Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра «Компьютерных технологий»

К.А. Ворошилов, А.В. Ефремов

Метод построения пошаговых ролевых игр на базе *XML*-скриптов и автоматов

Программирование с явным выделением состояний

Проектная документация

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

Санкт-Петербург
2004

Введение

С тех пор, как появились ролевые компьютерные игры, встал вопрос о программировании поведения персонажей, которыми управляет компьютер, а не игрок. Такие персонажи называются неигровым, или non-player characters (npc).

Наиболее распространенным решением для подобного рода задач являются **скрипты поведения** (behavior script) [1], которые позволяют описывать последовательность действий персонажа в зависимости от тех или иных факторов и событий. Поведение персонажей такого рода обычно характеризуется линейностью и предсказуемостью. При написании скриптов поведения такого рода нередко допускаются ошибки. Поэтому иногда "мертвые" персонажи продолжают совершать действия, реагируя на события.

В настоящей работе описан метод построения скриптов поведения (которые до этого строились эвристически) по графам переходов автоматов, являющихся наглядной спецификацией поведения объектов. С автоматным программированием можно ознакомиться на сайте http://is.ifmo.ru.

В работе изложен разработанный авторами формальный язык написания скриптов поведения по графам переходов. Для этого языка создан интерпретатор, позволяющий исполнять полученные скрипты. В заключение приведен пример игрового мира, наглядно иллюстрирующий все возможности и достоинства метода.

В этом примере в качестве основы для создания игрового пространства выбрана игра «Герои МатМеха» (Приложение 1), основной идеей которой является подготовка к сдаче и сдача всех зачетов одного из учебных семестров математико-механического факультета Санкт-Петербургского государственного университета в крайне сжатые сроки (одна неделя). Более подробная информация об игре приведена в указанном приложении.

Авторы решили написать аналогичную игру, воссоздав похожий игровой мир, но используя более близкую им атмосферу кафедры «Компьютерные технологии» факультета «Информационные технологии и программирование» Санкт-Петербургского государственного университета информационных технологий механики и оптики.

Этот игровой мир представляет собой фиксированный набор возможных местоположений персонажей (включая одного игрового персонажа, которым управляет пользователь), а также множество действий, которое он может совершать. Например, игровой персонаж может готовиться к некоторому предмету, общаться с другими персонажами, приобретая при этом знания, сдавать зачеты и экзамены, а также совершать ряд других повседневных действий, например, идти спать. Игровой персонаж обладает рядом параметров (знания по предметам, интеллект, настроение, усталость и т.п.), которые изменяются при совершении тех или иных действий. Игра заканчивается, если до определенного времени игрок успел сдать все зачеты и экзамены, в противном случае ему засчитывается поражение. Для обострения игровой ситуации, как и в прототипе, сохранение и восстановление в игре не предусмотрены.

Для реализации проекта была использована платформа *.NET*, которая обеспечивает наибольшую скорость и удобство разработки. При создании интерпретатора в качестве основного языка был использован язык C#, а для реализации предлагаемого формального языка - язык *XML* [2].

Исходный и исполняемый коды программы приведены на сайте http://is.ifmo.ru в разделе «Проекты».

1. Постановка задачи

Основной задачей проекта является разработка метода описания поведения персонажей в пошаговых ролевых играх. Для реализации метода ставятся следующие подзадачи:

- создание языка, удобного для работы в предметной области ролевых игр;
- построение интерпретатора и реализация скриптов поведения игрового и неигровых персонажей, описанных на созданном языке;

Подобный подход позволяет описать объектную модель игры, полностью отделив ее от внутренней структуры интерпретатора скриптов поведения. Отделение программы-исполнителя от конфигурации игрового мира и возможность изменения последнего без вмешательства в исходный код программы обеспечивает большую гибкость и возможность каждому пользователю настроить игровой мир «под себя».

Таким образом, создается программная база, которая позволяет проектировать аналогичные игровые миры при необходимости.

2. Метод описания поведения персонажей

Основным недостатком в описании поведения персонажей в ролевых пошаговых играх является то, что одновременно доступны все обработчики событий. Например, убитый персонаж может реагировать на попытки с ним заговорить, если в обработчике разговора не делалось проверки на допустимость подобного действия.

Авторы предлагают иной подход к программированию поведения персонажей, основанный на автоматном программировании (программировании с явным выделением состояний) [3].

Явное выделение состояний позволяет не допускать огрехов, подобных описанному выше. В аналогичной ситуации погибший персонаж переходит в состояние «мертвый», в котором на событие «разговор» просто не реагирует.

2.1. Основные положения

Основной единицей игрового мира является **объект**. Объектами являются все действующие лица игры, а также любые предметы, несущие смысловую нагрузку. Отметим, что здесь говорится об объектах, как о сущностях игрового мира, а не о термине объектно-ориентированного программирования.

Все объекты представляются в виде конечных автоматов с определенным числом состояний. Для каждого состояния объекта задается множество входных и выходных воздействий. В качестве входных воздействий используются **события**, а в качестве выходных воздействий – кратковременные **действия**.

Каждый объект обладает также рядом параметров, называемых **атрибутами** или **переменными** объекта. Основным атрибутом объекта (автомата) является его состояние. Набор и значения прочих атрибутов задаются для каждого объекта индивидуально.

2.2. Описание поведения персонажей

Основываясь на введенных понятиях, можно вывести следующую схему для построения описания поведения персонажей.

1. Из пространства игрового мира выделяются отдельные сущности.
2. Для сущностей строится схема их взаимодействия.
3. На основе этой схемы выделяются состояния объектов (автоматов).
4. Для каждого состояния задаются события и действия, связанные с этими событиями.
5. Полученные автоматы записываются на предлагаемом авторами формальном языке.

Таким образом, игровой мир разбивается на множество отдельных частей, реализация каждой из которых может независимо изменяться, качественно не влияя на остальные.

Полученное формальное описание мира, построенное в автоматной терминологии, может использоваться как база для дальнейшего программирования поведения объектов, так и как готовая к интерпретации программа.

Как отмечалось выше, в последующих разделах приводится пример языка для формального описания автоматов. Для этого языка построен интерпретатор, который позволяет трактовать описанную на этом языке модель поведения как конфигурацию игрового мира. Интерпретатор позволяет видеть работающую модель в действии, и (при наличии соответствующего визуального интерфейса) может быть использован как оболочка для самой игры.

3. Формальный язык для описания игрового мира. Основные идеи

В этом разделе приведена структура формального языка для описания поведения объектов. Пример реализации такого языка, построенный на основе языка *XML*, описан в разд. 5.

3.1. Течение игрового времени

Единицей измерения игрового времени является ход. Примерный состав хода рассматривается в разд. 4.5.

3.2. Объекты

В разд.2.1 отмечалось, что основным элементом игры является **объект**. Поведение объекта, в общем случае, описывается недетерминированным конечным автоматом с определенным набором состояний, так как в большинстве случаев моделируется поведение людей.

Как уже упоминалось, каждый объект игрового мира обладает определенным набором **переменных**, одна из которых кодирует состояния автомата, описывающего поведение объекта. В любом состоянии объект может реагировать на некоторые **события** (входные воздействия), каждое из которых имеет свой обработчик. Реакция бывает **локальной** (когда событие обрабатывается в одном или нескольких состояниях) и **глобальной** (когда оно обрабатывается вне зависимости от состояния).

3.3. Игровой персонаж

Игровой персонаж (или **герой**) является совершенно особым видом объекта. В отличие от остальных объектов, он управляется пользователем.

Для игрового персонажа в каждом состоянии определены **пользовательские команды**. Команда – это особый вид события, создаваемого пользователем. Реакция на пользовательскую команду также реализуется в **обработчике**.

3.4. Переменные и типы

Для каждого объекта может быть определен набор **переменных**. Они могут принимать различные значения в зависимости от их **типа**. Существующие типы переменных приведены на рис. 1.

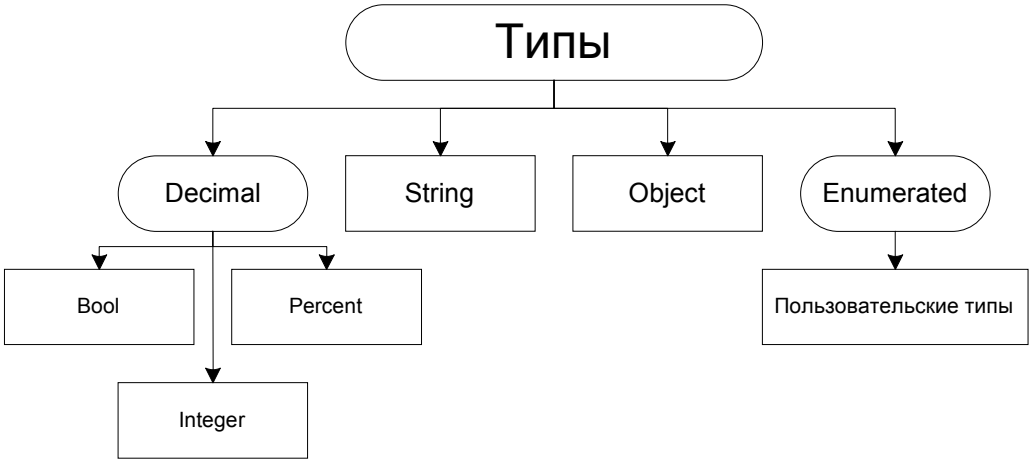


Рис. 1. Диаграмма типов

В интерпретаторе определены четыре базовых типа переменных:

- **Decimal**. Допустимые значения – вещественные числа с плавающей точкой. Для этого типа определен набор **подтипов**, каждый из которых является сужением вещественного типа на меньшую область значений:
  - **Integer.** Подтип целых чисел со знаком;
  - **Bool.** Булев тип. Принимает значения 0 и 1;
  - **Percent.** Процентный тип. Область значений – вещественные числа между нулем и единицей;
- **String**. Допустимые значения – строки;
- **Object**. Допустимые значения – объекты. Переменные этого типа позволяют осуществлять взаимодействия между объектами;
- **Enumerated**. Пользовательские перечислимые типы. Набор допустимых значений задается явно.

При объявлении переменной необходимо указать ее имя и тип (один из базовых или один из подтипов). Также возможно указать ее начальное значение.

3.5. События

Поведение объектов игры и взаимодействие между ними управляется с помощью **событий**. Каждое событие определяется именем и совокупностью параметров:

- **Активный объект**: объект, которому будет передано событие;
- **Объект взаимодействия**: объект, с которым осуществляется взаимодействие. Например, при обработке события "Сдача экзамена" объект-преподаватель может изменить состояние зачета объекта-студента;
- **Время обработки**: время, когда событие должно быть обработано объектом. Если при добавлении события указано время обработки, меньшее текущего, то событие никогда не будет обработано;
- **Период**: интервал между вызовами обработчиков события. Все события считаются периодическими, то есть выполняющимися каждые несколько ходов. Интервал указывается как количество ходов, через которые должно обрабатываться данное событие. После этого к его времени обработки прибавляется значение периода, и это событие снова помещается в очередь. Таким образом, для задания события, которое должно произойти только один раз, необходимо задать нулевой период. При этом время его следующей обработки никогда не наступит.

3.6. Пользовательские команды

Пользовательские команды – это особый вид событий, которые передаются игровому персонажу пользователем.

На каждом ходу пользователю предлагается выбор из подмножества команд, доступных для текущего состояния игрового персонажа. Команды бывают двух типов:

- **Команда действия**: при выборе команды этого типа выполняется связанный с ней обработчик и осуществляется очередной ход.
- **Команда взаимодействия**: при выборе команды этого типа пользователю предлагается список объектов, с которыми может осуществляться взаимодействие. Список строится по определенному **критерию**. Критерий состоит из интерфейса (разд. 2.8) и логического **условия**. В список попадают все объекты, реализующие указанный интерфейс и возвращающие ненулевое значение условия.

  Из списка объектов пользователь должен выбрать один. Ссылка на выбранный объект будет передана ассоциированному с командой обработчику.

3.7. Обработчики событий и команд

Обработчики описывают реакцию объекта на некоторое событие или пользовательскую команду. Поскольку пользовательские команды являются одним из видов событий, то обработчики для них реализуются одинаково.

Обработчики событий, как было отмечено выше, делятся на локальные и глобальные. Локальные обработчики задаются для каждого состояния в отдельности. Глобальные обработчики не зависят от состояния персонажа. Так как событие определяется только именем, то для одного и того же события могут существовать один или несколько локальных обработчиков и один глобальный обработчик.

Обработчики пользовательских команд объявляются только локально и всегда зависят от конкретного состояния игрового персонажа.

Любой обработчик представляет собой последовательность инструкций специального **скрипт-языка**. Детальное описание одного из вариантов реализации такого языка приведено в разд. 5.9.

3.8. Интерфейсы

Интерфейсы позволяют разделить объекты на группы и установить между ними отношения **наследования**. Каждому объекту сопоставляется один интерфейс (далее будем говорить, что объект его **реализует**). Различные объекты могут реализовать один и тот же интерфейс.

Интерфейс задается именем, совокупностью переменных и глобальных обработчиков событий. Для интерфейса определяется понятие наследования. В наследовании участвуют два интерфейса – **родительский** и **дочерний**. При наследовании дочерний интерфейс будет обладать как теми переменными и глобальными обработчиками, которые определены для родительского интерфейса, так и теми, которые определены для него самого.

Для наследования переменных и объектов определены два правила.

Правило наследования переменных. Если и в дочернем, и родительском интерфейсах объявлены переменные с одним и тем же именем, но разными значениями по умолчанию, то создается только одна переменная со значением, указанным в дочернем объекте.

Правило наследования глобальных обработчиков событий. Если в дочернем и родительском интерфейсах указаны обработчики одного и того же события, то обработчик

для дочернего интерфейса создается как объединение обоих обработчиков, причем обработчик из родительского интерфейса выполняется перед дочерним.

Это правило позволяет создавать обработчики событий, которые будут выполняться для всех объектов, реализующих данный интерфейс или любой из его дочерних интерфейсов.

Наследовать можно только один интерфейс. Если для некоторого интерфейса не указан его родительский интерфейс, то он считается унаследованным от общего интерфейса `Object`. Интерфейс `Object` обладает только строковой переменной `Name`, указывающей имя объекта.

Все правила наследования интерфейсов имеют силу и для реализации интерфейса объектом. Если объект реализует некий интерфейс, то будем говорить, что он реализует и все родительские по отношению к нему интерфейсы.

4. Интерпретатор

4.1. Устройство интерпретатора

На рис. 2 представлены отношения основных классов интерпретатора. Все манипуляции в игре совершает класс `Game`. Этот класс отвечает за основной ход игры.

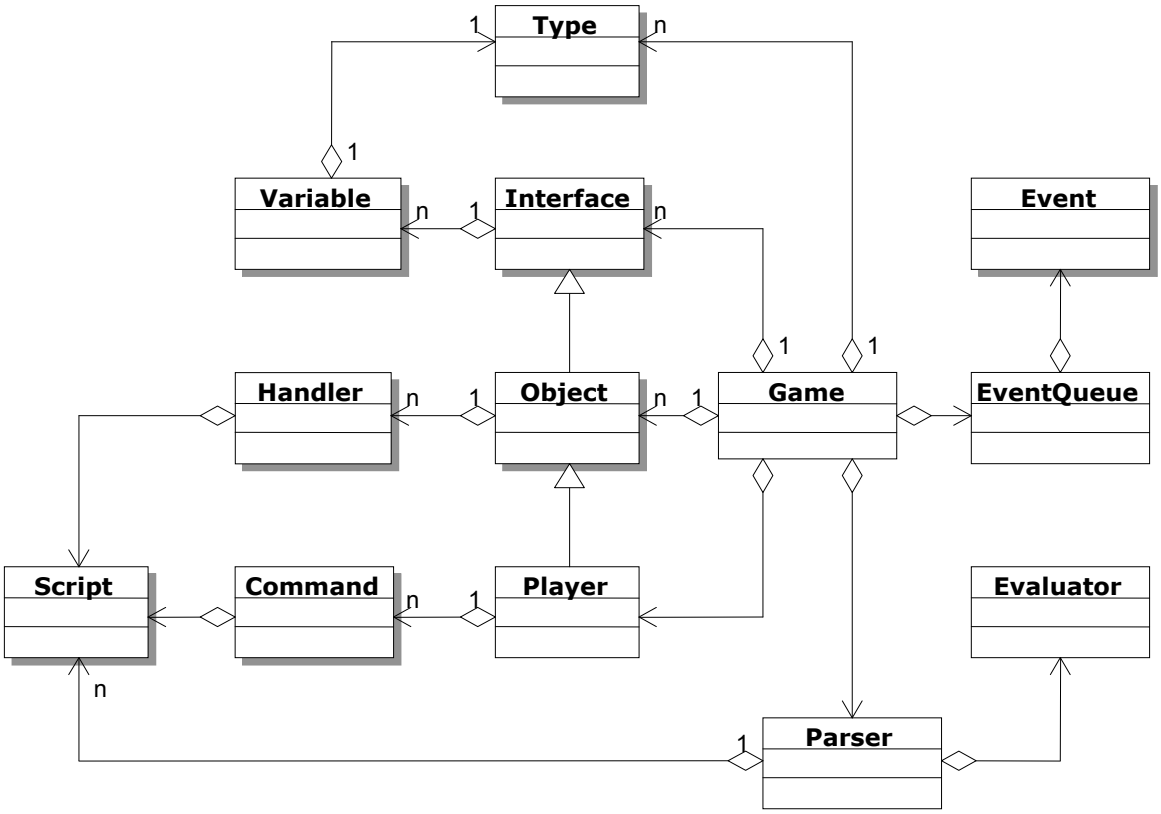


Рис. 2. Диаграмма классов интерпретатора

4.2. Парсер

Работа парсера заключается в переводе данных из формального описания (разд. 5) во внутреннее представление. В настоящей реализации интерпретатора предусмотрено два парсера – парсер конфигурационных файлов (`Parser`) и парсер выражений (`Evaluator`).

В процессе инициализации игры первый парсер разбирает конфигурационные файлы, проверяет содержащееся в них описание модели поведения на правильность и сохраняет полученную информацию в соответствующих структурах данных. На этом его работа завершается.

Парсер конфигурационных файлов реализован на базе *MSXML*-парсера, представляющего XML-файлы в виде удобной древовидной структуры данных.

Второй парсер служит для вычисления значений выражений, встречающихся в командах скрипт-языка (разд. 5). Поскольку значение подобного выражения зависит от контекста вызова, разбор совершается каждый раз при необходимости.

Парсер выражений реализован на базе контекстно-свободных грамматик, в частности, LL(1)-грамматики, разбор которой осуществляется наиболее просто и быстро [4].

4.3. Хранение данных

Все основные данные игры (типы, интерфейсы, объекты, переменные и состояния объектов) представляются в виде ассоциативных массивов. Доступ к данным осуществляется по ключу – имени соответствующей записи. Все эти структуры заполняются парсером конфигурационных файлов при инициализации программы.

Отношения агрегации, представленные на рис. 2, позволяют получить общее представление о хранении данных в игре.

4.4. Очередь событий

Для управления событиями в интерпретаторе введено понятие очереди событий. События для обработки извлекаются прямым просмотром очереди от начала до конца. Новые события добавляются в конец очереди, что позволяет им быть обработанными в тот же момент игры, когда они были добавлены. Подробнее извлечение и обработка событий изложены в разд. 4.5.

В предложенной авторами реализации каждому объекту обязательно передаются два события:

- `OnCreate`. Посылается один раз – при первом ходе;
- `OnTimer`. Обрабатывается в конце каждого хода.

4.5. Ход

Каждый ход делится на три фазы:

- глобальная обработка событий;
- обработка команд;
- локальная обработка событий.

На фазе глобальной обработки из **очереди событий** выбираются все те события, которые должны быть обработаны в текущий момент. Они (в том порядке, в котором они указаны в очереди) передаются соответствующим объектам для обработки. Время обработки для каждого события не изменяется для того, чтобы то же событие могло быть обработано в ходе третьей фазы. Если в процессе обработки было добавлено новое событие, которое должно быть обработано на этом ходе, то оно тоже будет обработано.

На второй фазе по текущему состоянию игрового персонажа строится и выдается пользователю набор доступных команд. Пользователь должен выбрать одну из них. После этого осуществляется вызов обработчика выбранной команды.

Для команды может быть указана ее длительность в ходах. По умолчанию, длительность команды равна одному ходу. Если задана нулевая длительность, то после выполнения соответствующего обработчика (внутри которого состояние игрового персонажа может измениться) вторая фаза будет повторена. Если же указана длительность, большая единицы, то соответствующее количество ходов игрок опрашиваться не будет. Этой возможностью рекомендуется пользоваться с особой осторожностью, так как она может стать причиной непредсказуемых результатов.

Если на данной фазе игровой персонаж находится в состоянии, для которого ни одной команды не задано (например, в состоянии «сон»), игрок также не опрашивается, и происходит переход непосредственно к третьей фазе.

Третья фаза – локальная обработка событий. Снова просматривается список событий, и из него выбираются те, которые должны быть обработаны на текущем ходе. Для каждого события рассматривается связанный с ним объект. Если в текущем состоянии объекта присутствует локальный обработчик этого события, то он вызывается. Затем время

обработки события увеличивается на его период. Если период равен нулю, то событие удаляется из списка.

После окончания третьей фазы происходит увеличение счетчика ходов (игрового времени) и переход к началу следующего хода.

4.6. Пользовательский интерфейс

В главном окне игры выводится информация о состоянии персонажа и предоставляется возможность выбора очередной команды. Если для команды требуется указание объекта взаимодействия, то всплывает окно, в котором выводится список всех объектов в игре, удовлетворяющих критерию выбора, и пользователю дается возможность выбрать один из них. В нижней части основного окна выводится лог для игрового персонажа. Из меню доступна возможность вывода логов для других объектов в отдельных окнах.

На рис. 3 приведен скриншот программы в работе. Пользователю здесь в качестве примера предлагается указать объект взаимодействия для выбранной им команды «Говорить». Также в отдельных окнах выведены логи для объектов `teacher` и `student1`.

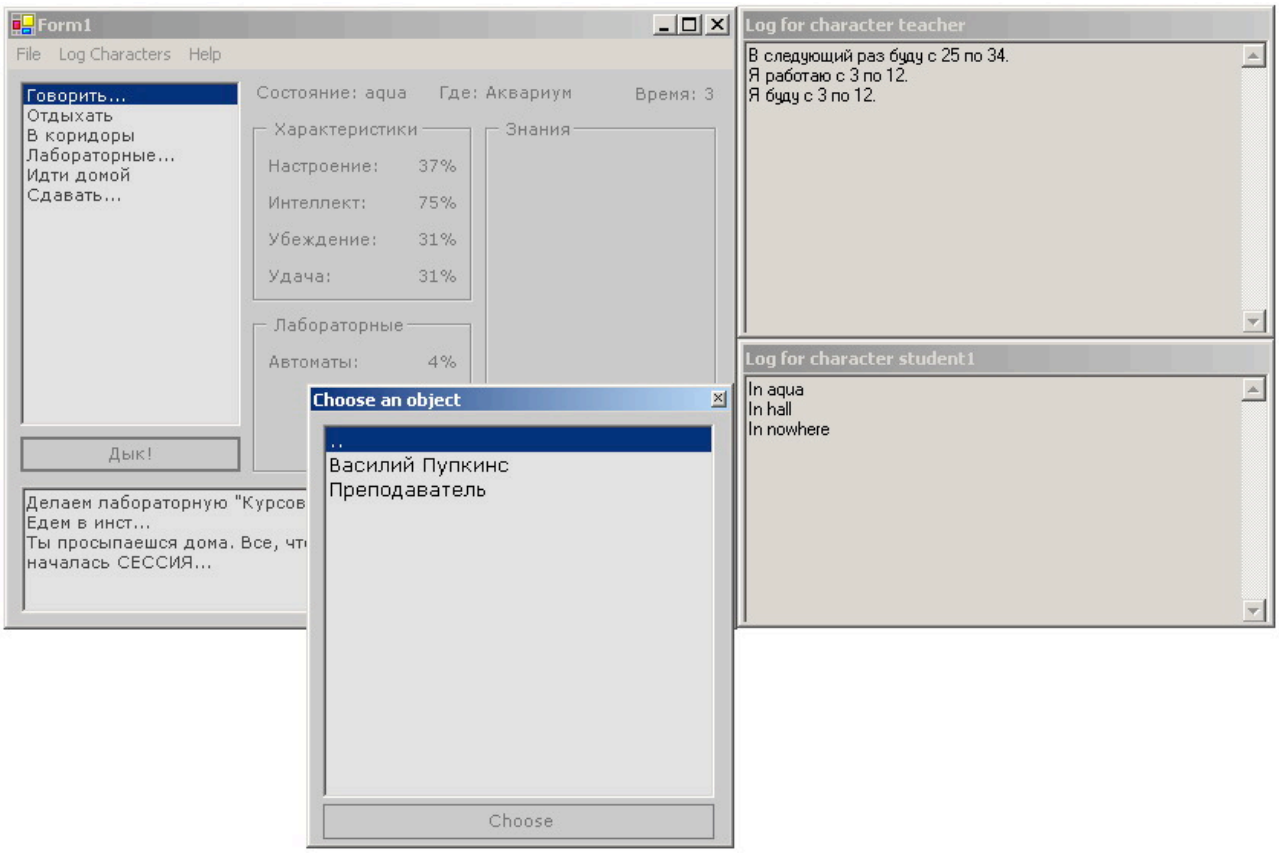


Рис. 3. Пользовательский интерфейс

5. Конфигурирование игрового мира

5.1. Основная конфигурация

Для описания конфигурации игрового мира используется язык тэговой разметки *XML*, позволяющий жестко задавать и хранить структурированную информацию. Также язык *XML* очень удобен для описания конечных автоматов (http://is.ifmo.ru). Примером такого описания является предложенный в работе [5] формат для задания внешнего вида видеопроигрывателя *Crystal Player*.

Вся конфигурация игры описывается в одном файле с названием `config.xml`. Для отдельных тэгов может быть определен атрибут `import`, который указывает, что содержимое тэга объявлено в другом файле. Имя файла задается значением атрибута. Такие тэги будем называть **импортируемыми**.

Тэги, группирующие информации в отдельные части, будем называть **блоками**. При описании формата тега необязательные атрибуты заключаются в квадратные скобки.

Структура конфигурационного файла приведена на рис. 4. Корневым элементом является тэг `config`. Он имеет три дочерних импортируемых тэга (блока), описывающих типы и интерфейсы, объекты и действия соответственно.

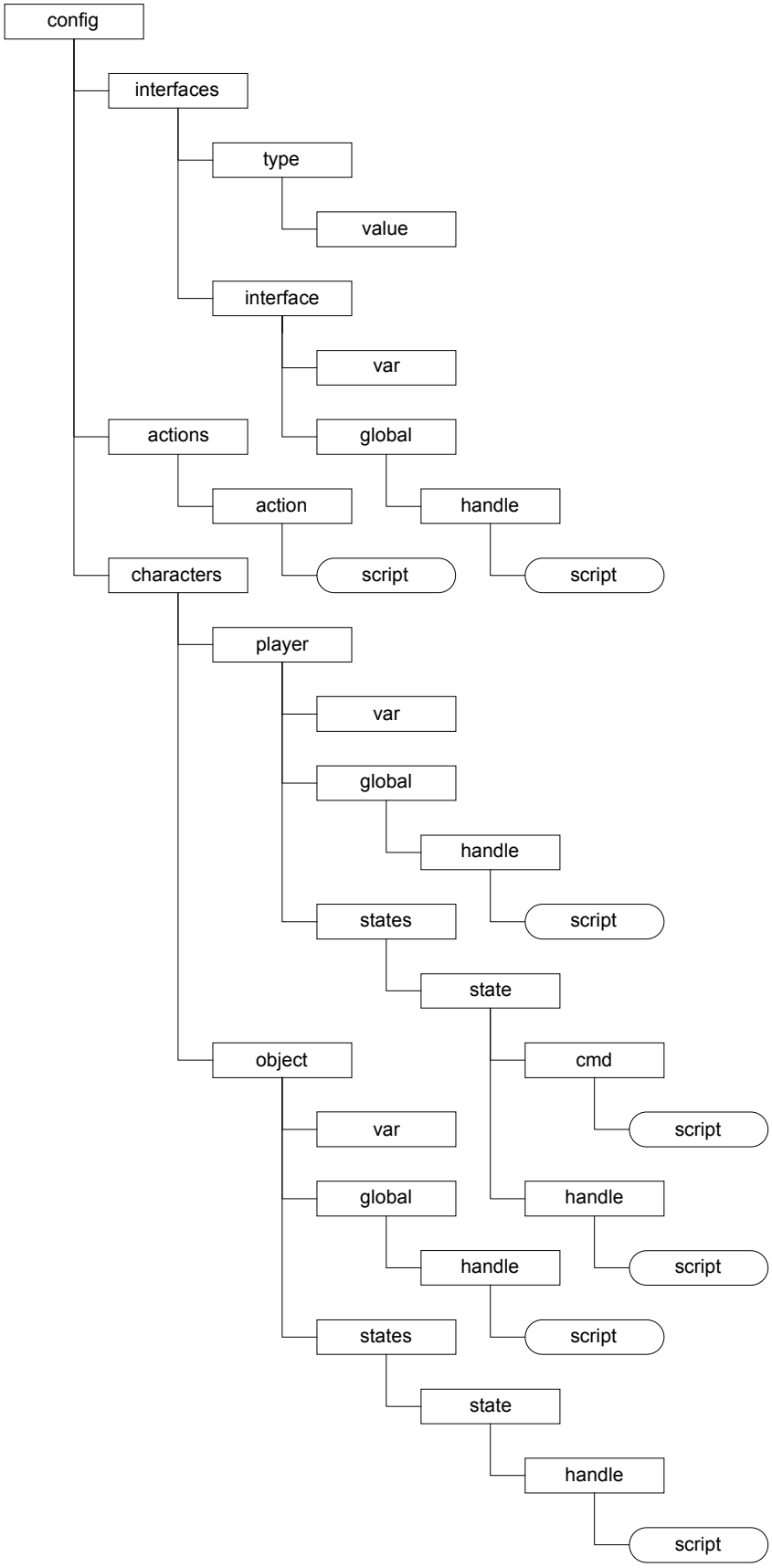


Рис. 4. Структура конфигурационного файла

5.2. Объявление пользовательских типов

В блоке `interfaces` объявляются пользовательские перечислимые типы и интерфейсы. Тип задается блоком `type` с обязательным атрибутом `name` – именем типа. Формат блока имеет следующую структуру:

```
<type name="type-name">
  <value name="type-value" [ display="display-string"] />
  ...
</type>
```

Внутри блока задаются допустимые значения для типа. Каждое значение описывается тегом `value`. Тэг обладает атрибутами, приводимыми ниже.

Атрибуты тэга <value>

Атрибут	Значение	Описание
`name`	имя значения	допустимое значение для определяемого типа
`display`	строка	текстовое представление для значения

5.3. Объявление переменных

Переменные принадлежат интерфейсам и объектам и объявляются в соответствующих блоках. Формат объявления переменной:

```
<var name="variable-name" [ type="type-name"] [ value="equation"] />
```

Атрибуты тэга <var>

Атрибут	Значение	Описание
`name`	имя переменной	имя переменной
`type`	имя типа	имя типа; может быть одним из базовых типов, подтипом или пользовательским перечислимым типом
`value`	выражение	начальное значение; тип выражения должен соответствовать указанному типу

5.4. Объявление интерфейсов

Интерфейсы также объявляются в блоке `interfaces`. Общий их формат можно представить следующей структурой:

```
<interface name="interface-name" [ extends="interface-name"] >
  { объявление переменных}
  ...
  <global>
    { объявление глобальных обработчиков событий}
  </global>
</interface>
```

Блок `interface` содержит объявления переменных и глобальных обработчиков событий. Глобальные обработчики задаются внутри блока `global`. Формат их задания более подробно описан в п. 0. Количество переменных и обработчиков событий не ограничено.

Атрибуты тэга <interface>

Атрибут	Значение	Описание
name	имя интерфейса	имя интерфейса
extends	имя интерфейса	имя родительского интерфейса; если не указано, родительским считается интерфейс Object

5.5. Объявление объектов

Объекты задаются в блоке characters. Общая структура объявления объекта сходна с объявлением интерфейса, однако содержит также и описание состояний объекта.

```
<object interface="interface-name">
  {объявление переменных}
  ...
  <global>
    {объявление глобальных обработчиков событий}
  </global>

  <states [init="state-name"]>
    <state name="state-name">
      {объявление локальных обработчиков событий}
    </state>
    ...
  </states>
</object>
```

Помимо переменных и блока глобальных событий, объект содержит также конечный набор состояний. Каждое состояние описывается блоком state, все блоки state заключаются в общий блок states. Для каждого состояния объявляется набор обработчиков локальных событий, которые будут вызываться в этом состоянии.

Атрибуты тэга <object>

Атрибут	Значение	Описание
interface	имя интерфейса	интерфейс, который реализуется объявляемым объектом

Атрибуты тэга <states>

Атрибут	Значение	Описание
init	имя состояния	состояние, в котором объект находится в начале игры; если не указано, то начальным считается первое по порядку объявления состояние

Атрибуты тэга <state>

Атрибут	Значение	Описание
name	имя состояния	имя объявляемого состояния

5.6. Объявление игрового персонажа

Поскольку игровой персонаж является объектом, объявляется он абсолютно также. Есть лишь два различия:

- тэг `object` заменяется на тэг `player`;
- в блоке `state` помимо локальных обработчиков присутствуют также объявления пользовательских команд.

Блок состояния для игрового персонажа выглядит следующим образом:

```
<state name="state-name">
  {объявление локальных обработчиков событий}
  ...
  {объявление обработчиков команд}
  ...
</state>
```

Объявление обработчиков команд более подробно описано в разд. 3.8.

5.7. Объявление обработчиков событий

Локальные и глобальные обработчики описываются блоком `handle` со следующей структурой:

```
<handle event="event-name">
  {инструкции скрипт-языка}
  ...
</handle>
```

Атрибуты тэга <handle>

Атрибут	Значение	Описание
`event`	имя события	событие, ассоциированное с обработчиком

Внутри блока находится **скрипт**, написанный на скрипт-языке.

5.8. Объявление обработчиков команд

Обработчики пользовательских команд задаются двумя различными блоками в зависимости от типа команды.

Команды действия описываются блоком `cmd`:

```
<cmd name="command-name">
  {инструкции скрипт-языка}
  ...
</cmd>
```

Атрибуты тэга <cmd>

Атрибут	Значение	Описание
`Name`	имя команды	название команды, которое будет показано пользователю

Команды взаимодействия описываются блоком cmdgroup:

```
<cmdgroup name="command-name" [ filter="interface-name"]
                              [ test="decimal-equation"]>
  { инструкции скрипт-языка}
  ...
</cmdgroup>
```

При выборе пользователем команды этого типа ему будет предложен список из возможных объектов для взаимодействия (разд. 3.6). Построение списка объектов осуществляется по критериям, задаваемым в атрибутах filter и test.

Атрибуты тэга <cmdgroup>

Атрибут	Значение	Описание
name	имя команды	название команды, которое будет показано пользователю
filter	имя интерфейса	интерфейс, который обязаны реализовывать выбираемые объекты; если интерфейс не указан, объекты фильтруются по интерфейсу Object
test	десятичное выражение	условие, которое должно выполняться для объекта; если атрибут не указан, выполнение условия для объектов не проверяется

Выборка объектов для взаимодействия происходит следующим образом: просматриваются все объекты (кроме игрового персонажа) и проверяется, реализуется ли объектом интерфейс, указанный в атрибуте filter. В случае положительного ответа вычисляется значение выражения из атрибута test, и если оно отлично от нуля, объект добавляется в список.

5.9. Скрипт-язык

Скрипт-язык используется для описания реакции персонажа на события. Скрипт-язык состоит из отдельных инструкций. Каждая инструкция может описывать изменение переменных объекта, добавление события, вывод информации в журнал и т.п. Существуют специальные управляющие инструкции, позволяющие создавать условные конструкции и ветвления.

В качестве скрипт-языка, как и для описания конфигурации, используется *XML*.

Инструкции скрипт-языка бывают следующих видов:

Инструкция установки значений переменных

```
<set var="variable-equation" value="equation"/>
```

Инструкция set устанавливает значение переменной, имя которой задается атрибутом var. Новое значение задается выражением, указанным в атрибуте value. Если базовые типы переменной и выражения не совпадают, то выдается ошибка.

Атрибуты инструкции <set>

Атрибут	Значение	Описание
var	выражение-переменная	выражение, описывающее переменную
value	выражение	выражение того же базового типа, что и переменная

Инструкция добавления события

```
<hang event="event-name" at="decimal-equation" [ period="period-value"]
      [ this="object-equation"] [ that="object-equation"] />
```

Инструкция hang добавляет новое событие в конец списка событий.

Атрибуты инструкции <hang>

Атрибут	Значение	Описание
event	имя события	имя ассоциированного события
at	десятичное выражение	ход, на котором должно быть обработано событие
period	период	период передачи события на обработку; в качестве значения может быть указана числовая константа или одно из ключевых слов: once (только раз), tick(каждый ход), hour(каждый час), day(каждый день).
this	выражение-объект	объект, ассоциированный с событием; если атрибут не указан, используется объект, для которого выполняется скрипт
that	выражение-объект	объект взаимодействия для события; если атрибут не указан, то событию он не ассоциируется

Инструкции вывода

```
<log format="format-string" [ param1="variable-equation"] [ ...]
     [ target="object-expression"] />
```

Каждый объект имеет журнал сообщений, позволяющий отслеживать его состояние и поведение. Журнал для всех объектов, кроме игрового персонажа, несет чисто отладочный характер и в игре не показывается. Журнал игрового персонажа доступен во время игры.

Инструкция log добавляет сообщение, сформированное по указанному формату и набору переменных, в журнал объекта, указанного в атрибуте target.

Атрибуты инструкции <log>

Атрибут	Значение	Описание
format	строка форматирования	строка форматирования; обычная строка, в которой можно указывать места подстановки значений переменных в виде {*n*}, где *n* – номер переменной
param*n*	выражение-переменная	набор переменных, которые будут подставлены в строку форматирования
target	выражение-объект	объект, в чей журнал будет занесено сообщение; если атрибут не указан, сообщение добавляется в журнал объекта, для которого вызван скрипт

Управляющие инструкции

Условная инструкция if вычисляет значение выражения и, если оно отлично от нуля, выполняет инструкции, описанные внутри блока.

```
<if test="decimal-equation">
  {инструкции скрипт-языка}
  ...
</if>
```

Инструкция выбора switch вычисляет значения выражений в атрибутах test блоков case до первого несовпадения с нулем. Как только это произойдет, будут выполнены все инструкции соответствующего блока case. Если все условия оказались равны нулю и в блоке switch присутствует блок default, то выполняются все инструкции из блока default. Таким образом, блок default аналогичен блоку case с условием "1". После выполнения всех инструкций одного из блоков происходит выход их инструкции switch.

```
<switch>
  <case test="decimal-equation">
    {инструкции скрипт-языка}
    ...
  </case>
  <case test="decimal-equation">
    {инструкции скрипт-языка}
    ...
  </case>

  [<default>
    {инструкции скрипт-языка}
    ...
  </default>]
</switch>
```

Атрибуты инструкций <if>, <case>

Атрибут	Значение	Описание
test	десятичное выражение	условие выполнения набора инструкций внутри блока

Инструкция вызова

Инструкция `call` выполняет все инструкции действия с указанным в атрибуте name именем. Подробнее действия описаны в разд. 5.10.

```
<call name="action-name"/>
```

Атрибуты инструкции <call>

Атрибут	Значение	Описание
name	имя действия	имя действия, инструкций которого должны быть выполнены

5.10. Действия

Действие представляет собой последовательный набор инструкций, объединенных в блок с указанным именем. Действия хранятся в импортируемом блоке `actions`:

```
<actions>
  <action name="action-name">
    {инструкции скрипт-языка}
    ...
   </action>
   ...
</actions>
```

Действия позволяют хранить часто используемые последовательности инструкций в отдельных блоках и обращаться к ним по именам. Это может существенно сократить размеры конфигурационных файлов.

5.11. Выражения

Выражения используются для вычисления значений переменных и параметров инструкций, зависящих от внешних условий. Например, в выражениях может применяться генератор случайных чисел, счетчик ходов и т.д.

Все вычисления ведутся в базовых типах. Подтипы используются только при присвоении значений переменным. Если полученное значение выходит за границы допустимых значений подтипа, то оно округляется до ближайшего допустимого значения.

Для описания синтаксически допустимых выражений в работе применяется форма Бэкуса – Наура. Ниже приведена грамматика, задающая правила построения выражений.

```
equation:
    variable-equation
    object-equation
    decimal-equation
    string-equation
    enum-equation

variable-equation:
    identifier
    object-equation -> identifier

object-equation:
    function
    variable-equation
    object-literal

object-literal:
    this
    that
    null

function:
    identifier ( )
    identifier ( params-list )

params-list:
    equation
    params-list , equation

string-equation:
    variable-equation
    function
    string-literal
    ( string-equation )
    string-equation . string-equation            оператор конкатенации

string-literal:
    ' string-characters '

string-characters:
    Любой символ кроме перевода строк и апострофа

identifier:
    alpha
    alpha alphanumeric-sequence

alphanumeric-sequence
    alphanumeric
    alphanumeric-sequence alphanumeric

digit-sequence:
    digit
    digit-sequence digit

number
    digit-sequence
    digit-sequence . digit-sequence

alpha:
    один из  _  A  ...  Z  a  ...  z

digit:
    один из  0  1  ... 9
```

```
alphanumeric:
    alpha
    digit

enum-equation:
    variable-equation
    enum-literal

enum-literal:
    # alphanumeric-sequence

decimal-equation:
    decimal-or-equation

decimal-or-equation:
    decimal-and-equation | decimal-and-equation

decimal-and-equation:
    decimal-equality-equation & decimal-equality-equation

decimal-equality-equation:
    decimal-arithmetic-equation == decimal-arithmetic-equation
    decimal-arithmetic-equation != decimal-arithmetic-equation
    decimal-arithmetic-equation <  decimal-arithmetic-equation
    decimal-arithmetic-equation <= decimal-arithmetic-equation
    decimal-arithmetic-equation >  decimal-arithmetic-equation
    decimal-arithmetic-equation >= decimal-arithmetic-equation

decimal-arithmetic-equation:
    + decimal-arithmetic-equation
    - decimal-arithmetic-equation
    decimal-plus-equation

decimal-plus-equation:
    decimal-mul-equation + decimal-mul-equation
    decimal-mul-equation - decimal-mul-equation

decimal-mul-equation:
    decimal-not-equation * decimal-not-equation
    decimal-not-equation / decimal-not-equation

decimal-not-equation:
    !decimal-not-equation
    ( decimal-equation )
    decimal-literal
    variable-equation
    function

decimal-literal:
    number
```

Объектные литералы `this` и `that` являются именами текущего объекта и объекта взаимодействия соответственно (в случае, когда последний определен). Литерал `null` не соответствует ни одному из существующих объектов и может быть трактован как «неопределенное значение».

Логическая операция '|' возвращает в качестве результата первый ненулевой аргумент, или ноль, если оба аргумента равны нулю.

Логическая операция '&' возвращает в качестве результата первый аргумент, если оба аргумента не равны нулю, и ноль - в противном случае.

Логическая операция '!' возвращает в качестве результата единицу, если аргумент был равен нулю, и ноль - в противном случае.

Все операции сравнения имеют своим результатом единицу, если равенство или неравенство выполнено, и ноль - в противном случае.

Подробнее о лексическом и синтаксическом разборе можно узнать из работы [4]. Также некоторое представление о них можно получить и из работы [6].

Функции

В выражениях доступны следующие функции:

`now()`	возвращает номер текущего хода
`rand()`	возвращает случайное число от 0.0 до 1.0
`eqstr(str1, str2)`	возвращает 1, если значения аргументов (выражений строкового типа) совпадают, и 0 в противном случае
`eqenum(enum1, enum2)`	возвращает 1, если значения аргументов (выражений перечислимого типа) совпадают, и 0 в противном случае
`eqobj(enum1, enum2)`	возвращает 1, если значения аргументов (выражений объектного типа) совпадают, и 0 в противном случае
`int(decimal1)`	округляет значение десятичного выражения до ближайшего целого
`ftos(decimal1)`	преобразует аргумент (десятичное выражение) к строковому типу
`stod(str1)`	преобразует максимально длинный префикс аргумента-выражения строкового типа к десятичному

6. Пример игрового мира

6.1. Общее описание

Для демонстрации возможностей интерпретатора приводится простейший игровой мир. Он состоит из игрового персонажа (`героя`), еще одного `студента`, `преподавателя`, `курсовой работы` и `амулета`.

Для того чтобы получить зачет и выиграть игру, `герой` должен сдать `преподавателю курсовую работу`. Готовность работы измеряется в процентах. Работа принимается преподавателем, когда ее готовность оценивается более чем 95 процентами. Готовность работы можно увеличивать двумя способами: приготовлением работы дома и сдачей работы преподавателю.

Чтобы сдавать работу преподавателю, необходимо назначить встречу, поговорив с ним. При сдаче работы ее готовность может, как увеличиться, так и уменьшиться в зависимости от характеристик героя и случайных факторов.

Герой может общаться со студентом, и, в зависимости от случайных факторов, студент либо рассказывает герою анекдот (настроение героя повышается), либо дарит герою амулет, повышающий удачу (если этот амулет еще у студента), либо студент молчит.

Все объекты игрового мира реализуют соответствующие интерфейсы. Иерархия интерфейсов приведена на рис. 5. Листинг файла конфигурации для этого игрового мира в силу большого объема вынесен в приложение 2.

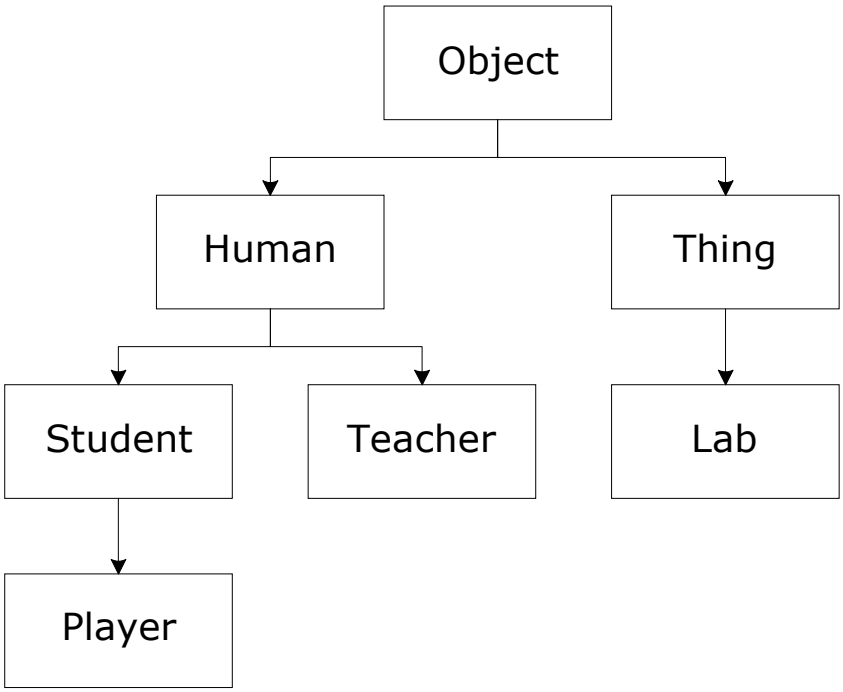


Рис. 5. Иерархия интерфейсов в игровом мире

6.2. Игровой персонаж

Игровой персонаж реализует интерфейс `Student`, поэтому часть его переменных унаследовано от этого интерфейса.

Переменные интерфейса 'Student'

Переменная	Тип	Описание
`k_program`	`integer`	знание программирования
`mood`	`integer`	настроение
`intelligence`	`integer`	интеллект
`charism`	`integer`	сила убеждения
`luck`	`integer`	удача

Игровой персонаж имеет также еще одну переменную `passed`, которая играет роль зачетки. По значению этой переменной определяется выигрыш или проигрыш пользователя.

Переменные объекта 'player'

Переменная	Тип	Описание
`passed`	`bool`	наличие сданного зачета

Игровой персонаж имеет три состояния (`home`, `ifmo`, `aqua`). Каждое состояние привязано к его местоположению. Диаграмма переходов, предназначенная для построения конфигурационных файлов, приведена на рис. 6.

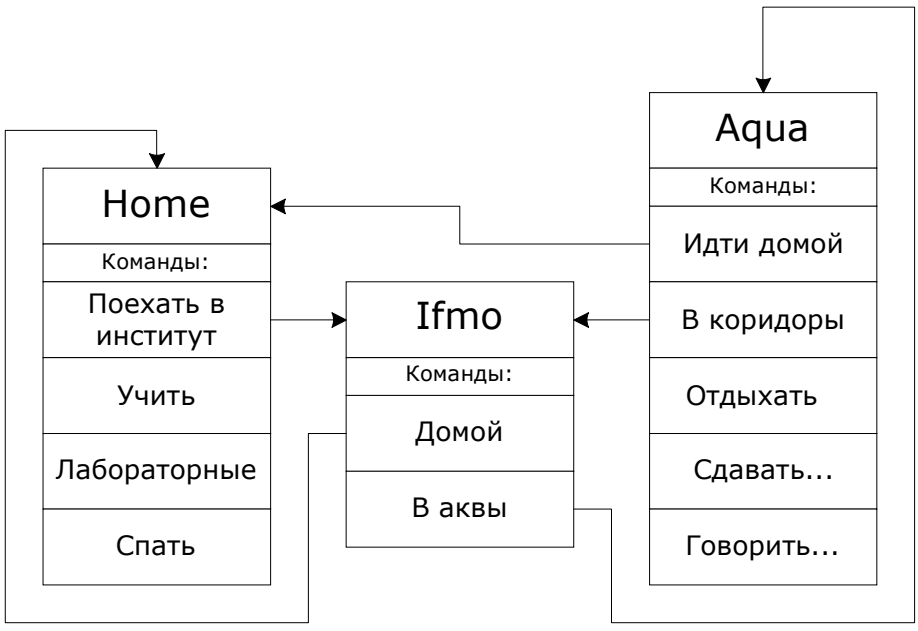


Рис. 6. Диаграмма переходов для объекта 'player'

В приведенной ниже таблице описываются команды для каждого состояния. Для команд взаимодействия в графе «Взаимодействие» приводится название интерфейса, играющего роль критерия (разд. 2.6).

Состояние			**Описание**
Home			герой находится дома
	Команда	**Взаимодействие**	**Действие**
	В институт		переход в институт [ifmo]
	Лабораторные…	Lab	готовить курсовую/лабораторную работу; настроение понижается
	Отдыхать		ничего не делать; настроение повышается
	Учить		изучать предмет; скорость изучения зависит от настроения, настроение понижается
Ifmo			герой находится в коридорах ЛИТМО
	Команда	**Взаимодействие**	**Действие**
	В аквариум		переход в аквариум [aqua]
	Домой		переход домой [home]
	Говорить…	Human	разговор с объектом, находящимся в коридоре
Aqua			герой находится в компьютерном классе
	Команда	**Взаимодействие**	**Действие**
	Идти домой		переход домой [home]
	В коридоры		переход в институт [ifmo]
	Лабораторные…	Lab	готовить курсовую/лабораторную работу; подготовка проходит медленнее, чем дома; понижает настроение
	Отдыхать		ничего не делать; повышает настроение
	Говорить…	Human	разговор с человеком, находящимся в компьютерном классе
	Сдавать…	Teaher	сдача курсовой преподавателю, находящемуся в компьютерном классе

Граф переходов игрового персонажа в стандартной форме приведен на рис. 7.

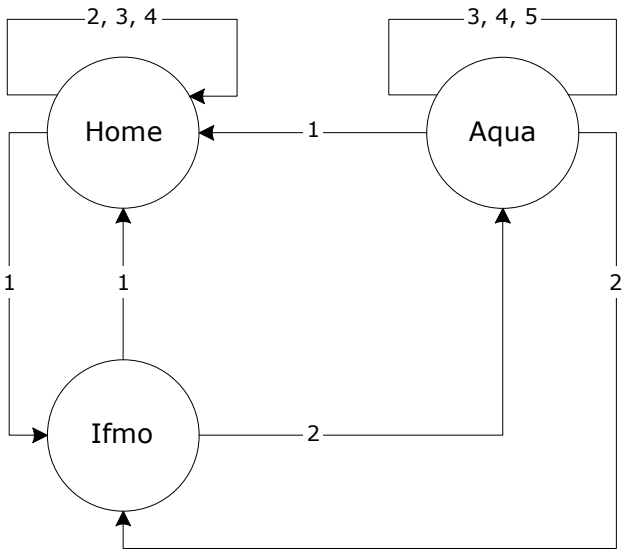


Рис. 7. Граф переходов игрового персонажа в стандартной форме

Для того чтобы не загромождать рисунок, команды, соответствующие переходам, отмечены на графе цифрами, соответствующими их номерам в приведенной выше таблице. Например, цифра 3 на стрелке, которая ведет из состояния "Home", обозначает третью команду в этом состоянии - "Отдыхать".

6.3. Преподаватель

Преподаватель имеет три состояния: отсутствует(away), занят (busy), готов к приему (ready). Ниже описываются переменные этого объекта.

Переменные объекта 'teacher'

Переменная	Тип	Описание
will_come	Integer	время следующего прихода преподавателя
will_away	Integer	время следующего ухода преподавателя
will_ready	Integer	время начала приема курсовой работы у героя
will_finish	Integer	время окончания приема курсовой работы у героя

Диаграмма переходов для преподавателя приведена на рис. 8.

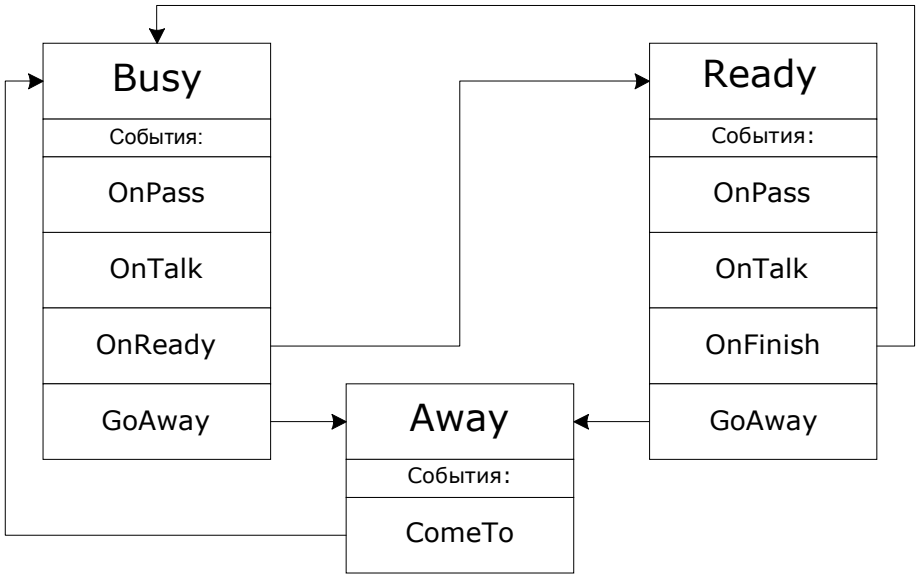


Рис. 8. Диаграмма переходов для объекта 'teacher'

В приведенной ниже таблице описываются компоненты графа переходов.

Состояние			**Описание**
Away			преподаватель отсутствует
	Событие	**Причина**	**Действие**
	OnCome	время идти в институт	переход в институт [busy] и назначение нового времени прихода/ухода
Busy			преподаватель занят
	Событие	**Причина**	**Действие**
	OnAway	время уходить	покинуть институт и перейти в состояние away
	OnReady	время начала приема	перейти в состояние приема
	OnPass	герой хочет сдавать курсовую работу	вежливо отказать
	OnTalk	герой хочет говорить	назначить встречу или сообщить об уже назначенной встрече
Aqua			преподаватель готов принимать курсовую
	Событие	**Причина**	**Действие**
	OnFinish	время окончания приема	перейти в состояние занятости
	OnPass	герой хочет сдавать курсовую работу	изменить готовность курсовой работы в зависимости от характеристик героя
	OnTalk	герой хочет говорить	сообщить о возможности сдачи курсовой работы

Граф переходов преподавателя в стандартной форме приведен на рис. 9. События на рисунке обозначены так же, как команды на графе переходов героя.

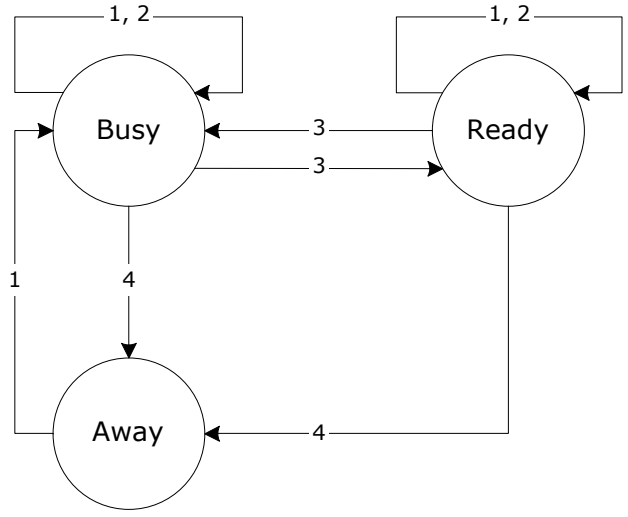


Рис. 9. Граф переходов преподавателя в стандартной форме

6.4. Студент

Объект "студент" имеет четыре состояния: `nowhere`, `hall`, `auditory` и `hall`, которые привязаны к его местоположению. Этот объект реализует интерфейс `Student`. Все его переменные унаследованы от этого интерфейса.

Диаграмма переходов студента приведена на рис. 10.

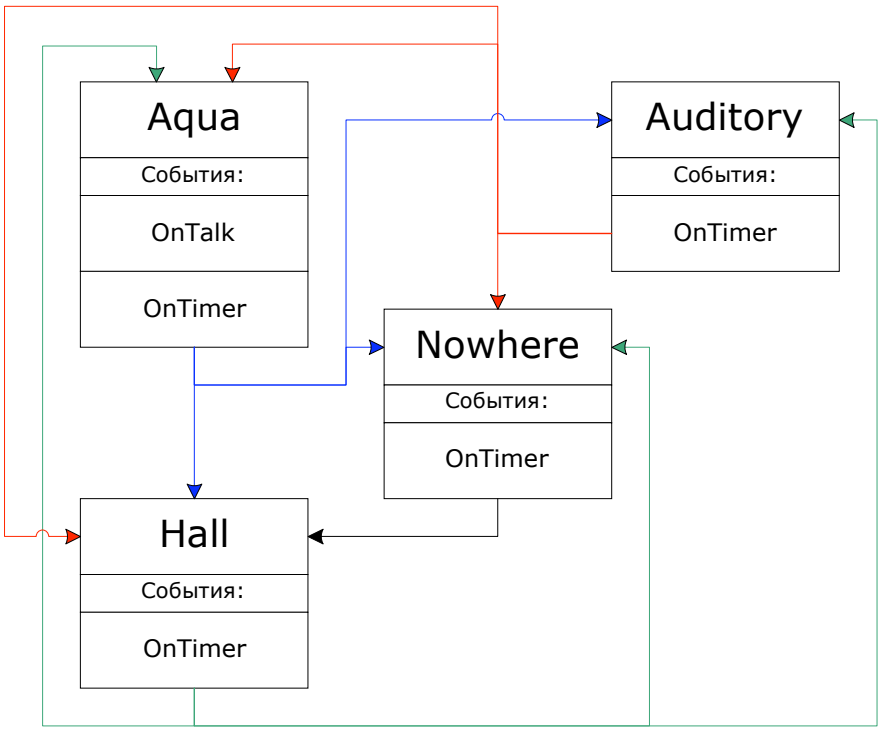


Рис. 10. Диаграмма переходов для объекта ‘student1’

В приведенной ниже таблице описываются компоненты диаграммы переходов:

Состояние			**Описание**
Aqua			студент находится в компьютерном классе
	Событие	**Причина**	**Действие**
	OnTalk	герой хочет говорить	молчать, рассказать анекдот или подарить амулет
	OnTimer	ход закончен	остаться, перейти в аудиторию [auditory], уйти домой [nowhere] или пойти в коридор [hall]
Auditory			студент находится в аудитории
	Событие	**Причина**	**Действие**
	OnTimer	ход закончен	остаться, перейти в компьютерный класс [aqua], уйти домой [nowhere] или пойти в коридор [hall]
Nowhere			студент находится дома
	Событие	**Причина**	**Действие**
	OnTimer	ход закончен	остаться или поехать в ЛИТМО [hall]
Hall			студент находится в коридорах ЛИТМО
	Событие	**Причина**	**Действие**
	OnTimer	ход закончен	остаться, перейти в компьютерный класс [aqua], уйти домой [nowhere] или пойти в аудиторию [auditory]

Граф переходов студента в стандартной форме приведен на рис. 11. Цифрами обозначены события. Конечный автомат для этого объекта является недетерминированным и выбор нового состояния зависит от случайных факторов.

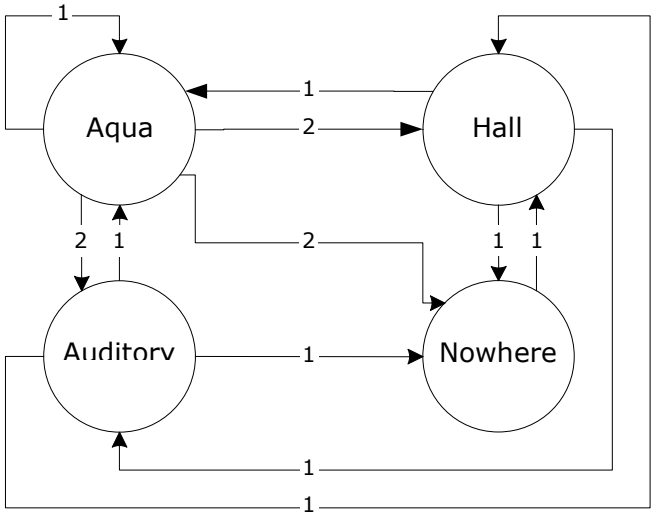


Рис. 11. Граф переходов студента в стандартной форме

6.5. Лабораторная работа

Лабораторная работа является объектом, реализующим интерфейс `Lab`. Она имеет только одно состояние и переменные, указанные ниже.

Переменные объекта 'lab'

Переменная	Тип	Описание
`Done`	Percent	степень готовности работы
`Step`	Percent	скорость приготовления работы; определяет долю работы, которая может быть сделана за один ход

6.6. Амулет

Амулет является объектом, реализующим интерфейс `Thing`. Он имеет только одно состояние и унаследованную от этого интерфейса переменную объектного типа `owner`. Эта переменная указывает на объект, являющийся в данный момент хозяином амулета. Амулет добавляет хозяину 10% удачи.

Амулет реагирует на событие `OnGive`. При получении этого события он меняет своего хозяина с текущего на объект взаимодействия. При этом производятся изменения характеристик объектов, связанные с эффектом от амулета.

7. Заключение

Результатом проекта можно считать подтверждение целесообразности применения автоматов для описания моделей поведения персонажей ролевых игр.

При этом удалось отделить сами модели поведения от внутренней структуры игры, что должно позволить создавать различные игры на основе одного "движка" без изменения программного кода.

8. Источники

[1] ***Behaviour scripts*** http://www.bioware.com

[2] ***Extensible Markup Language*** (XML) http://www.w3.org/XML

[3] ***Автоматное программирование*** http://www.is.ifmo.ru

[4] ***Ахо А., Сети Р., Ульман Д.*** Компиляторы. Принципы, технологии, инструменты. М.: Вильямс, 2001.

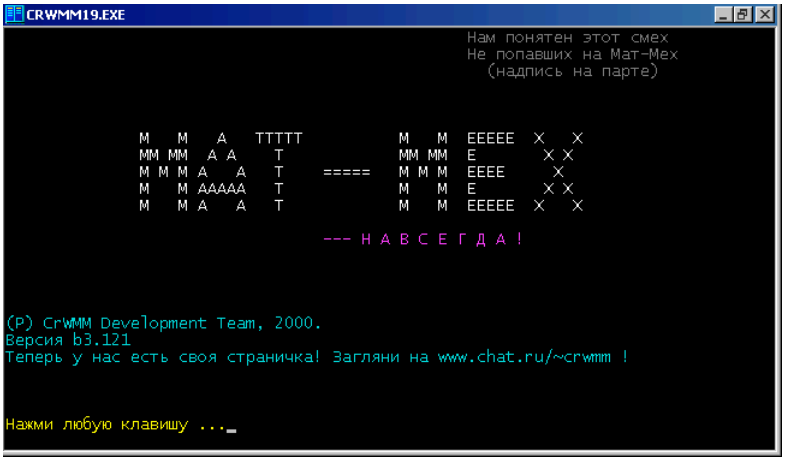
[5] ***Бондаренко К.А., Шалыто А.А.*** Разработка XML - формата для описания внешнего вида видеопроигрывателя с использованием конечных автоматов. http://is.ifmo.ru, раздел ”Проекты”.

[6] ***Штучкин А.А., Шалыто А.А.*** Совместное использование теории построения компиляторов и SWITCH-технологии (на примере построения калькулятора). http://is.ifmo.ru, раздел ”Проекты”.

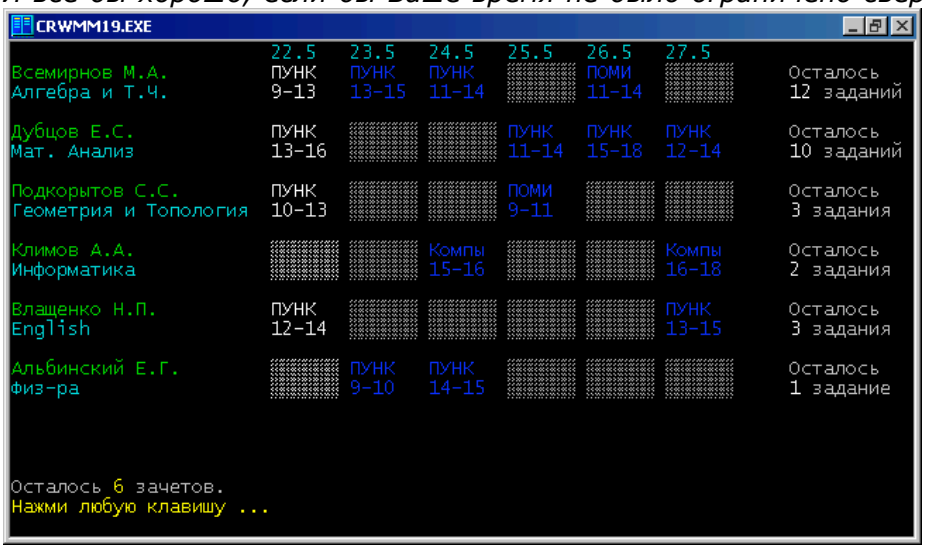
Приложение 1. Игра «Герои Мат-Меха»

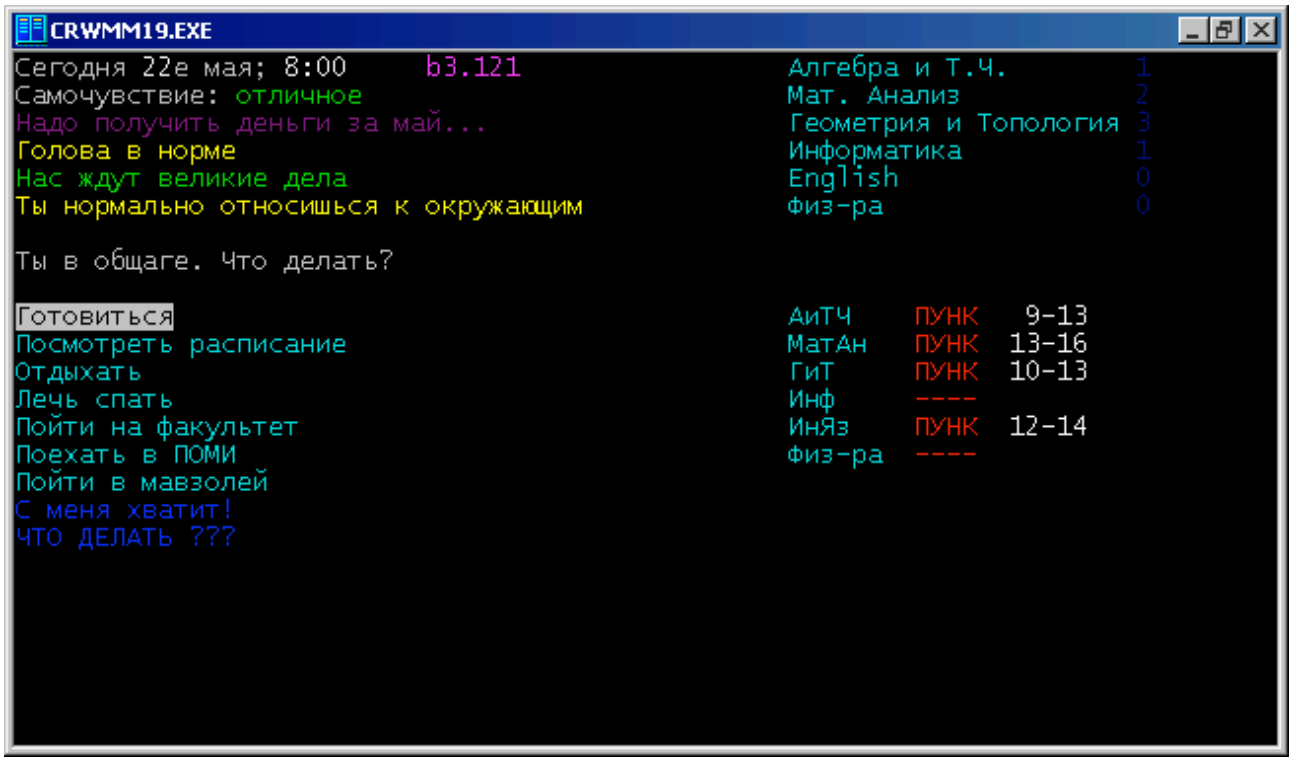
Для того чтобы дать представление об игре, прослужившей прототипом к данному проекту, представим выдержки из документации к ней и несколько скриншотов.

…Эта программа - некоторый синтез всех тех эмоций, что получил автор, пытаясь (с грехом пополам) выйти на сессию в конце второго семестра первого курса на мат-мехе. Правда, при этом автор находился в более выгодном положении, чем Вы - центральный персонаж этой игры, которому предстоит получить зачеты по 6-ти предметам практически с нуля…



…Вам предстоит получить шесть зачетов, начиная, практически, с пустого места. Это означает, что у Вас нет ни одного зачета. Чтобы получить зачет, нужно успешно сдать определенное количество заданий (по каждому предмету - свое). Для того чтобы успешно сдать некоторое количество заданий, нужно обладать некоторым уровнем знаний/умений в предмете. Чтобы что-либо знать/уметь, надо готовиться. Пока Вы готовитесь или сдаете задания, Ваше здоровье понижается. Увы, этот процесс может привести к гибели. Практически единственное действенное средство - отдыхать (и вовремя ложиться спать). И все бы хорошо, если бы Ваше время не было ограничено сверху шестью днями. Кроме этого, преподаватели бывают доступны отнюдь не в любое время и в любом месте, как хотелось бы, а только по определяемому в начале игры расписанию. В этом расписании указывается время и место, где их можно найти в каждый конкретный день. Редко (в некоторых весьма специфических случаях) в расписание вносятся некоторые дополнения, Вас об этом предупредят. И, ко всему прочему, в игре есть еще несколько других персонажей, которые живут самостоятельной жизнью. Большинство из них, впрочем, будет занято попытками получить зачет. При этом некоторые особо ловкие даже будут пытаться использовать для этого Вас... Впрочем, как правило, персонажи так же оказывают на Вас и некоторое положительное воздействие…





…Вашего персонажа характеризуют следующие показатели:

- *самочувствие;*
- *интеллект;*
- *выносливость;*
- *открытость.*

Самочувствие - характеризует, сколь далеки Вы от состояния бездыханного тела. Влияет на качество подготовки.

Интеллект - характеризует силу Ваших мозгов. Влияет на качество подготовки и процедуру получения зачета.

Выносливость - отражает Вашу защиту от отрицательного влияния зубрежки и нервных потрясений и физических нагрузок при получении зачетов.

Открытость - влияет на Ваши отношения с самостоятельными персонажами.

Кроме того, у Вас могут появиться какие-то деньги (которые можно тратить),

А также - уровни знаний по предмету…

Приложение 2. Листинг примерного игрового мира

Листинг файла config.xml

```
<?xml version="1.0" encoding="windows-1251" ?>
<config version="0.1">

    <!-- Players and objects interfaces -->
    <interfaces import="proto.xml"/>

    <characters>
        <!-- Only one player is supported -->
        <player import="player.xml"/>

        <!-- Student -->
        <object import="student1.xml"/>

        <!-- Преподаватель -->
        <object import="teacher.xml"/>

        <!-- Курсовая работа -->
        <object import="lab.xml"/>
        <object import="amulet.xml"/>

    </characters>
</config>
```

Листинг файла proto.xml

```
<?xml version="1.0" encoding="windows-1251" ?>
<interfaces>
    <!-- [ Data types] -->
    <type name="location">
        <value name="home" display="Дома" />
        <value name="nowhere" display="Нигде" />
        <value name="thathome" display="У себя" />
        <value name="aqua" display="Аквариум" />
        <value name="hall" display="Коридоры" />
        <value name="auditory" display="Аудитория" />
    </type>

    <!-- [ Interfaces] -->
    <interface name="Human">
        <var name="location" type="location" />
        <var name="name"     type="string"/>
    </interface>

    <interface name="Student" extends="Human">
        <!-- Knowledge -->
        <var name="k_program"     type="percent" value="rand() / 8"/>

        <!-- Primary skills -->
        <var name="mood"          type="percent" value="rand()"/>
        <var name="intelligence"  type="percent" value="0.25 + rand() / 2"/>
        <var name="charism"       type="percent" value="0.25 + rand() / 2"/>
        <var name="luck"          type="percent" value="0.25 + rand() / 2"/>
    </interface>


    <interface name="Thing">
        <var name="owner"         type="object"  value="null"/>
    </interface>


    <interface name="Lab" extends="Thing">
        <var name="done"          type="decimal" value="0"/>
        <var name="step"          type="decimal" value="1"/>
    </interface>
```

```
    <interface name="Player" extends="Student">
        <var name="passed"        type="decimal" value="0"/>

        <global>
            <handle event="OnCreate">
                <hang event="GameOver" at="96" period="once"/>
            </handle>

            <handle event="GameOver">
                <switch>
                    <case test="passed">
                        <game result="win" message="You win! Congratulations!"/>
                    </case>
                    <default>
                        <game result="lose" message="You have lost. Try Again!"/>
                    </default>
                </switch>
            </handle>
        </global>
    </interface>


    <interface name="Teacher" extends="Human">
    </interface>
</interfaces>
```

Листинг файла player.xml

```
<?xml version="1.0" encoding="windows-1251" ?>
<player interface="Player" id="player">
    <var name="name" value="'Герой'"/>

    <global>
        <handle event="OnCreate">
            <set var="location" value="#home"/>
            <log format="Ты просыпаешся дома. Все, что ты помнишь - это свое имя и то, что
началась СЕССИЯ..."/>
        </handle>
    </global>

    <states init="home">
        <state name="home">
            <cmd name="Поехать в институт">
                <set var="location" value="#hall"/>
                <set state="ifmo"/>
                <log format="Едем в инст..." param1="location"/>
            </cmd>

            <cmdalias name="Учить">
                <alias name="Программирование" bind="k_program"/>

                <command>
                    <set var="alias" value="alias + 0.01"/>
                    <set var="mood" value="mood - rand() / 5" />
                    <log format="Изучаем..."/>
                </command>
            </cmdalias>

            <cmdgroup name="Лабораторные..." filter="Lab">
                <set var="mood" value="mood - rand() / 5" />
                <set var="that->done" value="that->done + rand() * that->step"/>
                <log format="Делаем лабораторную &amp;quot;{0}&amp;quot;" param1="that->name"/>
            </cmdgroup>

            <cmd name="Отдыхать">
                <set var="mood" value="mood + rand() / 10" />
            </cmd>

            <cmd name="Спать">
```

```
                <log format="Штирлиц не спал. Это была привычка, выработанная годами."/>
            </cmd>
        </state>

        <!-- Аквариум -->
        <state name="aqua">
            <cmd name="Идти домой">
                <set state="home"/>
                <set var="location" value="#home"/>
            </cmd>
            <cmd name="В коридоры" tick="0">
                <set state="ifmo"/>
                <set var="location" value="#hall"/>
            </cmd>

            <cmd name="Отдыхать">
                <set var="mood" value="mood + rand() / 5" />
            </cmd>

            <cmdgroup name="Лабораторные..." filter="Lab">
                <set var="mood" value="mood - rand() / 3" />
                <set var="that->done" value="that->done + rand() * that->step"/>
                <log format="Делаем лабораторную &quot;{0}&quot;" param1="that->name"/>
            </cmdgroup>

            <cmdgroup name="Сдавать..." filter="Teacher" test="enumeq(that->location, #aqua)
and enumeq(location, #aqua)">
                <hang event="OnPass" at="now()" this="that" that="this"/>
            </cmdgroup>

            <cmdgroup name="Говорить..." filter="Human" test="enumeq(that->location, #aqua)
and enumeq(location, #aqua)">
                <hang event="OnTalk" at="now()" this="that" that="this"/>
            </cmdgroup>
        </state>

        <state name="ifmo">
            <cmd name="Домой">
                <set state="home"/>
                <set var="location" value="#home"/>
            </cmd>

            <cmd name="В аквы" tick="0">
                <set state="aqua"/>
                <set var="location" value="#aqua"/>
            </cmd>
        </state>
    </states>
</player>
```

Листинг файла student1.xml

```
<?xml version="1.0" encoding="windows-1251" ?>
<npc interface="Student" id="student1">

    <var name="name"     value="'Василий Пупкинс'"/>
    <var name="location" value="#nowhere"/>


    <global>
        <handle event="OnCreate">
            <set var="find('amulet')->owner" value="this"/>
        </handle>

        <handle event="OnTimer">
            <log format="In {0}" param1="this->location" />
        </handle>
    </global>
```

```
    <states>
        <state name="nowhere">
            <handle event="OnTimer">
                <probability>
                    <chance value="0.8">
                        <set var="location" value="#hall" />
                        <set state="hall" />
                    </chance>
                    <default>
                    </default>
                </probability>
            </handle>
        </state>


        <state name="aqua">
            <handle event="OnTalk">
                <probability>
                    <chance value="0.4">
                        <log target="that" format="рассказывает свежий анекдот с ostrie.ru. У
вас поднимается настроение."/>
                        <set var="that->mood" value="that->mood + rand() / 10"/>
                    </chance>

                    <chance value="0.05">
                        <if test="! objeq(find('amulet')->owner, this)">
                            <log target="that" format="молчит, проклятый!"/>
                        </if>
                        <if test="objeq(find('amulet')->owner, this)">
                            <log target="that" format="Вот тебе волшебная шняжка!"/>
                            <hang event="OnGive" this="find('amulet')" that="that"
at="now()"/>
                        </if>
                    </chance>

                    <default>
                        <log target="that" format="молчит, проклятый!"/>
                    </default>
                </probability>
            </handle>

            <handle event="OnTimer">
                <probability>
                    <chance value="0.1">
                        <set var="location" value="#hall" />
                        <set state="hall" />
                    </chance>
                    <chance value="0.1">
                        <set var="location" value="#auditory" />
                        <set state="auditory" />
                    </chance>
                    <chance value="0.1">
                        <set var="location" value="#nowhere" />
                        <set state="nowhere" />
                    </chance>
                    <default>
                    </default>
                </probability>
            </handle>
        </state>


        <state name="auditory">
            <handle event="OnTimer">
                <probability>
                    <chance value="0.7">
                        <set var="location" value="#aqua" />
                        <set state="aqua" />
                    </chance>
                    <chance value="0.1">
                        <set var="location" value="#hall" />
```

```
                    <set state="hall" />
                </chance>
                <chance value="0.1">
                    <set var="location" value="#nowhere" />
                    <set state="nowhere" />
                </chance>
                <default>
                </default>
            </probability>
        </handle>
    </state>


    <state name="hall">
        <handle event="OnTimer">
            <probability>
                <chance value="0.5">
                    <set var="location" value="#aqua" />
                    <set state="aqua" />
                </chance>
                <chance value="0.3">
                    <set var="location" value="#auditory" />
                    <set state="auditory" />
                </chance>
                <chance value="0.1">
                    <set var="location" value="#nowhere" />
                    <set state="nowhere" />
                </chance>
                <default>
                </default>
            </probability>
        </handle>
    </state>
</states>
</npc>
```

Листинг файла teacher.xml

```
<?xml version="1.0" encoding="windows-1251" ?>
<npc interface="Teacher" id="teacher">
    <var name="name" type="string" value="'Преподаватель'"/>
    <var name="location" value="#nowhere"/>

    <var name="will_come" type="integer"/>
    <var name="will_away" type="integer"/>

    <var name="will_ready"  type="integer" value="-1"/>
    <var name="will_finish" type="integer" value="-1"/>

    <var name="is_to_go" type="integer" value="0"/>

    <global>
        <handle event="OnCreate">
            <set var="will_come" value="1 + rand() * 4"/>
            <set var="will_away" value="will_come + 6 + rand() * 4"/>
            <hang event="ComeTo" at="will_come-1"  period="once"/>
            <hang event="GoAway" at="will_away"  period="once"/>
            <log format="Я буду с {0} по {1}." param1="will_come" param2="will_away"/>
        </handle>
    </global>

    <states>
        <!-- AWAY: Teacher is somewhere and not responsible -->
        <state name="away">
            <handle event="ComeTo">
                <!-- Make a new coming time -->
                <log format="Я работаю с {0} по {1}." param1="will_come" param2="will_away"/>

                <set var="will_come" value="will_away + 6 + rand() * 14"/>
                <set var="will_away" value="will_come + 6 + rand() * 4"/>
```

```
                <hang event="ComeTo" at="will_come"  period="once"/>
                <hang event="GoAway" at="will_away"  period="once"/>

                <log format="В следующий раз буду с {0}  по {1}." param1="will_come"
param2="will_away"/>


                <set var="location" value="#aqua"/>
                <set state="busy"/>
            </handle>

        </state>

        <!-- BUSY: Teacher is busy and cannot accept a lab, but he can assign availability
time -->
        <state name="busy">
            <handle event="OnPass">
                <switch>
                    <!-- No time assigned -->
                    <case test="will_ready == -1">
                        <log target="that" format="Не сейчас. Я занят. Назначьте встречу!"/>
                    </case>
                    <!-- Time is already assigned -->
                    <default>
                        <log target="that" format="Вам назначено! Приходите в свое время."/>
                    </default>
                </switch>
            </handle>

            <handle event="OnTalk">
                <switch>
                    <!-- No time assigned -->
                    <case test="will_ready == -1">
                        <set var="will_ready"  value="will_come + rand() * (will_away -
will_come - 4)"/>
                        <set var="will_finish" value="will_ready + 4"/>
                        <hang event="OnReady"  at="will_ready"  period="once"/>
                        <hang event="OnFinish" at="will_finish" period="once"/>

                        <log target="that" format="Приходите в следующий раз. Я приму вас с
{0}  по {1}." param1="will_ready" param2="will_finish"/>
                    </case>
                    <case test="(will_ready > now()) and (will_finish &lt; will_come)">
                        <log target="that" format="В этот раз я принимаю с {0}  по {1}."
param1="will_ready" param2="will_finish"/>
                    </case>
                    <case test="will_finish &lt; now()">
                        <log target="that" format="Похоже, вы не пришли в предыдущий раз.
Зря!"/>

                        <set var="will_ready"  value="-1"/>
                        <set var="will_finish" value="-1"/>
                    </case>
                    <!-- Time is already assigned -->
                    <default>
                        <log target="that" format="Я же сказал, приходите в следующий раз. Я
буду с {0}  по {1}." param1="will_ready" param2="will_finish"/>
                    </default>
                </switch>
            </handle>

            <handle event="OnReady">
                <set state="ready"/>
            </handle>

            <handle event="GoAway">
                <set var="location" value="#nowhere"/>
                <set state="away"/>
            </handle>
        </state>
```

```
        <!-- READY: Subject can accept lab -->
        <state name="ready">
            <handle event="OnPass">
                <set var="will_ready"  value="-1"/>
                <set var="will_finish" value="-1"/>

                <log target="that" format="Поехали..."/>
                <set var="find('lab')->done" value="find('lab')->done + rand()*0.2 - 0.08"/>
                <if test="find('lab')->done > 0.95">
                    <log target="that" format="Принято!"/>
                    <set var="that->passed" value="1"/>
                </if>
            </handle>

            <handle event="GoAway">
                <set var="is_to_go" value="1"/>
            </handle>

            <handle event="OnFinish">
                <set var="will_ready"  value="will_come + rand() * (will_away - will_come -
4)"/>
                <set var="will_finish" value="will_ready + 4"/>
                <hang event="OnReady"  at="will_ready"  period="once"/>
                <hang event="OnFinish" at="will_finish" period="once"/>

                <log target="find('player')" format="Приходите в следующий раз. Я приму вас с
{0} по {1}." param1="will_ready" param2="will_finish"/>
                <set state="busy"/>

                <if test="is_to_go">
                    <set var="is_to_go" value="0"/>
                    <hang event="GoAway" at="now()" period="once"/>
                </if>
            </handle>


            <handle event="OnTalk">
                <log target="find('player')" format="Сдавайте, раз уж пришли..."/>
            </handle>
        </state>
    </states>
</npc>
```

Листинг файла lab.xml

```
<?xml version="1.0" encoding="windows-1251" ?>
<object id="lab" interface="Lab">
    <var name="name" type="string" value="'Курсовая работа'"/>

    <var name="step" value="0.05"/>

    <states>
        <state name="default"/>
    </states>
</object>
```

Листинг файла amulet.xml

```
<?xml version="1.0" encoding="windows-1251" ?>
<object id="amulet" interface="Thing">
    <var name="name" type="string" value="'?жерелье лысости'"/>

    <global>
        <handle event="OnTimer">
            <log format="Owner = {0}" param1="owner->name"/>
        </handle>
    </global>
```

```
    <states>
        <state name="default">
            <handle event="OnGive">
                <log format="jмулет передан {0}" param1="that->name"/>
                <if test="! objeq(owner, null)">
                    <set var="owner->luck" value="owner->luck - 0.1"/>
                </if>
                <set var="owner" value="that"/>
                <if test="! objeq(owner, null)">
                    <set var="owner->luck" value="owner->luck + 0.1"/>
                </if>
                <log target="that" format="повышает удачу на 10%" />
            </handle>
        </state>
    </states>
</object>
```