Министерство образования и науки Российской Федерации

Государственное образовательное учреждение высшего профессионального образования

**"ЮЖНО-УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ"**

**Факультет Вычислительной математики и информатики**

**Кафедра системного программирования**

КУРСОВАЯ РАБОТА

бакалавра направления 010400 "Информационные технологии"

**Разработка компьютерной игры   
“future warfare”**

|  |  |
| --- | --- |
|  | Научный руководитель  к.ф.-м.н., доцент кафедры СП  Г.И. Радченко  Автор работы, студент группы ВМИ-346  М.С. Самсонов |

Челябинск-2012

# Оглавление

[Введение 3](#_Toc325564985)

[1. Архитектура системы 12](#_Toc325564986)

[1.1 Правила игры 13](#_Toc325564987)

[1.1.1 Основные типы модулей 13](#_Toc325564988)

[1.2 Диаграмма взаимодействия компонентов 14](#_Toc325564989)

[1.2.1 Агент 16](#_Toc325564990)

[1.2.2 Робот игрока 17](#_Toc325564991)

[1.2.3 Меню и интерфейс 18](#_Toc325564992)

[2. Реализация компьютерной игры “Future Warfare” 19](#_Toc325564993)

[2.1 Реализация основных компонентов 19](#_Toc325564995)

[2.1.1 Реализация мультиагентной системы 19](#_Toc325564996)

[2.1.2 Реализация модульного робота 22](#_Toc325564997)

[3. Тестирование и примеры работы системы 25](#_Toc325564998)

[3.1 Меню 25](#_Toc325565000)

[3.2 Пользовательский интерфейс 26](#_Toc325565001)

[3.3 Робот игрока 26](#_Toc325565002)

[3.4 Модульный робот 27](#_Toc325565003)

[3.4.1 Реконфигурация модульного робота 27](#_Toc325565004)

[3.4.2 Поведение модульных роботов 28](#_Toc325565005)

[Заключение 31](#_Toc325565006)

[Литература 32](#_Toc325565007)

[Приложение 1. Исходный код основных блоков мультиагентной системы 34](#_Toc325565008)

# Введение

АКТУАЛЬНОСТЬ ТЕМЫ

В последние годы отмечается быстрый рост спроса на компьютерные игры, и каждый год число персональных компьютеров в мире увеличивается на 12 процентов. Рынок игр становится более массовым. Появляется много людей, для которых игры на компьютере или приставке становятся не альтернативными, а равнозначными развлечениями.

Игровая индустрия на сегодняшний день по объемам продаж соревнуется с другими формами развлечений, такими как кино, музыка, спорт, и даже обогнала некоторых соперников. Например, в 2007 году объем мирового рынка игрового программного обеспечения обогнал музыкальную индустрию и достиг 37,5 млрд. долларов, в то время как объем музыкальной индустрии составил 31,8 млрд. долларов.

Компьютерные игры приносят многомиллиардный доход компаниям, которые занимаются их разработкой. Например, компания Activision на игре Call of Duty: Modern Warfare 3 за 16 дней успела заработать свыше миллиарда долларов.

На сегодняшний день разработано много компьютерных игр, в которых игроку предоставляется возможность вести боевые действия с роботами, оснащенными различными видами оружия, при этом игрок сам может играть за такого робота. Примерами таких игр являются Космические рейнджеры 2. Доминаторы и серия игр MechWarrior. Как правило, в этих играх рассматриваются роботы со статической структурой. Эти роботы являются противоположностью модульным роботам.

Модульные роботы способны самостоятельно менять свою конфигурацию адаптируясь к различной среде для выполнения различных задач. Такое свойство как самостоятельность предоставляет возможность использовать модульных роботов там, где человек не может помогать им вручную или дистанционно. Например, межпланетные исследовательские полеты, основной проблемой которых является расстояние между роботом и станцией управления [2].

В данной работе рассматривается разработка компьютерной игры, позволяющей игроку вести боевые действия с модульными роботами. Способность изменять свою конфигурацию во время боя, предоставляет модульным роботам возможность быстро менять тактику ведения боя в зависимости от ситуации.

ТЕОРИЯ ГРАФОВ В АРХИТЕКТУРЕ МОДУЛЬНЫХ РОБОТОВ

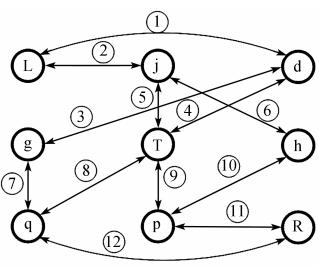
*Использование матрицы смежности*

Реконфигурируемые модульные роботы имеют множество конфигураций, адаптируясь к различным средам и задачам. Поиск нужной конфигурации усложняется тем, что возможное число конфигураций растет экспоненциально с увеличением числа модулей. В работе [4] для решения этой проблемы предлагают использовать матрицу смежности.

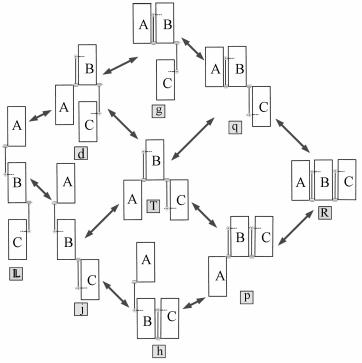
Смежные конфигурации можно перестроить друг в друга, в то время как не смежные конфигурации перестраиваются различными путями. Таким образом, конфигурация может быть представлена вершиной графа, а реконфигурация ребром графа. Такой граф может быть представлен матрицей смежности. Для примера возьмем модульного робота AMOEBA-I [4] (рисунок 1) и представим его конфигурации в виде графа (рисунок 2) (Более подробно эти конфигурации представлены на рисунке 3).



Рис. 1. Реконфигурируемый модульный робот AMOEBA-I [4]

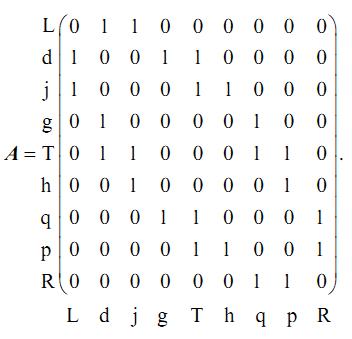


**Рис.** **2.** Представление конфигураций Модульного робота AMOEBA-I [4] в виде графа.



**Рис.** **3.** Подробное представление конфигураций Модульного робота AMOEBA-I [4].

Для этого модульного робота существует 9 конфигураций и 12 типов смежных отношений между ними. Каждое смежное отношение предполагает, что одна конфигурация может быть преобразована в другую. Этот граф можно представить в виде матрицы смежности.



Данный способ позволяет хранить все конфигурации модульного робота в удобной форме, а алгоритмы из теории графов обеспечивают быстрый поиск и переход в нужную конфигурацию. Что позволяет модульному роботу, находящемуся в опасной для него среде, быстро перейти в новую конфигурацию, если старая конфигурация не может справиться с заданием.

*Использование матрицы инцидентности*

Модульных роботов сложно моделировать из-за динамического изменения их структуры. При изменении количества модулей в структуре изменяется количество связей и число степеней свободы, что влияет на кинематику робота. Поэтому необходимо использовать метод, легко представляющий конфигурацию структуры.

В работе [1] предлагают представить модульную систему соответствующим графом, показывающим отношение, существующее между различными модулями. Этого можно достичь путем маркировки вершин и ребер графа таким образом, чтобы по метке можно было, определить к какому типу относится, та или иная вершина или ребро. Для примера возьмем модульного робота RobMAT [1] и представим одну из его конфигураций (рисунок 4) в виде взвешенного графа (рисунок 5).

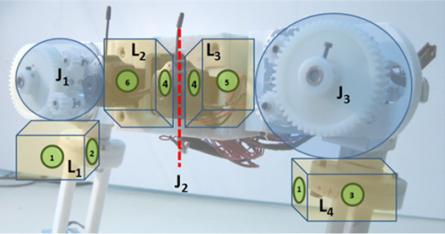


Рис. 4. Одна из конфигураций модульного робота RobMAT

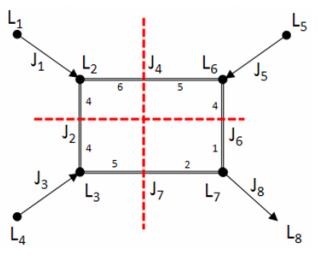
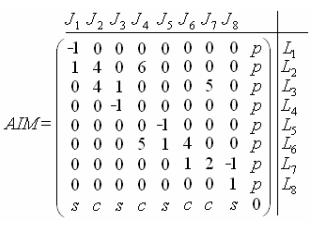


Рис. 5. Граф, представляющий конфигурацию модульного робота RobMAT [1]

В этом графе (рисунок 5) вершины представляют собой модули, а ребра связи между ними. Причем в эту конфигурацию входят связи и модули различных типов. Для представления такого графа в работе [1] используют матрицу инцидентности AIM, в которой последняя строка используется для определения типа связи, а последний столбец для определения тип модуля.



Таким образом, матрица AIM содержит достаточно информации, чтобы описать конфигурацию модульного робота с различными модулями и различными типами связей между ними. К тому же эту матрицу легко модифицировать при добавлении или удалении модуля. Это позволяет решить проблему описания конфигураций модульного робота с большим количеством различных типов модулей.

ИСПОЛЬЗОВАНИЕ МУЛЬТИАГЕНТНОЙ СИСТЕМЫ ДЛЯ ОРГАНИЗАЦИИ МОДУЛЬНЫХ РОБОТОВ

Модульные роботы могут содержать огромное количество различных модулей, каждый из которых мыслит самостоятельно. Для выполнения большинства задач им необходимо взаимодействовать друг с другом. Для решения этой проблемы в работе [3] предлагают использовать мультиагентную систему, т.е. можно представить модульного робота как мультиагентную систему, а каждый его модуль представить в виде агента этой системы.

Мультиагентная система – это система, содержащая группу агентов, которые могут взаимодействовать между собой.

Агент – нечто, что способно воспринимать свое окружение и изменять его своими действиями [8, 9]. Агент обладает следующими свойствами:

1. реактивность;
2. автономность;
3. целенаправленность;
4. непрерывность функционирования;
5. коммуникативность;

Свойство реактивности означает, что агент временами отвечает на изменения в окружении. Агент может перевести окружение из некоторого состояния в некоторое другое состояние, но не из любого в любое.

Свойство автономности означает, что агент сам контролирует свои действия.

Свойство целенаправленности означает, что у агента имеется определенная цель и его поведение подчинено этой цели.

Свойство непрерывности функционирования означает, что агент постоянно находится в состоянии исполнения.

Свойство коммуникативности означает, что агент может общаться с другими агентами, используя для этого некоторый язык.

Информация, которой обладает система агентов, является обычно распределенной. Агенты могут следить за окружением из разных положений и получать информацию в различные моменты времени. Таким образом, состояние окружения является частично обозримым для каждого агента.

Каждый агент для выполнения своего задания может выбрать себе роль исходя из своих возможностей. Играя каждый свою роль, агенты общаются между собой: координируют свои действия, обмениваются знаниями, ведут переговоры.

Таким образом, представив модульного робота как мультиагентную систему можно организовать успешное взаимодействие его модулей между собой.

ИНСТРУМЕНТЫ ДЛЯ РАЗРАБОТКИ ИГРЫ

Для создания качественной трехмерной компьютерной игры необходимо использовать качественные инструменты. Одним из таких инструментов является Unity [5] (рисунок 6). Он содержит огромное количество технических средств для создания компьютерных игр.



Рис. 6. Кроссплатформенный инструмент для разработки игр Unity

1. Встроенный физический движок Ageia PhysX™
2. Высоко оптимизированный графический "пайплайн" как для DirectX, так и для OpenGL
3. Потоковое аудио и видео
4. Поддержка трех языков скриптования: JavaScript, C#, Boo

Игровой движок Unity3D полностью интегрирован в Среду Разработки Unity. Тесная интеграция позволяет прямо в редакторе получить всё то, на что будет способна выпущенная игра.

ЦЕЛИ И ЗАДАЧИ ИССЛЕДОВАНИЯ

Целью данной работы является разработка прототипа трехмерной компьютерной игры “Future Warfare”, в рамках которой игрок может взаимодействовать с армией модульных роботов. Для достижения поставленной цели необходимо решить следующие задачи:

1. проанализировать доступные методы и библиотеки для разработки трехмерных компьютерных игр;
2. разработать несколько видов модулей для модульных роботов;
3. разработать робота для игрока;
4. разработать алгоритмы передвижения модульных роботов;
5. разработать алгоритмы реконфигурации модульных роботов;
6. реализовать прототип игры “Future Warfare”.

# Архитектура системы

Архитектура разрабатываемой системы изображена на рисунке 7. Разрабатываемая система состоит из четырех основных компонент.

1. Игрок + его робот.
2. Мультиагентная система.
3. Окружающая среда.
4. Меню и интерфейс.

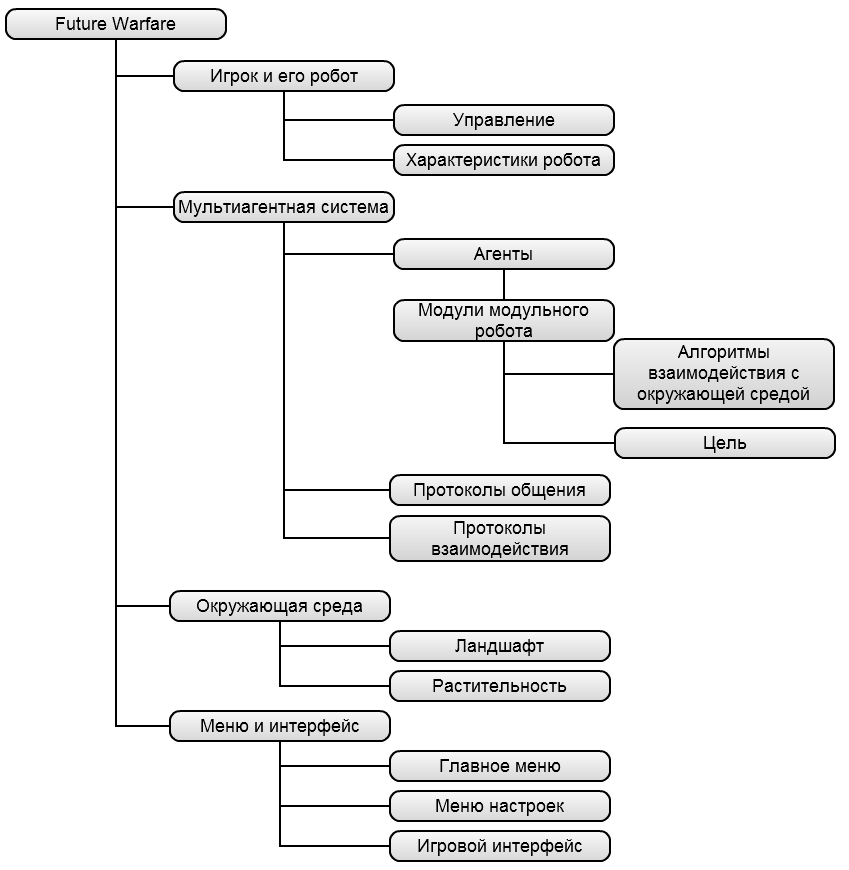


Рис. 7. Архитектура разрабатываемой системы

## Правила игры

В начале игры игроку выдается задание, выполнив которое он получает следующее. В процессе выполнения задания игрок взаимодействует с модульными роботами, которые могут быть как союзниками, так и врагами. Задания могут содержать различные задачи, такие как уничтожение вражеских объектов, отчистка территории от роботов противника, защита укрепленного объекта и т.п. В зависимости от задачи модульные роботы выбирают конфигурации подходящие для оптимального решения задачи. В определенный момент игры игрок получает технологию позволяющую взламывать модульных роботов. С помощью этой технологии игрок может заставить модульного робота воевать на его стороне.

Все модули в игре имеют сенсоры, с помощью которых они получают данные из окружающей среды. Например, сенсоры позволяют обнаружить противника на небольшом расстоянии. Кроме того модули могут общаться между собой, запрашивая помощь или передавая данные об окружающей среде.

### Основные типы модулей

1. Атакующий модуль – это модуль, состоящий из «тела», на котором расположены механизмы соединения с другими модулями, и «башни», на которой могут располагаться различные виды оружия. Атакующий модуль, обнаружив врага, открывает по нему огонь либо запрашивает помощь у других модулей. Не способен передвигаться самостоятельно.
2. Модуль передвижения – это модуль, состоящий из платформы, на которой расположены механизмы соединения с другими модулями, и элементов необходимых для передвижения платформы (как правило, это колеса или гусеницы). Модуль передвижения предназначен для перевозки модулей, которые не способны сами передвигаться. При этом он способен выбирать оптимальные пути передвижения.
3. Модуль-турель – это модуль, состоящий из «тела» и «башни», на которых расположены механизмы соединения с другими модулями (к «башне» крепятся исключительно атакующие модули, причем накладывается ограничение на вид оружия). «Башня» этого модуля вращается, работая по схеме Гатлинга, т.е. к «башне» крепятся атакующие модули, каждый из которых в течение одного цикла поворота делает единственный выстрел, затем перезаряжается. Это повышает огневую мощь и обеспечивает эффективное управление атакующими модулями. Не способен передвигаться самостоятельно.
4. Модуль защиты – это модуль, состоящий из «тела», на котором расположен один механизм соединения, и шарнира, к которому крепится металлическая пластина. Вращая шарнир, модуль подстраивается под защищаемый объект. Не способен передвигаться самостоятельно.
5. Многофункциональная платформа – это модуль, состоящий из «тела», на котором расположены механизмы соединения с другими модулями. Может подсоединять к себе Модуль-турель и модули защиты. Так же платформа имеет механизмы для подключения модулей передвижения (от 4-х), что позволяет платформе передвигаться.

## Диаграмма взаимодействия компонентов

Взаимодействие всех компонент разрабатываемой системы изображено на рисунке 8. Робот игрока является персонажем игры, с помощью которого игрок взаимодействует с игровым миром. Меню и интерфейс служат для представления данных о персонаже игрока и окружающем его мире пользователю в понятном для него виде. Мультиагентная система это набор агентов представляющих собой модули модульных роботов, и которые могут взаимодействовать с роботом игрока и окружающей средой. Окружающая среда это набор объектов (таких как ландшафт, деревья, свет) необходимых для создания игрового мира.

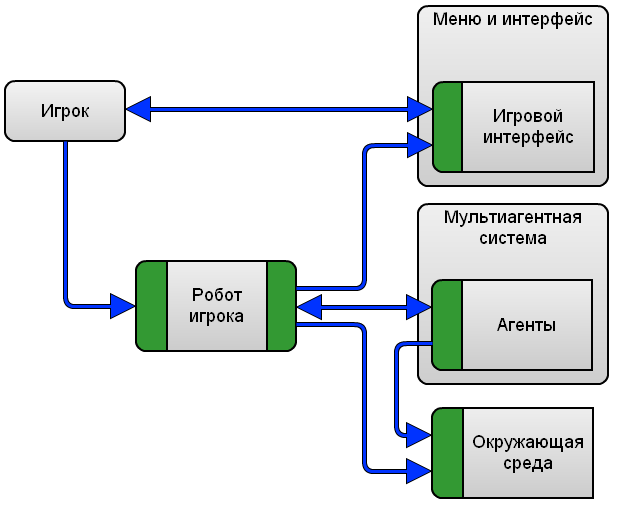


Рис. 8. Диаграмма взаимодействия компонентов

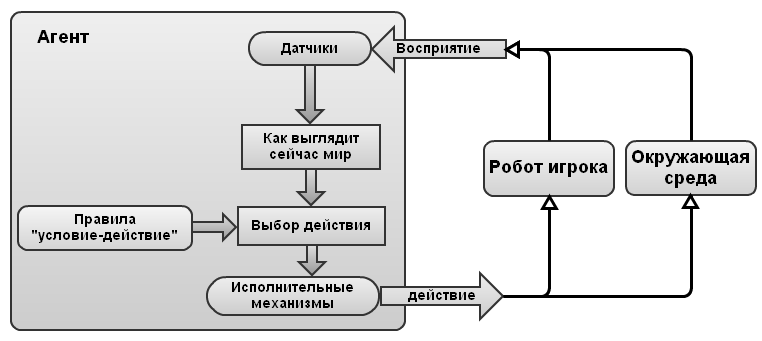
Рассмотрим основные компоненты подробнее.

### Агент

Для разрабатываемой системы были выбраны простые рефлексные агенты [7]. Подобные агенты выбирают действия на основе текущего акта восприятия, игнорируя всю остальную историю актов восприятия. Простой рефлексный агент состоит из четырех основных компонент.

1. Датчики.
2. Блок выбора действия.
3. Правила выбора действия.
4. Блок определения текущего состояния окружающей среды
5. Исполнительные механизмы.

Подробная диаграмма взаимодействия основных компонентов изображена на рисунке 9.



**Рис.** **9.** Диаграмма взаимодействия компонентов простого рефлексного агента

Датчики получают данные из окружающей среды и передают их в блок выбора действия. Блок выбора действия в свою очередь на основе полученных данных и имеющихся правил выбирает действие, которое будет выполнено одним или несколькими исполнительными механизмами.

### Робот игрока

Робот игрок это персонаж игры, которым управляет игрок и который содержит методы для взаимодействия с игровым миром. Этот робот состоит из трех основных компонент.

1. Блок управления.
2. Оружие.
3. Гусеничная платформа.

Подробная диаграмма взаимодействия основных компонентов изображена на рисунке 10.

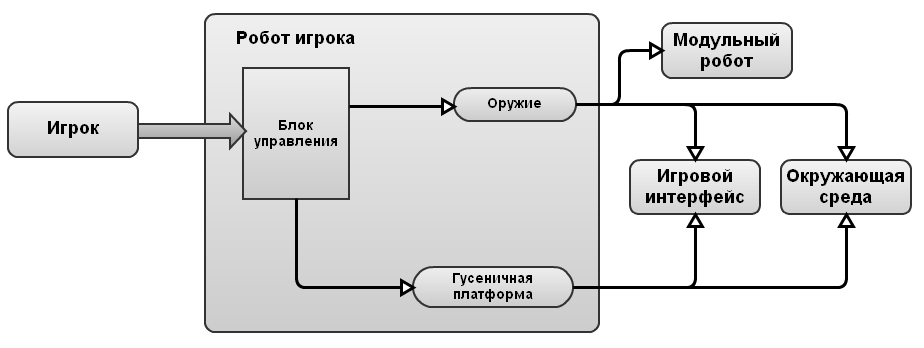


Рис. 10. Диаграмма взаимодействия компонентов Робота игрока

Блок управления обрабатывает входные данные с клавиатуры и мыши и инициализирует действия соответствующих компонент. Оружие наносит повреждения модульным роботам и окружающей среде, а также передает данные о своем состоянии в игровой интерфейс. Гусеничная платформа обеспечивает передвижение робота по игровому миру. Взаимодействуя с окружающей средой, она имитирует движение настоящей гусеничной платформы и позволяет передвигаться по неровной поверхности. Гусеничная платформа также передает данные о своем состоянии в игровой интерфейс.

### Меню и интерфейс

Различные меню и игровой интерфейс служат для взаимодействия игрока с игровым миром. В данной игре пользователю предлагается три основных меню и один игровой интерфейс.

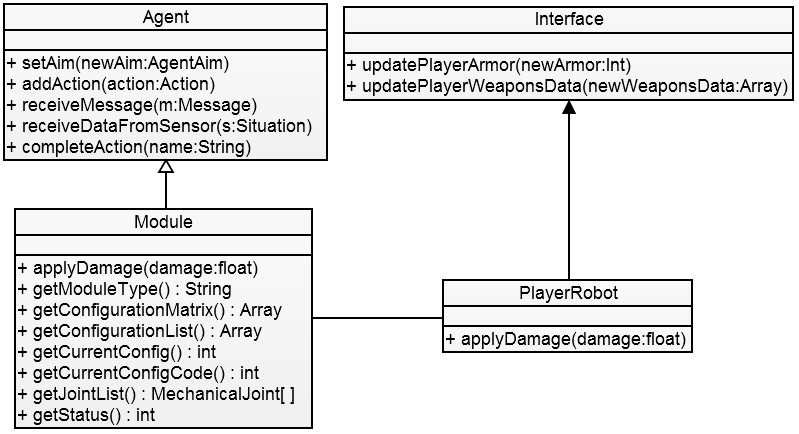
Игровой интерфейс получает данные о состоянии Робота игрока и отображает их на экране. Главное меню служит для того чтобы пользователь мог начать игру, выйти из нее или перейти в меню настроек. В меню настроек игрок может произвести различные настройки, изменив параметры игры. С помощью игрового меню игрок может приостановить игру изменить параметры игры и возобновить игру.

# Реализация компьютерной игры “Future Warfare”

1. **Реализация Компьютерной игры “Future Warfare”**

## Реализация основных компонентов

Основные компоненты системы реализованы четырьмя классами: Interface (интерфейс), PlayerRobot (робот игрока), Agent (агент мультиагентной системы) и Module (модуль модульного робота).



**Рис.** **11.** Схема классов, реализующих основные компоненты системы

### Реализация мультиагентной системы

Мультиагентная система реализована классом Agent (агент) и классами, определяющими его функциональные элементы: ActionBlock (блок выбора действий), Sensor (датчик), Action (действие агента), WorkingMemory (рабочая помять), AgentAim (цель агента).

Метод *howLooksWorldNow* представляет текущее состояние окружающей среды и состояние агента в виде специального кода, который впоследствии используется для генерации действий.

Блок действий реализован классом *ActionBlock,* полем которого является стек действий, причем на вершине стека находится действие с наивысшим приоритетом. Метод *generateActions* генерирует действия в соответствии с обстановкой и правилами, после чего помещает их в стек.

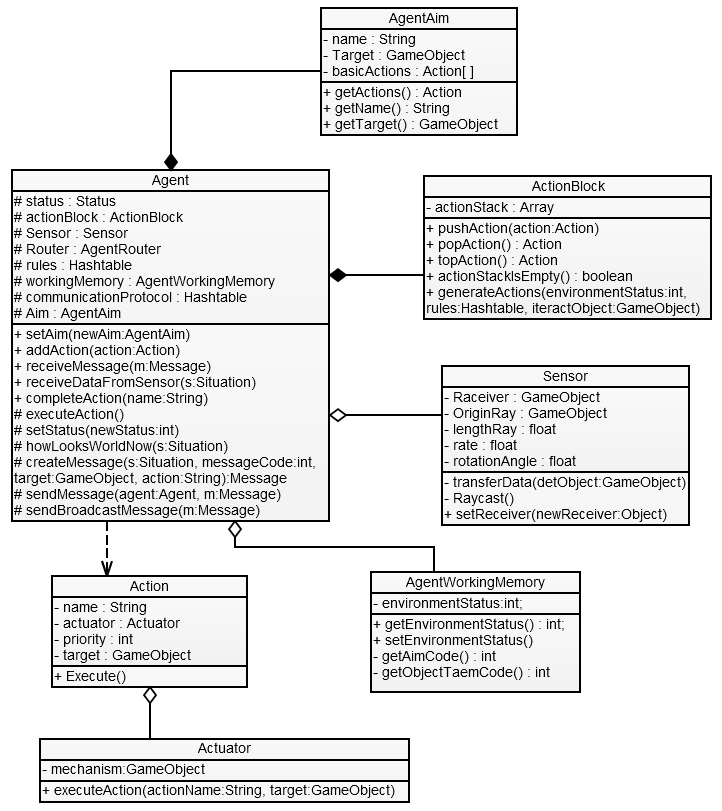


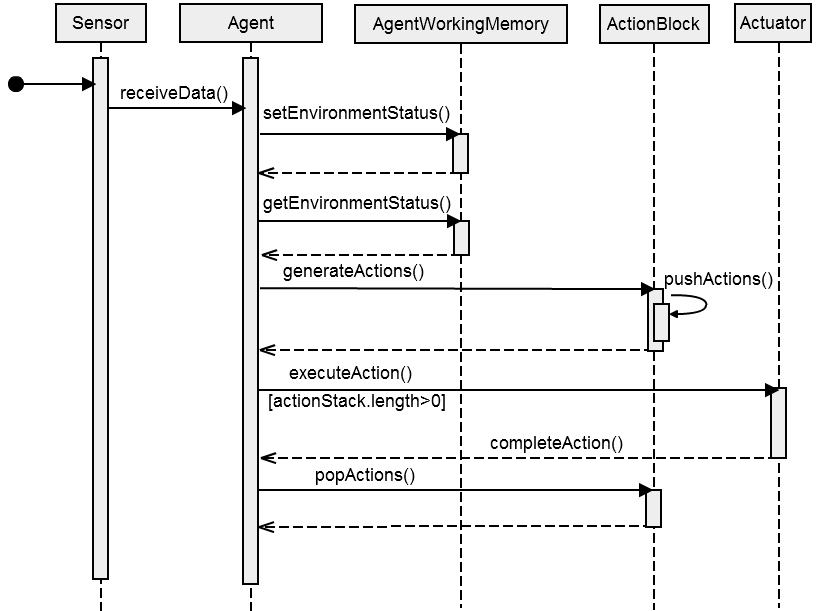
Рис. 12. Схема классов, реализующих мультиагентную систему

Датчики реализованы классом *Sensor,* поля которогозадают параметры датчиков (скорость вращения, максимальное расстояние). Метод *Raycast* запускает луч длиной *lengthRay,* который возвращает первый объект, попавшийся на пути, после чего передает полученный объект приемнику (Receiver) и поворачивается на заданный угол.

Рабочая память реализуется на основе класса *AgentWorkingMemory* и служит для хранения текущего состояния окружающей среды. Метод *setEnvironmentStatus* вызывает методы для кодирования данных об окружающей среде и формирует числовое значение определяющее состояние окружающей среды. Полученное таким способом числовое значение сохраняется в поле *environmentStatus*.

Цель агента реализована классом *AgentAim*, который определяет базовое поведение агента. Поле *basicActions* хранит начальный набор действий, которые необходимо выполнить для достижения цели.

Диаграмма последовательностей [6], изображенная на рисунке 13, подробно описывает взаимодействие основных компонентов агента.



**Рис.** **13.** Диаграмма последовательностей основных компонентов агента

Датчик агента, получив данные из окружающей среды (чаще всего это объект типа *GameObject*, который каким либо образом повлиял на датчик), передает их агенту. Агент передает полученные данные и данные о своем состоянии в рабочую память *AgentWorkingMemory*, где на основе этой информации формируется специальный код, однозначно идентифицирующий состояние окружающей среды. Далее агент передает новое состояние окружающей среды в блок выбора действий *ActionBlock*, где на их основе и на основе заранее заданных правил генерируются действия и заносятся в стек действий actionStack.

Полученный набор действий выполняется исполнительными механизмами *Actuator* в соответствии с приоритетом. После выполнения действия исполнительный механизм посылает агенту сообщение о том, что действие выполнено, а агент в свою очередь посылает сообщение блоку выбора действий о том, что действие необходимо удалить из стека действий actionStack. Исходный код основных блоков мультиагентной системы приведен в приложении 1.

### Реализация модульного робота

Модульный робот реализуется четырьмя классами: Module (модуль), Configuration (конфигурация), MechanicalJoint (механическое соединение), Reconfiguration (реконфигурация).

Модуль определяется классом *Module,* полями которого являются элементы (возможные конфигурации, элементы для объединения модулей), необходимые для взаимодействия с другими модулями.

Поле *configurationMatrix* представляет собой матрицу смежности, описывающую возможные конфигурации модульного робота, элементами которой являются числа 0 или 1. Значение 1 означает, что из конфигурации с индексом i модульный робот может перестроиться в конфигурацию с индексом j и наоборот. Значение 0 означает невозможность подобной перестройки.

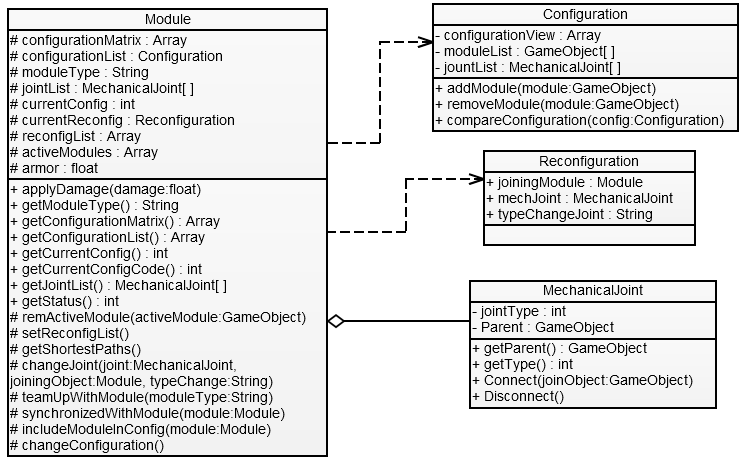


Рис. 14. Схема классов, реализующих модульного робота

Для перехода из одной конфигурации в другую модульный робот должен поочерёдно перестраиваться в промежуточные конфигурации пока не достигнет нужной. Метод *setReconfigList* определяет список промежуточных конфигураций. Для определения минимального количества промежуточных конфигураций используется метод *getShortestPaths*, который использует матрицу смежности *configurationMatrix* для поиска кратчайшего пути до новой конфигурации. Метод *changeConfiguration* определяет разницу между текущей и следующей конфигурациями и с помощью вызова метода *changeJoint* изменяет текущую конфигурацию.

Конфигурация модульного робота реализована классом *Configuration,* поля которого содержат данные, необходимые для описания конфигурации с помощью матрицы инцидентности.

Метод *compareConfiguration* сравнивает две конфигурации и возвращает массив реконфигураций Reconfiguration, которые необходимо выполнить модульному роботу, чтобы перестроится из одной конфигурации в другую.

Реконфигурация модульного робота представляет собой действие, разрывающее или устанавливающее соединения между модулями. Реконфигурация реализована классом *Reconfiguration* и содержит поля определяющие действие, которое необходимо совершить модульному роботу, чтобы перестроится в другую конфигурацию.

# Тестирование и примеры работы системы



## Меню

Главное меню игры (рисунок 15) состоит из логотипа “Future Warfare” и трех кнопок Start Game (начать игру), Options (настройки) и Quit (выход).



**Рис. 15.** Главное меню игры

При нажатии на кнопку Start Game загружается первый уровень (рисунок 16), и игрок начинает игру.



**Рис. 16.** Скриншот из игры

## Пользовательский интерфейс

Интерфейс состоит из трех полей содержащих информацию о состоянии робота игрока (рисунок …).

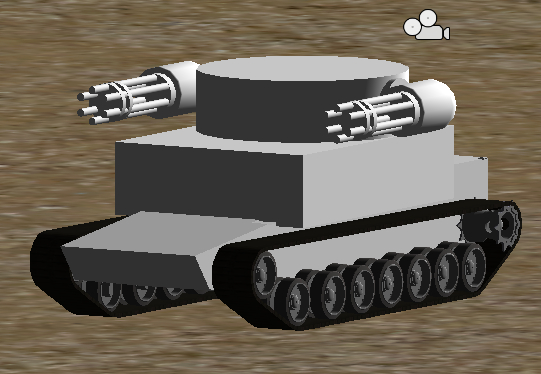


**Рис. 17.** Информация о состоянии робота игрока

При нажатии на левую кнопку мыши оружия робота начинают стрелять и передавать данные (количество оставшихся патронов) объекту класса Interface, где данные обрабатываются и выводятся на экран. При попадании в робота игрока из оружия, данные о текущем состоянии брони, также передаются в объект класса Interface.

## Робот игрока

Робот игрока состоит из платформы передвижения имитирующей гусеничную платформу и башни с различным оружием (рисунок 18).



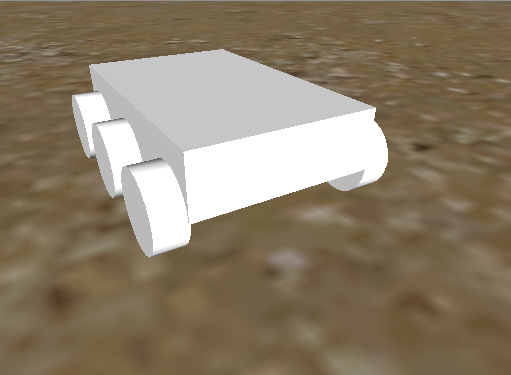
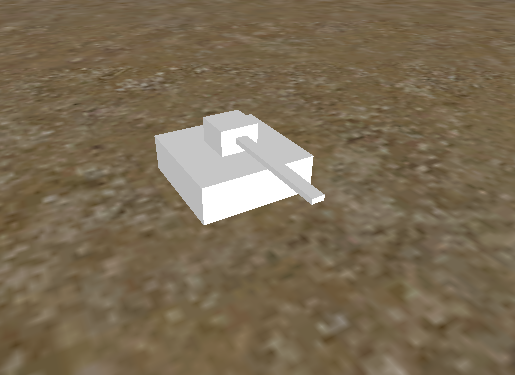
**Рис. 18.** Робот игрока

Движение осуществляется клавишами “w” – вперед, “s” – назад, “a” – влево, “d” – вправо. Движение мышью обеспечивает вращение камеры, башни и оружия, колесо прокрутки используется для приближения и отдаление камеры.

## Модульный робот

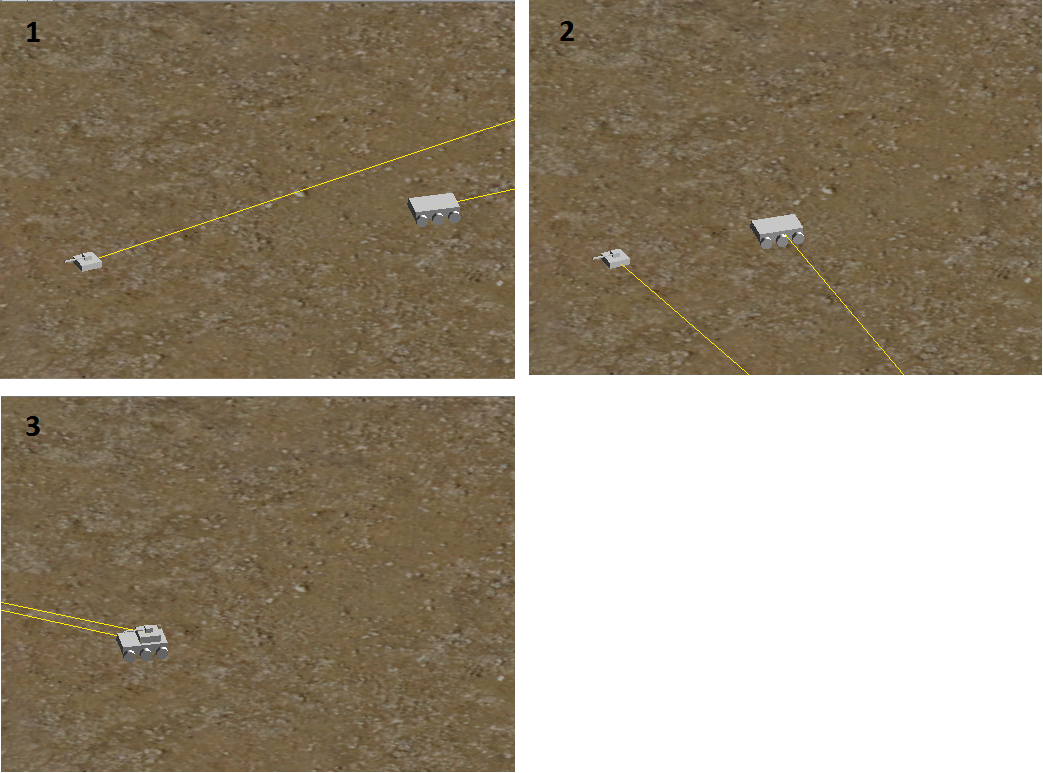
### Реконфигурация модульного робота

При тестировании использовался модульный робот, состоящий из двух модулей, модуля передвижения и атакующего модуля (рисунок 19).

**Рис. 19.** Модуль передвижения и атакующий модуль

Для проведения тестирования реконфигурации модульных роботов было необходимо подать на сенсоры данные, которые приводили к вызову соответствующих функций. При сравнении текущей и заранее заданной конфигурации, атакующий модуль решил, что для смены конфигурации ему необходимо ничего не делать, а модуль передвижения решил, что ему необходимо подъехать к атакующему модулю, поставить его на себя и закрепить механическое соединение (рисунок 20).



**Рис. 20.** Реконфигурация модульного робота

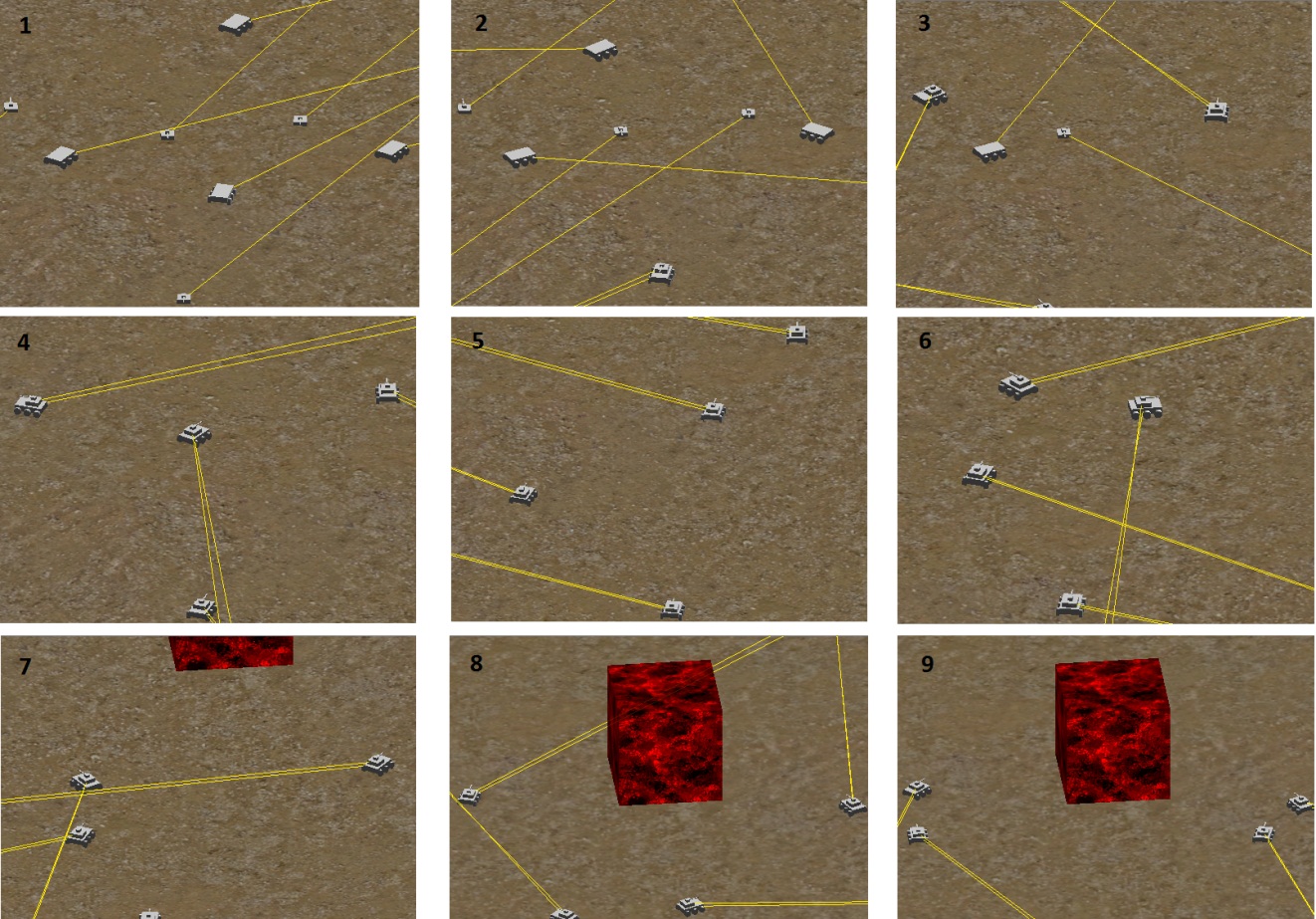
### Поведение модульных роботов

Для проведения тестирования поведения модульных роботов было необходимо создать группу модульных роботов, назначить им цель и провести наблюдение за их поведением.

Была создана группа модульных роботов, состоявшая из 4 модулей передвижения и 4 атакующих модулей, и назначена цель «защита объекта». При запуске игрового уровня каждый модуль начинает выполнять базовые действия определяемые целью. Цель «защита объекта» определяет следующие базовые действия:

1. Найти ближайший свободный модуль, необходимый для создания приемлемой для защиты объекта конфигурации;
2. Перестроится в конфигурацию;
3. Подъехать к защищаемому объекту на приемлемое расстояние;
4. Защищать объект, т.е. с помощью сенсоров определять появление вражеских объектов и соответствующим образом реагировать на их появление;

На рисунке 16 (желтыми линиями обозначается работа сенсоров) представлены скриншоты работы программы, где изображены сборка модульных роботов и выполнение ими поставленной задачи (защита объекта). Защищаемым объектом выступает куб красного цвета.



**Рис. 21.** Работа модульных роботов

Для проверки работы сенсоров был использован робот игрока, который является вражеским объектом по отношению к созданной группе модульных роботов. Подъехав роботом игрока на достаточное расстояние, можно убедиться в том, что модульный робот улавливает сенсором появление вражеского объекта и начинает выполнять действия соответствующие заданным правилам. Атакующий модуль открывает огонь по вражескому объекту, а модуль передвижения начинает двигаться к объекту.

При уничтожении одного из модулей составляющих конфигурацию, второй модуль начинает искать ему замену. Находит ближайший свободный модуль, объединяется с ним, перестраивается в нужную конфигурацию и продолжает защищать объект.

# Заключение

Данная работа посвящена разработке прототипа компьютерной игры, в которой игрок взаимодействует с модульными роботами.

ОСНОВНЫЕ РЕЗУЛЬТАТЫ РАБОТЫ

1. Разработаны два вида модулей с различным функционалом.
2. Разработан робот для игрока.
3. Разработаны алгоритмы передвижения модульного робота.
4. Разработаны алгоритмы реконфигурации модульных роботов.
5. Разработан прототип игры “Future Warfare”

НАПРАВЛЕНИЯ ДАЛЬНЕЙШИХ ИССЛЕДОВАНИЙ

В ходе дальнейших исследований предполагается улучшить анимацию и трехмерные модели модульных роботов, анимацию стрельбы и добавить звуковые эффекты, новые модули и различные виды оружия. Также предполагается улучшить алгоритмы передвижения модульных роботов для предотвращения столкновений. Также есть возможность улучшить поведение интеллектуальных агентов (модульных роботов), с помощью написания более сложных правил поведения.

# Литература

1. [Baca](http://www.springerlink.com/content/?Author=Jos%c3%a9+Baca) [J.](http://www.springerlink.com/content/?Author=Jos%c3%a9+Baca), [Yerpes](http://www.springerlink.com/content/?Author=Ariadna+Yerpes) [A](http://www.springerlink.com/content/?Author=Ariadna+Yerpes)., [Ferre](http://www.springerlink.com/content/?Author=Manuel+Ferre) [M](http://www.springerlink.com/content/?Author=Manuel+Ferre)., [Juan A.,](http://www.springerlink.com/content/?Author=Juan+A.+Escalera) [Aracil](http://www.springerlink.com/content/?Author=Rafael+Aracil) [E](http://www.springerlink.com/content/?Author=Juan+A.+Escalera)., [Aracil](http://www.springerlink.com/content/?Author=Rafael+Aracil) [R](http://www.springerlink.com/content/?Author=Rafael+Aracil). Modelling of Modular Robot Configuration Using Graph Theory, [Lecture Notes in Computer Science](http://www.springerlink.com/content/0302-9743/) // Hybrid Artificial Intelligence Systems. Vol.5271. 2008. pp. 649-656.
2. [Busquets](http://www.springerlink.com/content/?Author=D%c3%addac+Busquets) D., [Sierra](http://www.springerlink.com/content/?Author=Carles+Sierra) [C](http://www.springerlink.com/content/?Author=Carles+Sierra)., [López R.](http://www.springerlink.com/content/?Author=Ramon+L%c3%b3pez+de+M%c3%a0ntaras) A Multiagent Approach to Qualitative Landmark-Based Navigation // Autonomous Robots. Vol.15. №2. 2003. pp. 129-154.
3. Carrillo E., Duhaut D. Methods for Collective Displacement of Modular Self-reconfigurable Robots // Climbing and Walking Robots. Vol.10. 2006. pp. 641-648.

1. [JinGuo Liu](http://www.springerlink.com/content/?Author=JinGuo+Liu), [YueChao Wang](http://www.springerlink.com/content/?Author=YueChao+Wang), [Bin Li](http://www.springerlink.com/content/?Author=Bin+Li), [ShuGen Ma](http://www.springerlink.com/content/?Author=ShuGen+Ma) and [DaLong Tan](http://www.springerlink.com/content/?Author=DaLong+Tan). Center-configuration selection technique for the reconfigurable modular robot // Science in China Series F: Information Sciences, Vol.50. №5. 2007. pp. 697-710.
2. Will Goldstone. Unity Game Development Essentials. 2009. 316 p.
3. Арлоу Д., Нейштадт И. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование. 2-е издание. СПб: Символ-Плюс, 2007. 624 с.
4. Интеллектуальные агенты. URL: <http://www.itfru.ru/index.php/intellectual-agents>. (дата обращения 14.3.2012).
5. [Миков А.И.](http://www.intuit.ru/lector/284.html), [Замятина Е.Б.](http://www.intuit.ru/lector/285.html) Распределенные интеллектуальные системы на основе агентов // [Распределенные системы и алгоритмы](http://www.intuit.ru/goto/course/distrsa/). 2008. URL: <http://www.intuit.ru/department/algorithms/distrsa/10/> (дата обращения 22.12.2011).
6. Стюарт Р., Питер Н. Искусственный интеллект: Современный подход // Интеллектуальные агенты. 2-е издание. М.: Вильямс, 2006. С. 75-109.
7. Шампандар А. Искусственный интеллект в компьютерных играх. М.: Вильямс. 2007, 768 с.

# Приложение 1. Исходный код основных блоков мультиагентной системы

Исходный код класса Sensor:

public var Receiver:GameObject;// Объект принимающий данные с сенсора

public var OriginRay:GameObject;// Объект определяющий начальное положение луча

public var lengthRay:float=50.0;// длина луча

public var rate:float=3.0;// скорость вращения луча

private var rotateAngle=0.0;// текущий угол поворота

private function transferData(detectedObject:GameObject){

var s=new Situation();

s.situationCode=1;

s.target=detectedObject;

Receiver.SendMessage("receiveDataFromSensor", s);

}

public function setReceiver(newReceiver:Object){

Receiver=newReceiver;

}

private function Raycast(){

if(Receiver!=null)

{

var hit:RaycastHit;

rotateAngle=Mathf.Repeat(rotateAngle+rate, 360.0);

var dirtion=OriginRay.transform.TransformDirection(

Vector3.forward);

Debug.DrawRay(OriginRay.transform.position, direction\*lengthRay, Color.yellow);

if(Physics.Raycast(OriginRay.transform.position, direction, hit, lengthRay))

{

transferData(hit.collider.gameObject);

}

OriginRay.transform.localRotation=Quaternion.Euler(0, rotateAngle, 0);

}

}

public function FixedUpdate(){

Raycast();

}

Исходный код класса Agent:

public class Agent extends MonoBehaviour{

protected var status:int;//статус агента, busy-занять, free-свободен

protected var actionBlock:ActionBlock;//блок дествий, обеспечивет выбор внешних дайствий

protected var rules:Hashtable;

protected var workingMemory:AgentWorkingMemory;

protected var communicationProtocol:Hashtable;

protected var isDestroyed:int;

protected var Aim:AgentAim;

public var Router:AgentsRouter;

public var Sensor:GameObject;//сенсор получает информацию из окружающей среды

public function getStatus(){

return status;

}

public function setAim(newAim:AgentAim){

Aim=newAim;

actionBlock.clearActionStack();

var basicActions=Aim.getActions();

for(var i=0; i<basicActions.length; i++)

{

addAction(basicActions[i]);

}

}

public function addAction(action:Action){

if(actionBlock.actionStackIsEmpty() ||

action.getPriority()>actionBlock.topAction().priority)

{

actionBlock.pushAction(action);

}

}

//функция обрабатывает входящее сообщение

public function receiveMessage(message:Message){

if(message.messageCode==1){

receiveDataFromSensor(message.situation);

}else{

if(!isDestroyed && gameObject!=null)

{

gameOject.SendMessage(

communicationProtocol[message.messageCode],

message.situation.target);

}

}

}

public function receiveDataFromSensor(s:Situation){

if(s.target.tag=="interactObject")

{

howLooksWorldNow(s);

ationBlock.generateActions(

workingMemory.getEnvironmentStatus(), rules, s.target);

}

}

public function completeAction(name:String){

if(actionBlock.topAction()!=null &&

ationBlock.topAction().name==name)

{

actionBlock.popAction();

}

}

protected function executeAction(){

if(!actionBlock.actionStackIsEmpty())

{

actionBlock.topAction().Execute();

}

}

protected function howLooksWorldNow(s:Situation){

if(!isDestroyed)

workingMemory.setEnvironmentStatus(s.target,

gameObject.GetComponent("Team").teamCode, status, Aim);

}

protected function createMessage(situationCode:int, messageCode:int, target:GameObject, action:String){

var situation=new Situation();

situation.situationCode=situationCode;

situation.target=target;

return new Message(this, messageCode, action, situation);

}

//функция изменяет статус агента, входные параметры: новый статус

protected function setStatus(newStatus:int){

status=newStatus;

}

//функция посылает сообщение указанному агенту

protected function sendMessage(agent:Agent, message:Message){

Router.sendMessage(agent, message);

}

//функция посылает сообщение всем агентам

protected function sendBroadcastMessage(message:Message){

Router.sendBroadcastMessage(message);

}

}

Исходный код класса ActionBlock:

public class ActionBlock{

private var actionStack:Array;

public function pushAction(action:Action){

actionStack.Push(action);

}

public function popAction(){

return actionStack.Pop();

}

public function topAction(){

if(actionStack.length>0){

return actionStack[actionStack.length-1];

}else return null;

}

public function getActionStack(){

return actionStack;

}

public function clearActionStack(){

actionStack.Clear();

}

public function ActionBlock(){

actionStack=new Array();

}

public function actionStackIsEmpty(){

return actionStack.length==0;

}

public function generateActions(environmentStatus:int,

rules:Hashtable, iteractObject:GameObject){

var actions=rules[environmentStatus];

if(actions!=null)

{

for(var i=0; i<actions.length; i++)

{

if(actionStackIsEmpty() ||

ations[i].getPriority()>topAction().priority){

actions[i].changeTarget(iteractObject);

pushAction(actions[i]);

}

}

}

}

}