4=0 x5=1 x7=0 e1]
Послана строка: "юг"
Состояние изменилось A1: 8 -> 13 [x1=0 x2=1 x3=0 x4=0 x5=0 x7=0 e1]
Состояние изменилось A1: 13 -> 8 [x1=0 x2=1 x3=0 x4=0 x5=0 x7=0 e1]
Послана строка: "юг"
Состояние изменилось A1: 8 -> 13 [x1=0 x2=1 x3=0 x4=0 x5=0 x7=0]
Состояние изменилось A3: 1 -> 2 [x1=0 x2=0 x3=0 x6=1 x7=0 e3]
Состояние изменилось A0: 3 -> 4 [x1=1 x2=1 e2]
Получена строка: "<20hp 20mhp 100m 100mm 100mv 100mmv 89600exp
89600mexp>     Клетка"
Получена строка: "  Ты в клетке. Всюду запекшаЯсЯ кровь. Уборщики
должно быть очень"
Получена строка: "небрежны здесь! ИмеетсЯ знак на стене."
Получена строка: "Единственный выход - на север."
Получена строка: "[Видимые выходы: север]"
Получена строка: "Трусливый монстр привЯзан здесь."
Послана строка: "убить монстр"
Состояние изменилось A3: 2 -> 4 [x1=0 x2=1 x3=0 x6=1 x7=0 e3]
Состояние изменилось A1: 13 -> 8 [x1=0 x2=0 x3=0 x4=0 x5=1 x7=1 e1]
Состояние изменилось A1: 8 -> 1 [x1=0 x2=0 x3=0 x4=0 x5=1 x7=1]
Получена строка: "<20hp 20mhp 100m 100mm 99mv 100mmv 89600exp
89600mexp> Все, туда не пройти."
Получена строка: "<20hp 20mhp 100m 100mm 99mv 100mmv 89600exp
89600mexp> Твой удар школьным мечом лишь щекочет трусливого
монстра."
Получена строка: "Трусливый монстр легко ранен. "
Послана строка: "кол ма"
Состояние изменилось A3: 4 -> 6 [x1=0 x2=0 x3=1 x6=1 x7=0 e3]
Получена строка: "<20hp 20mhp 100m 100mm 99mv 100mmv 89600exp
89600mexp> Твой магический снарЯд едва дотрагиваЯсь поражает
трусливого монстра."
Получена строка: "Трусливый монстр ранен. "
Состояние изменилось A3: 6 -> 4 [x1=0 x2=0 x3=1 x6=1 x7=0 e3 e4]
Послана строка: "кол ма"
Состояние изменилось A3: 4 -> 6 [x1=0 x2=0 x3=1 x6=1 x7=0 e3]
Получена строка: "<20hp 20mhp 50m 100mm 99mv 100mmv 89600exp
89600mexp> "
Получена строка: "Твой удар школьным мечом лишь щекочет трусливого
монстра."
Получена строка: "Трусливый монстр едва дотрагиваЯсь терзает тебЯ
."
Получена строка: "Трусливый монстр серьезно ранен. "
Получена строка: "<18hp 20mhp 50m 100mm 99mv 100mmv 89600exp
89600mexp> Твой магический снарЯд едва дотрагиваЯсь поражает
трусливого монстра."
Получена строка: "Трусливый монстр МЕРТВ!!"
Послана строка: "улыб"
Получена строка: "Ты получаешь 124 EXP."
Состояние изменилось A3: 6 -> 4 [x1=0 x2=0 x3=1 x6=1 x7=0 e2 e3 e4]
```

```
Послана строка: "кол ма"
Состояние изменилось A3: 4 -> 6 [x1=0 x2=0 x3=1 x6=1 x7=0 e2 e3]
Получена строка: "До следующего уровнЯ осталось 89476."
Получена строка: "Вырванный коготь трусливого монстра падает на
землю."
Получена строка: "Ты берешь 5 серебрЯных монет из трупа трусливого
монстра."
Получена строка: "Ты жертвуешь труп трусливого монстра со всем
содержимым своему божеству."
Получена строка: "Семидар дает тебе одну серебрЯную монетку за твою
жертву."
```

Заключение

Использование автоматного подхода позволило формально описать логику игры со сложной системой правил, и реализовать искусственный интеллект, который в большинстве случаев не уступает естественному. Программа, построенная на автоматах, получается простой для модификации. Наличие проектной и программной документации позволяет легче разобраться в устройстве робота.

Несмотря на наличие опыта написания программ у автора сложилось впечатление, что без использования автоматного подхода эту задачу было бы не решить.

Выполнение данной работы на основании автоматного подхода проходило следующим образом: сначала были разработаны автоматы, затем выполнена их программная реализация. В результате тестирования были обнаружен ряд несоответствий реального поведения системы требуемому. Ведение отладочного протокола позволило обнаружить причины этого несоответствия сравнительно быстро, исправить автоматы и внести изменения в программную реализацию. После этого программа заработала правильно.

Источники

1. *Сайт кафедры* «Информационные системы» СПбГУ ИТМО. http://is.ifmo.ru
2. *Сайт игры* «Аладон» http://www.aladon.ru
3. *Кузнецов Д. В., Шалыто А. А.* Система управления танком для игры «Robocode». Вариант 2. http://is.ifmo.ru/projects/robocode2/
4. *Лопатухина А. Д.* Модель искусственного интеллекта игрового бота. http://is.ifmo.ru/projects/gamebot/
5. *Марков С. М., Шалыто А. А.* Система управления травоядным существом для игры «Terrarium». http://is.ifmo.ru/projects/terrarium/
6. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. http://is.ifmo.ru/books/switch/1
7. *Каталог текстовых многопользовательских ролевых игр.* http://www.mudconnector.com

Приложение. Исходные коды

```
package ru.ifmo.rain.abdrashitov.elf;
/**
 * Абстрактный класс автомата. Содержит общие для автоматов данного проекта свойства и методы.
 * Все автоматы для повышения производительности выделены в отдельные потоки. Класс абстрактного
 * автомата позволяет производить управление потоками автоматов (их запуск и остановку).
 * Также класс поддерживает работу менеджера событий. Для реализации конкретного автомата
 * прописывается процедура инициализации и функция {@link Automaton#nextStep()}.
 */
public abstract class Automaton implements Runnable {
    /**
     * Функция перехода в следующее состояние. Реализации этой функции следует на основе
     * анализа входных переменных перевести автомат в новое состояние вызовом
     * функции {@link Automaton#setState(int)} и произвести необходимые выходные воздействия.
     * @return Возвращает истину, если переход произведен. Если текущее состояние входных
     * переменных подразумевает переход по умолчанию в текущее состояние без выполнения
     * выходных воздействий возвращается ложь. Значение используется для экономии системных
     * ресурсов.
     */
    protected abstract boolean nextStep();
    /**
     * Функция возвращает название автомата. Название будет использоваться для ведения лога.
     * @return Строковое представление названия автомата.
     */
    abstract protected String getName();
    /**
     * Функция выводит значения входных переменных в виде строки. Используется для ведения лога.
     * @return Текстовое представление значений переменных.
     */
    abstract protected String getVariablesDump();

    /**
     * Текстовый поток лога.
     */
    private Logger logger;
    /**
     * Менеджер событий.
     */
    private VariableNotifier notifier;
    /**
     * Состояние автомата.
     */
    private int y = 1;
    /**
     * Идентификатор события изменения состояния автомата.
     */
    private Object yEvent = new Object();
    /**
     * Переменная характеризующая активность потока автомата. Установка переменной
     * значения ложь приводит к остановки текущего потока.
     */
    private boolean isThreadActive;
    /**
     * Идентификатор события остановки автомата и уничтожения потока, в котором он выполняется.
     */
    private Object isThreadActiveEvent = new Object();

    /**
     * Конструктор по умолчанию инициализирует внутренние объекты.
     */
    protected Automaton() {
        notifier = new VariableNotifier();
        notifier.addChangeListener(isThreadActiveEvent, this);
    }
    /**
     * Функция установки потока, в который будут выводиться логи. Поток лога обязательно должен
     * быть установлен перед запуском автомата.
     * @param logger Ссылка на поток лога.
     */
    protected void setLogger(Logger logger) {
        this.logger = logger;
```

```
}
/**
 * Функция добавления слушателя изменений состояния автомата.
 * @param listener Представитель слушателя.
 */
protected void addStateListener(Object listener) {
    notifier.addChangeListener(yEvent, listener);
}
/**
 * Функция получения состояния автомата.
 * @return Состояние автомата.
 */
protected int getState() {
    return y;
}
/**
 * Функция перехода в новое состояние.
 * @param state Номер нового состояния.
 */
protected void setState(int state) {
    if (this.y != state) {
        this.y = state;
        notifier.eventHappened(yEvent);
    }
}
/**
 * В процедуре содержится основной цикл работы автомата.
 * В цикле автомат переходит по входным переменным из одного состояния
 * в другое, если переход существует. Если переход отсутствует, предполагается
 * переход в текущее состояния без выходных воздействий, в этом случае поток
 * автомата усыпляется, до наступления изменений во входных переменных.
 */
public void run() {
    y = 1;
    int yOld = y;
    isThreadActive = true;
    String name = getName();
    while (isThreadActive) {
        if (!nextStep() && isThreadActive) {
            try {
                synchronized(this) {
                    this.wait(1000);
                }
            } catch (InterruptedException e) {
            }
        }
        else {
            if (logger != null) {
                synchronized (logger) {
                    StringBuffer logbuf = new StringBuffer();
                    logbuf.append("Состояние изменилось ");
                    logbuf.append(name);
                    logbuf.append(": ");
                    logbuf.append(yOld);
                    logbuf.append(" -> ");
                    logbuf.append(y);
                    logbuf.append(" [");
                    logbuf.append(getVariablesDump());
                    logbuf.append("]");
                    logger.printToLog(logbuf.toString());
                }
            }
            yOld = y;
        }
    }
}
/**
 * Процелура запуска автомата. Создает новый поток, в котором начинается выполнение
 * основного цикла работы автомата.
 */
protected void start() {
    (new Thread(this)).start();
}
/**
```

```
     * Процедура остановки автомата. После остановки автомата уничтожается поток,
     * в котором автомат выполнялся.
     */
    protected void stop() {
        isThreadActive = false;
        notifier.eventHappened(isThreadActiveEvent);
    }
}

package ru.ifmo.rain.abdrashitov.elf;

import java.util.HashMap;
import java.util.regex.Pattern;
import java.util.regex.Matcher;

/**
 * Класс конфигурации робота. Обеспечивает загрузку параметров из текстового
 * представления и их хранение с возможностью получения значения параметра
 * по его названию.
 */
public class BotConfig {
    /**
     * Хеш таблица, хранящая параметры конфигурации.
     */
    private HashMap params;

    /**
     * Инициализирует обьект и производит загрузку параметров конфигурации
     * из строкового представления.
     * @param configStr Строка содержащая строковое представление конфигурации.
     */
    protected BotConfig(CharSequence configStr) {
        params = new HashMap();
        loadConfig(configStr);
    }

    /**
     * Функция загрузки параметров из строкового представления. Строка должна содержать
     * параметры в формате: "имя = значение;".
     * @param configStr Строка из которой будет производится загрузка.
     */
    protected void loadConfig(CharSequence configStr) {
        params.clear();
        Pattern cfgLinePattern = Pattern.compile("\\s*([^=\\s]+)\\s*=\\s*([^;]*);");
        Matcher cfgMatch = cfgLinePattern.matcher(configStr);
        while (cfgMatch.find()) {
            params.put(cfgMatch.group(1), cfgMatch.group(2));
        }
    }
    /**
     * Функция получения значения параметра конфигурации робота.
     * @param parameterName Название параметра.
     * @return Значение параметра.
     */
    protected String getParameter(String parameterName) {
        if (params.containsKey(parameterName)) {
            return (String)(params.get(parameterName));
        }
        else {
            return "";
        }
    }
}

package ru.ifmo.rain.abdrashitov.elf;

/**
 * Автоматный класс управления соединением. Выполняется в потоке созданным классом
 * робота, после выполнения инициализации всех начальных частей робота (некоторые
 * части робота не являются начальными и подгружаются по мере их необходимости, к ним
 * относятся автоматы A1-A5).
 */
public class Connector extends Automaton {
    /**
```

```
 * Ссылка на сокет. Ссылка используется для управления соединением и опроса
 * переменных.
 */
private MSocket socket;
/**
 * Ссылка на МАД-адаптер. МАД-адаптер используется для выходного воздействия
 * и опроса переменных.
 */
private MAdapter adapter;
/**
 * Ссылка на робота. Ссылка используется для опроса переменных.
 */
private MBot bot;
/**
 * Автомат A1 "Перемещение". Создается и уничтожается при переходе
 * в определенные состояния.
 */
private Walk walk;
/**
  * Автомат A2 "Сон". Создается и уничтожается при переходе
  * в определенные состояния.
  */
private Rest rest;
/**
  * Автомат A3 "Битва". Создается и уничтожается при переходе
  * в определенные состояния.
  */
private Fight fight;
/**
  * Автомат A4 "Вода". Создается и уничтожается при переходе
  * в определенные состояния.
  */
private Water water;
/**
  * Автомат A5 "Еда". Создается и уничтожается при переходе
  * в определенные состояния.
  */
private Food food;
/**
 * Ссылка на поток, в который записывается лог. Используется для создания автоматов A1-A5.
 */
private Logger logger;
/**
 * Входная переменная x1. Переключатель активности робота.
 */
private boolean x1;
/**
 * Входная переменная x2. Индикатор активности сокета.
 */
private boolean x2;
/**
 * Сохраненное для лога сообщение e1.
 */
private boolean e1log;
/**
 * Сохраненное для лога сообщение e2.
 */
private boolean e2log;
/**
 * Сохраненное для лога сообщение e3.
 */
private boolean e3log;
/**
 * Сохраненное для лога сообщение e4.
 */
private boolean e4log;
/**
 * Сохраненное для лога сообщение e5.
 */
private boolean e5log;
/**
 * Выполнение установки необходимых ссылок и непосредственной подготовки
 * автомата к работе. Перед запуском автомата необходимо зарегестрировать его
 * во всех менеджерах событий, отслеживающих изменения входных переменных этого автомата.
```

```
 * @param bot Робот, которым производится управление.
 * @param socket Сокет связи с сервером.
 * @param adapter Адаптер правил конкретной версии игры.
 */
protected void init(MBot bot, MSocket socket, MAdapter adapter) {
    this.bot = bot;
    this.socket = socket;
    this.adapter = adapter;
    this.bot.addActiveListener(this);
    this.socket.addConnectedListener(this);
    this.socket.addConnectionErrorListener(this);
    this.adapter.addLoginDoneListener(this);
    this.adapter.addLoginErrorListener(this);
    this.adapter.addLogoutDoneListener(this);
    this.adapter.addLogoutErrorListener(this);
}
/**
 * Функция установки потока, в который будут выводиться логи. Поток лога обязательно должен
 * быть установлен перед запуском автомата.
 * @param logger Ссылка на поток лога.
 */
protected void setLogger(Logger logger) {
    this.logger = logger;
    super.setLogger(logger);
}
/**
 * Обновление значений входных переменных. Процедура получает информацию от источников
 * данных и вычисляет новые значения входных переменных.
 */
private void getInput() {
    x1 = bot.getActive();
    x2 = socket.getConnected();
    e1log = e1t();
    e2log = e2t();
    e3log = e3t();
    e4log = e4t();
    e5log = e5t();
}
/**
 * Получение сообщения e1. Сообщение об ошибке соединения.
 * @return Истина, если сообщение есть.
 */
private boolean e1() {
    if (e1t()) {
        socket.setConnectionError(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e1. Сообщение об ошибке соединения.
 * @return Истина, если сообщение есть.
 */
private boolean e1t() {
    return socket.getConnectionError();
}
/**
 * Получение сообщения e2. Сообщение о выполнении входа в игру.
 * @return Истина, если сообщение есть.
 */
private boolean e2() {
    if (e2t()) {
        adapter.setLoginDone(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e2. Сообщение о выполнении входа в игру.
 * @return Истина, если сообщение есть.
 */
private boolean e2t() {
```

```
    return adapter.getLoginDone();
}
/**
 * Получение сообщения e3. Сообщение об ошибке входа в игру.
 * @return Истина, если сообщение есть.
 */
private boolean e3() {
    if (e3t()) {
        adapter.setLoginError(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e3. Сообщение об ошибке входа в игру.
 * @return Истина, если сообщение есть.
 */
private boolean e3t() {
    return adapter.getLoginError();
}
/**
 * Получение сообщения e4. Сообщение о выполнении выхода из игры.
 * @return Истина, если сообщение есть.
 */
private boolean e4() {
    if (e4t()) {
        adapter.setLogoutDone(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e4. Сообщение о выполнении выхода из игры.
 * @return Истина, если сообщение есть.
 */
private boolean e4t() {
    return adapter.getLogoutDone();
}
/**
 * Получение сообщения e5. Сообщение об ошибке выхода из игры.
 * @return Истина, если сообщение есть.
 */
private boolean e5() {
    if (e5t()) {
        adapter.setLogoutError(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e5. Сообщение об ошибке выхода из игры.
 * @return Истина, если сообщение есть.
 */
private boolean e5t() {
    return adapter.getLogoutError();
}
/**
 * Выходное воздействие z1. Установление соединения.
 */
private void z1() {
    socket.connect();
}
/**
 * Выходное воздействие z2. Вход в систему.
 */
private void z2() {
    adapter.login();
}
/**
 * Выходное воздействие z3. Активация автоматов a1-a5.
 */
```

```
private void z3() {
    synchronized(this) {
        try {
            this.wait(1000); //wait for receiving all information from the server
        } catch (InterruptedException e) {
        }
    }
    walk = new Walk();
    rest = new Rest();
    fight = new Fight();
    food = new Food();
    water = new Water();
    walk.setLogger(logger);
    rest.setLogger(logger);
    fight.setLogger(logger);
    food.setLogger(logger);
    water.setLogger(logger);
    walk.init(adapter, rest, fight, water, food);
    rest.init(adapter, fight, walk, water, food);
    fight.init(adapter, rest);
    food.init(adapter, walk, rest);
    water.init(adapter, walk, rest);
    walk.start();
    rest.start();
    fight.start();
    food.start();
    water.start();
}
/**
 * Выходное воздействие z4. Выход из системы.
 */
private void z4() {
    adapter.logout();
    synchronized(this) {
        try {
            this.wait(1000);
        } catch (InterruptedException e) {
        }
    }
}
/**
 * Выходное воздействие z5. Уничтожение потоков для автоматов A1, A2, A3, A4, A5.
 */
private void z5() {
    walk.stop();
    rest.stop();
    fight.stop();
    food.stop();
    water.stop();
    walk = null;
    rest = null;
    fight = null;
    food = null;
    water = null;
}
/**
 * Переопределение процедуры остановки автомата. В дополнение к базовым действиям по остановке
 * автомата выполняется разрыв соединения с сервером.
 */
protected void stop() {
    super.stop();
    socket.disconnect();
}

protected String getName() {
    return "A0";
}
protected String getVariablesDump() {
    StringBuffer buf = new StringBuffer();
    if (x1) buf.append("x1=1"); else buf.append("x1=0");
    if (x2) buf.append(" x2=1"); else buf.append(" x2=0");
    if (e1log) buf.append(" e1");
    if (e2log) buf.append(" e2");
    if (e3log) buf.append(" e3");
```

```
    if (e4log) buf.append(" e4");
    if (e5log) buf.append(" e5");
    return buf.toString();
}
protected boolean nextStep() {
    boolean waitNeed = false;
    int y = getState();
    getInput();
    switch (y) {
        case 1:
            if (x1) {
                z1();
                setState(2);
            }
            else {
                waitNeed = true;
            }
            break;
        case 2:
            if (e1()) {
                setState(6);
            }
            else if (x2) {
                z2();
                setState(3);
            }
            else {
                waitNeed = true;
            }
            break;
        case 3:
            if (e3() || !x2) {
                setState(6);
            }
            else if (x2 && e2()) {
                z3();
                setState(4);
            }
            else {
                waitNeed = true;
            }
            break;
        case 4:
            if (!x2) {
                z5();
                setState(1);
            }
            else if (x2 && !x1) {
                z4();
                setState(5);
            }
            else {
                waitNeed = true;
            }
            break;
        case 5:
            if (e4() || !x2) {
                z5();
                setState(1);
            }
            else if (x2 && e5()) {
                z4();
                setState(5);
            }
            else {
                waitNeed = true;
            }
            break;
        case 6:
            if (!x1) {
                setState(1);
            }
            else {
                waitNeed = true;
```

```
                }
                break;
        }
        return !waitNeed;
    }
}

package ru.ifmo.rain.abdrashitov.elf;

/**
 * Автоматный класс "Битва". Создается автоматом управления соединением при переходе
 * в состояние игры. После инициализации для работы автомата создается отдельный поток.
 * Автомат управляет боевыми действиями персонажа в игровом мире. В качестве источника
 * входных воздействий выступает анализатор входного потока и автомат A2.
 */
public class Fight extends Automaton {
    /**
     * Ссылка на МАД-адаптер. МАД-адаптер используется для выходных воздействий (ведения
     * боя) и опроса переменных (результаты боя).
     */
    private MAdapter adapter;
    /**
     * Ссылка на автомат A2 "Сон".
     */
    private Rest rest;
    /**
     * Входная переменная x1. Индикатор низкого уровня жизни и запрета атаки.
     */
    private boolean x1;
    /**
     * Входная переменная x2. Индикатор наличия зарегестрированного противника.
     */
    private boolean x2;
    /**
     * Входная переменная x3. Индикатор боевых действий.
     */
    private boolean x3;
    /**
     * Входная переменная x6. Индикатор наличия магической энергии.
     */
    private boolean x6;
    /**
     * Входная переменная x7. Переменная запрещающая получения опыта.
     */
    private boolean x7;
    /**
     * Сохраненное для лога сообщение e1.
     */
    private boolean e1log;
    /**
     * Сохраненное для лога сообщение e2.
     */
    private boolean e2log;
    /**
     * Сохраненное для лога сообщение e3.
     */
    private boolean e3log;
    /**
     * Сохраненное для лога сообщение e4.
     */
    private boolean e4log;

    /**
     * Выполнение установки необходимых ссылок и непосредственной подготовки автомата к работе.
     * Перед запуском автомата необходимо зарегестрировать его во всех менеджерах событий,
     * отслеживающих изменения входных переменных этого автомата.
     * @param adapter Адаптер правил конкретной версии игры.
     * @param rest Ссылка на автомат A2 "Сон".
     */
    protected void init(MAdapter adapter, Rest rest) {
        this.adapter = adapter;
        this.rest = rest;
        rest.addStateListener(this);
        adapter.addLowHPListener(this);
```

```
    adapter.addIsMonsterHereListener(this);
    adapter.addIsInFightListener(this);
    adapter.addAttackErrorListener(this);
    adapter.addMonsterKilledListener(this);
    adapter.addLowManaListener(this);
    adapter.addTooMuchExpListener(this);
    adapter.addHandAttackDoneListener(this);
    adapter.addMagicAttackDoneListener(this);
}
/**
 * Обновление значений входных переменных. Процедура снимает информацию с источников
 * данных и вычисляет новые значения входных переменных.
 */
private void getInput() {
    int y2 = rest.getState();
    x1 = adapter.getLowHP() || (y2 == 7) || (y2 == 9) || (y2 == 5);
    x2 = adapter.getIsMonsterHere();
    x3 = adapter.getIsInFight();
    x6 = !adapter.getLowMana();
    x7 = adapter.getTooMuchExp();
    e1log = e1t();
    e2log = e2t();
    e3log = e3t();
    e4log = e4t();
}
/**
 * Получение сообщения e1. Сообщение об ошибке нападения на противника.
 * @return Истина, если сообщение есть.
 */
private boolean e1() {
    if (e1t()) {
        adapter.setAttackError(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e1. Сообщение об ошибке нападения на противника.
 * @return Истина, если сообщение есть.
 */
private boolean e1t() {
    return adapter.getAttackError();
}
/**
 * Получение сообщения e2. Сообщение о победе над противником.
 * @return Истина, если сообщение есть.
 */
private boolean e2() {
    if (e2t()) {
        adapter.setMonsterKilled(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e2. Сообщение о победе над противником.
 * @return Истина, если сообщение есть.
 */
private boolean e2t() {
    return adapter.getMonsterKilled();
}
/**
 * Получение сообщения e3. Сообщение о выполнении немагического удара.
 * @return Истина, если сообщение есть.
 */
private boolean e3() {
    if (e3t()) {
        adapter.setHandAttackDone(false);
        return true;
    } else {
        return false;
    }
```

```
}
/**
 * Проверка наличия сообщения e3. Сообщение о выполнении немагического удара.
 * @return Истина, если сообщение есть.
 */
private boolean e3t() {
    return adapter.getHandAttackDone();
}
//adapter.getMagicAttackDone();
/**
 * Получение сообщения e4. Сообщение о выполнении магического удара.
 * @return Истина, если сообщение есть.
 */
private boolean e4() {
    if (e4t()) {
        adapter.setMagicAttackDone(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e4. Сообщение о выполнении магического удара.
 * @return Истина, если сообщение есть.
 */
private boolean e4t() {
    return adapter.getMagicAttackDone();
}
/**
 * Выходное воздействие z1. Атака текущей цели.
 */
private void z1() {
    adapter.killMonster();
}
/**
 * Выходное воздействие z2. Нанесение рукопашного удара по противнику.
 */
private void z2() {
    adapter.doHandAttack();
}
/**
 * Выходное воздействие z3. Нанесение магического удара по противнику.
 */
private void z3() {
    adapter.doMagicAttack();
}

protected String getName() {
    return "A3";
}
protected String getVariablesDump() {
    StringBuffer buf = new StringBuffer();
    if (x1) buf.append("x1=1"); else buf.append("x1=0");
    if (x2) buf.append(" x2=1"); else buf.append(" x2=0");
    if (x3) buf.append(" x3=1"); else buf.append(" x3=0");
    if (x6) buf.append(" x6=1"); else buf.append(" x6=0");
    if (x7) buf.append(" x7=1"); else buf.append(" x7=0");
    if (e1log) buf.append(" e1");
    if (e2log) buf.append(" e2");
    if (e3log) buf.append(" e3");
    if (e4log) buf.append(" e4");
    return buf.toString();
}
protected boolean nextStep() {
    boolean waitNeed = false;
    int y = getState();
    getInput();
    switch (y) {
        case 1:
            if (x3) {
                setState(3);
            }
            else if (!x1 && !x3 && !x7) {
                setState(2);
```

```
                }
                else {
                    waitNeed = true;
                }
                break;
            case 2:
                if (x3) {
                    setState(3);
                }
                else if ((x1 && !x3 && !x7) || (x7 && !x3)) {
                    setState(1);
                }
                else if (!x1 && x2 && !x3 && !x7) {
                    setState(4);
                    z1();
                }
                else {
                    waitNeed = true;
                }
                break;
            case 3:
                setState(4);
                break;
            case 4:
                if (e1() || (!x3 && e2())) {
                    setState(2);
                }
                else if (x3 && !x6) {
                    z2();
                    setState(5);
                }
                else if (x3 && x6) {
                    z3();
                    setState(6);
                }
                else {
                    waitNeed = true;
                }
                break;
            case 5:
                if (e3()) {
                    setState(4);
                }
                else {
                    waitNeed = true;
                }
                break;
            case 6:
                if (e4()) {
                    setState(4);
                }
                else {
                    waitNeed = true;
                }
                break;
        }
        return !waitNeed;
    }
}

package ru.ifmo.rain.abdrashitov.elf;

/**
 * Автоматный класс "Еда". Создается автоматом управления соединением при переходе
 * в состояние игры. После инициализации для работы автомата создается отдельный поток.
 * Автомат управляет питанием персонажа в игровом мире. В качестве входных воздействий
 * выступают данные получаемые от анализатора входного потока и состояние автоматов A1-A2.
 */
public class Food extends Automaton {
    /**
     * Ссылка на МАД-адаптер. МАД-адаптер используется для выходного воздействия
     * и опроса переменных.
     */
    private MAdapter adapter;
```

```
/**
 * Ссылка на автомат A1 "Перемещение".
 */
private Walk walk;
/**
 * Ссылка на автомат A2 "Сон".
 */
private Rest rest;
/**
 * Входная переменная x1. Индикатор голода персонажа.
 */
private boolean x1;
/**
 * Входная переменная x2. Индикатор отсутствия еды у персонажа.
 */
private boolean x2;
/**
 * Входная переменная x3. Персонаж находится в продовольственном магазине.
 */
private boolean x3;
/**
 * Входная переменная x8. Персонаж спит.
 */
private boolean x8;
/**
 * Сохраненное для лога сообщение e1.
 */
private boolean e1log;
/**
 * Сохраненное для лога сообщение e2.
 */
private boolean e2log;
/**
 * Сохраненное для лога сообщение e3.
 */
private boolean e3log;
/**
 * Сохраненное для лога сообщение e4.
 */
private boolean e4log;
/**
 * Выполнение установки необходимых ссылок и непосредственной подготовки автомата к работе.
 * Перед запуском автомата необходимо зарегестрировать его во всех менеджерах событий,
 * отслеживающих изменения входных переменных этого автомата.
 * @param adapter Адаптер правил конкретной версии игры.
 * @param walk Ссылка на автомат A1 "Перемещение".
 * @param rest Ссылка на автомат A2 "Сон".
 */
protected void init(MAdapter adapter, Walk walk, Rest rest) {
    this.adapter = adapter;
    this.walk = walk;
    this.rest = rest;
    this.walk.addStateListener(this);
    this.rest.addStateListener(this);
    this.adapter.addFoodNeedListener(this);
    this.adapter.addOutOfFoodListener(this);
    this.adapter.addEatDoneListener(this);
    this.adapter.addEatErrorListener(this);
    this.adapter.addBuyFoodDoneListener(this);
    this.adapter.addBuyFoodErrorListener(this);
}
/**
 * Обновление значений входных переменных. Процедура снимает информацию с источников
 * данных и вычисляет новые значения входных переменных.
 */
private void getInput() {
    int y1 = walk.getState();
    int y2 = rest.getState();
    x1 = adapter.getFoodNeed();
    x2 = adapter.getOutOfFood();
    x3 = (y1 == 5);
    x8 = (y2 == 6) || (y2 == 7);
    e1log = e1t();
    e2log = e2t();
```

```
    e3log = e3t();
    e4log = e4t();
}
/**
 * Получение сообщения e1. Сообщение о выполнении действия {@link Food#z1()}.
 * @return Истина, если сообщение есть.
 */
private boolean e1() {
    if (e1t()) {
        adapter.setEatDone(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e1. Сообщение о выполнении действия {@link Food#z1()}.
 * @return Истина, если сообщение есть.
 */
private boolean e1t() {
    return adapter.getEatDone();
}
/**
 * Получение сообщения e2. Сообщение об ошибке выполнения действия {@link Food#z1()}.
 * @return Истина, если сообщение есть.
 */
private boolean e2() {
    if (e2t()) {
        adapter.setEatError(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e2. Сообщение об ошибке выполнения действия {@link Food#z1()}.
 * @return Истина, если сообщение есть.
 */
private boolean e2t() {
    return adapter.getEatError();
}
/**
 * Получение сообщения e3. Сообщение о выполнении действия {@link Food#z2()}.
 * @return Истина, если сообщение есть.
 */
private boolean e3() {
    if (e3t()) {
        adapter.setBuyFoodDone(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e3. Сообщение о выполнении действия {@link Food#z2()}.
 * @return Истина, если сообщение есть.
 */
private boolean e3t() {
    return adapter.getBuyFoodDone();
}
/**
 * Получение сообщения e4. Сообщение об ошибке выполнения действия {@link Food#z2()}.
 * @return Истина, если сообщение есть.
 */
private boolean e4() {
    if (e4t()) {
        adapter.setBuyFoodError(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e4. Сообщение об ошибке выполнения действия {@link Food#z2()}.
```

```
 * @return Истина, если сообщение есть.
 */
private boolean e4t() {
    return adapter.getBuyFoodError();
}
/**
 * Выходное воздействие z1. Поесть.
 */
private void z1() {
    adapter.doEat();
}
/**
 * Выходное воздействие z2. Пополнить запас еды.
 */
private void z2() {
    adapter.doBuyFood();
}
/**
 * Выходное воздействие z3. Сделать значительную паузу в выполнении автомата.
 */
private void z3() {
    synchronized(this) {
        try {
            this.wait(60000);
        } catch (InterruptedException e) {
        }
    }
}

protected String getName() {
    return "A5";
}
protected String getVariablesDump() {
    StringBuffer buf = new StringBuffer();
    if (x1) buf.append("x1=1"); else buf.append("x1=0");
    if (x2) buf.append(" x2=1"); else buf.append(" x2=0");
    if (x3) buf.append(" x3=1"); else buf.append(" x3=0");
    if (x8) buf.append(" x8=1"); else buf.append(" x8=0");
    if (e1log) buf.append(" e1");
    if (e2log) buf.append(" e2");
    if (e3log) buf.append(" e3");
    if (e4log) buf.append(" e4");
    return buf.toString();
}
protected boolean nextStep() {
    boolean waitNeed = false;
    int y = getState();
    getInput();
    switch (y) {
        case 1:
            if (x1) {
                setState(2);
            }
            else {
                waitNeed = true;
            }
            break;
        case 2:
            if (x2) {
                setState(3);
            }
            else if (!x2) {
                setState(6);
            }
            else {
                waitNeed = true;
            }
            break;
        case 3:
            if (x3) {
                setState(4);
            }
            else {
                waitNeed = true;
```

```
                }
                break;
            case 4:
                if (x2) {
                    z2();
                    setState(5);
                }
                else if (!x2) {
                    setState(2);
                }
                else {
                    waitNeed = true;
                }
                break;
            case 5:
                if (e4()) {
                    setState(8);
                }
                else if (e3()) {
                    setState(4);
                }
                else {
                    waitNeed = true;
                }
                break;
            case 6:
                if (x2) {
                    setState(3);
                }
                else if (!x1 && !x2) {
                    setState(1);
                }
                else if (x1 && !x2 && !x8) {
                    z1();
                    setState(7);
                }
                else {
                    waitNeed = true;
                }
                break;
            case 7:
                if (e2()) {
                    setState(1);
                    z3();
                }
                else if (e1()) {
                    setState(6);
                }
                else {
                    waitNeed = true;
                }
                break;
            case 8:
                waitNeed = true;
                break;
        }
        return !waitNeed;
    }
}
```

```
package ru.ifmo.rain.abdrashitov.elf;

/**
 * Интерфейс ведения лог файла и вывода игровой информации на экран.
 */
public interface Logger {
    /**
     * Вывод игровой информации на экран.
     * @param str Строка для вывода.
     */
    public void printToScreen(String str);
    /**
     * Вывод информации в лог.
     * @param str Строка для вывода.
```

```
     */
    public void printToLog(String str);
}

package ru.ifmo.rain.abdrashitov.elf;

/**
 * Абстрактный класс МАД-адаптера. Адаптер является промежуточным звеном между сервером
 * и автоматами роботов. Он преобразует текстовую информацию идущую от сервера в
 * значения входных переменных и сообщений для автоматов, согласно правилам конкретного МАДа.
 * Выходные воздействия автоматов преобразуются в текстовую форму и отсылаются
 * на сервер, в случаях когда это воздействие не относится к управлению локальными
 * обьектами.
 */
public abstract class MAdapter {
    /**
     * Ссылка на конфигурацию робота.
     */
    protected BotConfig config;
    /**
     * Ссылка на сокет, через который устанавливается соединение.
     */
    protected MSocket socket;
    /**
     * Менеджер событий.
     */
    private VariableNotifier notifier;

    /**
     * Флаг завершения входа в систему.
     */
    private boolean loginDone = false;
    /**
     * Идентификатор события изменения флага завершения входа в систему {@link MAdapter#loginDone}.
     */
    private Object loginDoneEvent = new Object();
    /**
     * Флаг ошибки входа в систему.
     */
    private boolean loginError = false;
    /**
     * Идентификатор события изменения флага ошибки входа в систему {@link MAdapter#loginError}.
     */
    private Object loginErrorEvent = new Object();
    /**
     * Флаг завершения выхода из системы.
     */
    private boolean logoutDone = false;
    /**
     * Идентификатор события изменения флага завершения выхода из системы {@link
MAdapter#logoutDone}.
     */
    private Object logoutDoneEvent = new Object();
    /**
     * Флаг ошибки выхода из системы.
     */
    private boolean logoutError = false;
    /**
     * Идентификатор события изменения флага ошибки выхода из системы {@link MAdapter#logoutError}.
     */
    private Object logoutErrorEvent = new Object();
    /**
     * Индикатор достижения цели перемещения.
     */
    private boolean targetReached = false;
    /**
     * Идентификатор события изменения индикатора достижения цели перемещения
     * {@link MAdapter#targetReached}.
     */
    private Object targetReachedEvent = new Object();
    /**
     * Флаг завершения шага по направлению к цели.
     */
    private boolean stepDone = false;
```

```
    /**
     * Идентификатор события изменения флага завершения шага по направлению к цели
     * {@link MAdapter#stepDone}.
     */
    private Object stepDoneEvent = new Object();
    /**
     * Индикатор низкого уровня жизненой силы.
     */
    private boolean lowHP = true;
    /**
     * Идентификатор события изменения нидикатора низкого уровня жизненой силы {@link
MAdapter#lowHP}.
     */
    private Object lowHPEvent = new Object();
    /**
     * Индикатор низкого уровня магической энергии.
     */
    private boolean lowMana = true;
    /**
     * Идентификатор события изменения индикатора низкого уровня магической энергии
     * {@link MAdapter#lowMana}.
     */
    private Object lowManaEvent = new Object();
    /**
     * Индикатор низкого уровня бодрости.
     */
    private boolean lowMove = true;
    /**
     * Идентификатор события изменения индикатора низкого уровня бодрости {@link MAdapter#lowMove}.
     */
    private Object lowMoveEvent = new Object();
    /**
     * Индикатор неполной жизненой силы.
     */
    private boolean notFullHP = true;
    /**
     * Идентификатор события изменения индикатора неполной жизненой силы {@link MAdapter#notFullHP}.
     */
    private Object notFullHPEvent = new Object();
    /**
     * Индикатор неполной магической энергии.
     */
    private boolean notFullMana = true;
    /**
     * Идентификатор события изменения индикатора неполной магической энергии
     * {@link MAdapter#notFullMana}.
     */
    private Object notFullManaEvent = new Object();
    /**
     * Индикатор неполной бодрости.
     */
    private boolean notFullMove = true;
    /**
     * Идентификатор события изменения индикатора неполной бодрости {@link MAdapter#notFullMove}.
     */
    private Object notFullMoveEvent = new Object();
    /**
     * Индикатор количества опыта.
     * Если количество опыта становится слишком большим, принимает значение истина.
     */
    private boolean tooMuchExp = true;
    /**
     * Идентификатор события изменения индикатора количества опыта {@link MAdapter#tooMuchExp}.
     */
    private Object tooMuchExpEvent = new Object();
    /**
     * Индикатор наличия в текущей комнате зарегестрированого противника.
     */
    private boolean isMonsterHere = false;
    /**
     * Идентификатор события соответствующего изменению {@link MAdapter#isMonsterHere}.
     */
    private Object isMonsterHereEvent = new Object();
    /**
```

```
 * Индикатор участия в бою.
 */
private boolean isInFight = false;
/**
 * Идентификатор события изменения идентификатора участия в бою {@link MAdapter#isInFight}.
 */
private Object isInFightEvent = new Object();
/**
 * Флаг ошибки нападения на противника.
 */
private boolean attackError = false;
/**
 * Идентификатор события изменения флага ошибки нападения на противника
 * {@link MAdapter#attackError}.
 */
private Object attackErrorEvent = new Object();
/**
 * Флаг отражающий факт убийства зарегестрированного противника.
 */
private boolean monsterKilled = false;
/**
 * Идентификатор события соответствующего флагу {@link MAdapter#monsterKilled}.
 */
private Object monsterKilledEvent = new Object();
/**
 * Флаг выполнения рукопашного удара по противнику.
 */
private boolean handAttackDone = false;
/**
 * Идентификатор события изменения флага {@link MAdapter#handAttackDone}.
 */
private Object handAttackDoneEvent = new Object();
/**
 * Флаг выполнения магического удара по противнику.
 */
private boolean magicAttackDone = false;
/**
 * Идентификатор события изменения флага {@link MAdapter#magicAttackDone}.
 */
private Object magicAttackDoneEvent = new Object();
/**
 * Индикатор взволнованости персонажа. Если персонаж взволнован или отравлен, он не может
 * полноценно восстанавливать свои силы, и это учитывается в управлении восстановлением
 * боеспособности персонажа.
 */
private boolean unrest = false;
/**
 * Идентификатор события изменения индикатора взволнованости персонажа {@link MAdapter#unrest}.
 */
private Object unrestEvent = new Object();
/**
 * Флаг подтверждения того, что персонаж лег спать.
 */
private boolean sleepDone = false;
/**
 * Идентификатор события изменения флага подтверждения сна {@link MAdapter#sleepDone}.
 */
private Object sleepDoneEvent = new Object();
/**
 * Флаг подтверждения того, что персонаж сел отдохнуть.
 */
private boolean restDone = false;
/**
 * Идентификатор события изменения флага подтверждения отдыха {@link MAdapter#restDone}.
 */
private Object restDoneEvent = new Object();
/**
 * Флаг подтверждения того, что пресонаж проснулся (если спал) и встал.
 */
private boolean wakeDone = false;
/**
 * Идентификатор события изменения флага подтверждения бодрствования {@link MAdapter#wakeDone}.
 */
private Object wakeDoneEvent = new Object();
```

```
    /**
     * Индикатор того, что в текущей комнате на персонажа не может никто напасть.
     *
    private boolean peace = false;/**/
    /**
     * Идентификатор события изменения индикатора мирного места.
     *
    private Object peaceEvent = new Object();/**/
    /**
     * Индикатор жажды персонажа.
     */
    private boolean drinkNeed = false;
    /**
     * Идентификатор события изменения индикатора жажды персонажа {@link MAdapter#drinkNeed}.
     */
    private Object drinkNeedEvent = new Object();
    /**
     * Индикатор отсутствия запасов воды.
     */
    private boolean outOfWater = false;
    /**
     * Идентификатор события изменения индикатора отсутствия запасов воды {@link
MAdapter#outOfWater}.
     */
    private Object outOfWaterEvent = new Object();
    /**
     * Флаг подтверждения выполнения команды {@link MAdapter#doDrink()}.
     */
    private boolean drinkDone = false;
    /**
     * Идентификатор события изменения флага {@link MAdapter#drinkDone}.
     */
    private Object drinkDoneEvent = new Object();
    /**
     * Флаг ошибки выполнения команды {@link MAdapter#doDrink()}.
     */
    private boolean drinkError = false;
    /**
     * Идентификатор события изменения флага {@link MAdapter#drinkError}.
     */
    private Object drinkErrorEvent = new Object();
    /**
     * Флаг подтверждения выполнения команды {@link MAdapter#doBuyDrink()}.
     */
    private boolean buyDrinkDone = false;
    /**
     * Идентификатор события изменения флага {@link MAdapter#buyDrinkDone}.
     */
    private Object buyDrinkDoneEvent = new Object();
    /**
     * Флаг ошибки выполнения команды {@link MAdapter#doBuyDrink()}.
     */
    private boolean buyDrinkError = false;
    /**
     * Идентификатор события изменения флага {@link MAdapter#buyDrinkError}.
     */
    private Object buyDrinkErrorEvent = new Object();
    /**
     * Индикатор голода персонажа.
     */
    private boolean foodNeed = false;
    /**
     * Идентификатор события изменения индикатора голода персонажа {@link MAdapter#foodNeed}.
     */
    private Object foodNeedEvent = new Object();
    /**
     * Индикатор отсутствия запасов еды.
     */
    private boolean outOfFood = false;
    /**
     * Идентификатор события изменения индикатора отсутствия запасов еды {@link MAdapter#outOfFood}.
     */
    private Object outOfFoodEvent = new Object();
    /**
```

```
 * Флаг подтверждения выполнения команды {@link MAdapter#doEat()}.
 */
private boolean eatDone = false;
/**
 * Идентификатор события изменения флага {@link MAdapter#eatDone}.
 */
private Object eatDoneEvent = new Object();
/**
 * Флаг ошибки выполнения команды {@link MAdapter#doEat()}.
 */
private boolean eatError = false;
/**
 * Идентификатор события изменения флага {@link MAdapter#eatError}.
 */
private Object eatErrorEvent = new Object();
/**
 * Флаг подтверждения выполнения команды {@link MAdapter#doBuyFood()}.
 */
private boolean buyFoodDone = false;
/**
 * Идентификатор события изменения флага {@link MAdapter#buyFoodDone}.
 */
private Object buyFoodDoneEvent = new Object();
/**
 * Флаг ошибки выполнения команды {@link MAdapter#doBuyFood()}.
 */
private boolean buyFoodError = false;
/**
 * Идентификатор события изменения флага {@link MAdapter#buyFoodError}.
 */
private Object buyFoodErrorEvent = new Object();

/**
 * Инициализирует обьект адаптера. При инициализации необходимо указать ссылку на
 * параметры настройки персонажа, для корректной дальнейшей работы с ним.
 * @param config Конфигурация робота.
 */
protected MAdapter(BotConfig config) {
    this.config = config;
    notifier = new VariableNotifier();
}
/**
 * Функция установки ссылки на сокет, через которое будет осуществлятся соединение.
 * Ссылку необходимо выставить до соединения с сервером.
 * @param socket Ссылка на сокет.
 */
protected void setSocket(MSocket socket) {
    this.socket = socket;
}

/**
 * Функция проверки завершения входа в систему.
 * @return Истина - вход выполнен, ложь - вход не выполнен.
 */
protected boolean getLoginDone() {
    return loginDone;
}
/**
 * Функция проверки наличия ошибок входа в систему.
 * @return Истина - ошибка присутствует, ложь - ошибка отсутствует.
 */
protected boolean getLoginError() {
    return loginError;
}
/**
 * Функция проверки завершения выхода из системы.
 * @return Истина - выход выполнен, ложь - выход не выполнен.
 */
protected boolean getLogoutDone() {
    return logoutDone;
}
/**
 * Функция проверки наличия ошибок выхода из системы.
 * @return Истина - ошибка присутствует, ложь - ошибка отсутствует.
```

```
 */
protected boolean getLogoutError() {
    return logoutError;
}
/**
 * Функция проверки достижения цели перемещения.
 * @return Истина - цель достигнута, ложь - цель не достигнута.
 */
protected boolean getTargetReached() {
    return targetReached;
}
/**
 * Функция проверки флага выполнения шага по направлению к цели.
 * @return Истина - шаг подтвержден, ложь - шаг еще не подтвержден.
 */
protected boolean getStepDone() {
    return stepDone;
}
/**
 * Функция проверки низкого уровня жизненой силы.
 * @return Истина - низкий уровень жизненой силы, ложь - высокий.
 */
protected boolean getLowHP() {
    return lowHP;
}
/**
 * Функция проверки низкого уровня магической энергии.
 * @return Истина - низкий уровень магической энергии, ложь - высокий.
 */
protected boolean getLowMana() {
    return lowMana;
}
/**
 * Функция проверки низкого уровня бодрости.
 * @return Истина - персонаж скоро не сможет двигаться, иначе ложь.
 */
protected boolean getLowMove() {
    return lowMove;
}
/**
 * Функция проверки количества жизненой силы.
 * @return Истина - неполная жизненая сила, ложь - максимум жизненой силы достигнут.
 */
protected boolean getNotFullHP() {
    return notFullHP;
}
/**
 * Функция проверки количества магической энергии.
 * @return Истина - неполная магическая энергия, ложь - достигнут максимум магической энергии.
 */
protected boolean getNotFullMana() {
    return notFullMana;
}
/**
 * Функция проверки количества единиц бодрости.
 * @return Истина - неполная бодрость, ложь - достигнут максимум единиц бодрости.
 */
protected boolean getNotFullMove() {
    return notFullMove;
}
/**
 * Функция проверки количества опыта необходимого до следующего уровня персонажа.
 * @return Истина - опыта слишком много, необходимо приостановить получение опыта,
 * ложь - нормальное количество опыта.
 */
protected boolean getTooMuchExp() {
    return tooMuchExp;
}
/**
 * Функция проверки наличия зарегестрированного противника в текущей комнате.
 * @return Истина - противник присутствует, ложь - отсутствует.
 */
protected boolean getIsMonsterHere() {
    return isMonsterHere;
```

```
}
/**
 * Функция проверки, находится ли персонаж в состоянии битвы.
 * @return Истина - персонаж дерется, ложь - битвы нет.
 */
protected boolean getIsInFight() {
    return isInFight;
}
/**
 * Функция возвращающая значение флага ошибки нападения на противника.
 * @return Истина - произошла ошибка, ложь - ошибки не было.
 */
protected boolean getAttackError() {
    return attackError;
}
/**
 * Функция возвращает значение флага, отражающего факт победы над противником.
 * @return Истина - победа, ложь - битва не начата или продолжается.
 */
protected boolean getMonsterKilled() {
    return monsterKilled;
}
/**
 * Функция возвращает значение флага ошибки рукопашного удара по противнику.
 * @return Истина - произошла ошибка, ложь - ошибки не было.
 */
protected boolean getHandAttackDone() {
    return handAttackDone;
}
/**
 * Функция возвращает значение флага ошибки магического удара по противнику.
 * @return Истина - произошла ошибка, ложь - ошибки не было.
 */
protected boolean getMagicAttackDone() {
    return magicAttackDone;
}
/**
 * Функция возвращает значение индикатора взволнованности персонажа {@link MAdapter#unrest}.
 * @return Истина - персонаж не сможет полноценно восстанавливать силы, ложь - персонаж спокоен.
 */
protected boolean getUnrest() {
    return unrest;
}
/**
 * Функция возвращает значение флага подтверждения того, что персонаж лег спать.
 * @return Истина - персонаж уснул, ложь - команда спать еще не выполнена.
 */
protected boolean getSleepDone() {
    return sleepDone;
}
/**
 * Функция возвращает значение флага подтверждения того, что персонаж сел отдохнуть.
 * @return Истина - персонаж сел отдохнуть, ложь - команда отдыхать еще не выполнена.
 */
protected boolean getRestDone() {
    return restDone;
}
/**
 * Функция возвращает значение флага подтверждения того, что персонаж встал.
 * @return Истина - персонаж проснулся и встал, ложь - команда встать еще не выполнена.
 */
protected boolean getWakeDone() {
    return wakeDone;
}
/**
 * Функция возвращает значение индикатора жажды персонажа.
 * @return Истина - персонаж испытывает жажду, ложь - иначе.
 */
protected boolean getDrinkNeed() {
    return drinkNeed;
}
/**
 * Функция возвращает значение индикатора отсутствия запасов воды у персонажа.
 * @return Истина - запасы воды исчерпаны, ложь - есть запас воды.
```

```
 */
protected boolean getOutOfWater() {
    return outOfWater;
}
/**
 * Функция возвращает значение флага выполнения команды пить.
 * @return Истина - подтверждение выполненя, ложь - команда еще не выполнена.
 */
protected boolean getDrinkDone() {
    return drinkDone;
}
/**
 * Функция возвращает значение флага ошибки выполнения команды пить.
 * @return Истина - произошла ошибка выполнения команды, ложь - ошибки нет.
 */
protected boolean getDrinkError() {
    return drinkError;
}
/**
 * Функция возвращает значение флага выполнения команды пополнить запас воды.
 * @return Истина - подтверждение выполненя, ложь - команда еще не выполнена.
 */
protected boolean getBuyDrinkDone() {
    return buyDrinkDone;
}
/**
 * Функция возвращает значение флага ошибки выполнения команды пополнить запас воды.
 * @return Истина - произошла ошибка выполнения команды, ложь - ошибки нет.
 */
protected boolean getBuyDrinkError() {
    return buyDrinkError;
}
/**
 * Функция возвращает значение индикатора голода персонажа.
 * @return Истина - персонаж испытывает голод, ложь - иначе.
 */
protected boolean getFoodNeed() {
    return foodNeed;
}
/**
 * Функция возвращает значение индикатора отсутствия запасов еды у персонажа.
 * @return Истина - запасы еды исчерпаны, ложь - есть запас еды.
 */
protected boolean getOutOfFood() {
    return outOfFood;
}
/**
 * Функция возвращает значение флага выполнения команды есть.
 * @return Истина - подтверждение выполненя, ложь - команда еще не выполнена.
 */
protected boolean getEatDone() {
    return eatDone;
}
/**
 * Функция возвращает значение флага ошибки выполнения команды есть.
 * @return Истина - произошла ошибка выполнения команды, ложь - ошибки нет.
 */
protected boolean getEatError() {
    return eatError;
}
/**
 * Функция возвращает значение флага выполнения команды пополнить запас еды.
 * @return Истина - подтверждение выполненя, ложь - команда еще не выполнена.
 */
protected boolean getBuyFoodDone() {
    return buyFoodDone;
}
/**
 * Функция возвращает значение флага ошибки выполнения команды пополнить запас еды.
 * @return Истина - произошла ошибка выполнения команды, ложь - ошибки нет.
 */
protected boolean getBuyFoodError() {
    return buyFoodError;
}
```

```
/**
 * Функция возвращает значение индикатора мирного места {@link MAdapter#peace}.
 * @return Истина - текущая комната является мирным местом, ложь - иначе.
 *
protected boolean getPeace() {
    return peace;
}/**/
/**
 * Функция добавления слушателя изменений флага завершения входа в систему.
 * @param listener Представитель слушателя.
 */
protected void addLoginDoneListener(Object listener) {
    notifier.addChangeListener(loginDoneEvent, listener);
}
/**
 * Функция добавления слушателя изменений флага ошибки входа в систему.
 * @param listener Представитель слушателя.
 */
protected void addLoginErrorListener(Object listener) {
    notifier.addChangeListener(loginErrorEvent, listener);
}
/**
 * Функция добавления слушателя изменений флага завершения выхода из системы.
 * @param listener Представитель слушателя.
 */
protected void addLogoutDoneListener(Object listener) {
    notifier.addChangeListener(logoutDoneEvent, listener);
}
/**
 * Функция добавления слушателя изменений флага ошибки выхода из системы.
 * @param listener Представитель слушателя.
 */
protected void addLogoutErrorListener(Object listener) {
    notifier.addChangeListener(logoutErrorEvent, listener);
}
/**
 * Функция добавления слушателя достижения цели при перемещении.
 * @param listener Представитель слушателя.
 */
protected void addTargetReachedListener(Object listener) {
    notifier.addChangeListener(targetReachedEvent, listener);
}
/**
 * Функция добавления слушателя выполнения шага по направлению к цели.
 * @param listener Представитель слушателя.
 */
protected void addStepDoneListener(Object listener) {
    notifier.addChangeListener(stepDoneEvent, listener);
}
/**
 * Функция добавления слушателя изменений индикатора низкого уровня жизненой силы.
 * @param listener Представитель слушателя.
 */
protected void addLowHPListener(Object listener) {
    notifier.addChangeListener(lowHPEvent, listener);
}
/**
 * Функция добавления слушателя изменений индикатора низкого уровня магической энергии.
 * @param listener Представитель слушателя.
 */
protected void addLowManaListener(Object listener) {
    notifier.addChangeListener(lowManaEvent, listener);
}
/**
 * Функция добавления слушателя изменений индикатора низкого уровня бодрости.
 * @param listener Представитель слушателя.
 */
protected void addLowMoveListener(Object listener) {
    notifier.addChangeListener(lowMoveEvent, listener);
}
/**
 * Функция добавления слушателя изменений индикатора неполной жизненой силы.
 * @param listener Представитель слушателя.
 */
```

```
protected void addNotFullHPListener(Object listener) {
    notifier.addChangeListener(notFullHPEvent, listener);
}
/**
 * Функция добавления слушателя изменений индикатора неполной магической энергии.
 * @param listener Представитель слушателя.
 */
protected void addNotFullManaListener(Object listener) {
    notifier.addChangeListener(notFullManaEvent, listener);
}
/**
 * Функция добавления слушателя изменений индикатора неполной бодрости.
 * @param listener Представитель слушателя.
 */
protected void addNotFullMoveListener(Object listener) {
    notifier.addChangeListener(notFullMoveEvent, listener);
}
/**
 * Функция добавления слушателя изменений индикатора количества очков опыта.
 * @param listener Представитель слушателя.
 */
protected void addTooMuchExpListener(Object listener) {
    notifier.addChangeListener(tooMuchExpEvent, listener);
}
/**
 * Функция добавления слушателя изменений индикатора присутствия противника.
 * @param listener Представитель слушателя.
 */
protected void addIsMonsterHereListener(Object listener) {
    notifier.addChangeListener(isMonsterHereEvent, listener);
}
/**
 * Функция добавления слушателя изменений индикатора участия персонажа в битве.
 * @param listener Представитель слушателя.
 */
protected void addIsInFightListener(Object listener) {
    notifier.addChangeListener(isInFightEvent, listener);
}
/**
 * Функция добавления слушателя изменений флага ошибки нападения.
 * @param listener Представитель слушателя.
 */
protected void addAttackErrorListener(Object listener) {
    notifier.addChangeListener(attackErrorEvent, listener);
}
/**
 * Функция добавления слушателя изменений флага, отражающего факт победы над противником.
 * @param listener Представитель слушателя.
 */
protected void addMonsterKilledListener(Object listener) {
    notifier.addChangeListener(monsterKilledEvent, listener);
}
/**
 * Функция добавления слушателя изменений флага выполнения рукопашного удара.
 * @param listener Представитель слушателя.
 */
protected void addHandAttackDoneListener(Object listener) {
    notifier.addChangeListener(handAttackDoneEvent, listener);
}
/**
 * Функция добавления слушателя изменений флага выполнения магического удара.
 * @param listener Представитель слушателя.
 */
protected void addMagicAttackDoneListener(Object listener) {
    notifier.addChangeListener(magicAttackDoneEvent, listener);
}
/**
 * Функция добавления слушателя изменений индикатора взволнованности персонажа.
 * @param listener Представитель слушателя.
 */
protected void addUnrestListener(Object listener) {
    notifier.addChangeListener(unrestEvent, listener);
}
/**
```

```
     * Функция добавления слушателя изменений флага выполнения команды спать.
     * @param listener Представитель слушателя.
     */
    protected void addSleepDoneListener(Object listener) {
        notifier.addChangeListener(sleepDoneEvent, listener);
    }
    /**
     * Функция добавления слушателя изменений флага выполнения команды отдыхать.
     * @param listener Представитель слушателя.
     */
    protected void addRestDoneListener(Object listener) {
        notifier.addChangeListener(restDoneEvent, listener);
    }
    /**
     * Функция добавления слушателя изменений флага выполнения команды встать.
     * @param listener Представитель слушателя.
     */
    protected void addWakeDoneListener(Object listener) {
        notifier.addChangeListener(wakeDoneEvent, listener);
    }
    /**
     * Функция добавления слушателя изменений индикатора жажды персонажа.
     * @param listener Представитель слушателя.
     */
    protected void addDrinkNeedListener(Object listener) {
        notifier.addChangeListener(drinkNeedEvent, listener);
    }
    /**
     * Функция добавления слушателя изменений индикатора отсутствия запаса воды.
     * @param listener Представитель слушателя.
     */
    protected void addOutOfWaterListener(Object listener) {
        notifier.addChangeListener(outOfWaterEvent, listener);
    }
    /**
     * Функция добавления слушателя изменений флага подтверждения выполнения команды пить.
     * @param listener Представитель слушателя.
     */
    protected void addDrinkDoneListener(Object listener) {
        notifier.addChangeListener(drinkDoneEvent, listener);
    }
    /**
     * Функция добавления слушателя изменений флага ошибки выполнения команды пить.
     * @param listener Представитель слушателя.
     */
    protected void addDrinkErrorListener(Object listener) {
        notifier.addChangeListener(drinkErrorEvent, listener);
    }
    /**
     * Функция добавления слушателя изменений флага подтверждения пополнения запаса воды.
     * @param listener Представитель слушателя.
     */
    protected void addBuyDrinkDoneListener(Object listener) {
        notifier.addChangeListener(buyDrinkDoneEvent, listener);
    }
    /**
     * Функция добавления слушателя изменений флага ошибки пополнения запаса воды.
     * @param listener Представитель слушателя.
     */
    protected void addBuyDrinkErrorListener(Object listener) {
        notifier.addChangeListener(buyDrinkErrorEvent, listener);
    }
    /**
     * Функция добавления слушателя изменений индикатора голода персонажа.
     * @param listener Представитель слушателя.
     */
    protected void addFoodNeedListener(Object listener) {
        notifier.addChangeListener(foodNeedEvent, listener);
    }
    /**
     * Функция добавления слушателя изменений индикатора отсутствия запаса еды.
     * @param listener Представитель слушателя.
     */
    protected void addOutOfFoodListener(Object listener) {
```

```
    notifier.addChangeListener(outOfFoodEvent, listener);
}
/**
 * Функция добавления слушателя изменений флага подтверждения выполнения команды есть.
 * @param listener Представитель слушателя.
 */
protected void addEatDoneListener(Object listener) {
    notifier.addChangeListener(eatDoneEvent, listener);
}
/**
 * Функция добавления слушателя изменений флага ошибки выполнения команды есть.
 * @param listener Представитель слушателя.
 */
protected void addEatErrorListener(Object listener) {
    notifier.addChangeListener(eatErrorEvent, listener);
}
/**
 * Функция добавления слушателя изменений флага подтверждения пополнения запаса еды.
 * @param listener Представитель слушателя.
 */
protected void addBuyFoodDoneListener(Object listener) {
    notifier.addChangeListener(buyFoodDoneEvent, listener);
}
/**
 * Функция добавления слушателя изменений флага ошибки пополнения запаса еды.
 * @param listener Представитель слушателя.
 */
protected void addBuyFoodErrorListener(Object listener) {
    notifier.addChangeListener(buyFoodErrorEvent, listener);
}
/**
 * Функция добавления слушателя изменений индикатора мирного места.
 * @param listener Представитель слушателя.
 *
protected void addPeaceListener(Object listener) {
    notifier.addChangeListener(peaceEvent, listener);
}/**/
/**
 * Функция установки флага завершения входа в систему.
 * @param loginDone Новое значение флага. Истина - вход выполнен,
 * ложь - вход не выполнен.
 */
protected void setLoginDone(boolean loginDone) {
    if (this.loginDone != loginDone) {
        this.loginDone = loginDone;
        notifier.eventHappened(loginDoneEvent);
    }
}
/**
 * Функция установки флага ошибки входа в систему.
 * @param loginError Новое значение флага. Истина - ошибка присутствует,
 * ложь - ошибка отсутствует.
 */
protected void setLoginError(boolean loginError) {
    if (this.loginError != loginError) {
        this.loginError = loginError;
        notifier.eventHappened(loginErrorEvent);
    }
}
/**
 * Функция установки флага завершения выхода из системы.
 * @param logoutDone Новое значение флага. Истина - выход выполнен,
 * ложь - выход не выполнен.
 */
protected void setLogoutDone(boolean logoutDone) {
    if (this.logoutDone != logoutDone) {
        this.logoutDone = logoutDone;
        notifier.eventHappened(logoutDoneEvent);
    }
}
/**
 * Функция установки флага ошибки выхода из системы.
 * @param logoutError Новое значение флага. Истина - ошибка присутствует,
 * ложь - ошибка отсутствует.
```

```
     */
    protected void setLogoutError(boolean logoutError) {
        if (this.logoutError != logoutError) {
            this.logoutError = logoutError;
            notifier.eventHappened(logoutErrorEvent);
        }
    }
    /**
     * Функция оповещения о нахождении у цели перемещения.
     * @param targetReached Истина - цель достигнута, ложь - цель не достигнута.
     */
    protected void setTargetReached(boolean targetReached) {
        if (this.targetReached != targetReached) {
            this.targetReached = targetReached;
            notifier.eventHappened(targetReachedEvent);
        }
    }
    /**
     * Функция установки флага выполнения шага по направлению к цели.
     * @param stepDone Новое значение флага. Истина - шаг выполнен, ложь - шаг не выполнен.
     */
    protected void setStepDone(boolean stepDone) {
        if (this.stepDone != stepDone) {
            this.stepDone = stepDone;
            notifier.eventHappened(stepDoneEvent);
        }
    }
    /**
     * Функция установки индикатора низкого уровня жизненой силы. Низкий уровень означает низкую
     * вероятность победы над противником из-за небольшого количества жизненой силы.
     * @param lowHP Новое значение индикатора. Истина - низкий уровень жизненой силы, иначе ложь.
     */
    protected void setLowHP(boolean lowHP) {
        if (this.lowHP != lowHP) {
            this.lowHP = lowHP;
            notifier.eventHappened(lowHPEvent);
        }
    }
    /**
     * Функция установки индикатора низкого уровня магической энергии.
     * Если персонаж не магической профессии, то этот флаг всегда должен иметь значение истина.
     * @param lowMana Новое значение индикатора. Истина - низкий уровень магической энергии, иначе
ложь.
     */
    protected void setLowMana(boolean lowMana) {
        if (this.lowMana != lowMana) {
            this.lowMana = lowMana;
            notifier.eventHappened(lowManaEvent);
        }
    }
    /**
     * Функция установки индикатора низкого уровня бодрости. Низкий уровень означает то, что
персонаж
     * не сможет двигаться втечении ближайших ходов.
     * @param lowMove Новое значение индикатора. Истина - низкий уровень бодрости, иначе ложь.
     */
    protected void setLowMove(boolean lowMove) {
        if (this.lowMove != lowMove) {
            this.lowMove = lowMove;
            notifier.eventHappened(lowMoveEvent);
        }
    }
    /**
     * Функция установки индикатора неполной жизненой силы.
     * @param notFullHP Новое значение индикатора. Истина - неполная жизненая сила, иначе ложь.
     */
    protected void setNotFullHP(boolean notFullHP) {
        if (this.notFullHP != notFullHP) {
            this.notFullHP = notFullHP;
            notifier.eventHappened(notFullHPEvent);
        }
    }
    /**
     * Функция установки индикатора неполной магической энергии.
```

```
     * @param notFullMana Новое значение индикатора. Истина - неполная магическая энергия, иначе
ложь.
     */
    protected void setNotFullMana(boolean notFullMana) {
        if (this.notFullMana != notFullMana) {
            this.notFullMana = notFullMana;
            notifier.eventHappened(notFullManaEvent);
        }
    }
    /**
     * Функция установки индикатора неполной бодрости.
     * @param notFullMove Новое значение индикатора. Истина - неполная бодрость, иначе ложь.
     */
    protected void setNotFullMove(boolean notFullMove) {
        if (this.notFullMove != notFullMove) {
            this.notFullMove = notFullMove;
            notifier.eventHappened(notFullMoveEvent);
        }
    }
    /**
     * Функция установки значения индикатора превышеного опыта.
     * @param tooMuchExp Истина - опыта слишком много, необходимо приостановить получение опыта,
     * ложь - нормальное количество опыта.
     */
    protected void setTooMuchExp(boolean tooMuchExp) {
        if (this.tooMuchExp != tooMuchExp) {
            this.tooMuchExp = tooMuchExp;
            notifier.eventHappened(tooMuchExpEvent);
        }
    }
    /**
     * Функция установки значения индикатора наличия зарегестрированного противника
     * в текущей комнате.
     * @param isMonsterHere Истина - противник присутствует, ложь - нет противника.
     */
    protected void setIsMonsterHere(boolean isMonsterHere) {
        if (this.isMonsterHere != isMonsterHere) {
            this.isMonsterHere = isMonsterHere;
            notifier.eventHappened(isMonsterHereEvent);
        }
    }
    /**
     * Функция установки значения индикатора нахождения персонажа в битве.
     * @param isInFight Истина - персонаж сражается, ложь - нет битвы.
     */
    protected void setIsInFight(boolean isInFight) {
        if (this.isInFight != isInFight) {
            this.isInFight = isInFight;
            notifier.eventHappened(isInFightEvent);
        }
    }
    /**
     * Функция установки значения флага ошибки нападения на противника.
     * @param attackError Истина - произошла ошибка, ложь - ошибки не было.
     */
    protected void setAttackError(boolean attackError) {
        if (this.attackError != attackError) {
            this.attackError = attackError;
            notifier.eventHappened(attackErrorEvent);
        }
    }
    /**
     * Функция установки значения флага, отражающего факт победы над противником.
     * Победой считается не только убийство противника, но также его капитуляция.
     * @param monsterKilled Истина - противник побежден, ложь - битва не началась или еще
продолжается.
     */
    protected void setMonsterKilled(boolean monsterKilled) {
        if (this.monsterKilled != monsterKilled) {
            this.monsterKilled = monsterKilled;
            notifier.eventHappened(monsterKilledEvent);
        }
    }
    /**
```

```
 * Функция установки значения флага подтверждения рукопашного удара по противнику.
 * @param handAttackDone Истина - удар выполнен, ложь - удар не подтвержден.
 */
protected void setHandAttackDone(boolean handAttackDone) {
    if (this.handAttackDone != handAttackDone) {
        this.handAttackDone = handAttackDone;
        notifier.eventHappened(handAttackDoneEvent);
    }
}
/**
 * Функция установки значения флага подтверждения магического удара по противнику.
 * @param magicAttackDone Ислина - удар выполнен, ложь - удар не подтвержден.
 */
protected void setMagicAttackDone(boolean magicAttackDone) {
    if (this.magicAttackDone != magicAttackDone) {
        this.magicAttackDone = magicAttackDone;
        notifier.eventHappened(magicAttackDoneEvent);
    }
}
/**
 * Функция установки значения индикатора взволнованности персонажа.
 * @param unrest Ислина - персонаж не может в полной мере восстанавливать силы,
 * ложь - персонаж спокоен.
 */
protected void setUnrest(boolean unrest) {
    if (this.unrest != unrest) {
        this.unrest = unrest;
        notifier.eventHappened(unrestEvent);
    }
}
/**
 * Функция установки значения флага подтверждения выполнения команды спать.
 * @param sleepDone Ислина - персонаж лег спать, ложь - персонаж стоит.
 */
protected void setSleepDone(boolean sleepDone) {
    if (this.sleepDone != sleepDone) {
        this.sleepDone = sleepDone;
        notifier.eventHappened(sleepDoneEvent);
    }
}
/**
 * Функция установки значения флага подтверждения выполнения команды отдыхать.
 * @param restDone Ислина - персонаж сел отдохнуть, ложь - персонаж стоит.
 */
protected void setRestDone(boolean restDone) {
    if (this.restDone != restDone) {
        this.restDone = restDone;
        notifier.eventHappened(restDoneEvent);
    }
}
/**
 * Функция установки значения флага подтверждения выполнения команды встать.
 * @param wakeDone Ислина - персонаж встал, ложь - персонаж отдыхает.
 */
protected void setWakeDone(boolean wakeDone) {
    if (this.wakeDone != wakeDone) {
        this.wakeDone = wakeDone;
        notifier.eventHappened(wakeDoneEvent);
    }
}
/**
 * Функция установки значения индикатора жажды персонажа.
 * @param drinkNeed Ислина - персонаж хочет пить, ложь - персонаж не испытывает жажды.
 */
protected void setDrinkNeed(boolean drinkNeed) {
    if (this.drinkNeed != drinkNeed) {
        this.drinkNeed = drinkNeed;
        notifier.eventHappened(drinkNeedEvent);
    }
}
/**
 * Функция установки значения индикатора отсутствия запаса воды.
 * @param outOfWater Ислина - запас воды иссяк, ложь - у персонажа есть запас воды.
 */
```

```
protected void setOutOfWater(boolean outOfWater) {
    if (this.outOfWater != outOfWater) {
        this.outOfWater = outOfWater;
        notifier.eventHappened(outOfWaterEvent);
    }
}
/**
 * Функция установки значения флага подтверждения выполнения команды пить.
 * @param drinkDone Ислина - выполнение команды подтверждено, ложь - команда еще не выполнена.
 */
protected void setDrinkDone(boolean drinkDone) {
    if (this.drinkDone != drinkDone) {
        this.drinkDone = drinkDone;
        notifier.eventHappened(drinkDoneEvent);
    }
}
/**
 * Функция установки значения флага ошибки выполнения команды пить.
 * @param drinkError Ислина - выполнение команды неудачно, ложь - ошибок не происходило.
 */
protected void setDrinkError(boolean drinkError) {
    if (this.drinkError != drinkError) {
        this.drinkError = drinkError;
        notifier.eventHappened(drinkErrorEvent);
    }
}
/**
 * Функция установки значения флага подтверждения пополнения запаса воды.
 * @param buyDrinkDone Ислина - запас воды пополнен, ложь - команда еще не выполнена.
 */
protected void setBuyDrinkDone(boolean buyDrinkDone) {
    if (this.buyDrinkDone != buyDrinkDone) {
        this.buyDrinkDone = buyDrinkDone;
        notifier.eventHappened(buyDrinkDoneEvent);
    }
}
/**
 * Функция установки значения флага ошибки пополнения запаса воды.
 * @param buyDrinkError Ислина - произошла ошибка, ложь - ошибок не происходило.
 */
protected void setBuyDrinkError(boolean buyDrinkError) {
    if (this.buyDrinkError != buyDrinkError) {
        this.buyDrinkError = buyDrinkError;
        notifier.eventHappened(buyDrinkErrorEvent);
    }
}
/**
 * Функция установки значения индикатора голода персонажа.
 * @param foodNeed Ислина - персонаж хочет есть, ложь - персонаж не испытывает голода.
 */
protected void setFoodNeed(boolean foodNeed) {
    if (this.foodNeed != foodNeed) {
        this.foodNeed = foodNeed;
        notifier.eventHappened(foodNeedEvent);
    }
}
/**
 * Функция установки значения индикатора отсутствия запаса еды.
 * @param outOfFood Ислина - запас еды иссяк, ложь - у персонажа есть запас еды.
 */
protected void setOutOfFood(boolean outOfFood) {
    if (this.outOfFood != outOfFood) {
        this.outOfFood = outOfFood;
        notifier.eventHappened(outOfFoodEvent);
    }
}
/**
 * Функция установки значения флага подтверждения выполнения команды есть.
 * @param eatDone Ислина - выполнение команды подтверждено, ложь - команда еще не выполнена.
 */
protected void setEatDone(boolean eatDone) {
    if (this.eatDone != eatDone) {
        this.eatDone = eatDone;
        notifier.eventHappened(eatDoneEvent);
```

```
    }
}
/**
 * Функция установки значения флага ошибки выполнения команды есть.
 * @param eatError Ислина - выполнение команды неудачно, ложь - ошибок не происходило.
 */
protected void setEatError(boolean eatError) {
    if (this.eatError != eatError) {
        this.eatError = eatError;
        notifier.eventHappened(eatErrorEvent);
    }
}
/**
 * Функция установки значения флага подтверждения пополнения запаса еды.
 * @param buyFoodDone Ислина - запас еды пополнен, ложь - команда еще не выполнена.
 */
protected void setBuyFoodDone(boolean buyFoodDone) {
    if (this.buyFoodDone != buyFoodDone) {
        this.buyFoodDone = buyFoodDone;
        notifier.eventHappened(buyFoodDoneEvent);
    }
}
/**
 * Функция установки значения флага ошибки пополнения запаса еды.
 * @param buyFoodError Ислина - произошла ошибка, ложь - ошибок не происходило.
 */
protected void setBuyFoodError(boolean buyFoodError) {
    if (this.buyFoodError != buyFoodError) {
        this.buyFoodError = buyFoodError;
        notifier.eventHappened(buyFoodErrorEvent);
    }
}
/**
 * Функция установки значения индикатора мирного места.
 * @param peace Ислина - текущая комната является мирным местом, ложь - иначе.
 *
protected void setPeace(boolean peace) {
    if (this.peace != peace) {
        this.peace = peace;
        notifier.eventHappened(peaceEvent);
    }
}/**/
/**
 * Процедура входа в систему. Для входа в систему серверу должны быть отосланы
 * соответствующие команды, обычно содержащие логин и пароль к игровому персонажу.
 * Также необходимо обнулить соответствующие переменные.
 */
abstract protected void login();
/**
 * Процедура выхода из системы. Для выхода из системы серверу должна быть отослана
 * соответствующая команда, зависящая от конкретной версии правил. Перед выполнением
 * входа необходимо обнулить соответствующие переменные.
 */
abstract protected void logout();
/**
 * Процедура построения пути к мирной комнате. После выполнения процедуры автоматами
 * робота может быть вызвана процедура {@link MAdapter#doStepToTarget()},
 * в результате чего должен быть сделан шаг по направлению к мирной комнате.
 */
abstract protected void setTargetToPeacefulPlace();
/**
 * Процедура построения пути к противнику. После выполнения процедуры автоматами
 * робота может быть вызвана процедура {@link MAdapter#doStepToTarget()},
 * в результате чего должен быть сделан шаг по направлению к противнику.
 * Желательно при построении пути до противника в качестве цели
 * выбирать случайного противника из заранее известного набора доступных
 * для персонажа противников.
 */
abstract protected void setTargetToBattlePlace();
/**
 * Процедура построения пути к магазину еды. После выполнения процедуры автоматами
 * робота может быть вызвана процедура {@link MAdapter#doStepToTarget()},
 * в результате чего должен быть сделан шаг по направлению к магазину,
 * в котором можно будет приобрести еду.
```

```
     */
    abstract protected void setTargetToShop();
    /**
     * Процедура построения пути к роднику. После выполнения процедуры автоматами
     * робота может быть вызвана процедура {@link MAdapter#doStepToTarget()},
     * в результате чего должен быть сделан шаг по направлению к месту,
     * где можно будет набрать воду в контейнер.
     */
    abstract protected void setTargetToSpring();
    /**
     * Процедура выполнения шага по направлению к выбраной цели. Вызовами процедур
     * {@link MAdapter#setTargetToBattlePlace()}, {@link MAdapter#setTargetToPeacefulPlace()},
     * {@link MAdapter#setTargetToShop()} и {@link MAdapter#setTargetToSpring()}
     * устанавливается цель перемещения. После чего каждый шаг должен выполнятся по
     * направлению к установленой цели.
     */
    abstract protected void doStepToTarget();
    /**
     * Процедура нападения на зарегестрированного противника, находящегося в одной комнате с
персонажем.
     * После нападения бой ведется немагическими {@link MAdapter#doHandAttack()}
     * и магическими ударами {@link MAdapter#doMagicAttack()}. Процедура должна обнулять флаги
     * ошибки нападения {@link MAdapter#attackError} и флага победы над противником
     * {@link MAdapter#monsterKilled}.
     */
    abstract protected void killMonster();
    /**
     * Процедура рукопашного удара по противнику. Процедура должна отослать на сервер команду,
     * соответствующую рукопашному удару в битве, зависящую от конкретной версии правил сервера и
     * параметров персонажа (расы и профессии). Например на Аладоне воин достаточного уровня
     * может выполнять команду "выпад". Процедура должна обнулять флаг подтверждения рукопашного
     * удара {@link MAdapter#handAttackDone}. Для персонажа, неспособного наносить сложные
     * немагические удары, процедура должна делать паузу и не выполнять никаких действий. А флаг
     * подтверждения всегда должен быть установлен в значение истина.
     */
    abstract protected void doHandAttack();
    /**
     * Процедура магического удара по противнику. Процедура должна отослать на сервер команду,
     * соответствующую магическому удару в битве, зависящую от конкретной версии правил сервера и
     * параметров персонажа (расы и профессии). Например на Аладоне маг может выполнять заклинание
     * "магический снаряад". Процедура должна обнулять флаг подтверждения магического
     * удара {@link MAdapter#magicAttackDone}. Для немагического персонажа не вызывается.
     */
    abstract protected void doMagicAttack();
    /**
     * Процедура посылает команду спать персонажу. Во время сна жизненая сила, магическая энергия
     * и единицы бодрости восстанавливаются значительно быстрее, чем в бодрствующем состоянии.
     * Текстовое представление команды зависит от правил конкретного сервера. Для отсылки команды
     * на сервер используется метод {@link MSocket#sendText(String)}. Процедура должна
     * устанавливать флаг {@link MAdapter#sleepDone} в значение ложь.
     */
    abstract protected void doSleep();
    /**
     * Процедура посылает команду отдыхать персонажу. Во время отдыха жизненая сила, магическая
энергия
     * и единицы бодрости восстанавливаются медленнее, чем в состоянии сна.
     * Текстовое представление команды зависит от правил конкретного сервера. Для отсылки команды
     * на сервер используется метод {@link MSocket#sendText(String)}. Процедура должна
     * устанавливать флаг {@link MAdapter#restDone} в значение ложь.
     */
    abstract protected void doRest();
    /**
     * Процедура посылает команду встать персонажу. В результате команды персонаж должен проснуться,
     * если спал, и встать. Текстовое представление команды зависит от правил конкретного сервера.
     * Для отсылки команды на сервер используется метод {@link MSocket#sendText(String)}.
     * Процедура должна устанавливать флаг {@link MAdapter#wakeDone} в значение ложь.
     */
    abstract protected void doWake();
    /**
     * Процедура посылает команду пить персонажу. В результате команды персонаж должен глотнуть
     * воды из запасов. Текстовое представление команды зависит от правил конкретного сервера.
     * Для отсылки команды на сервер используется метод {@link MSocket#sendText(String)}.
     * Процедура должна устанавливать флаги {@link MAdapter#drinkDone} и {@link MAdapter#drinkError}
     * в значение ложь.
```

```
     */
    abstract protected void doDrink();
    /**
     * Процедура заставляет персонаж пополнить запас воды. В результате команды персонаж должен
наполнить
     * емкость для воды из источника или купить уже наполненную емкость. Текстовое представление
команды
     * зависит от правил конкретного сервера. Для отсылки команды на сервер используется метод
     * {@link MSocket#sendText(String)}. Процедура должна устанавливать флаги
     * {@link MAdapter#buyDrinkDone} и {@link MAdapter#buyDrinkError} в значение ложь.
     */
    abstract protected void doBuyDrink();
    /**
     * Процедура посылает команду есть персонажу. В результате команды персонаж должен есть пищу
     * из своих запасов. Текстовое представление команды зависит от правил конкретного сервера.
     * Для отсылки команды на сервер используется метод {@link MSocket#sendText(String)}.
     * Процедура должна устанавливать флаги {@link MAdapter#eatDone} и {@link MAdapter#eatError}
     * в значение ложь.
     */
    abstract protected void doEat();
    /**
     * Процедура заставляет персонаж пополнить запас еды. В результате команды персонаж должен
купить
     * некоторое количество еды. Текстовое представление команды зависит от правил конкретного
сервера.
     * Для отсылки команды на сервер используется метод {@link MSocket#sendText(String)}.
     * Процедура должна устанавливать флаги {@link MAdapter#buyFoodDone} и
     * {@link MAdapter#buyFoodError} в значение ложь.
     */
    abstract protected void doBuyFood();
    /**
     * Процедура обработки входящего трафика. Когда от сервера приходит строка, она
     * принимается обьектом {@link MSocket} и перенаправляется в данную процедуру.
     * Процедура должна производить анализ данных, которые можно извлечь из строки
     * с учетом особенностей конкретного МАДа. Процедура должна поддерживать в
     * корректном состоянии все флаги и индикаторы адаптера.
     * @param inputStr Строка для обработки.
     */
    abstract protected void procesLine(String inputStr);
}

package ru.ifmo.rain.abdrashitov.elf;

import ru.ifmo.rain.abdrashitov.elf.gui.MainForm;

import java.io.*;

/**
 * Класс робота. Выполняет загрузку настроек персонажа и выбор соответствующего
 * адаптера. При создании класс создает экземпляры классов управления соединением,
 * сокета и адаптера. Вновь созданному сокету передаются параметры соединения.
 * Вновь созданному адаптеру передаются настройки персонажа. После установки
 * параметров для класса сокет и автомата управления соединением создаются
 * отдельные потоки. Класс предоставляет доступ к управлению активностью робота.
 */
public class MBot {
    /**
     * Менеджер событий.
     */
    private VariableNotifier notifier;
    /**
     * Флаг активности робота. Истина - робот включен, ложь - робот выключен.
     */
    private boolean active = false;
    /**
     * Идентификатор события изменения состояния активности.
     */
    private Object activeEvent = new Object();
    /**
     * Ссылка на созданый автомат А0 "Управление соединением".
     */
    private Connector connector;

    /**
```

```
 * Инициализирует робота конфигурацией загруженной из строкового представления.
 * @param configStr Строковое представление конфигурации.
 * @param logger Лог робота.
 */
public MBot(CharSequence configStr, Logger logger) {
    notifier = new VariableNotifier();
    BotConfig config = new BotConfig(configStr);
    MAdapter adapter;
    if (((String)config.getParameter("adapter")).equals("SampleAdapter")) {
        adapter = new SampleAdapter(config);
    }
    else {
        adapter = new VoidAdapter(config);
    }
    MSocket socket = new MSocket(config);
    adapter.setSocket(socket);
    socket.setAdapter(adapter);
    socket.setLogger(logger);
    connector = new Connector();
    connector.setLogger(logger);
    connector.init(this, socket, adapter);

    //--------------
    connector.start();
}

/**
 * Функция возвращает состояние активности робота.
 * @return Состояние активности робота. Истина - робот включен,
 * @return ложь - робот выключен.
 */
protected boolean getActive() {
    return active;
}
/**
 * Установка значения активности робота.
 * @param active Новое состояние активности. Истина - робот включен, ложь - выключен.
 */
private void setActive(boolean active) {
    if (this.active != active) {
        this.active = active;
        notifier.eventHappened(activeEvent);
    }
}
/**
 * Функция добавления слушателя изменений активности робота.
 * @param listener Представитель слушателя.
 */
protected void addActiveListener(Object listener) {
    notifier.addChangeListener(activeEvent, listener);
}
/**
 * Функция включения робота. После включения робот установит соединение с сервером,
 * выполнит вход, определит положение персонажа и начнет действовать.
 */
public void startRobot() {
    setActive(true);
}
/**
 * Функция выключения робота. После выключения робот завершит текущие действия,
 * возможно изменит сонфигурацию (сохранит положение) и отключится от сервера.
 */
public void stopRobot() {
    setActive(false);
}
/**
 * Функция уничтожения робота. Если не вызвать функцию уничтожения робота, не будет
 * остановлен автомат A0 "Управление соединением" и его поток продолжит использование
 * системных ресурсов.
 */
public void disposeRobot() {
    setActive(false);
    connector.addStateListener(this);
    synchronized(this) {
```

```
            while (connector.getState() != 1) {
                try {
                    this.wait();
                } catch (InterruptedException e) {
                }
            }
        }
        connector.stop();
    }

    /**
     * Процедура для тестового запуска робота.
     * @param args Параметры запуска игнорируются.
     */
    public static void main(String[] args) {
        if (args.length > 0) {
            final PrintStream log;
            if (args.length > 1) {
                PrintStream nps = System.out;
                try {
                    nps = new PrintStream(new FileOutputStream(args[1]), true);
                } catch (FileNotFoundException e) {
                    System.out.println("Error: FileNotFoundException while creating log file.");
                }
                log = nps;
            } else {
                log = System.out;
            }
            BufferedReader cfgfile = null;
            try {
                cfgfile = new BufferedReader(new FileReader(args[0]));
            } catch (FileNotFoundException e) {
                System.out.println("Error: FileNotFoundException while loading configuration
file.");
                System.exit(0);
            }
            StringBuffer configStr = new StringBuffer();
            while (true) {
                try {
                    String str = cfgfile.readLine();
                    if (str == null) {
                        break;
                    }
                    configStr.append(str);
                } catch (IOException e) {
                    System.out.println("Error: IOException while loading configuration file.");
                    System.exit(0);
                }
            }
            MBot bot = new MBot(configStr, new Logger() {
                public void printToScreen(String str) {}
                public void printToLog(String str) {
                    log.println(str);
                }
            });
            bot.startRobot();
            BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
            while (true) {
                try {
                    String command = input.readLine();
                    if (command == null) {
                        break;
                    }
                    else if (command.equals("quit")) {
                        break;
                    }
                } catch (IOException e) {
                }
            }
            bot.stopRobot();
            bot.disposeRobot();
        }
        else {
            System.out.println("Starting graphical version configured for local demo-server.");
```

```
            System.out.println();
            System.out.println("To start console version: MUDBot <config> [<log>]");
            System.out.println("<config> - file that contains robot configuration");
            System.out.println("<log> - log file");
            new MainForm().setVisible(true);
        }
    }
}
```

```
package ru.ifmo.rain.abdrashitov.elf;

import java.io.*;
import java.net.InetSocketAddress;
import java.net.Socket;

/**
 * Класс обеспечивающий соединение с МАД сервером. Для чтения из соединения создается
 * отдельный поток. При получении данных вызывается их обработчик {@link MAdapter}.
 * Также класс предоставляет функцию отправки данных на сервер.
 */
public class MSocket implements Runnable {
    /**
     * Ссылка на текущий обработчик входящего трафика.
     */
    private MAdapter adapter;
    /**
     * TCP/IP сокет выполняющий связь с сервером.
     */
    private Socket socket;
    /**
     * Выходной канал TCP/IP сокета. Через него производится весь вывод на сервер.
     */
    private OutputStreamWriter socketOut;
    /**
     * Конфигурация бота. Из конфигурации берутся параметры host и port для установки
     * соединения с сервером по адресу host и порту port.
     */
    private BotConfig config;
    /**
     * Менеджер событий.
     */
    private VariableNotifier notifier;
    /**
     * Текстовый лог.
     */
    private Logger logger;
    /**
     * Флаг активности соединения с сервером.
     */
    private boolean connected = false;
    /**
     * Идентификатор события изменения активности сокета.
     */
    private Object connectedEvent = new Object();
    /**
     * Флаг наличия ошибок соединения с сервером.
     */
    private boolean connectionError = false;
    /**
     * Идентификатор события изменения флага наличия ошибок соединения.
     */
    private Object connectionErrorEvent = new Object();

    /**
     * Инициализирует сокет.
     * @param config Ссылка на конфигурацию робота, из которой будет взят
     * адрес MUD сервера.
     */
    protected MSocket(BotConfig config) {
        this.config = config;
        notifier = new VariableNotifier();
    }

    /**
```

```
     * Устанавливает ссылку на обработчик входящего трафика. При получении данных от
     * сервера, они будут перенаправлены на обработку адаптером, настроенным на
     * работу с конкретным МАДом. Ссылку на адаптер необходимо выставить до соединения
     * с сервером.
     * @param adapter Ссылка на адаптер.
     */
    protected void setAdapter(MAdapter adapter) {
        this.adapter = adapter;
    }
    /**
     * Функция установки потока, в который будут выводиться логи. Поток лога обязательно должен
     * быть установлен перед началом работы с сокетом.
     * @param logger Ссылка на лог.
     */
    protected void setLogger(Logger logger) {
        this.logger = logger;
    }
    /**
     * Процедура начала отслеживания входного потока. Выполняется в отдельном потоке.
     * Процедуру необходимо вызвать путем создания нового потока сразу после установки
     * соединения. При разрыве связи с сервером выполнение процедуры прекращается
     * и поток уничтожается.
     */
    public void run() {
        String inputStr;
        try {
            BufferedReader reader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            while (true) {
                inputStr = reader.readLine();
                if (inputStr == null) {
                    break;
                }
                if (inputStr.length() != 0) {
                    if (logger != null) {
                        logger.printToLog("Получена строка: \"" + inputStr + "\"");
                        logger.printToScreen(inputStr);
                    }
                    adapter.procesLine(inputStr);
                }
            }
        } catch (IOException e) {
        }
        setConnected(false);
    }
    /**
     * Функция добавления слушателя изменений активности соединения с сервером.
     * @param listener Представитель слушателя.
     */
    protected void addConnectedListener(Object listener) {
        notifier.addChangeListener(connectedEvent, listener);
    }
    /**
     * Функция возвращает значение активности соединения с сервером. Истина - соединение
     * активно, ложь - соединения нет.
     * @return Значение активности соединения.
     */
    protected boolean getConnected() {
        return connected;
    }
    /**
     * Процедура установки флага активности соединения. Истина - соединение активно,
     * в противном случае - не активно.
     * @param connected Новое значение флага активности.
     */
    private void setConnected(boolean connected) {
        if (this.connected != connected) {
            this.connected = connected;
            notifier.eventHappened(connectedEvent);
        }
    }
    /**
     * Функция добавления слушателя ошибок соединения с сервером.
     * @param listener Представитель слушателя.
```

```
     */
    protected void addConnectionErrorListener(Object listener) {
        notifier.addChangeListener(connectionErrorEvent, listener);
    }
    /**
     * Функция сообщает о наличии ошибок соединения с сервером.
     * @return Истина - ошибки есть, ложь - ошибок нет.
     */
    protected boolean getConnectionError() {
        return connectionError;
    }
    /**
     * Процедура установки флага наличия ошибок соединения. Флаг истина - ошибки есть,
     * в противном случае - ошибок не было.
     * @param connectionError Новое значение флага ошибок.
     */
    protected void setConnectionError(boolean connectionError) {
        if (this.connectionError != connectionError) {
            this.connectionError = connectionError;
            notifier.eventHappened(connectionErrorEvent);
        }
    }
    /**
     * Процедура выполняет соединение с сервером. В случае успешного соединения
     * устанавливается соответствующий флаг, в противном случае устанавливается
     * флаг ошибки соединения.
     */
    protected void connect() {
        socket = new Socket();
        setConnected(false);
        //setConnectionError(false);
        String host = config.getParameter("host");
        Integer port = new Integer(config.getParameter("port"));
        try {
            socket.connect(new InetSocketAddress(host, port.intValue()));
            socketOut = new OutputStreamWriter(socket.getOutputStream());
            setConnected(true);
            new Thread(this).start();
        } catch (IOException e) {
            setConnectionError(true);
        }
    }
    protected void disconnect() {
        //setConnectionError(false);
        try {
            socket.close();
            socket = null;
            setConnected(false);
        } catch (IOException e) {
            setConnectionError(true);
        }
    }
    /**
     * Процедура отправки текста на сервер. Текстовые команды робота отправляются
     * на сервер через установленное TCP/IP соединение. В случае разрыва связи во
     * время отправления об этом информируется автомат управления соединением (или
     * другой предусмотренный слушатель этого события).
     * @param outputStr Строка на отправку. Символ ';' является разделителем команд.
     */
    protected void sendText(String outputStr) {
        try {
            int i = 0;
            int j;
            while ((j = outputStr.indexOf(';', i)) != -1) {
                if (logger != null) {
                    logger.printToLog("Послана строка: \"" + outputStr.substring(i, j) + "\"");
                    logger.printToScreen(">>> " + outputStr.substring(i, j));
                }
                socketOut.write(outputStr.substring(i, j) + "\n");
                i = j + 1;
            }
            if (logger != null) {
                logger.printToLog("Послана строка: \"" + outputStr.substring(i, outputStr.length())
+ "\"");
```

```
                logger.printToScreen(">>> " + outputStr.substring(i, outputStr.length()));
            }
            socketOut.write(outputStr.substring(i, outputStr.length()) + "\n");
            socketOut.flush();
        } catch (IOException e) {
            setConnected(false);
        }
    }
}

package ru.ifmo.rain.abdrashitov.elf;

/**
 * Автоматный класс "Сон". Создается автоматом управления соединением при переходе
 * в состояние игры. После инициализации для работы автомата создается отдельный поток.
 * Автомат контролирует боеспособность и управляет восстановлением боеспособности
 * персонажа в игровом мире. В качестве входных воздействий выступают состояние автоматов
 * A3-A5 и данные получаемые от анализатора входного потока.
 */
public class Rest extends Automaton {
    /**
     * Ссылка на МАД-адаптер. МАД-адаптер используется для выходного воздействия (восстановления
     * боеспособности персонажа) и опроса переменных (состояния боеспособности).
     */
    private MAdapter adapter;
    /**
     * Ссылка на автомат A1 "Перемещение".
     */
    private Walk walk;
    /**
     * Ссылка на автомат A3 "Битва".
     */
    private Fight fight;
    /**
      * Автомат A4 "Вода". Создается и уничтожается при переходе
      * в определенные состояния.
      */
    private Water water;
    /**
      * Автомат A5 "Еда". Создается и уничтожается при переходе
      * в определенные состояния.
      */
    private Food food;
    /**
     * Входная переменная x1. Неполная жизненая сила персонажа.
     */
    private boolean x1;
    /**
     * Входная переменная x2. Неполная магическая энергия персонажа.
     */
    private boolean x2;
    /**
     * Входная переменная x3. Неполная бодрость персонажа. Бодрость отвечает за способность
перемещатся.
     */
    private boolean x3;
    /**
     * Входная переменная x4. Персонаж участвует в битве.
     */
    private boolean x4;
    /**
     * Входная переменная x5. Ошибка выполнения выходного воздействия {@link Rest#z1()}.
     */
    private boolean x5;
    /**
     * Входная переменная x9. Текущая комната - мирное место.
     */
    private boolean x9;
    /**
     * Входная переменная x10. Мало едениц бодрости. Дальнейшие перемещения станут невозможными.
     */
    private boolean x10;
    /**
     * Входная переменная x11. Мало жизненой силы. Участие в битвах опасно для жизни персонажа.
```

```
 */
private boolean x11;
/**
 * Входная переменная x12. Необходимо бодрствовать.
 */
private boolean x12;
/**
 * Сохраненное для лога сообщение e1.
 */
private boolean e1log;
/**
 * Сохраненное для лога сообщение e2.
 */
private boolean e2log;
/**
 * Сохраненное для лога сообщение e3.
 */
private boolean e3log;
/**
 * Выполнение установки необходимых ссылок и непосредственной подготовки автомата к работе.
 * Перед запуском автомата необходимо зарегестрировать его во всех менеджерах событий,
 * отслеживающих изменения входных переменных этого автомата.
 * @param adapter Адаптер правил конкретной версии игры.
 * @param fight Ссылка на автомат A3 "Битва".
 * @param walk Ссылка на автомат A1 "Перемещение".
 * @param water Ссылка на автомат A4 "Вода".
 * @param food Ссылка на автомат A5 "Еда".
 */
protected void init(MAdapter adapter, Fight fight, Walk walk, Water water, Food food) {
    this.adapter = adapter;
    this.fight = fight;
    this.walk = walk;
    this.water = water;
    this.food = food;
    this.fight.addStateListener(this);
    this.walk.addStateListener(this);
    this.water.addStateListener(this);
    this.food.addStateListener(this);
    this.adapter.addNotFullHPListener(this);
    this.adapter.addNotFullManaListener(this);
    this.adapter.addNotFullMoveListener(this);
    this.adapter.addLowMoveListener(this);
    this.adapter.addLowHPListener(this);
    this.adapter.addUnrestListener(this);
    this.adapter.addSleepDoneListener(this);
    this.adapter.addRestDoneListener(this);
    this.adapter.addWakeDoneListener(this);
    //this.adapter.addPeaceListener(this);
}
/**
 * Обновление значений входных переменных. Процедура снимает информацию с источников
 * данных и вычисляет новые значения входных переменных.
 */
private void getInput() {
    int y1 = walk.getState();
    int y3 = fight.getState();
    int y4 = water.getState();
    int y5 = food.getState();
    x1 = adapter.getNotFullHP();
    x2 = adapter.getNotFullMana();
    x3 = adapter.getNotFullMove();
    x10 = adapter.getLowMove();
    x11 = adapter.getLowHP();
    x5 = adapter.getUnrest();
    //x9 = adapter.getPeace();
    x9 = (y1 == 7);
    x4 = adapter.getIsInFight() || (y3 == 3) || (y3 == 4) || (y3 == 5) || (y3 == 6);
    x12 = (y4 == 6) || (y5 == 6);
    e1log = e1t();
    e2log = e2t();
    e3log = e3t();
}
/**
 * Получение сообщения e1. Сообщение о выполнении действия {@link Rest#z1()}.
```

```
 * @return Истина, если сообщение есть.
 */
private boolean e1() {
    if (e1t()) {
        adapter.setSleepDone(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e1. Сообщение о выполнении действия {@link Rest#z1()}.
 * @return Истина, если сообщение есть.
 */
private boolean e1t() {
    return adapter.getSleepDone();
}
/**
 * Получение сообщения e2. Сообщение о выполнении действия {@link Rest#z2()}.
 * @return Истина, если сообщение есть.
 */
private boolean e2() {
    if (e2t()) {
        adapter.setRestDone(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e2. Сообщение о выполнении действия {@link Rest#z2()}.
 * @return Истина, если сообщение есть.
 */
private boolean e2t() {
    return adapter.getRestDone();
}
/**
 * Получение сообщения e3. Сообщение о выполнении действия {@link Rest#z3()}.
 * @return Истина, если сообщение есть.
 */
private boolean e3() {
    if (e3t()) {
        adapter.setWakeDone(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e3. Сообщение о выполнении действия {@link Rest#z3()}.
 * @return Истина, если сообщение есть.
 */
private boolean e3t() {
    return adapter.getWakeDone();
}
/**
 * Выходное воздействие z1. Лечь спать.
 */
private void z1() {
    adapter.doSleep();
}
/**
 * Выходное воздействие z2. Сесть отдохнуть.
 */
private void z2() {
    adapter.doRest();
}
/**
 * Выходное воздействие z3. Проснуться и/или встать.
 */
private void z3() {
    adapter.doWake();
}
```

```
protected String getName() {
    return "A2";
}
protected String getVariablesDump() {
    StringBuffer buf = new StringBuffer();
    if (x1) buf.append("x1=1"); else buf.append("x1=0");
    if (x2) buf.append(" x2=1"); else buf.append(" x2=0");
    if (x3) buf.append(" x3=1"); else buf.append(" x3=0");
    if (x4) buf.append(" x4=1"); else buf.append(" x4=0");
    if (x5) buf.append(" x5=1"); else buf.append(" x5=0");
    if (x9) buf.append(" x9=1"); else buf.append(" x9=0");
    if (x10) buf.append(" x10=1"); else buf.append(" x10=0");
    if (x11) buf.append(" x11=1"); else buf.append(" x11=0");
    if (x12) buf.append(" x12=1"); else buf.append(" x12=0");
    if (e1log) buf.append(" e1");
    if (e2log) buf.append(" e2");
    if (e3log) buf.append(" e3");
    return buf.toString();
}
protected boolean nextStep() {
    boolean waitNeed = false;
    int y = getState();
    getInput();
    switch(y) {
        case 1:
            if (x10 || x11) {
                setState(2);
            }
            else {
                waitNeed = true;
            }
            break;
        case 2:
            if (!x1 && !x2 && !x3) {
                setState(1);
            }
            else if (x10 && (x1 || x2 || x3) && !x9 && !x4) {
                z2();
                setState(4);
            }
            else if ((x1 || x2 || x3) && x9 && !x4 && !x12) {
                z1();
                setState(6);
            }
            else {
                waitNeed = true;
            }
            break;
        case 3:
            if (e3()) {
                setState(2);
            }
            else {
                waitNeed = true;
            }
            break;
        case 4:
            if (x4) {
                setState(2);
            }
            else if (e2() && !x4) {
                setState(5);
            }
            else {
                waitNeed = true;
            }
            break;
        case 5:
            if (x4) {
                setState(2);
            }
            else if (!x3 && !x4) {
                z3();
                setState(3);
```

```
                }
                else {
                    waitNeed = true;
                }
                break;
            case 6:
                if (e1()) {
                    setState(7);
                }
                else if (x5) {
                    z2();
                    setState(8);
                }
                else {
                    waitNeed = true;
                }
                break;
            case 7:
                if ((!x1 && !x2 && !x3 && x9) || !x9 || x12) {
                    z3();
                    setState(3);
                }
                else {
                    waitNeed = true;
                }
                break;
            case 8:
                if (e2()) {
                    setState(9);
                }
                else {
                    waitNeed = true;
                }
                break;
            case 9:
                if ((!x1 && !x2 && !x3 && x9) || !x9) {
                    z3();
                    setState(3);
                }
                else if ((x1 || x2 || x3) && !x5 && x9 && !x12) {
                    z1();
                    setState(6);
                }
                else {
                    waitNeed = true;
                }
                break;
        }
        return !waitNeed;
    }
}

package ru.ifmo.rain.abdrashitov.elf;

import java.util.regex.Pattern;
import java.util.regex.Matcher;
import java.util.Random;

/**
 * Пример написания МАД-адаптера. Класс адаптера является наследником
 * абстрактного класса {@link MAdapter}. Пример построен с учетом правил
 * сервера Аладон (www.aladon.ru:9000). И настроен на работу с персонажем
 * низкого уровня в зоне "MUD-школа". Персонаж должен быть обязательно
 * магической профессии. Класс тестировался на персонаже эльф-маг-нейтрал
 * на локальной версии сервера Aladon на платформе Win32.
 */
public class SampleAdapter extends MAdapter {
    /**
     * Лог, в который сохраняется текстовый поток приходящий от сервера.
     */
    private StringBuffer textLog = new StringBuffer();
    private Matcher loginDoneMatcher =
            Pattern.compile("^(Добро пожаловать в мир АЛАДОН.)|(Этим персонажем уже играют.)|" +
            "(Переподключение.)",
```

```
                    Pattern.MULTILINE).matcher(textLog);
    private int loginDonePosition = 0;
    private Matcher loginErrorMatcher =
            Pattern.compile("^((Твое имЯ. )?Такого имени тут нет.)|(Неправильный пароль.)|" +
            "(Лимит одновременных персонажей)",
                    Pattern.MULTILINE).matcher(textLog);
    private int loginErrorPosition = 0;
    private Matcher logoutDoneMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?Да, все хорошее когда-нибудь
заканчиваетсЯ...Заходи еще.",
                    Pattern.MULTILINE).matcher(textLog);
    private int logoutDonePosition = 0;
    private Matcher logoutErrorMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?(Нет! Ты еще сражаешьсЯ.$)|" +
            "(Кровь бьет в висках...не времЯ)|" +
            "(Твой хозЯин так хорош собой...не времЯ)",
                    Pattern.MULTILINE).matcher(textLog);
    private int logoutErrorPosition = 0;
    private Matcher movementMatcher =
            Pattern.compile("^\\[Видимые выходы:.*\\]",
                    Pattern.MULTILINE).matcher(textLog);
    private int movementPosition = 0;
    private Matcher monsterMatcher =
            Pattern.compile("^(Трусливый монстр привЯзан здесь)|(Некий монстр привЯзан здесь)|" +
            "(Агрессивный монстр здесь)|(Трусливый агрессивный монстр привЯзан здесь)",
                    Pattern.MULTILINE).matcher(textLog);
    private int monsterPosition = 0;
    private Matcher inFightMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?[a-zA-Za-яA-Я ]*( в прекрасном состоЯнии)|" +
            "( легко ранен)|" +
            "( ранен)|( серьезно ранен)|( выглЯдит очень плохо)|( в ужасном состоЯнии)",
                    Pattern.MULTILINE).matcher(textLog);
    private int inFightPosition = 0;
    private boolean possibleInFight = false;
    private Matcher attackErrorMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?(Таких тут нет)|(А-ах... Ты ведь
расслаблЯешьсЯ...)|" +
            "(В твоих снах или как)",
                    Pattern.MULTILINE).matcher(textLog);
    private int attackErrorPosition = 0;
    private Matcher monsterKilledMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?[a-zA-Za-яA-Я ]*( МЕРТВО!!)|( МЕРТВ!!)|(
МЕРТВА!!)|" +
            "(Ты убегаешь из драки)",
                    Pattern.MULTILINE).matcher(textLog);
    private int monsterKilledPosition = 0;
    private Matcher magicAttackDoneMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?(Ты не можешь хорошо)|(Ты терЯешь концентрацию)|"
+
            "(Что-то плохо получаетсЯ)|(ТебЯ что-то отвлекает)|" +
            "(У тебЯ не хватает энергии)|(Твой магический снарЯд)|(Колдовать заклинание на кого)",
                    Pattern.MULTILINE).matcher(textLog);
    private int magicAttackDonePosition = 0;
    private Matcher handAttackDoneMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?",
                    Pattern.MULTILINE).matcher(textLog);//hand attack is ignored
    private int handAttackDonePosition = 0;
    private Matcher unrestFalseMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?Ты понемногу успокаиваешьсЯ",
                    Pattern.MULTILINE).matcher(textLog);
    private int unrestFalsePosition = 0;
    private Matcher unrestTrueMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?Ты слишком взволнован недавней дракой",
                    Pattern.MULTILINE).matcher(textLog);
    private int unrestTruePosition = 0;
    private Matcher sleepDoneMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?(Ты ложишьсЯ спать)|(Ты уже спишь)",
                    Pattern.MULTILINE).matcher(textLog);
    private int sleepDonePosition = 0;
    private Matcher restDoneMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?(Ты просыпаешьсЯ и отдыхаешь)|(Ты уже отдыхаешь)|"
+
            "(Ты садишьсЯ отдыхать)",
                    Pattern.MULTILINE).matcher(textLog);
```

```
    private int restDonePosition = 0;
    private Matcher wakeDoneMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?(Ты просыпаешьсЯ и встаешь)|(Ты встаешь)|" +
            "(Ты уже стоишь)|" +
            "(Ты уже сражаешьсЯ)",
                    Pattern.MULTILINE).matcher(textLog);
    private int wakeDonePosition = 0;
    private Matcher drinkNeedMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?Ты хочешь пить",
                    Pattern.MULTILINE).matcher(textLog);
    private int drinkNeedPosition = 0;
    private Matcher outOfWaterMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?(Там уже пусто)|(Тут нет такого)",
                    Pattern.MULTILINE).matcher(textLog);
    private int outOfWaterPosition = 0;
    private Matcher drinkUnNeedMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?Ты утолЯешь свою жажду",
                    Pattern.MULTILINE).matcher(textLog);
    private int drinkUnNeedPosition = 0;
    private Matcher drinkDoneMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?(Ты пьешь воду из)|(Там уже пусто)|(Тут нет
такого)",
                    Pattern.MULTILINE).matcher(textLog);
    private int drinkDonePosition = 0;
    private Matcher drinkErrorMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?(Тебе уже хватит)|(Тебе уже не выпить ни капли)",
                    Pattern.MULTILINE).matcher(textLog);
    private int drinkErrorPosition = 0;
    private Matcher buyDrinkDoneMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?Ты покупаешь бурдюк",
                    Pattern.MULTILINE).matcher(textLog);
    private int buyDrinkDonePosition = 0;
    private Matcher buyDrinkErrorMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?(Адепт Селены говорит тебе)|" +
            "(Ты не можешь делать это здесь)",
                    Pattern.MULTILINE).matcher(textLog);
    private int buyDrinkErrorPosition = 0;
    private Matcher foodNeedMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?Ты хочешь есть",
                    Pattern.MULTILINE).matcher(textLog);
    private int foodNeedPosition = 0;
    private Matcher outOfFoodMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?У тебЯ нет этого",
                    Pattern.MULTILINE).matcher(textLog);
    private int outOfFoodPosition = 0;
    private Matcher foodUnNeedMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?Тебе уже не хочетсЯ есть",
                    Pattern.MULTILINE).matcher(textLog);
    private int foodUnNeedPosition = 0;
    private Matcher eatDoneMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?(Ты ешь хлеб)|(У тебЯ нет этого)",
                    Pattern.MULTILINE).matcher(textLog);
    private int eatDonePosition = 0;
    private Matcher eatErrorMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?(Ты наедаешьсЯ до отвала)|(Твой живот уже туго
набит)",
                    Pattern.MULTILINE).matcher(textLog);
    private int eatErrorPosition = 0;
    private Matcher buyFoodDoneMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?Ты покупаешь хлеб",
                    Pattern.MULTILINE).matcher(textLog);
    private int buyFoodDonePosition = 0;
    private Matcher buyFoodErrorMatcher =
            Pattern.compile("^(<[\\x20-\\xFF]*> )?(Адепт Селены говорит тебе)|" +
            "(Ты не можешь делать это здесь)",
                    Pattern.MULTILINE).matcher(textLog);
    private int buyFoodErrorPosition = 0;

    private int mapRoomPosition = 0;
    private Matcher mapRoom912 =
            Pattern.compile("^  Ты находишьсЯ в комнате с клетками. Вокруг - 4 клетки.\\n" +
            "МЯгкий белый свет льетсЯ с потолка.\\n" +
            "Конечно же, на стене имеетсЯ большой знак.\\n" +
            "Выходы идут во все главные направленИЯ и вниз.",
```

```
                    Pattern.MULTILINE).matcher(textLog);
    private Matcher mapRoom913 =
            Pattern.compile("^  Ты в клетке. Всюду запекшаЯсЯ кровь. Уборщики должно быть очень\\n"
+
            "небрежны здесь! ИмеетсЯ знак на стене.\\n" +
            "Единственный выход - на юг.",
                    Pattern.MULTILINE).matcher(textLog);
    private Matcher mapRoom914 =
            Pattern.compile("^  Ты в клетке. Всюду запекшаЯсЯ кровь. Уборщики должно быть очень\\n"
+
            "небрежны здесь! ИмеетсЯ знак на стене.\\n" +
            "Единственный выход - на восток.",
                    Pattern.MULTILINE).matcher(textLog);
    private Matcher mapRoom915 =
            Pattern.compile("^  Ты в клетке. Всюду запекшаЯсЯ кровь. Уборщики должно быть очень\\n"
+
            "небрежны здесь! ИмеетсЯ знак на стене.\\n" +
            "Единственный выход - на север.",
                    Pattern.MULTILINE).matcher(textLog);
    private Matcher mapRoom916 =
            Pattern.compile("^  Ты в клетке. Всюду запекшаЯсЯ кровь. Уборщики должно быть очень\\n"
+
            "небрежны здесь! ИмеетсЯ знак на стене.\\n" +
            "Единственный выход - на запад.",
                    Pattern.MULTILINE).matcher(textLog);
    private Matcher mapRoom917 =
            Pattern.compile("^  Ты в квадратной белой комнате. Стены пусты и без окон.\\n" +
            "МЯгкий белый свет льетсЯ с потолка.\\n" +
            "Конечно же, на стене имеетсЯ знак.",
                    Pattern.MULTILINE).matcher(textLog);
    private Matcher mapRoom918 =
            Pattern.compile("^  Ты находишьсЯ в узкой комнате. Всюду на полках - товары и
пакеты.\\n" +
            "МЯгкий белый свет льетсЯ с потолка.\\n" +
            "Конечно же, на стене имеетсЯ знак.",
                    Pattern.MULTILINE).matcher(textLog);

    private Matcher promptMatcher =
            Pattern.compile("^<(\\d+)hp (\\d+)mhp (\\d+)m (\\d+)mm (\\d+)mv (\\d+)mmv (\\d+)exp
(\\d+)mexp>",
                    Pattern.MULTILINE).matcher(textLog);
    private int promptPosition = 0;

    /**
     * Номер комнаты в которой находится персонаж. Если комната неопределена принимает
     * значение 0.
     */
    private int currentRoom = 0;
    /**
     * Цель перемещений персонажа.
     * <br>0 - нет цели или цель "мирная комната".
     * <br>1 - цель "противник".
     * <br>2 - цель "родник".
     * <br>3 - цель "магазин продуктов".
     */
    private int currentTarget = 0;
    /**
     * Генератор случайных чисел.
     */
    private Random random = new Random();
    /**
     * Команда движения на север.
     */
    private String north = "север";
    /**
     * Команда движения на юг.
     */
    private String south = "юг";
    /**
     * Команда движения на запад.
     */
    private String west = "запад";
    /**
     * Команда движения на восток.
```

```
 */
private String east = "восток";
/**
 * Команда движения вверх.
 */
private String up = "верх";
/**
 * Команда движения вниз.
 */
private String down = "низ";

protected SampleAdapter(BotConfig config) {
    super(config);
}

protected void doStepToTarget() {
    //setStepDone(false);
    switch (currentRoom) {
        case 912:
            if (currentTarget == 0) {
                setTargetReached(true);
                setStepDone(true);
            }
            else if (currentTarget == 1) {
                int i = random.nextInt(4);
                if (i == 0) {
                    socket.sendText(north);
                }
                else if (i == 1) {
                    socket.sendText(west);
                }
                else if (i == 2) {
                    socket.sendText(south);
                }
                else {
                    socket.sendText(east);
                }
            }
            else if ((currentTarget == 2) || (currentTarget == 3)) {
                socket.sendText(down);
            }
            break;
        case 913:
            socket.sendText(south);
            break;
        case 914:
            socket.sendText(east);
            break;
        case 915:
            socket.sendText(north);
            break;
        case 916:
            socket.sendText(west);
            break;
        case 917:
            if (currentTarget == 0) {
                socket.sendText(up);
            }
            else if (currentTarget == 1) {
                socket.sendText(up);
            }
            else if ((currentTarget == 2) || (currentTarget == 3)) {
                socket.sendText(south);
            }
            break;
        case 918:
            if (currentTarget == 0) {
                socket.sendText(north);
            }
            else if (currentTarget == 1) {
                socket.sendText(north);
            }
            else if ((currentTarget == 2) || (currentTarget == 3)) {
                setTargetReached(true);
```

```
                setStepDone(true);
            }
            break;
        default:
            socket.sendText("эмоция чешет репу, не зная куда пойти.");
            synchronized(this) {
                try {
                    this.wait(5000); //Елси персонаж находится в неизвестном месте.
                } catch (InterruptedException e) {
                }
            }
            socket.sendText("см");
            break;
    }
}
protected void login() {
    //setLoginError(false);
    socket.sendText("2");
    socket.sendText(config.getParameter("name"));
    socket.sendText(config.getParameter("password"));
    socket.sendText("да");
    socket.sendText("смотреть");
    socket.sendText("непризыв");
    socket.sendText("встать");
    socket.sendText("приглашение <%hhp %Hmhp %mm %Mmm %vmv %Vmmv %xexp %Xmexp>");
}
protected void logout() {
    //setLogoutDone(false);
    //setLogoutError(false);
    socket.sendText("конец");
}
protected void setTargetToPeacefulPlace() {
    currentTarget = 0;
    setTargetReached(false);
}
protected void setTargetToBattlePlace() {
    currentTarget = 1;
    setTargetReached(false);
}
protected void setTargetToShop() {
    currentTarget = 3;
    setTargetReached(false);
}
protected void setTargetToSpring() {
    currentTarget = 2;
    setTargetReached(false);
}
protected void killMonster() {
    //setAttackError(false);
    //setMonsterKilled(false);
    socket.sendText("убить монстр");
}
protected void doHandAttack() {
    //setHandAttackDone(false);
    //socket.sendText("выпад"); //Персонаж не выполняет сложных немагических ударов.
    synchronized(this) {
        try {
            this.wait(1000);
        } catch (InterruptedException e) {
        }
    }
}
protected void doMagicAttack() {
    //setMagicAttackDone(false);
    socket.sendText("кол ма");
}
protected void doSleep() {
    //setSleepDone(false);
    socket.sendText("спать");
}
protected void doRest() {
    //setRestDone(false);
    socket.sendText("отдых");
}
```

```
protected void doWake() {
    //setWakeDone(false);
    socket.sendText("встать");
}
protected void doDrink() {
    //setDrinkDone(false);
    //setDrinkError(false);
    socket.sendText("пить бурдюк");
}
protected void doBuyDrink() {
    //setBuyDrinkDone(false);
    //setBuyDrinkError(false);
    socket.sendText("брос бурдюк;жерт бурдюк;куп бурдюк");
}
protected void doEat() {
    //setEatDone(false);
    //setEatError(false);
    socket.sendText("есть хлеб");
}
protected void doBuyFood() {
    //setBuyFoodDone(false);
    //setBuyFoodError(false);
    socket.sendText("куп 3*хлеб");
}
protected void procesLine(String inputStr) {
    //System.out.println(inputStr);
    textLog.append(inputStr).append("\n");
    if (loginDoneMatcher.find(loginDonePosition)) {
        setLoginDone(true);
        loginDonePosition = loginDoneMatcher.start() + 1;
    }
    if (loginErrorMatcher.find(loginErrorPosition)) {
        setLoginError(true);
        loginErrorPosition = loginErrorMatcher.start() + 1;
    }
    if (logoutDoneMatcher.find(logoutDonePosition)) {
        setLogoutDone(true);
        logoutDonePosition = logoutDoneMatcher.start() + 1;
    }
    if (logoutErrorMatcher.find(logoutErrorPosition)) {
        setLogoutError(true);
        logoutErrorPosition = logoutErrorMatcher.start() + 1;
    }
    if (movementMatcher.find(movementPosition)) {
        setStepDone(true);
        movementPosition = movementMatcher.start() + 1;
    }
    if (monsterMatcher.find(monsterPosition)) {
        setIsMonsterHere(true);
        monsterPosition = monsterMatcher.start() + 1;
    }
    if (inFightMatcher.find(inFightPosition)) {
        setIsInFight(true);
        possibleInFight = true;
        inFightPosition = inFightMatcher.start() + 1;
    }
    if (attackErrorMatcher.find(attackErrorPosition)) {
        setAttackError(true);
        attackErrorPosition = attackErrorMatcher.start() + 1;
    }
    if (monsterKilledMatcher.find(monsterKilledPosition)) {
        setMonsterKilled(true);
        monsterKilledPosition = monsterKilledMatcher.start() + 1;
        socket.sendText("улыб");
    }
    if (magicAttackDoneMatcher.find(magicAttackDonePosition)) {
        setMagicAttackDone(true);
        magicAttackDonePosition = magicAttackDoneMatcher.start() + 1;
    }
    if (handAttackDoneMatcher.find(handAttackDonePosition)) {
        setHandAttackDone(true);
        handAttackDonePosition = handAttackDoneMatcher.start() + 1;
    }
    if (unrestFalseMatcher.find(unrestFalsePosition)) {
```

```
            setUnrest(false);
            unrestFalsePosition = unrestFalseMatcher.start() + 1;
        }
        if (unrestTrueMatcher.find(unrestTruePosition)) {
            setUnrest(true);
            unrestTruePosition = unrestTrueMatcher.start() + 1;
        }
        if (sleepDoneMatcher.find(sleepDonePosition)) {
            setSleepDone(true);
            sleepDonePosition = sleepDoneMatcher.start() + 1;
        }
        if (restDoneMatcher.find(restDonePosition)) {
            setRestDone(true);
            restDonePosition = restDoneMatcher.start() + 1;
        }
        if (wakeDoneMatcher.find(wakeDonePosition)) {
            setWakeDone(true);
            wakeDonePosition = wakeDoneMatcher.start() + 1;
        }
        if (drinkNeedMatcher.find(drinkNeedPosition)) {
            setDrinkNeed(true);
            drinkNeedPosition = drinkNeedMatcher.start() + 1;
        }
        if (drinkUnNeedMatcher.find(drinkUnNeedPosition)) {
            setDrinkNeed(false);
            drinkUnNeedPosition = drinkUnNeedMatcher.start() + 1;
        }
        if (outOfWaterMatcher.find(outOfWaterPosition)) {
            setOutOfWater(true);
            outOfWaterPosition = outOfWaterMatcher.start() + 1;
        }
        if (buyDrinkDoneMatcher.find(buyDrinkDonePosition)) {
            setOutOfWater(false);
            setBuyDrinkDone(true);
            buyDrinkDonePosition = buyDrinkDoneMatcher.start() + 1;
        }
        if (drinkDoneMatcher.find(drinkDonePosition)) {
            setDrinkDone(true);
            drinkDonePosition = drinkDoneMatcher.start() + 1;
        }
        if (drinkErrorMatcher.find(drinkErrorPosition)) {
            setDrinkError(true);
            drinkErrorPosition = drinkErrorMatcher.start() + 1;
        }
        if (buyDrinkErrorMatcher.find(buyDrinkErrorPosition)) {
            setBuyDrinkError(true);
            buyDrinkErrorPosition = buyDrinkErrorMatcher.start() + 1;
        }
        if (foodNeedMatcher.find(foodNeedPosition)) {
            setFoodNeed(true);
            foodNeedPosition = foodNeedMatcher.start() + 1;
        }
        if (foodUnNeedMatcher.find(foodUnNeedPosition)) {
            setFoodNeed(false);
            foodUnNeedPosition = foodUnNeedMatcher.start() + 1;
        }
        if (outOfFoodMatcher.find(outOfFoodPosition)) {
            setOutOfFood(true);
            outOfFoodPosition = outOfFoodMatcher.start() + 1;
        }
        if (buyFoodDoneMatcher.find(buyFoodDonePosition)) {
            setOutOfFood(false);
            setBuyFoodDone(true);
            buyFoodDonePosition = buyFoodDoneMatcher.start() + 1;
        }
        if (eatDoneMatcher.find(eatDonePosition)) {
            setEatDone(true);
            eatDonePosition = eatDoneMatcher.start() + 1;
        }
        if (eatErrorMatcher.find(eatErrorPosition)) {
            setEatError(true);
            eatErrorPosition = eatErrorMatcher.start() + 1;
        }
        if (buyFoodErrorMatcher.find(buyFoodErrorPosition)) {
```

```
    setBuyFoodError(true);
    buyFoodErrorPosition = buyFoodErrorMatcher.start() + 1;
}

if (mapRoom912.find(mapRoomPosition)) {
    currentRoom = 912;
    //setPeace(true);
    mapRoomPosition = mapRoom912.start() + 1;
    if (currentTarget == 0) {
        setTargetReached(true);
    }
}
else if (mapRoom913.find(mapRoomPosition)) {
    currentRoom = 913;
    mapRoomPosition = mapRoom913.start() + 1;
    if (currentTarget == 1) {
        setTargetReached(true);
    }
}
else if (mapRoom914.find(mapRoomPosition)) {
    currentRoom = 914;
    mapRoomPosition = mapRoom914.start() + 1;
    if (currentTarget == 1) {
        setTargetReached(true);
    }
}
else if (mapRoom915.find(mapRoomPosition)) {
    currentRoom = 915;
    mapRoomPosition = mapRoom915.start() + 1;
    if (currentTarget == 1) {
        setTargetReached(true);
    }
}
else if (mapRoom916.find(mapRoomPosition)) {
    currentRoom = 916;
    mapRoomPosition = mapRoom916.start() + 1;
    if (currentTarget == 1) {
        setTargetReached(true);
    }
}
else if (mapRoom917.find(mapRoomPosition)) {
    currentRoom = 917;
    //setPeace(true);
    mapRoomPosition = mapRoom917.start() + 1;
    /*if (currentTarget == 0) {
        setTargetReached(true);
    }*/
}
else if (mapRoom918.find(mapRoomPosition)) {
    currentRoom = 918;
    //setPeace(true);
    mapRoomPosition = mapRoom918.start() + 1;
    if ((currentTarget == 2) || (currentTarget == 3)) {
        setTargetReached(true);
    }
}
/**
 * Кусок необходимо помещать в конец процедуры для улучшения оценки бесполезных
 * частей входящего трафика.
 */
if (promptMatcher.find(promptPosition)) {
    Integer hp = new Integer(promptMatcher.group(1));
    Integer maxHp = new Integer(promptMatcher.group(2));
    Integer mana = new Integer(promptMatcher.group(3));
    Integer maxMana = new Integer(promptMatcher.group(4));
    Integer move = new Integer(promptMatcher.group(5));
    Integer maxMove = new Integer(promptMatcher.group(6));
    //Integer exp = new Integer(promptMatcher.group(7));
    Integer maxExp = new Integer(promptMatcher.group(8));
    if ((hp.intValue() <= 16) && (hp.intValue() < maxHp.intValue())) {//maxHp.intValue()) {
        setLowHP(true);
    }
    else {
        setLowHP(false);
```

```
}
if (hp.intValue() >= maxHp.intValue()) {
    setNotFullHP(false);
}
else {
    setNotFullHP(true);
}

if (mana.intValue() < 50) {
    setLowMana(true);
}
else {
    setLowMana(false);
}
if (mana.intValue() >= maxMana.intValue()) {
    setNotFullMana(false);
}
else {
    setNotFullMana(true);
}

if (move.intValue() < 5) {
    setLowMove(true);
}
else {
    setLowMove(false);
}
if (move.intValue() >= maxMove.intValue()) {
    setNotFullMove(false);
}
else {
    setNotFullMove(true);
}

if (maxExp.intValue() <= 250) {
    setTooMuchExp(true);
}
else {
    setTooMuchExp(false);
}
setIsMonsterHere(false);
if (!possibleInFight) {
    setIsInFight(false);
}
else {
    possibleInFight = false;
}
//setPeace(false);
promptPosition = promptMatcher.start() + 1;
movementPosition = Math.max(promptPosition - 1, movementPosition);
mapRoomPosition = Math.max(promptPosition - 1, mapRoomPosition);
loginDonePosition = Math.max(promptPosition - 1, loginDonePosition);
loginErrorPosition = Math.max(promptPosition - 1, loginErrorPosition);
logoutDonePosition = Math.max(promptPosition - 1, logoutDonePosition);
logoutErrorPosition = Math.max(promptPosition - 1, logoutErrorPosition);
monsterPosition = Math.max(promptPosition - 1, monsterPosition);
inFightPosition = Math.max(promptPosition - 1, inFightPosition);
attackErrorPosition = Math.max(promptPosition - 1, attackErrorPosition);
monsterKilledPosition = Math.max(promptPosition - 1, monsterKilledPosition);
magicAttackDonePosition = Math.max(promptPosition - 1, magicAttackDonePosition);
handAttackDonePosition = Math.max(promptPosition - 1, handAttackDonePosition);
unrestFalsePosition = Math.max(promptPosition - 1, unrestFalsePosition);
unrestTruePosition = Math.max(promptPosition - 1, unrestTruePosition);
sleepDonePosition = Math.max(promptPosition - 1, sleepDonePosition);
restDonePosition = Math.max(promptPosition - 1, restDonePosition);
wakeDonePosition = Math.max(promptPosition - 1, wakeDonePosition);
drinkNeedPosition = Math.max(promptPosition - 1, drinkNeedPosition);
drinkUnNeedPosition = Math.max(promptPosition - 1, drinkUnNeedPosition);
buyDrinkDonePosition = Math.max(promptPosition - 1, buyDrinkDonePosition);
outOfWaterPosition = Math.max(promptPosition - 1, outOfWaterPosition);
drinkDonePosition = Math.max(promptPosition - 1, drinkDonePosition);
drinkErrorPosition = Math.max(promptPosition - 1, drinkErrorPosition);
buyDrinkErrorPosition = Math.max(promptPosition - 1, buyDrinkErrorPosition);
foodNeedPosition = Math.max(promptPosition - 1, foodNeedPosition);
```

```
            foodUnNeedPosition = Math.max(promptPosition - 1, foodUnNeedPosition);
            buyFoodDonePosition = Math.max(promptPosition - 1, buyFoodDonePosition);
            outOfFoodPosition = Math.max(promptPosition - 1, outOfFoodPosition);
            eatDonePosition = Math.max(promptPosition - 1, eatDonePosition);
            eatErrorPosition = Math.max(promptPosition - 1, eatErrorPosition);
            buyFoodErrorPosition = Math.max(promptPosition - 1, buyFoodErrorPosition);
        }

    }
}
```

```
package ru.ifmo.rain.abdrashitov.elf;

import java.util.HashMap;
import java.util.ArrayList;
import java.util.ListIterator;

/**
 * Класс пробуждающий потоки по событию. Пробуждаются все потоки усыпленные на определенные
 * обьекты, соответствующие зарегестрированным событиям. Событием является изменение
 * определенной переменной. Данный подход позволяет значительно сэкономить системные ресурсы.
 */
public class VariableNotifier {
    /**
     * Хеш таблица, в которой хранятся списки зарегестрированных слушателей определенного
     * события.
     */
    private HashMap eventmap;

    /**
     * Инициализирует таблицу слушателей событий.
     */
    protected VariableNotifier() {
        eventmap = new HashMap();
    }

    /**
     * Добавляет слушателя изменений зарегестрированного обьекта. Слушатель представлен
     * обьектом на который он засыпает в ожидании события.
     * @param event Обьект, изменения которого отслеживаются.
     * @param listener Представитель слушателя.
     */
    protected void addChangeListener(Object event, Object listener) {
        if (!eventmap.containsKey(event)) {
            ArrayList newlist = new ArrayList();
            eventmap.put(event, newlist);
        }
        ArrayList list = (ArrayList) eventmap.get(event);
        list.add(listener);
    }
    /**
     * Вызывается при изменении переменной event. Будятся все слушатели изменений прерменной,
     * уснувшие на своего зарегестрированного представителя.
     * @param event Переменная которая изменилась.
     */
    protected void eventHappened(Object event) {
        if (eventmap.containsKey(event)) {
            ListIterator list = ((ArrayList) eventmap.get(event)).listIterator();
            while (list.hasNext()) {
                Object listener = list.next();
                synchronized (listener) {
                    listener.notify();
                }
            }
        }
    }
}
```

```
package ru.ifmo.rain.abdrashitov.elf;

/**
 * Пустой адаптер. Используется, когда запрашиваемый конфигурацией адаптер не найден.
 * Не выполняет никаких действий.
 */
```

```
public class VoidAdapter extends MAdapter {
    public VoidAdapter(BotConfig config) {
        super(config);
    }
    protected void login() {
        System.out.println("Can't login: void version of adapter.");
        setLoginDone(false);
        setLoginError(true);
    }
    protected void logout() {
    }
    protected void setTargetToPeacefulPlace() {
    }
    protected void setTargetToBattlePlace() {
    }
    protected void setTargetToShop() {
    }
    protected void setTargetToSpring() {
    }
    protected void doStepToTarget() {
    }
    protected void killMonster() {
    }
    protected void doHandAttack() {
    }
    protected void doMagicAttack() {
    }
    protected void doSleep() {
    }
    protected void doRest() {
    }
    protected void doWake() {
    }
    protected void doDrink() {
    }
    protected void doBuyDrink() {
    }
    protected void doEat() {
    }
    protected void doBuyFood() {
    }

    protected void procesLine(String inputStr) {
        System.out.println(inputStr);
    }
}

package ru.ifmo.rain.abdrashitov.elf;

/**
 * Автоматный класс "Перемещение". Создается автоматом управления соединением при переходе
 * в состояние игры. После инициализации для работы автомата создается отдельный поток.
 * Автомат управляет перемещениями персонажа в игровом мире. В качестве входных воздействий
 * выступают состояния автоматов A2-A5 и данные получаемые от анализатора входного потока.
 */
public class Walk extends Automaton {
    /**
     * Ссылка на МАД-адаптер. МАД-адаптер используется для выходного воздействия (перемещения)
     * и опроса переменных (результата перемещения).
     */
    private MAdapter adapter;
    /**
     * Ссылка на автомат A2 "Сон".
     */
    private Rest rest;
    /**
     * Ссылка на автомат A3 "Битва".
     */
    private Fight fight;
    /**
     * Ссылка на автомат A4 "Вода".
     */
    private Water water;
    /**
```

```
     * Ссылка на автомат A5 "Еда".
     */
    private Food food;
    /**
     * Входная переменная x1. Индикатор необходимости мирного места.
     */
    private boolean x1;
    /**
     * Входная переменная x2. Индикатор необходимости противника.
     */
    private boolean x2;
    /**
     * Входная переменная x3. Индикатор необходимости магазина.
     */
    private boolean x3;
    /**
     * Входная переменная x4. Индикатор необходимости родника.
     */
    private boolean x4;
    /**
     * Входная переменная x5. Индикатор достижения поставленной цели.
     */
    private boolean x5;
    /**
     * Входная переменная x7. Тормоз. Выполнение некоторых действий требует нахождения
     * на одном месте.
     */
    private boolean x7;
    /**
     * Сохраненное для лога сообщение e1.
     */
    private boolean e1log;
    /**
     * Выполнение установки необходимых ссылок и непосредственной подготовки автомата к работе.
     * Перед запуском автомата необходимо зарегестрировать его во всех менеджерах событий,
     * отслеживающих изменения входных переменных этого автомата.
     * @param adapter Адаптер правил конкретной версии игры.
     * @param rest Ссылка на автомат A2 "Сон".
     * @param fight Ссылка на автомат A3 "Битва".
     * @param water Ссылка на автомат A4 "Вода".
     * @param food Ссылка на автомат A5 "Еда".
     */
    protected void init(MAdapter adapter, Rest rest, Fight fight, Water water, Food food) {
        this.adapter = adapter;
        this.rest = rest;
        this.fight = fight;
        this.water = water;
        this.food = food;
        this.rest.addStateListener(this);
        this.fight.addStateListener(this);
        this.water.addStateListener(this);
        this.food.addStateListener(this);
        this.adapter.addTargetReachedListener(this);
        this.adapter.addStepDoneListener(this);
        this.adapter.addTooMuchExpListener(this);
    }
    /**
     * Обновление значений входных переменных. Процедура снимает информацию с источников
     * данных и вычисляет новые значения входных переменных.
     */
    private void getInput() {
        int y2 = rest.getState();
        int y3 = fight.getState();
        int y4 = water.getState();
        int y5 = food.getState();
        x1 = (y2 > 1) || adapter.getTooMuchExp();
        x2 = (y3 == 2);
        x3 = (y5 == 3) || (y5 == 4) || (y5 == 5);
        x4 = (y4 == 3) || (y4 == 4) || (y4 == 5);
        x5 = adapter.getTargetReached();
        x7 = (y2 == 3) || (y2 == 4) || (y2 == 5) || (y2 == 3) || (y2 == 6) || (y2 == 7) || (y2 == 8)
|| (y2 == 9) ||
                (y3 == 3) || (y3 == 4) || (y3 == 5) || (y3 == 6);
        e1log = e1t();
```

```
}
/**
 * Получение сообщения e1. Сообщение о выполнении шага по направлению к цели.
 * @return Истина, если сообщение есть.
 */
private boolean e1() {
    if (e1t()) {
        adapter.setStepDone(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e1. Сообщение о выполнении шага по направлению к цели.
 * @return Истина, если сообщение есть.
 */
private boolean e1t() {
    return adapter.getStepDone();
}
/**
 * Выходное воздействие z1. Построение пути до ближайшего мирного места.
 */
private void z1() {
    adapter.setTargetToPeacefulPlace();
}
/**
 * Выходное воздействие z2. Построение пути до любого противника.
 */
private void z2() {
    adapter.setTargetToBattlePlace();
}
/**
 * Выходное воздействие z3. Построение пути до ближайшего магазина.
 */
private void z3() {
    adapter.setTargetToShop();
}
/**
 * Выходное воздействие z4. Построение пути до ближайшего родника.
 */
private void z4() {
    adapter.setTargetToSpring();
}
/**
 * Выходное воздействие z5. Выполнение шага по построенному пути.
 */
private void z5() {
    adapter.doStepToTarget();
}
/**
 * Выходное воздействие z6. Пауза в выполнении автомата.
 */
private void z6() {
    synchronized(this) {
        try {
            this.wait(1000);
        } catch (InterruptedException e) {
        }
    }
}

protected String getName() {
    return "A1";
}
protected String getVariablesDump() {
    StringBuffer buf = new StringBuffer();
    if (x1) buf.append("x1=1"); else buf.append("x1=0");
    if (x2) buf.append(" x2=1"); else buf.append(" x2=0");
    if (x3) buf.append(" x3=1"); else buf.append(" x3=0");
    if (x4) buf.append(" x4=1"); else buf.append(" x4=0");
    if (x5) buf.append(" x5=1"); else buf.append(" x5=0");
    if (x7) buf.append(" x7=1"); else buf.append(" x7=0");
    if (e1log) buf.append(" e1");
```

```
    return buf.toString();
}
protected boolean nextStep() {
    boolean waitNeed = false;
    int y = getState();
    getInput();
    switch (y) {
        case 1:
            if (x4) {
                z4();
                setState(2);
            }
            else if (!x4 && x3) {
                z3();
                setState(4);
            }
            else if (!x4 && !x3 && x1) {
                z1();
                setState(6);
            }
            else if (!x4 && !x3 && !x1 && x2) {
                z2();
                setState(8);
            }
            else {
                waitNeed = true;
            }
            break;
        case 2:
            if (!x4) {
                setState(1);
            }
            else if (x4 && !x5 && !x7) {
                z5();
                setState(10);
            }
            else if (x4 && x5) {
                setState(3);
            }
            else {
                waitNeed = true;
            }
            break;
        case 3:
            if (!x4) {
                setState(1);
            }
            else {
                waitNeed = true;
            }
            break;
        case 4:
            if (!x3) {
                setState(1);
            }
            else if (x3 && !x5 && !x7) {
                z5();
                setState(11);
            }
            else if (x3 && x5) {
                setState(5);
            }
            else {
                waitNeed = true;
            }
            break;
        case 5:
            if (!x3) {
                setState(1);
            }
            else {
                waitNeed = true;
            }
            break;
```

```
case 6:
    if (!x1 || x3 || x4) {
        setState(1);
    }
    else if (x1 && !x3 && !x4 && !x5 & !x7) {
        z5();
        setState(12);
    }
    else if (x1 && !x3 && !x4 && x5) {
        setState(7);
    }
    else {
        waitNeed = true;
    }
    break;
case 7:
    if (!x1 || x3 || x4) {
        setState(1);
    }
    else {
        waitNeed = true;
    }
    break;
case 8:
    if (!x2 || x1 || x3 || x4) {
        setState(1);
    }
    else if (x2 && !x1 && !x3 && !x4 && !x5 && !x7) {
        z5();
        setState(13);
    }
    else if (x2 && !x1 && !x3 && !x4 && x5) {
        z6();
        setState(9);
    }
    else {
        waitNeed = true;
    }
    break;
case 9:
    if (x7) {
        waitNeed = true;
    }
    else if (!x7 && (!x2 || x1 || x3 || x4)) {
        setState(1);
    }
    else if (!x7 && x2 && !x1 && !x3 && !x4) {
        z2();
        setState(8);
    }
    else {
        waitNeed = true;
    }
    break;
case 10:
    if (e1()) {
        setState(2);
    }
    else {
        waitNeed = true;
    }
    break;
case 11:
    if (e1()) {
        setState(4);
    }
    else {
        waitNeed = true;
    }
    break;
case 12:
    if (e1()) {
        setState(6);
    }
```

```
                else {
                    waitNeed = true;
                }
                break;
            case 13:
                if (e1()) {
                    setState(8);
                }
                else {
                    waitNeed = true;
                }
                break;
        }
        return !waitNeed;
    }
}

package ru.ifmo.rain.abdrashitov.elf;

/**
 * Автоматный класс "Вода". Создается автоматом управления соединением при переходе
 * в состояние игры. После инициализации для работы автомата создается отдельный поток.
 * Автомат управляет утолением жажды персонажа в игровом мире. В качестве входных воздействий
 * выступают данные получаемые от анализатора входного потока и состояние автоматов A1-A2.
 */
public class Water extends Automaton {
    /**
     * Ссылка на МАД-адаптер. МАД-адаптер используется для выходного воздействия
     * и опроса переменных.
     */
    private MAdapter adapter;
    /**
     * Ссылка на автомат A1 "Перемещение".
     */
    private Walk walk;
    /**
     * Ссылка на автомат A2 "Сон".
     */
    private Rest rest;
    /**
     * Входная переменная x1. Индикатор жажды персонажа.
     */
    private boolean x1;
    /**
     * Входная переменная x2. Индикатор отсутствия запасов воды.
     */
    private boolean x2;
    /**
     * Входная переменная x3. Персонаж находится у источника воды.
     */
    private boolean x3;
    /**
     * Входная переменная x8. Персонаж спит.
     */
    private boolean x8;
    /**
     * Сохраненное для лога сообщение e1.
     */
    private boolean e1log;
    /**
     * Сохраненное для лога сообщение e2.
     */
    private boolean e2log;
    /**
     * Сохраненное для лога сообщение e3.
     */
    private boolean e3log;
    /**
     * Сохраненное для лога сообщение e4.
     */
    private boolean e4log;
    /**
     * Выполнение установки необходимых ссылок и непосредственной подготовки автомата к работе.
     * Перед запуском автомата необходимо зарегестрировать его во всех менеджерах событий,
```

```
 * отслеживающих изменения входных переменных этого автомата.
 * @param adapter Адаптер правил конкретной версии игры.
 * @param walk Ссылка на автомат A1 "Перемещение".
 * @param rest Ссылка на автомат A2 "Сон".
 */
protected void init(MAdapter adapter, Walk walk, Rest rest) {
    this.adapter = adapter;
    this.walk = walk;
    this.rest = rest;
    this.walk.addStateListener(this);
    this.rest.addStateListener(this);
    this.adapter.addDrinkNeedListener(this);
    this.adapter.addOutOfWaterListener(this);
    this.adapter.addDrinkDoneListener(this);
    this.adapter.addDrinkErrorListener(this);
    this.adapter.addBuyDrinkDoneListener(this);
    this.adapter.addBuyDrinkErrorListener(this);
}
/**
 * Обновление значений входных переменных. Процедура снимает информацию с источников
 * данных и вычисляет новые значения входных переменных.
 */
private void getInput() {
    int y1 = walk.getState();
    int y2 = rest.getState();
    x1 = adapter.getDrinkNeed();
    x2 = adapter.getOutOfWater();
    x3 = (y1 == 3);
    x8 = (y2 == 6) || (y2 == 7);
    e1log = e1t();
    e2log = e2t();
    e3log = e3t();
    e4log = e4t();
}
/**
 * Получение сообщения e1. Сообщение о выполнении действия {@link Water#z1()}.
 * @return Истина, если сообщение есть.
 */
private boolean e1() {
    if (e1t()) {
        adapter.setDrinkDone(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e1. Сообщение о выполнении действия {@link Water#z1()}.
 * @return Истина, если сообщение есть.
 */
private boolean e1t() {
    return adapter.getDrinkDone();
}
/**
 * Получение сообщения e2. Сообщение об ошибке выполнения действия {@link Water#z1()}.
 * @return Истина, если сообщение есть.
 */
private boolean e2() {
    if (e2t()) {
        adapter.setDrinkError(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e2. Сообщение об ошибке выполнения действия {@link Water#z1()}.
 * @return Истина, если сообщение есть.
 */
private boolean e2t() {
    return adapter.getDrinkError();
}
/**
 * Получение сообщения e3. Сообщение о выполнении действия {@link Water#z2()}.
```

```
 * @return Истина, если сообщение есть.
 */
private boolean e3() {
    if (e3t()) {
        adapter.setBuyDrinkDone(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e3. Сообщение о выполнении действия {@link Water#z2()}.
 * @return Истина, если сообщение есть.
 */
private boolean e3t() {
    return adapter.getBuyDrinkDone();
}
/**
 * Получение сообщения e4. Сообщение об ошибке выполнения действия {@link Water#z2()}.
 * @return Истина, если сообщение есть.
 */
private boolean e4() {
    if (e4t()) {
        adapter.setBuyDrinkError(false);
        return true;
    } else {
        return false;
    }
}
/**
 * Проверка наличия сообщения e4. Сообщение об ошибке выполнения действия {@link Water#z2()}.
 * @return Истина, если сообщение есть.
 */
private boolean e4t() {
    return adapter.getBuyDrinkError();
}
/**
 * Выходное воздействие z1. Выпить воды.
 */
private void z1() {
    adapter.doDrink();
}
/**
 * Выходное воздействие z2. Пополнить запас воды.
 */
private void z2() {
    adapter.doBuyDrink();
}
/**
 * Выходное воздействие z3. Сделать значительную паузу в выполнении автомата.
 */
private void z3() {
    synchronized(this) {
        try {
            this.wait(60000);
        } catch (InterruptedException e) {
        }
    }
}

protected String getName() {
    return "A4";
}
protected String getVariablesDump() {
    StringBuffer buf = new StringBuffer();
    if (x1) buf.append("x1=1"); else buf.append("x1=0");
    if (x2) buf.append(" x2=1"); else buf.append(" x2=0");
    if (x3) buf.append(" x3=1"); else buf.append(" x3=0");
    if (x8) buf.append(" x8=1"); else buf.append(" x8=0");
    if (e1log) buf.append(" e1");
    if (e2log) buf.append(" e2");
    if (e3log) buf.append(" e3");
    if (e4log) buf.append(" e4");
    return buf.toString();
```

```
}
protected boolean nextStep() {
    boolean waitNeed = false;
    int y = getState();
    getInput();
    switch (y) {
        case 1:
            if (x1) {
                setState(2);
            }
            else {
                waitNeed = true;
            }
            break;
        case 2:
            if (x2) {
                setState(3);
            }
            else if (!x2) {
                setState(6);
            }
            else {
                waitNeed = true;
            }
            break;
        case 3:
            if (x3) {
                setState(4);
            }
            else {
                waitNeed = true;
            }
            break;
        case 4:
            if (x2) {
                z2();
                setState(5);
            }
            else if (!x2) {
                setState(2);
            }
            else {
                waitNeed = true;
            }
            break;
        case 5:
            if (e4()) {
                setState(8);
            }
            else if (e3()) {
                setState(4);
            }
            else {
                waitNeed = true;
            }
            break;
        case 6:
            if (x2) {
                setState(3);
            }
            else if (!x1 && !x2) {
                setState(1);
            }
            else if (x1 && !x2 && !x8) {
                z1();
                setState(7);
            }
            else {
                waitNeed = true;
            }
            break;
        case 7:
            if (e2()) {
                setState(1);
```

```
                    z3();
                }
                else if (e1()) {
                    setState(6);
                }
                else {
                    waitNeed = true;
                }
                break;
            case 8:
                waitNeed = true;
                break;
        }
        return !waitNeed;
    }
}

package ru.ifmo.rain.abdrashitov.elf.gui;

import ru.ifmo.rain.abdrashitov.elf.MBot;
import ru.ifmo.rain.abdrashitov.elf.Logger;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class MainForm extends JFrame {
    public static final String TITLE = "МПРИ-робот";
    public static final int WIDTH = 320;
    public static final int HEIGHT = 200;

    private JScrollPane textPanel;
    private JScrollPane logPanel;
    private JPanel mainPanel;
    private JTextArea logArea;
    private JTextArea textArea;
    private JButton startStopButton;
    private JButton textLogButton;

    private String STARTBUTTON = "Включить";
    private String STOPBUTTON = "Выключить";
    private String TEXTBUTTON = "Текст";
    private String LOGBUTTON = "Протокол";

    private MBot robot;

    public MainForm() {
        super(TITLE);
        setSize(WIDTH,  HEIGHT);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setContentPane(createContentPane());
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                actionOnClose();
            }
        });
        robot = createRobot();
    }

    private MBot createRobot() {
        String configStr =
                "host=localhost;" +
                "port=4000;" +
                "adapter=SampleAdapter;" +
                "name=персонаж;" +
                "password=пароль;";
        return new MBot(configStr, new Logger() {
            public void printToScreen(String str) {
                textArea.append(str);
                textArea.append("\n");

textPanel.getVerticalScrollBar().setValue(textPanel.getVerticalScrollBar().getMaximum());
                mainPanel.repaint();
            }
```

```
            public void printToLog(String str) {
                logArea.append(str);
                logArea.append("\n");

logPanel.getVerticalScrollBar().setValue(logPanel.getVerticalScrollBar().getMaximum());
                mainPanel.repaint();
            }
        });
    }

    private void actionOnClose() {
        robot.disposeRobot();
    }

    private Font getTextFont() {
        Font font = new Font("Monospaced", Font.PLAIN, 12);
        return font;
    }

    private Container createContentPane() {
        Box result = Box.createVerticalBox();
        mainPanel = new JPanel();
        mainPanel.setLayout(new BorderLayout());
        logArea = new JTextArea();
        logArea.setEditable(false);
        logArea.setFont(getTextFont());
        logPanel = new JScrollPane(logArea);
        textArea = new JTextArea();
        textArea.setEditable(false);
        textArea.setFont(getTextFont());
        textPanel = new JScrollPane(textArea);
        mainPanel.add(logPanel, BorderLayout.CENTER);
        result.add(mainPanel);
        Box buttons = Box.createHorizontalBox();
        startStopButton = new JButton("Start");
        startStopButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (startStopButton.getText().equals(STARTBUTTON)) {
                    robot.startRobot();
                    startStopButton.setText(STOPBUTTON);
                } else {
                    robot.stopRobot();
                    startStopButton.setText(STARTBUTTON);
                }
            }
        });
        buttons.add(Box.createHorizontalGlue());
        buttons.add(startStopButton);
        buttons.add(Box.createHorizontalGlue());
        textLogButton = new JButton("Text");
        textLogButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (textLogButton.getText().equals(TEXTBUTTON)) {
                    mainPanel.add(textPanel);
                    mainPanel.remove(logPanel);

textPanel.getVerticalScrollBar().setValue(textPanel.getVerticalScrollBar().getMaximum());
                    mainPanel.repaint();
                    textLogButton.setText(LOGBUTTON);
                } else {
                    mainPanel.add(logPanel);
                    mainPanel.remove(textPanel);

logPanel.getVerticalScrollBar().setValue(logPanel.getVerticalScrollBar().getMaximum());
                    mainPanel.repaint();
                    textLogButton.setText(TEXTBUTTON);
                }
            }
        });
        buttons.add(textLogButton);
        buttons.add(Box.createHorizontalGlue());
        result.add(buttons);
        return result;
    }
```

```
}
```