МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

02.03.03 МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ И АДМИНИСТРИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

БАКАЛАВРСКАЯ РАБОТА

на тему Разработка алгоритмов взаимодействия программных агентов в виртуальной среде

Студент А. В. Демидович _____

Руководитель Н. И. Лиманова _____

Допустить к защите

Заведующий кафедрой к. тех. н, доцент, А.В. Очеповский _____

«_____»_____20_____г.

Тольятти 2016

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав. кафедрой «Прикладная математика и информатика»
_____ А.В. Очеповский

«____»_____ 2016 г.

ЗАДАНИЕ
на выполнение бакалаврской работы

Студент_Демидович Александр Валерьевич

1. Тема: Разработка алгоритмов взаимодействия программных агентов в виртуальной среде
2. Срок сдачи студентом законченной выпускной квалификационной работы 19.06.2016
3. Исходные данные к выпускной квалификационной работе учебная литература по теме программных агентов, требования к функциональным характеристикам: обеспечение полного контроля управления приложением путем взаимодействия с интерфейсом, использование двухмерных моделей; наличие интуитивно-понятного интерфейса; существующие разработки подобных приложений
4. Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов, разделов) проанализировать научную и учебную литературу по проблеме реализации обучающих приложений; рассмотреть основные принципы разработки программных агентов; проанализировать работу программных агентов для определения основных функций и структурных компонентов моделируемого приложения; создать структуру приложения и описать алгоритмы работы основных его модулей; обосновать выбор средств реализации приложения; реализовать приложение программными средствами; протестировать реализованное программное приложение

5. Ориентировочный перечень графического и иллюстративного материала блок-схемы работы приложения, графики, диаграммы, экранные формы, демонстрирующие работоспособность программного продукта; презентация
6. Дата выдачи задания « 11 » января 2016 г.

Заказчик Директор, ИП Лебеденко Артур Владимирович	_____	А.В. Лебеденко
Руководитель выпускной квалификационной работы	_____	Н.И. Лиманова
Задание принял к исполнению	_____	А.В. Демидович

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав. кафедрой «Прикладная математика и информатика»
_____ А.В. Очеповский

«____»_____2016 г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения бакалаврской работы

Студента Демидович Александра Валерьевича
по теме Разработка алгоритмов взаимодействия программных агентов в виртуальной среде

Наименование раздела работы	Плановый срок выполнения раздела	Фактический срок выполнения раздела	Отметка о выполнении	Подпись руководителя
Выбор и утверждение темы ВКР	14.01.2016	14.01.2016	Выполнено	
Подбор, изучение и проработка практических материалов на исследуемом предприятии и анализ литературы программных агентов	07.03.2016 - 20.03.2016	07.03.2016 - 20.03.2016	Выполнено	
Разработка и согласование с руководителем	28.02.2016- 06.03.2016	28.02.2016- 06.03.2016	Выполнено	

первой главы ВКР				
Разработка и согласование с руководителем второй главы ВКР	07.03.2016 - 20.03.2016	07.03.2016 - 20.03.2016	Выполнено	
Разработка и согласование с руководителем третьей главы ВКР	28.03.2016 - 10.04.2016	28.03.2016 - 10.04.2016	Выполнено	
Оформление пояснительной записки ВКР	21.03.2016 - 27.03.2016	21.03.2016 - 27.03.2016	Выполнено	
Подготовка доклада и графического материала к защите	28.03.2016 - 20.04.2016	28.03.2016 - 20.04.2016	Выполнено	
Оформление презентации и ВКР	12.04.2016 - 20.05.2016	12.04.2016 - 20.05.2016	Выполнено	
Предварительная защита ВКР	16.06.2016	16.06.2016	Выполнено	
Корректировка ВКР согласно сделанным замечаниям	17.06.2016	17.06.2016	Выполнено	
Сдача пояснительной записки ВКР и реализованного программного приложения	19.06.2016	19.06.2016	Выполнено	

Руководитель выпускной квалификационной работы _____ Н.И. Лиманова

Задание принял к исполнению _____ А.В. Демидович

Аннотация

Темой данной выпускной квалификационной работы является: «Разработка алгоритмов взаимодействия программных агентов в виртуальной среде».

Работа выполнена студентом ТГУ, института математики, физики и информационных технологий, группы МОб-1201, Демидович Александром Валерьевичем.

Объект исследования - процесс имитации жизни магазина, **предмет** исследования – алгоритмы взаимодействия агентов.

Целью данной работы является разработка алгоритмов взаимодействия программных агентов в виртуальной среде

Для достижения цели решились следующие задачи:

1. Проанализированы основные бизнес-процессы деятельности предприятия;
2. Проанализирована основные операции в области работы администратора, кассира, работника зала и кладовщика;
3. Разработка модели программных агентов, изобразив их на рисунке;
4. Разработка алгоритмов программных агентов на основе модели;
5. Разработка понятного пользовательского интерфейса с учетом модели и возможностей влиять на виртуальную среду;
6. Проведено тестирование для выявления основных ошибок в работе алгоритмов агентов;
7. Осуществлено внедрение.

Выпускная квалификационная работа состоит из введения, трех глав, заключения, списка литературы и приложений.

В работе использованы современные системы и технологии программирования, Unity и C#.

В первой главе описывается анализ деятельности предприятия для определения ключевых моментов работы персонала, а также даем определение

агента, его возможности и обосновываем выбор инструментальной среды разработки.

Во второй главе проводится разработка и рассмотрение алгоритмов взаимодействия агентов в виртуальной сети.

В третьей главе приводится реализация разработанных алгоритмов и тестирование программной системы.

В заключении приводятся основные выводы по работе, достигнутые в ходе выполнения работы.

В приложении предоставлены фрагменты программного кода и другие дополнительные материалы.

В данной работе содержится 72 страницы, на которых 3 таблицы, 29 рисунков, 20 источников используемой литературы, 6 приложений.

Оглавление

Введение 4

Глава 1 Анализ известных программных систем 7

на основе использования интеллектуальных агентов 7

1.1 Автономные космические машины 7

1.2 Интернет-агенты 9

1.3 Агенты в логистике 11

1.4 Сравнительная характеристика известных систем 12

1.5 Анализ деятельности предприятия 13

Глава 2 Принципы построения программных агентов и алгоритмы их взаимодействия в виртуальной среде 17

2.1 Понятие интеллектуального агента 17

2.2 Архитектуры интеллектуальных агентов 19

2.3 Обоснование выбора инструментальной среды разработки и языка программирования 22

2.4 Описание моделей интеллектуальных агентов в компании 24

2.5 Алгоритм взаимодействия администратора в магазине и его реализация 25

2.6 Алгоритмы поведения покупателей в магазине и их реализация 27

2.7 Алгоритм взаимодействия кассира и работника зала в магазине и его реализация 28

2.8 Алгоритм взаимодействия работника склада в магазине и его реализация ... 31

Глава 3 Реализация разработанных алгоритмов и тестирование программной системы 33
3.1 Реализация основных классов приложения 33
3.2 Разработка основного интерфейса приложения 37
3.3 Журнал событий работы агентов 41
3.4 Руководство пользователя эксплуатации программной системы 44
Заключение 46
Список используемой литературы 47
Приложение А Код агента администратора 49
Приложение Б Код агента кладовщика 51
Приложение В Код агента работника 52
Приложение Г Код ядра виртуального мира 53
Приложение Д Код агента покупателя 59
Приложение Е Техническое задание. 60

Введение

Широкое распространение в мире получили системы, основанные на применении методов и технологий искусственного интеллекта. Такие системы существенно повышают возможности компьютера по решению количества задач и увеличивают эффект, произведенный на экономическую часть. Кроме того, такие системы являются важнейшими инструментами в решении глобальных вопросов традиционного программирования: дорогое обслуживание сложных систем; высокая стоимость разработки проекта ввиду его длительности; использование системы повторно и тому подобные продукты искусственного интеллекта на коммерческом рынке в 2013 году оценивались в 900 млн. долларов.

Последнее время, в мире является популярным направление, основанное на программных агентах. Рассматривать такие системы как «подпрограмма» неправильно, они скорее являются полноценными автономными работниками с ограниченной автономностью и некоторой независимостью от пользователя.

Программные агенты существенно позволяют увеличить производительность работы системы при выполнении тех задач, в которых на пользователя накладывается ключевая нагрузка по координации разных операций.

Агенты, которые должны работать в быстро изменяющихся, непредсказуемых или открытых средах, где существует значительная вероятность того, что действия могут вывести из строя систему, известны как интеллектуальные агенты или иногда автономные агенты. Например, когда космический зонд делает свой длинный перелет с Земли до отдаленных планет, наземной команде, как правило, требуется постоянно отслеживать прогресс и решать как справляться с неожиданными ситуациями. Это затратно и если решение необходимо принять быстро это просто непрактично. По этим причинам, такие организации как федеральное космическое агентство серьезно изучают

возможность создания более автономных зондов – предоставляя им более расширенные возможности принятия решений и ответственностей.

Поиск в интернете ответов на специфические вопросы может быть долгим и утомительным процессом. Поэтому, созданы агенты системы на основе таких решений, которые позволят нам не заниматься отбором релевантной информации. Агент, как правило, получит вопрос, который бы синтезировал куски информации с различных источников из интернета. Отказ может произойти, если конкретный ресурс был недоступен, (возможно, из-за сбоя в сети), или когда результаты не могут быть достигнуты.

Таким образом, актуальность исследования заключается в том, что во многих сферах деятельности мы можем разработать алгоритм агента и тем самым упростим жизнь людям и сохраним другие ресурсы. В данной работе рассматривается написание алгоритмов для программных агентов и их работа в виртуальной среде. За основу среды взят магазин и персонал, который взаимодействует друг с другом.

Объектом исследования является процесс имитации жизни магазина.

Предметом исследования является алгоритм агентов.

Целью данной работы является разработка алгоритмов взаимодействия программных агентов в виртуальной среде.

Для достижения цели необходимо выполнить следующие задачи:

1. Проанализировать основные бизнес-процессы деятельности предприятия;
2. Проанализировать основные операции в области работы администратора, кассира, работника зала и кладовщика;
3. Разработать модель программных агентов, изобразив их на рисунке;
4. Разработать алгоритмы программных агентов на основе модели;
5. Разработать понятный пользовательский интерфейс с учетом модели и возможностей влиять на виртуальную среду;

6. Провести тестирование для выявления основных ошибок в работе алгоритмов агентов;
7. Сделать внедрение

В ходе выполнения работы создано приложение, которое позволяет:

- имитировать работу торгового зала
- осуществлять взаимодействие с виртуальной средой
- следить за текущей информацией в виртуальной среде
- получать каждодневные отчеты по ситуации в виртуальной среде

Выпускная квалификационная работа состоит из введения, трех глав, заключения, списка литературы и приложений.

В первой главе описывается анализ деятельности предприятия для определения ключевых моментов работы персонала, а также даем определение агента, его возможности и обосновываем выбор инструментальной среды разработки.

Во второй главе проводится разработка и рассмотрение алгоритмов взаимодействия агентов в виртуальной сети.

В третьей главе приводится реализация разработанных алгоритмов и тестирование программной системы.

В заключении приводятся основные выводы по работе, достигнутые в ходе выполнения работы.

Глава 1 Анализ известных программных систем на основе использования интеллектуальных агентов

1.1 Автономные космические машины

Осваивая космос, люди столкнулись с проблемой автоматизации космических машин ввиду того, что управление в режиме реального времени не представляется возможным. Поэтому, было предложено использовать агентов, которые бы осуществляли анализ текущего окружения и выбирали действие максимально удовлетворяющим проектным целям. На рис.1.1 изображен космический аппарат «СОЮЗ».

Рисунок 1.1 – Космический аппарат "СОЮЗ"

Например, так как корректировка движения космического аппарата не всегда обязательно ручной труд, то данную задачу можно передать агентам. Также

для планирования и координации поведением элементов аппарата используют агентов.

И теперь, во многом, люди, используя космические аппараты, полагаются на работу агентов. Ручной труд используется лишь для корректировки или для одобрения применения решений, выбранным агентом [1; 11].

Например, французская компания Airbus, которая участвует в построении, разработке и поддержки самолетов также нацелена на осваивание космоса. Руководство компании решило разработать систему, состоящую из агентов, которые бы решали многочисленные проблемы с жизненным циклом продукта (самолета или космического аппарата). На рис.1.2 изображено воздушное судно компании Airbus, запустившее использование агентов в программной системе.

Рисунок 1.2 – Агенты в Airbus

Компания до этого не занималась разработками данный сферы и лишь в 2012 была создан проект ARUM. Во многом, это случилось из-за того, что процесс перехода к серийному производству в сфере гражданского авиастроения иногда вызывает сложные изменениями в продукте и производственных

операциях в процессе сборки. Такие проблемы достаточно сильно увеличивают срок производства и реализации воздушного судна. Кроме того, компания сталкивается с осложнениями запуска производства, связанную с техническими сложностями и мелкосерийностью самолетов. Также производство аппаратов по индивидуальному заказу клиента приводит к наибольшим рискам. На сегодняшний день жизненный цикл продукта начинает уменьшаться, и производство новых версий запускается все в большем количестве. Данные факторы очень сильно влияют на разработку новых проектов и такой переход к новому серийному или мелкосерийному изготовлению трудностями для производственных инженеров из области авиастроения. Разработка проекта ARUM направлена на создание IT-решения для уменьшения рисков, для планирования и поддержки принятия решений при переходе к серийному производству непростых технических работ в сфере строения воздушных судов.

1.2 Интернет-агенты

При развитии интернета также возросло количество сайтов. На данный момент карта интернета очень обширна и поиск какой-либо информации занимает большое количество времени.

На рис.1.3 изображена карта интернета, в которой ежесекундно взаимодействую тысячи агентов.

Такая система позволяет нам получать актуальную информацию без использования лишних операций отсеиваний старой информации.

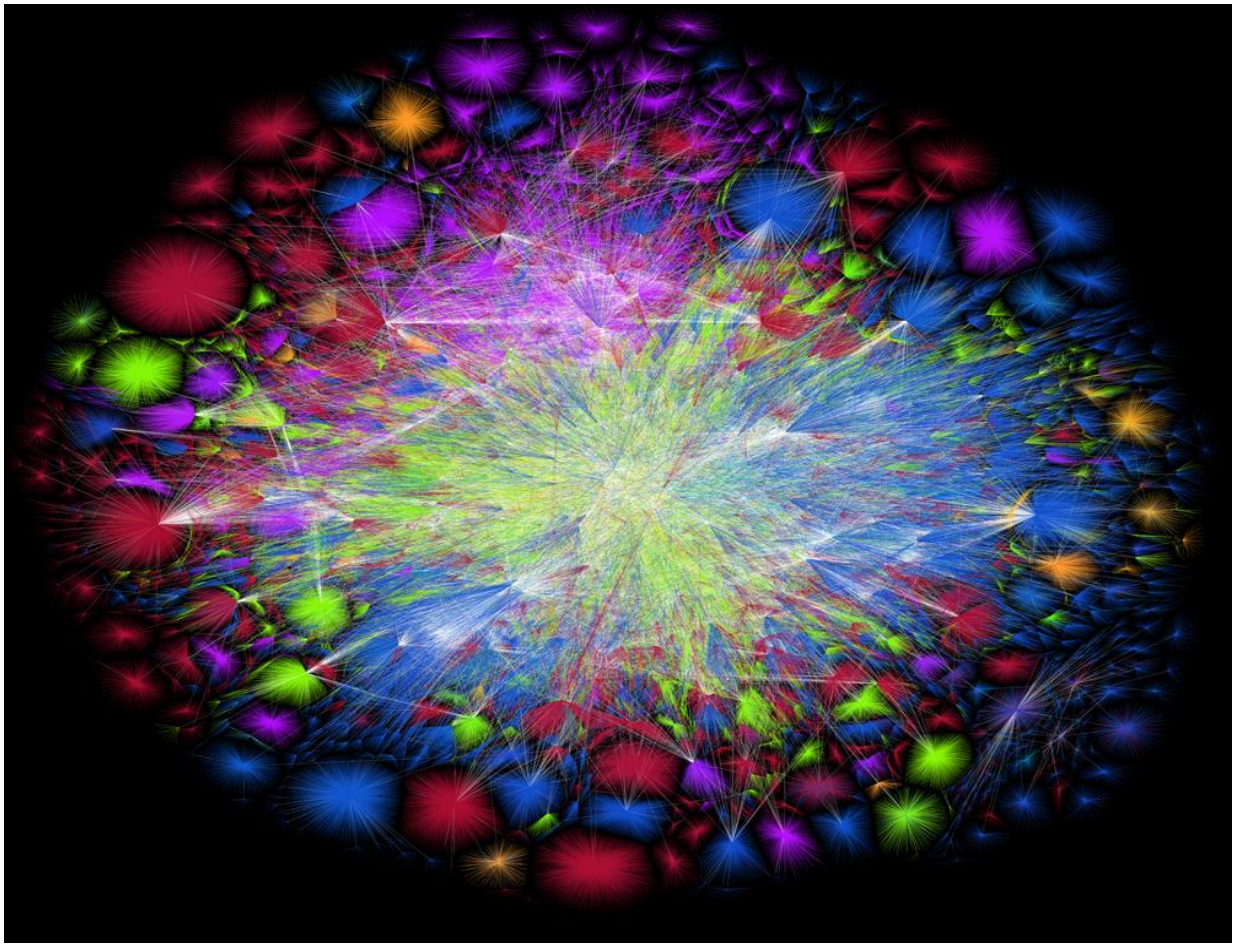

Рисунок 1.3 – Карта интернета

Агенты используют базы данных для поиска релевантных результатов. Для этого пользователь вводит фразу и результатом будет список страниц, которые подходят своим наполнением к фразе.

На данный момент почти все поисковики используют агентов для работы с пользователями и выявлением желаний пользователей, при этом составляя логические цепочки.

Кроме того, для того, чтобы поиск страниц происходил достаточно быстро – существуют агенты, которые собирают статистики и обновляют базу данных. Тем самым поиск необходимой информации занимает очень короткое время.

1.3 Агенты в логистике

Наша мировая экономика основана на торговле и не всегда торговля товаром осуществляется из рук в руки. Большинство товаров, которые используются людьми, прибывают из других стран, где производство данных товаров выгодно. Чтобы доставить товар людям приходилось вручную собирать информацию о доступных маршрутах и времени доставке, рисках и многих других параметрах. Но сейчас этим занимаются агенты, которые могут в реальном времени определить наилучший маршрут с наименьшими затратами и выгодной скоростью доставки.

На рис.1.4 изображена программная система компании Magenta, в которой интегрированы программные агенты.

Данная компания привлекла к разработке агентов ввиду их высоких показателей скорости работы и отклика. Использовав агентов в качестве помощников, программное обеспечение повысило уровень предлагаемых услуг.

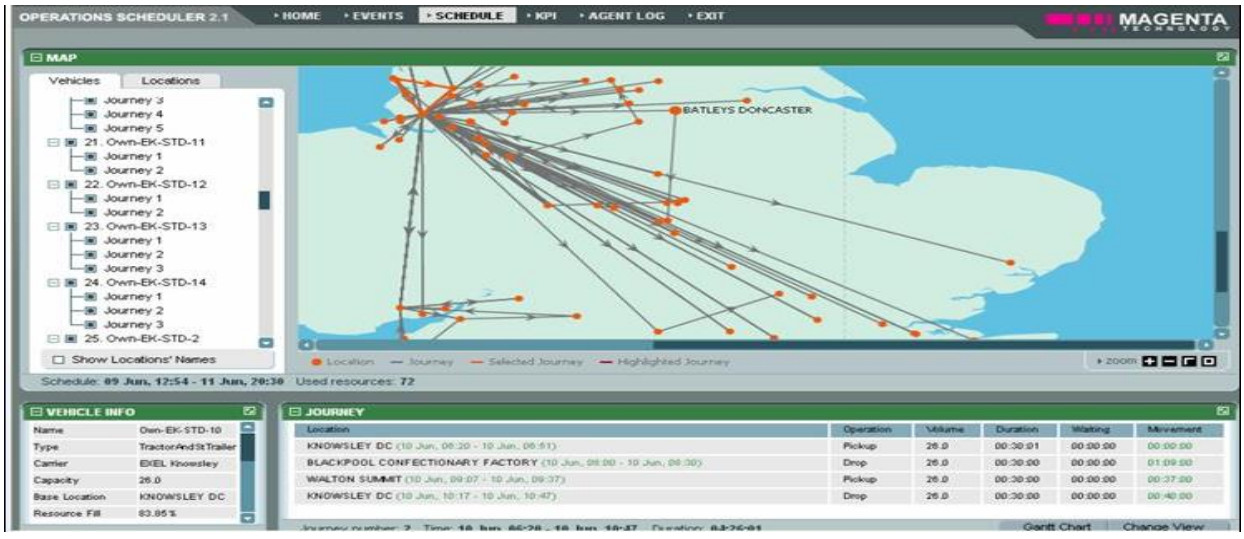

Рисунок 1.4 – Компания MAGENTA использует агентов в логистике

Использование агентов помогает решить многие конфликты в доставке, которые могли возникнуть во время транспортировки груза. Также помощь в принятии решений и уведомление об их принятии агентами – полезная функция для управления компанией.

1.4 Сравнительная характеристика известных систем

Данные системы имеют совершенно различные окружения, где они взаимодействуют друг с другом и окружением. Также в них заложены совершенно разные задачи, цели и пути их достижения.

Тем не менее, агенты имеют и схожие принципы, как, например, свобода действий работы в окружении, влияние на среду, обучение и многое другое. Однако, в данном случае, агенты, использующие в логистической системе, не имеют таких задач как обучение, а в двух других системах они обязательны. Но влиять на окружение имеют возможность все три системы и поэтому, это выводит полезность систем на высокий уровень.

Агенты, использующие в системе логистики просты по сравнению с двумя остальными, но наиболее сложно-структурированными агентами является система поддержки космических аппаратов. Такая система имеет большое количество взаимодействующих единиц и по своему функционалу они все разнообразны и в большинстве своем – независимы.

То есть для решения какой-либо проблемы агент должен найти решение без посторонней помощи и в критический момент совершить это действие.

Соберем всю информацию в таблицу ниже.

Таблица 1.1 – Сводная характеристика агентов в программных системах

Характеристика	Агенты в системе космических	Интернет агенты	Агенты в логистических

	аппаратов		системах
Внутренняя модель	Развитая	Развитая	Развитая
Обучаемость	Присутствует	Присутствует	Отсутствует
Мотивация	Развитая система мотивации	Примитивная система мотивации	Примитивная система мотивации
Адаптивность	Высокая	Высокая	Малая
Модульная архитектура	Есть	Есть	Нет
Память	Есть	Есть	Нет
Реакция	Высокая	Медленная	Высокая

Программные системы, использующие агентов в качестве помощников или частей управления программой, выигрывают значительное количество времени по решению любой сложности задачи.

1.5 Анализ деятельности предприятия

Индивидуальный Предприниматель Лебедев Артур Владимирович предоставляет собой компанию с широким спектром современного оборудования световых приборов как отечественного, так и зарубежного производства для различных отраслей экономики и социальной сферы. На рис.1.5 изображен текущий логотип компании.

Рисунок 1.5 – Логотип компании

Представительство компании работает в городе Тольятти и имеет гибкие возможности работы с клиентами – как через интернет или телефон, так через магазин.

Компания сотрудничает с предприятиями крупного, малого и среднего бизнеса.

В сферу деятельности входит:

- Продажа световых приборов физическим лицам;
- Продажа световых приборов юридическим лицам;
- Доставка и установка световых приборов;

Продажа световых приборов – основное направление предприятия. Кроме световых приборов компания распространяет дополнительное оборудование.

Полный список категорий:

1) Люстры
   1.1) Потолочные
   1.2) С пультом дистанционного управления
   1.3) Люстры с вентилятором
   1.4) Подвесные
   1.5) Еврокаркасы
   1.6) Люстры в детскую
2) Светильники споты

3) Светильники

3.1) Настенные / Бра

3.2) Настенно-потолочные

3.3) Точечные светильники (для натяжных потолков и пластика)

3.4) Настольные лампы

3.5) Торшеры

3.6) Светодиодные прожекторы

3.7) Светодиодные светильники

3.8) Светильники с солнечной батареей

3.9) Уличные, садово-парковые светильники

4) Лампы

5) Электроустановочные изделия

6) Встраиваемые светильники

На предприятии ведется финансовая и экономическая деятельность, в которой используется компьютерная техника для выполнения финансовых операций как между продавцом и торговцем, так и внутренний учет.

На рис.1.6 изображена организационная структура предприятия на данный момент.

Рисунок 1.6 – Организационная структура предприятия

Основным персоналом являются:

1) Директор
2) Работник склада
3) Кассиры
4) Работники зала

Взаимодействие работников между собой позволяет фирме быстро реагировать на запросы клиентов. Однако, на предприятии присутствует текучка кассиров и работников зала, и поэтому, наем работников и все остальные вопросы по кадрам входит в обязанности директора.

В остальном же, предприятие имеет стандартный план работы каждой позиции персонала.

Глава 2 Принципы построения программных агентов и алгоритмы их взаимодействия в виртуальной среде

2.1 Понятие интеллектуального агента

Дадим определение термина агента. Ввиду того, что эта работа о разработке алгоритмов агентов – конечно же, мы должны понимать, чем же является агент. Однако, здесь нет никакой договоренности: нет общепринятого определения термина агент и, действительно, существует много дебатов и споров на эту тему [12; 14; 20]. По существу, в то время как существует общее мнение о том, что автономия является центральным понятием агента, существует небольшое соглашение за пределами него. Часть трудностей состоит в том, что различные атрибуты, ассоциируемые с агентами, имеют разную важность для разных областей. Таким образом, для некоторых приложений, способность агентов учиться из их опыта имеет первостепенное значение; а для других приложений, учится не только не важно, но и нежелательно [18-20].

Тем не менее, некоторая классификация определения важна – в противном случае, существует опасность, что этот термин потеряет всякий смысл. Определение, представленное здесь, является заимствованным и адаптировано из следующей формулировки: Агент представляет собой компьютерную систему, которая находится в некоторой среде и которая способна к автономной деятельности в этой среде в целях удовлетворения своих проектных задач [13; 15].

Есть несколько моментов, на которые нужно обратить внимание в этом определении. Во-первых, определение относится к «агентам», а не к «интеллектуальным агентам». Различие сделано умышлено: это обсуждается более подробно ниже. Во-вторых, определение ничего не говорит о том, в какой среде агент находится. Опять же, это сделано намеренно: агенты могут находиться

во многих типах окружающей сред. В-третьих, мы не дали определение автономии. Как и сами агенты, автономия является несколько сложнее концепцией, чтобы его заменить, но я имею в виду это в том смысле, что агенты способны действовать без вмешательства человека или других систем: они контролируют как свое собственное состояние, так и свое поведение[14]. Рис.2.7 дает нам абстрактный вид верхнего уровня агента.

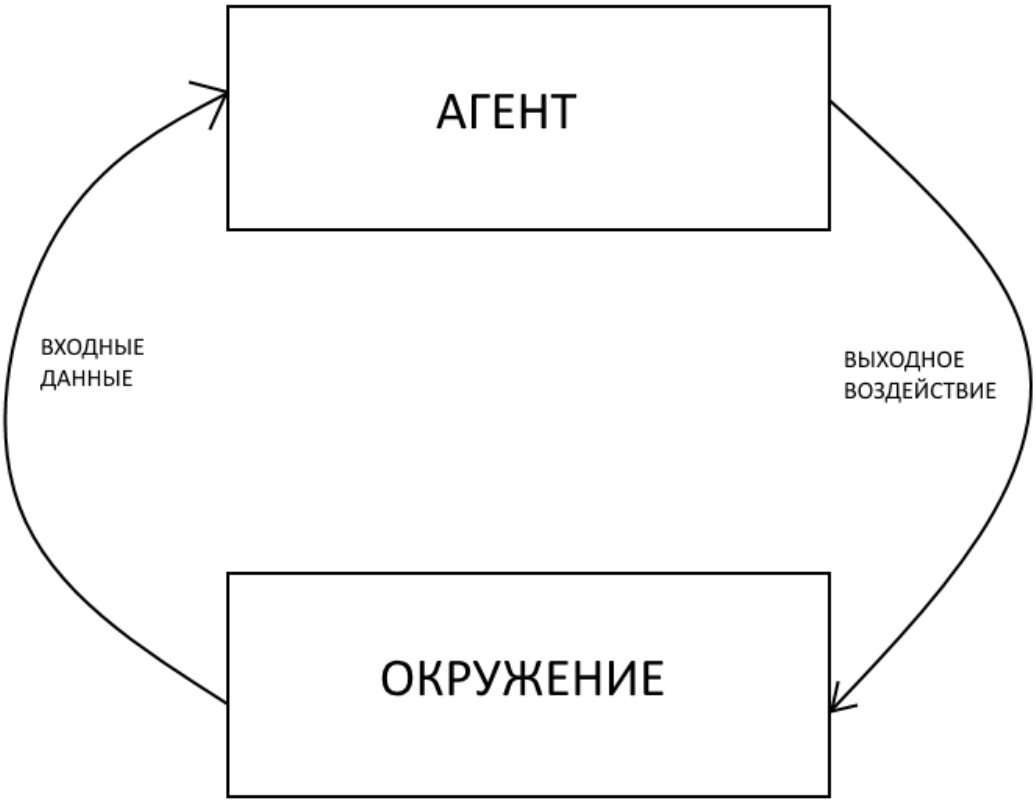

Рисунок 2.7 – Агент и его окружение.

На этом изображении мы можем увидеть результат действия, порожденное агентом, для того, чтобы повлиять на его окружение. В большинстве областей разумной сложности агент не будет иметь полный контроль над своим окружением. Он будет иметь, в лучшем случае, частичный контроль над тем окружением, где оно может влиять на него. С точки зрения агента это означает,

что то же самое действие, совершаемое дважды в очевидно идентичных обстоятельствах, может иметь совершенно разные эффекты и в частности вообще может не иметь желаемого эффекта. Таким образом, агенты во всех, кроме самых тривиальных систем, должны быть готовы к возможности неудачи. Мы можем подытожить эту ситуацию, формально сказав, что окружение не является детерминированным [16].

Как правило, агенты будут иметь репертуар действий, доступных для него. Этот набор возможных действий представляет агентов со способностями влияний: это его способность изменять среду. Обратите внимание, что не все действия могут быть выполнены во всех случаях. Например, действие «подними стол» применяется лишь в тех случаях, когда вес стола достаточно мал, что агент может поднять его. Кроме того, действие «купи Ferrari» потерпит неудачу, если область недостатка капитала будет доступна, чтобы сделать это. Поэтому, действия имеют предварительные условия, связанные с ними, которые определяют возможные ситуации, в которых они могут быть применены [17].

Ключевой проблемой агента является то, что решить, какое из действий следует выполнить, для того, чтобы наилучшим образом удовлетворить свои проектные задачи. Архитектуры агентов мы рассмотрим далее. Сложность процесса принятия решения может зависеть от целого набора различных свойств окружения.

2.2 Архитектуры интеллектуальных агентов

Перейдем к архитектуре агентов. Важно знать о том, какие архитектуры существует, чтобы выбрать наиболее подходящую для реализации наших алгоритмов. На данный момент выделяют четыре класса агентов:

- Агенты, основанные на логике – в которой принятие решений осуществляется через логический вывод;

- противодействующие агенты – в которых принятие решений осуществляется в той или иной форме прямой отображение ситуации к действию;
- вера-желание-намерение – агенты, в которых принятие решений зависит от манипуляций структур данных, представляющих убеждения, желания и намерения агента;
- многоуровневая архитектура – это архитектура, в которой процесс принятия решений реализуется с помощью различных слоев программного обеспечения, каждый из которых является более или менее явным образом рассуждением об окружающей среде в различных уровнях абстракции.

Каждая архитектура работает в своем окружение, поэтому, программисты редко используют несколько типов архитектур в одном окружении [7; 8]. В основном, в одном окружении мы можем проследить взаимодействие нескольких агентов одного типа архитектуры, которые выполняют проектные задачи, делегируя их между собой. На рис.2.8 изображена структура взаимодействия агентов для принятия решения.

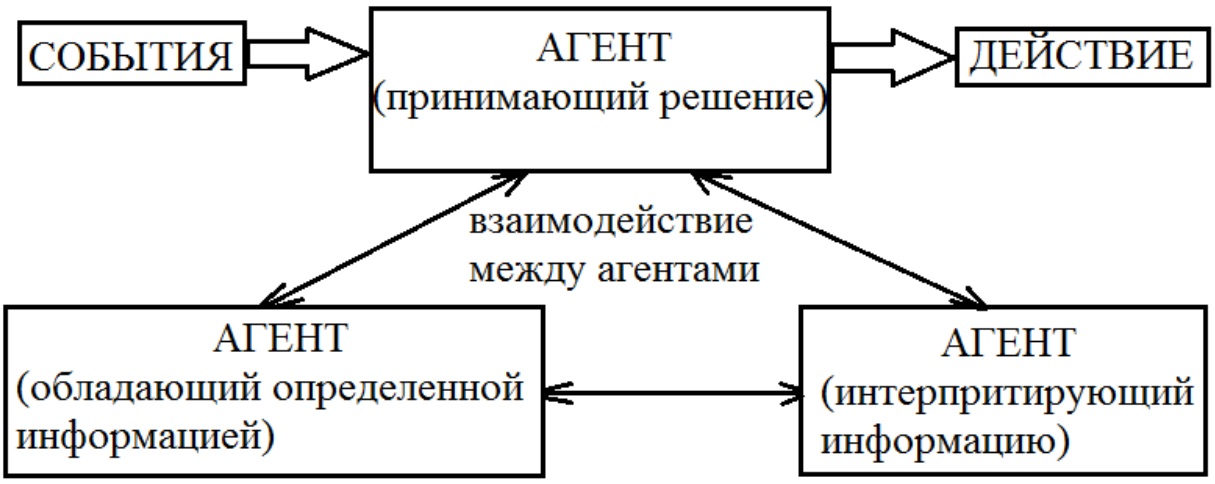

Рисунок 2.8 – Взаимодействие агентов для принятия решения

Типологию агентов мы можем разделить на когнитивных агентов и реактивных. Когнитивные или интеллектуальные агенты имеют улучшенный и дополняемой знаковой моделью внешнего мира и это добивается из-за присутствия в их базы знаний, конструкций анализа и решений действий [9; 10]. Однако, реактивные агенты не имеют расширенного представления внешнего окружения, ни функционала многоступенчатых рассуждений, ни определенного количества свободных ресурсов. Таким образом, мы видим еще одно важное различие между реактивными и интеллектуальным агентами, относящийся к потенциалу прогнозирования модификации окружения и, как следствие, своего будущего.

Сделаем сравнительный анализ этих типов. В качестве критериев используем: внутренняя модель внешнего мира; рассуждения; мотивация; память; реакция; адаптивность; модульная архитектура; состав многоагентной системы

Таблица 2.2 – Сравнительный анализ типов агентов

Характеристики	Реактивные агенты	Интеллектуальные агенты
Мотивация	Простая мотивация и побуждения, основанные на выживание	Сложная система побуждений , включающая желания, убеждения, намерения
Реакция	Быстрая	Медленная
Внутренняя модель внешнего мира	Примитивная	Развитая
Состав многоагентной системы	Множественное число зависимых друг от друга агентов	Небольшое количество автоматизированных агентов
Рассуждения	Обычные, одношаговые	Составные и

	рассуждения	рефлексивные рассуждения
Адаптивность	Высокая	Малая
Модульная архитектура	Нет	Есть
Память	Нет	Есть

Данный анализ показывает нам, что при создании собственной мультиагентной системы следует учитывать тип и их функциональность исходя из проектных целей. И из-за неправильно подобранной системы, эффективность будет низкая, а активность агентов высокая. Тем самым мы будем наблюдать деятельность агентов, которые не приносят результатов, а лишь создают видимость работы.

2.3 Обоснование выбора инструментальной среды разработки и языка программирования

Исследовав наиболее используемые технологии разработки прикладных приложений можно начать выделять условий, характеризующих технологии с целью разработки на основе которых, методики выбора оптимального средства создания прикладного приложения в зависимости от конкретных задач.

Разработка любого прикладного приложения предписывает определенные требования в зависимости от задач: архитектура, предоставление взаимодействия с информацией, а также от параметров самой технологии. Разработка приложения с графическим интерфейсом является непростой задачей и поэтому была выбрана бесплатная платформа разработки – Unity3D, которая предоставляет разработчикам широкий спектр возможностей и является наиболее популярной средой написания прикладных приложений на множество операционных систем. Исходя из этих требований, можно выделить критерии сравнения следующих языков программирования, которые используются в Unity3D: C# и JavaScript.

JavaScript – прототипно-ориентированный сценарный язык программирования.

JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса.

На JavaScript оказали влияние многие языки, при разработке была цель сделать язык похожим на Java, но при этом лёгким для использования непрограммистами. Языком JavaScript не владеет какая-либо компания или организация, что отличает его от ряда языков программирования, используемых в веб-разработке.

Название «JavaScript» является зарегистрированным товарным знаком компании Oracle Corporation.

C# – объектно-ориентированный язык программирования. Разработан в 1998–2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework и впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270 [1-3].

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML [6].

Переняв многое от своих предшественников – языков C++, Pascal, Модула, Smalltalk и, в особенности, Java – C#, опираясь на практику их использования,

исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов [4; 5] (между тем допускается множественное наследование интерфейсов).

Таким образом, в результате обзора языков программирования для написания прикладного приложения был выбран C#.

2.4 Описание моделей интеллектуальных агентов в компании

Модель агентов в ИП изображена на рис.2.9.

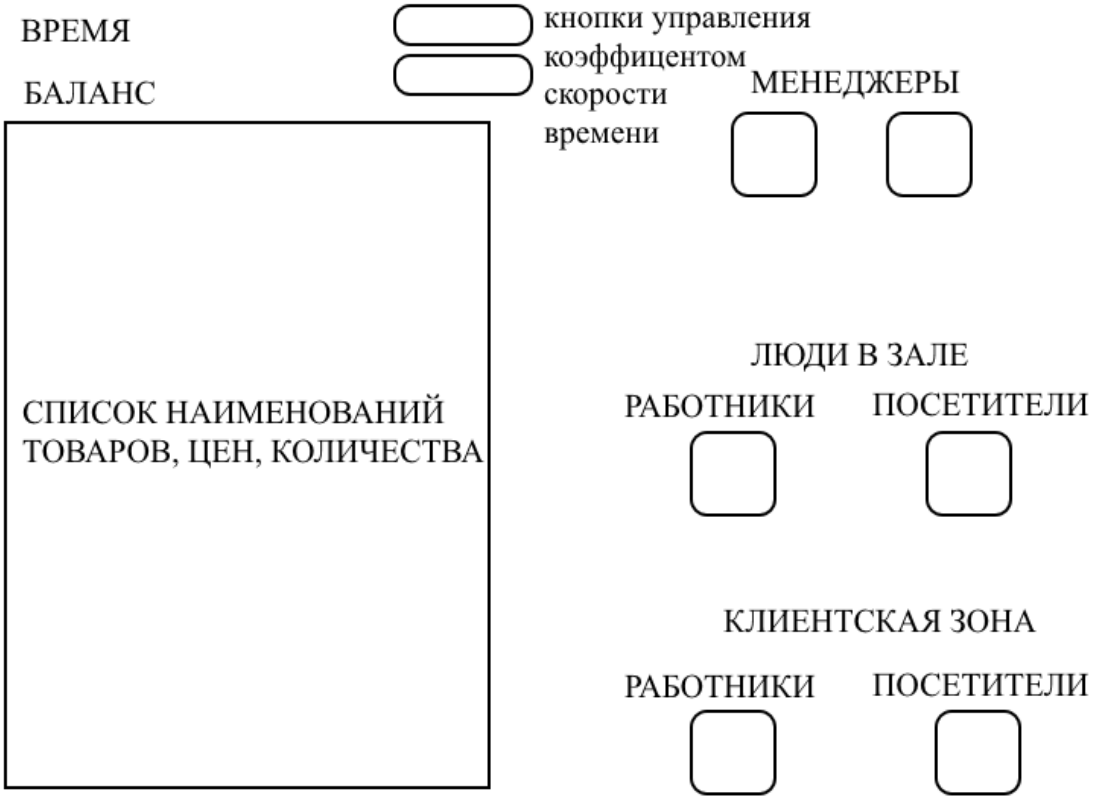

Рисунок 2.9 – Модель агентов

Текущее расположение всех частей и моделей в системе показано на данном рисунке. Модели агентов располагаются на правой стороне. Менеджерами являются один управляющий и один работник склада. Люди в зале также делятся

на работников зала и посетителей. В клиентской зоне располагаются кассиры и посетители с покупками.

2.5 Алгоритм взаимодействия администратора в магазине и его реализация

Работа администратора в магазине является очень важной, поэтому, при разработке данного агента, были выделены следующие его обязанности:

- Проверяет работников на рабочих местах;
- Проверяет выделение денег для закупки товаров;
- Выделяет деньги в день зарплаты;
- Проверяет каждый месяц ситуацию с балансом и нанимает или увольняет работников.

Проверка работников осуществляется алгоритмом, который просматривает всех работников и выделяет среди них тех, кто должен работать в этот день и пришел на работу, а тех, кто не пришел – делает выговор. Работник, получивший больше 3 выговоров – увольняется. Остальные работники распределяются по рабочим местам в соответствии со свободными позициями.

Рис.2.10 наглядно демонстрирует модель взаимодействия администратора.

Алгоритм агента администратора также подразумевает изменение состояния агента извне. Изменение состояний возможно также окружению, в котором находится агент.

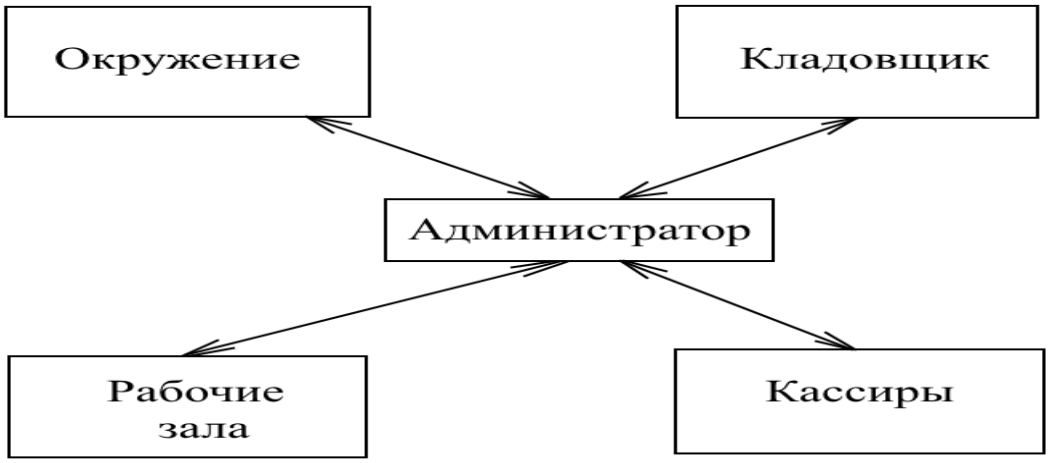

Рисунок 2.10 – Модель взаимодействия администратора

Работник склада каждый день выставляет счет на покупку товаров. Администратор проверяет данную сумму на возможность оплаты и выделяет денег. Если денег не хватает на то, чтобы оплатить покупку всего товара – он выделяет часть нужных денег.

Зарплата выплачивается раз в месяц и составляет сумму всех работников. Администратор проверяет сумму зарплаты и баланс фирмы и если баланс позволяет – выплачивает, а если нет, то отдает товар вместо недостающей суммы.

Проверка ситуации с балансом каждый раз может привести к найму новых работников или их увольнению. Однако если увольнять некого, то такая фирма закрывается. Тем самым мы видим, что данная схема старта бизнеса – невыгодна. Однако, сто процентов, что данная схема не сработает в реальном мире, программа дать не может. Предусмотреть все варианты развития событий не было целью данной работы.

2.6 Алгоритмы поведения покупателей в магазине и их реализация

Поведение покупателей в магазине происходит следующим образом:

1) Покупатель появляется в магазине со списком товаров, который он хочет купить.
2) По прошествии 20 минут покупатель кладет товар в корзину. Если нужного товара нет, он игнорирует данную позицию.
3) Покупатель выбирает наименее загруженную очередь к кассе и ждет взаимодействия с покупателем.
4) Покупатель приобретает товар и удаляется из окружения.

При появлении покупателя в магазине должно пройти 20 минут, перед тем, как он найдет нужный товар и встанет в очередь, где находятся меньшее количество других покупателей. Это сделано для того, чтобы показать более реальные развития событий в магазине. Причем магазин закрывается ровно в 8 часов и покупатели, которые не успели приобрести товар, просто удаляются из магазина или из очереди.

На рис.2.11 изображена модель взаимодействия гостей в виртуальном окружении.

На изменение состояния агента-гостя может влиять одно только окружение, в котором агент находится и агент-кассир. Остальные агенты не имеют никакой связи и никак с ним не взаимодействуют.

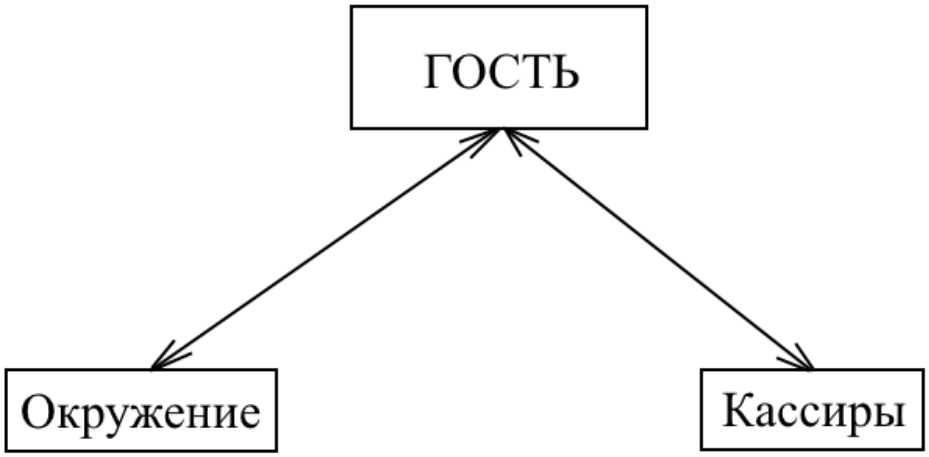

Рисунок 2.11 – Модель взаимодействия гостей

Выбирая товар в корзину, покупатель также смотрит на свой баланс, который генерируется при появлении агента в магазине. Если суммы не хватает на конкретный товар в списке – он его не берет. Или если товара нет на полках стеллажей магазина – он тоже игнорирует данную позицию и забывает про него.

Встав в очередь, покупатель бездействует и ожидает того взаимодействия с кассиром. После совершения покупки агент удаляется из очереди и окружения. Кроме того, у него списываются деньги в размере цены товара и двадцати процентов наценки.

2.7 Алгоритм взаимодействия кассира и работника зала в магазине и его реализация

Кассир, как и работник зала, являются важной составляющей в жизни магазина. Поэтому, для создания агентов, которые осуществляют взаимодействие в виртуальной среде, используем следующую логику поведения. Кассир осуществляет взаимодействие напрямую с покупателями и имеет следующий жизненный цикл:

- Кассир каждый день появляется на работке с 95% вероятностью;

- После появления, менеджер назначает его на кассу и он начинает работу;
- Обслуживает покупателей в течение 15 минут и приглашает следующего в очереди.
- После завершения трудового дня он закрывает кассу и не взаимодействует ни с кем.
- Если кассир пропускает работу более двух раз – его увольняют и агент удаляется из магазина.

На рис. 2.12 изображена модель взаимодействия кассиров в виртуальном окружении.

Рисунок 2.12 – Модель взаимодействия кассиров

Поведение агента работника зала описывается следующим образом:

- Работник зала каждый день появляется на работе с 95% вероятностью;
- После появления на работе, менеджер назначает работника в зал и он сразу начинает работу;

- Каждые 10 минут работник зала выкладывает одну единицу товара из склада на стеллажи;
- Если все стеллажи заполнены, работник бездействует;
- После завершения трудового дня он закрывает позицию и прекращает взаимодействие с кем-либо;
- При получении 3 выговоров – работник торгового зала увольняется и агент удаляется из магазина.

На рис.2.13 изображена модель взаимодействия работников зала в виртуальном окружении.

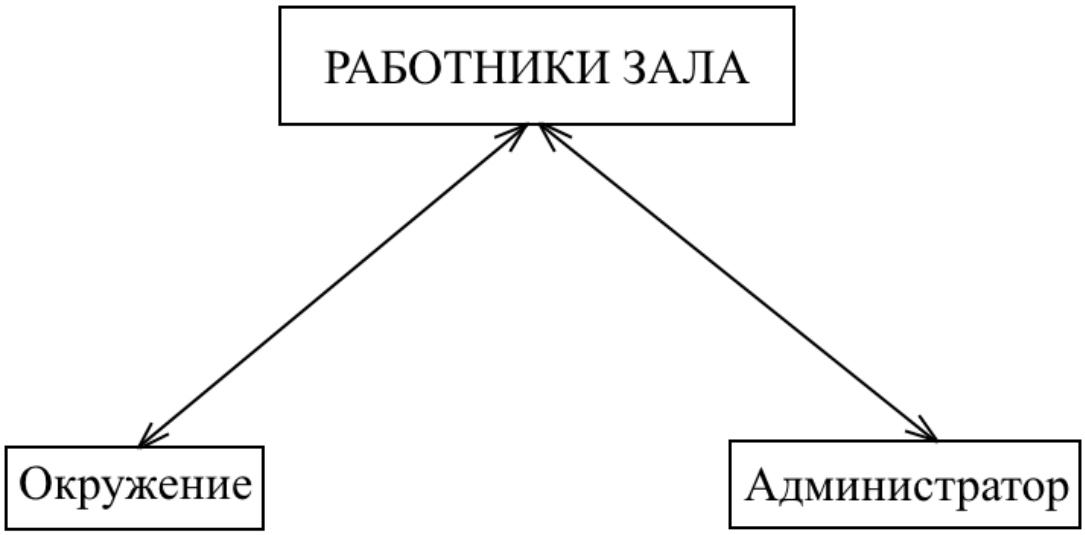

Рисунок 2.13 – Модель взаимодействия работников зала

Поведения агентов описаны в одном классе, поэтому, они взаимозаменяемые. То есть, если менеджер видит, что кассиров нет на кассах – он может заменить им работником торгового зала.

2.8 Алгоритм взаимодействия работника склада в магазине и его реализация

Агент, который занимает должность работника склада, имеет следующие обязанности:

- Каждый день появляется на работе и приступает к работе сразу, без указания администратора;
- Первым делом проверяет наполненность склада товаром, и если один из позиций товаров имеет наполненность менее 50%, то агент собирает полную информацию по каждой из позиций и выставляет счет на закупку товаров;
- Как только администратор выделяет денег – кладовщик проверяет, выделилась ли нужная сумма и если выделилась вся запрошенная сумма, то он закупает весь товар, а если часть, то закупает исходя из выделенной суммы;
- Кроме того, если администратор запросит выделить продуктов на сумму, которую он пришлет, кладовщик удалит нужное количество товара со склада и вернет сумму, которая вышла в итоге;
- При завершении рабочего дня агент закрывает позицию и бездействует остальное время.

На рис.2.14 изображена модель взаимодействия кладовщика в виртуальном окружении.

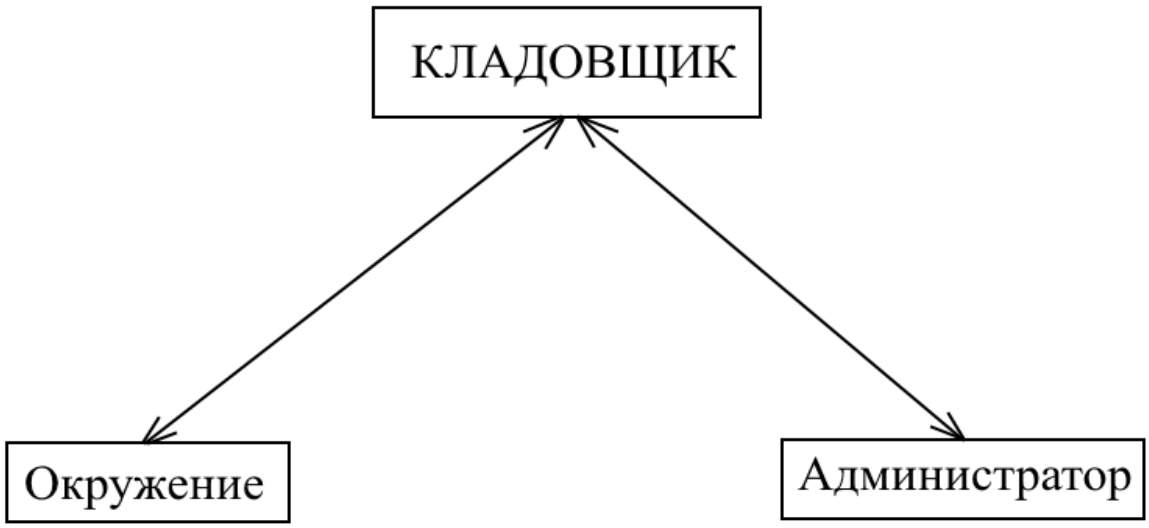

Рисунок 2.14 – Модель взаимодействия кладовщика

Данный агент взаимодействует лишь с администратором и окружением. Он не может быть заменен никаким другим агентом и имеет сложную структуру существования в системе.

Глава 3 Реализация разработанных алгоритмов и тестирование программной системы

3.1 Реализация основных классов приложения

Реализация классов произведена на основе алгоритмов, рассмотренных во второй главе.

На рис.3.15 изображена структура класса Employee в системе.

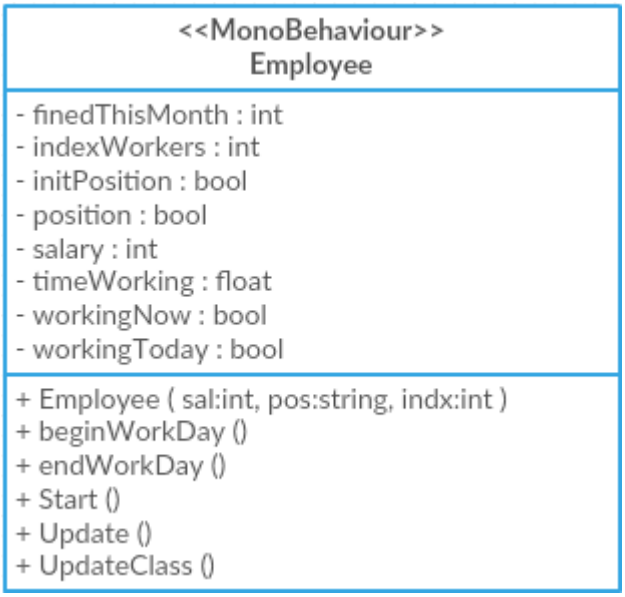

Рисунок 3.15 – Структура класса Employee

Класс Employee относится к алгоритмам программных агентов кассиров и работников торгового зала. Данный класс имеет 8 полей и 6 методов.

На рис.3.16 изображена структура класса Storekeeper в системе.

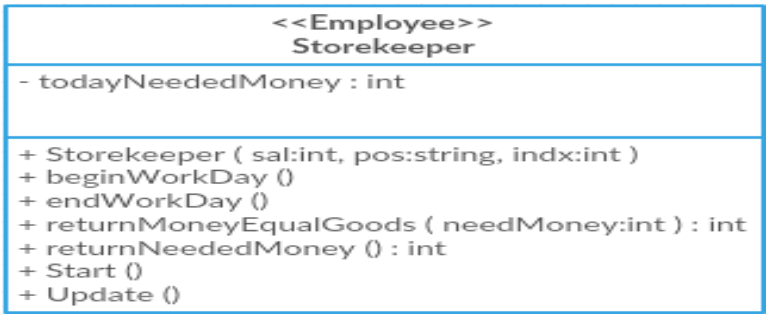

Рисунок 3.16 – Структура класса Storekeeper

Класс Storekeeper наследует класс Employee и относится к программному агенту работника склада. В классе 9 полей и 7 методов.

На рис.3.17 изображена структура класса manager в системе.

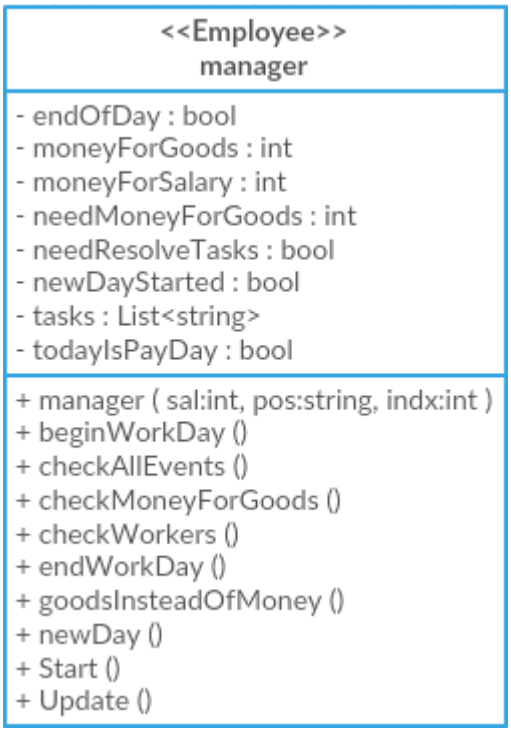

Рисунок 3.17 – Структура класса manager

Класс manager наследует класс Employee как и класс Storekeeper. Он относится к программному агенту – менеджеру. В классе присутствует 16 полей и 10 методов.

На рис.3.18 изображена структура класса guests в системе.

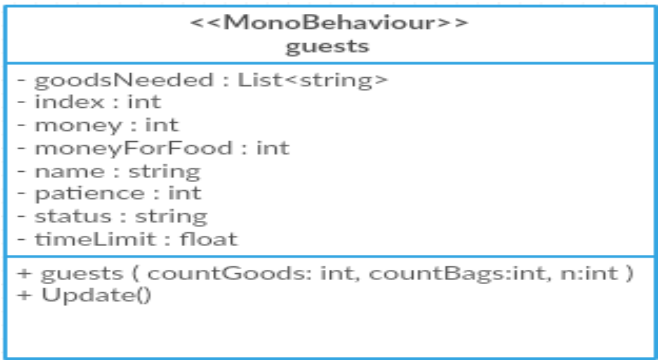

Рисунок 3.18 – структура класса guests

Данный класс относится к программному агенту гостя. Класс содержит 8 полей и 2 метода.

Класс core изображен на рис.3.19.

В классе core реализованы все основные функции по поддержки жизнеобеспечения виртуальной среды. Каждую секунду система обновляет состояние всех объектов и изменяет состояния как программных агентов, так и других множеств объектов.

В обновление входит поддержка перехода виртуального магазина в режим работы или закрытие. Также данный класс содержит переменные, которые пользователь может изменять в интерфейсе, тем самым влияя на окружение и взаимодействующих там программных агентов.

Все агенты, которые существуют в виртуальной среде, регистрируют свою ссылку в данном классе. Это сделано для того, чтобы окружение знало, какие объекты должны быть оповещены в случае изменение статуса.

Рисунок 3.19 – Структура класса core

Класс core служит для имитации виртуальной системы работы магазина. Программные агенты взаимодействуют с данным классом как с реальным окружением.

3.2 Разработка основного интерфейса приложения

Для того, чтобы пользователь видел работу агентов – интерфейс приложения был разработан используя 2D графику платформы Unity3D.

На рис.3.20 изображен интерфейс реализованной системы.

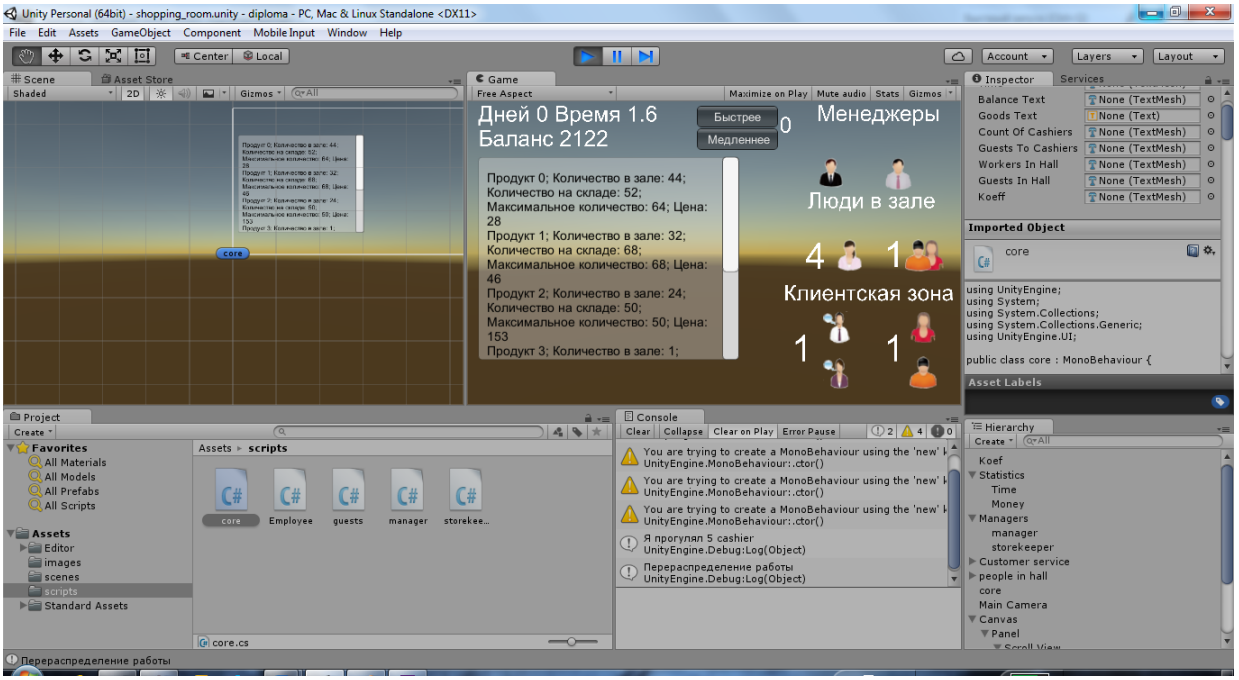

Рисунок 3.20 – Интерфейс приложения

Объекты, выводимые в интерфейсе, условно, делятся на 3 части. На рис.3.21 изображена первая часть интерфейса, которая выводит основную информацию о виртуальной среде, связанную с временной частью и балансом магазина.

Рисунок 3.21 – Интерфейсная часть 1

Также позволяет нам взаимодействовать с течением времени, используя кнопки. Кнопка «Быстрее» ускорит время в 2 раза. Кнопка «Медленнее» замедлит время в 2 раза. Коэффициент ускорения времени не может быть меньше нуля. Коэффициент равный нулю запускает обычный режим отсчета времени. Поле вывода времени и баланса используют шрифты большого размера для более удобного отслеживания основных параметров. На рис.3.22 изображена панель вывода информации о товарах.

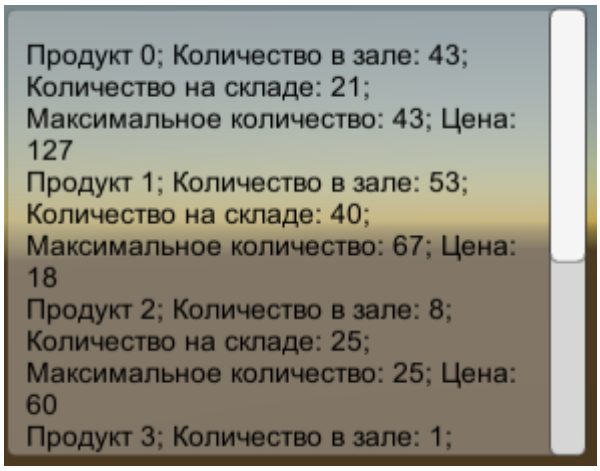

Рисунок 3.22 – Интерфейсная часть 2

Панель сделана с заполняемым контентом и использует прокрутку внутреннего контейнера. По каждой позиции товаров выводится его название, количество товаров выставленного на стеллажи в зале, количество товара на складе и закупочная цена. Обновление происходит с каждым обновлением экрана.

Последняя часть интерфейса отображена на рис.3.23 и показывает пользователю агентов, которые сейчас взаимодействуют в виртуальной системе.

Рисунок 3.23 – Интерфейсная часть 3

Менеджерами являются агент-администратор и агент-кладовщик. Людьми в зале служат агенты-покупатели и агенты работники торгового зала. В клиентской зоне находятся также агенты-покупатели и агенты-кассиры.

Изображения гостей, которые используются в программе, отображены на рис.3.24.

Рисунок 3.24 – Модели интерфейса

В интерфейсе рядом, слева с данными изображениями находится цифра, которая показывает количество агентов взаимодействующих в системе.

Рисунок 3.25 используется в качестве отображения работника торгового зала.

Рисунок 3.25 – Модель интерфейса

И, кроме того, используется рядом цифра, отображающая количество агентов, относящихся к этой должности.

На рис.3.26 изображены кассиры и также используется рядом с ними цифра, означающая количество агентов взаимодействующих с покупателями.

Рисунок 3.26 – Модели интерфейса

Менеджеры, которые используются в приложении, изображены на рис.3.27.

Рисунок 3.27 – Модели интерфейса

Данные изображения не имеют рядом с собой цифр потому, что существует только один администратор и один кладовщик.

3.3 Журнал событий работы агентов

Для того, чтобы пользователь знал что происходит в системе пользователь получает каждодневный отчет о происходящих событиях в магазине.

Данный отчет содержит всю информацию и хранится в txt файле. Файл создается при запуске приложения и обновляется в течение жизненного цикла всей программы.

На рис.3.28 изображен пример журнала событий в текстовом редакторе.

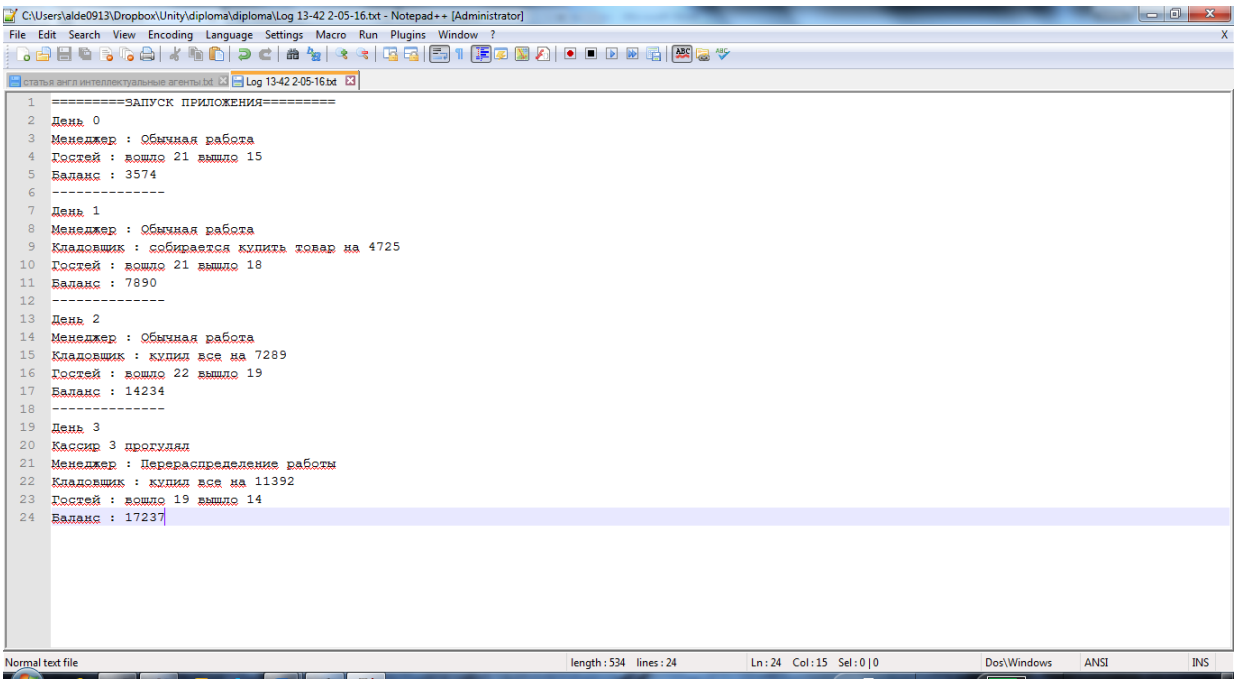

Рисунок 3.28 – Файл отчета

Название файла отчета формируется из слова «Log», времени запуска приложения и текущей даты.

Заполняемость файла происходит каждый рабочий день в приложении и формат заполнения следующий: первым словом является название агента – от кого данная запись и последующая часть – информативная. Какого-либо ограничения на количество запись, совершаемым агентом – нет. Чем больше

информативности мы можем получить от агента – тем удобнее нам будет понимать логику решений взаимодействия с агентами.

В основном, агент протоколирует то, как он воздействует на систему – каждый шаг, который был выполнен, он не записывает. Не имеет смысла писать то, что, например, агент кладовщик в данный момент пытается найти количество товара на складе, которого меньше 50%. Ввиду того, что данное условие может быть не выполнено и оно никак не воздействует на окружающую среду – пользователю это знать не стоит.

Перечень сообщений и обозначений описан в таблице ниже.

Таблица 3.3 – Обзор системного лога сообщений

Сообщение	**Описание**
День 0	Означает, что сейчас идет день, порядковый номер который 0.
Менеджер : Обычная работа	Означает, что агент-администратор взаимодействует с другими агентами-кассирами и агентами работниками зала, расставляя их по позициям соответствующих их должностям.
Менеджер : Перераспределение работы	Означает, что агент-администратор взаимодействует с другими агентами-кассирами и агентами работниками зала, расставляя их по позициям в соответствии с количеством занятых позиций. Чтобы на каждой из позиций был хотя бы 1 работник.
Менеджер : Не работаем сегодня	Означает, что агент-администратор взаимодействует со всеми агентами,

	уведомляя их, что магазин не работает сегодня по причине нехватки персонала.
Менеджер : Выплатить всю зарплату деньгами	Означает, что агент-администратор выплачивает всем зарплату из баланса магазина.
Менеджер : Выплатить всю зарплату деньгами и товаром	Означает, что агент-администратор выплачивает зарплату из баланса магазина и остатки – товаром.
Менеджер : Был нанят кассир	Означает, что агент-администратор нанял агента-кассира.
Менеджер : Был нанят рабочий зала	Означает, что агент-администратор нанял агента рабочего зала.
Кладовщик : собирается купить товар на 4725	Означает, что агент-кладовщик закупит товар на выделенные деньги.
Кладовщик : купил все на 7289	Означает, что агент-кладовщик закупит товар на все деньги.
Кладовщик : Вернуть товар вместо денег 450	Означает, что агент-кладовщик вернет товара на указанную сумму.
Кассир 3 прогулял	Означает, что агент-кассир под номером 3 не вышел на работу в рабочий день.
Гостей : вошло 21 вышло 18	Означает, что агентов-гостей вошло в магазин 21, а было обслужено на кассе 18.
Баланс : 7890	Означает, что в данный рабочий день баланс составляет 7890.

Данный список может быть расширен при желании заказчика или обновлении алгоритмов агентов.

3.4 Руководство пользователя эксплуатации программной системы

При запуске приложения мы увидим сразу сгенерированный виртуальный мир, в котором уже все агенты функционируют.

На рис.29 изображена запущенная программа. Программные агенты взаимодействуют между собой, поддерживая виртуальное окружение в работоспособном состоянии.

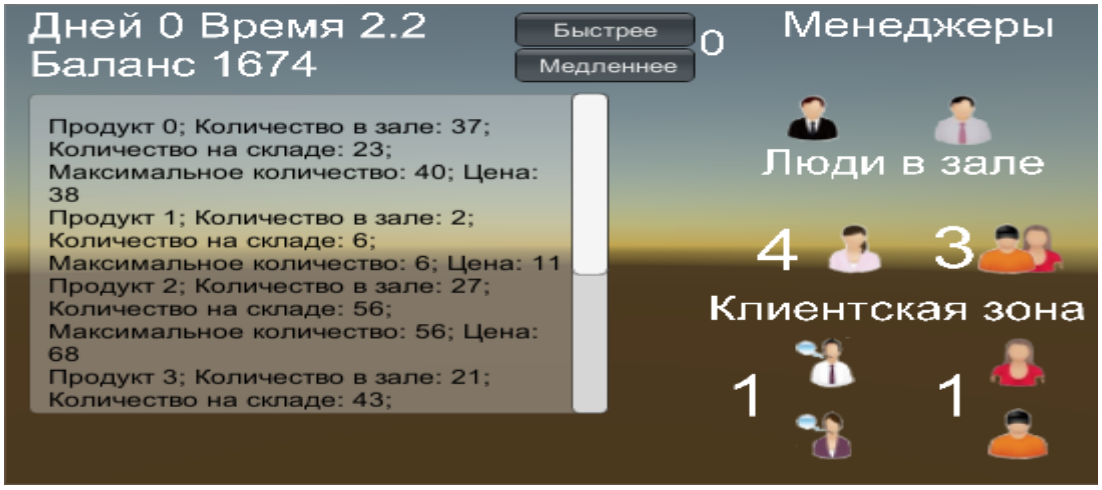

Рисунок 3.29 – Запущенная программа

В свою очередь пользователь получает актуальную информацию о мире и может ускорить течение времени. Завершить работу приложения можно нажатием кнопки «ESC». Все сообщения, сгенерированные во время работы, будут сохранены в локальной папке приложения.

Виртуальный мир имеет при запуске приложения следующую настройку:

- День равен нулю.
- Баланс магазина равен одной тысяче.

- Количество позиций администратора равно 1 и нанят 1 агент-администратор.
- Количество позиций кладовщика равно 1 и нанят 1 агент-кладовщик.
- Количество позиций для кассира равно 2.
- Количество нанятых агентов-кассиров равно 2.
- Количество позиций для работников зала равно 4.
- Количество нанятых агентов-работников зала равно 2.
- Количество позиций товаров равно двадцати.
- Количество максимального товара генерируется в промежутке от единицы до ста.
- Цена на товар генерируется в промежутке от пяти до двухсот.
- Количество товаров в зале вычисляется делением максимального количества товара на два.
- Количество товара на складе равно максимальному количеству товаров.
- Имя товара складывается из слова «Продукт» и числа – порядкового номера в списке товаров.
- Значение коэффициента скорости времени равно нулю.
- Количество людей при открытии магазина в зале равно трем.

Текущий набор является среднестатистическим по своим характеристикам и позволяет наглядно продемонстрировать работу алгоритмов агентов в виртуальной среде.

Данный набор был согласован с заказчиком и может быть изменен по желанию. Все цифры, приведенные в данных настройках, хранятся в переменных, поэтому, их изменение на другие числа никак не повлияет на поведение логики основного виртуального окружения и алгоритмов программных агентов.

Заключение

Использование программных агентов становится неотъемлемой составляющей жизни компьютера, современных фирм и деятельности современных людей.

В данной бакалаврской работе был выделен объект исследования, проанализирована предметная область, определены цели и задачи работы. Проведен обзор существующих разработок типов агентов и выявлена архитектура для реализации подходящего алгоритма программных агентов в компании.

При анализе разработок было выявлено, что для решения данных задач бакалаврской работы потребуется написать свои реализации алгоритмов программных агентов, а не использовать существующих, так как существующие разработки были сделаны для конкретных целей, поэтому и наша разработка будет носить такой же характер.

Был спроектирован интерфейс пользователя и разработаны алгоритмы агентов, а также основное виртуальное окружение, на которое также может влиять каждый из агентов.

Далее алгоритмы, интерфейс и виртуальное окружение были реализованы на платформе Unity3D с использованием языка программирования C#.

В ходе выполнения работы реализованы следующие функции:

- имитация работы торгового зала (окружения)
- взаимодействие агентов с виртуальным окружением
- отображение текущей информацией в виртуальной среде
- выгрузка ежедневных отчетов по ситуации в виртуальной среде.

Список используемой литературы

1. Бен Албахари, Джозеф Албахари. C# 5.0. Справочник. Полное описание языка 2013. — 1008 с. — ISBN 978-5-8459-1819-2
2. Гончаров А.К., Чернов А.А. Планирование сеансов приёма информации с космических аппаратов орбитальной группировки при ограниченном количестве приёмных комплексов // Космонавтика и ракетостроение. – 2014. – № 3. —229с.
3. Джон Скит. C# для профессионалов: тонкости программирования, 3-е издание, новый перевод 2014. — 608 с. — ISBN 978-5-8459-1909-0
4. Кристиан Нейгел и др. C# 5.0 и платформа .NET 4.5 для профессионалов, 2013. — 1440 с. — ISBN 978-5-8459-1850-5
5. Эндрю Троелсен. Язык программирования C# 5.0 и платформа .NET 4.5, 6-е издание, 2013. — 1312 с. — ISBN 978-5-8459-1814-7
6. Э. Стиллмен, Дж. Грин. Изучаем C#. 2-е издание, 2012. — 704 с. — ISBN 978-5-4461-0105-4
7. Рациональный агент. [Электронный ресурс]. – URL: http://ru.wikipedia.org/wiki/Рациональный_агент (дата обращения: 20.03.2016).
8. Многоагентная система. [Электронный ресурс]. – URL: http://ru.wikipedia.org/wiki/Многоагентная_система (дата обращения: 22.03.2016)
9. Интеллектуальные агенты. [Электронный ресурс]. – URL: http://www.itfru.ru/index.php/intellectual-agents (дата обращения: 23.03.2016)
10. Принятие решений. [Электронный ресурс]. – URL: http://club-energy.ru/8_2.php (дата обращения: 25.03.2016)
11. Интеллектуальный агент. [Электронный ресурс] – URL: https://ru.wikipedia.org/wiki/Интеллекутальный_агент (дата обращения: 26.03.2016)

12. Mikkel Birkegaard Andersen, Thomas Bolander, Hans P. van Ditmarsch, and Martin Holm Jensen. Bisimulation for single-agent plausibility models. In Stephen Cranefield and Abhaya Nayak, editors, Australasian Conference on Artificial Intelligence, volume 8272 of Lecture Notes in Computer Science, pages 277–288. Springer, 2013
13. Tom Everitt, Jan Leike, Marcus Hutter. Sequential Extensions of Causal and Evidential Decision Theory, pages 132-139, 2015
14. Ximing Wen, Yongmei Liu and Quan Yu and. Multi-agent epistemic explanatory diagnosis via reasoning about actions. In Francesca Rossi, editor, IJCAI. IJCAI/AAAI, pages 1180-1189, 2013.
15. Guido Boella, Edith Elkind, Bastin Tony Roy Savarimuthu, Frank Dignum, Martin K. Purvis. PRIMA 2013: Principles and Practice of Multi-Agent Systems, pages 345-378, 2013.
16. Hosny A. Abbas, Mohammed H. Amin, Samir I. Shaheen. On the Adoption of Multi-Agent Systems for the Development of Industrial Control Networks: A Case Study, pages 33-50, 2015.
17. Wunder, M.; Kaisers, M.; Yaros, J.; and Littman, M. 2011. Using iterated reasoning to predict opponent strategies. In International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), 593–600.
18. Alejandro Torreño, Eva Onaindia, Óscar Sapena. A flexible coupling approach to multi-agent planning under incomplete information. Knowledge and Information Systems 38(1), pages 141-178, 2014
19. Chopra, A.K., Singh, M.P.: Agent communication. In Weiss, G., ed.: Multiagent Systems, 2nd edition. MIT Press (2013)
20. J. Barbosa and P. Leitão, "Simulation of multi-agent manufacturing systems using AgentBased Modelling platforms," in Industrial Informatics (INDIN), 2011 9th IEEE International Conference on, 2011, pp. 477-482.

Приложение А

Код агента администратора

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
public class manager : Employee{
        public static int needMoneyForGoods=0;
        public static int moneyForGoods = 0;
        public static bool todayIsPayDay = false;
        public static int moneyForSalary = 0;
        public List<string> tasks = new List<string>();
        public bool needResolveTasks=false;
        public static bool newDayStarted=true;
        public static bool endOfDay=false;
   public manager(int sal, string pos, int indx):base(sal, pos, indx){
      salary = sal;
      position = pos;    }
          void Start () {      }
          void newDay(){
      workingNow = true;
      workingToday = true;
      needResolveTasks = true;
                  tasks.Add ("checkWorkers");
                  tasks.Add ("checkMoneyForGoods");          }
          void Update () {
                  if (newDayStarted) {
        if (todayIsPayDay)          {
          tasks.Add("checkDayPay");
          todayIsPayDay = false;          }
        newDay ();
                        newDayStarted=false;                  }
      if (!endOfDay || !needResolveTasks)       {
        checkAllEvents();
        if (tasks.Count == 0)
          needResolveTasks = false;       }   }
          void checkAllEvents(){
                  for (int i=0; i<tasks.Count; i++)
                        switch (tasks [i]) {
           case "checkWorkers":
              if (!core.once)                   return;
              checkWorkers();
                              tasks.RemoveAt(i);
                              break;
                           case "checkDayPay":
                              checkDayPay();
                              tasks.RemoveAt(i);
                              break;
                           case "checkMoneyForGoods":
                              checkMoneyForGoods();
                              tasks.RemoveAt(i);
                              break;
           case "forecastPayments":
              if (core.reportForNextMonth() < 1)
                 core.needDismissEmployee();
              else
```

```
                core.hirePeopleIfNeeded();
          break;                          }          }
        void checkWorkers ()    {
  core.fineEmployee();
  switch (core.returnCountOfWorkers())       {
     case 1:
        core.manageThisWorkersAsUsual();
        core.initCashiersAndGuests = true;
        Debug.Log("Обычная работа");
        break;
     case 2:
        core.initMiddlePos();
        core.initCashiersAndGuests = true;
        Debug.Log("Перераспределение работы");
        break;
     case 3:
        core.closeTodayWork();
        core.hirePeopleIfNeeded();
        Debug.Log("Не работаем сегодня");
        break;       }         }
       void checkDayPay ()         {
  Debug.Log("checkDayPay start " + moneyForSalary);
                if (core.balance >= moneyForSalary) {
                        core.balance -= moneyForSalary;
                        moneyForSalary = 0;
     Debug.Log("checkDayPay all money");                    } else {
     goodsInsteadOfSalary();
     Debug.Log("checkDayPay goods instead money");                  }          }
       void goodsInsteadOfSalary ()          {
  moneyForSalary -= storekeeper.returnMoneyEqualGoods(moneyForSalary);    }
       void checkMoneyForGoods (){
  needMoneyForGoods = storekeeper.returnNeededMoneyForFood();
                if (core.balance >= needMoneyForGoods) {
                        moneyForGoods = needMoneyForGoods;
     core.balance -= needMoneyForGoods;
                        needMoneyForGoods = 0;             } else {
     moneyForGoods = needMoneyForGoods - (needMoneyForGoods - core.balance);             }
  storekeeper.buyGoods(moneyForGoods);          }
public override void beginWorkDay()    {
  newDayStarted = true;
  endOfDay = false;    }
public override void endWorkDay()    {
  newDayStarted = false;
  endOfDay                        =                        true;                                                                  }}
```

Приложение Б

Код агента кладовщика

```
using UnityEngine;
using System.Collections;
public class storekeeper : Employee{
   static int todayNeededMoney = 0;
   public storekeeper(int sal, string pos, int indx):base(sal, pos, indx)    {  salary = sal;  position = pos;    }
            void Start () {                    }void Update () { }
   public static void buyGoods(int m) {
      if (m == 0)            return;
      if (m == todayNeededMoney)  { core.dayLog.Add("Кладовщик : купил все на "+m);
         for (int i = 0; i < core.goodsKeep.Count; i++)
            core.goodsKeep[i].count = core.goodsKeep[i].maxCount;
         todayNeededMoney = 0;} else { core.dayLog.Add("Кладовщик : собирается купить что-то на " + m);
         //ищем минимальную цену товара, чтобы узнать хватит ли у нас денег
         int minCost = core.goodsKeep[0].cost;
         for (int i = 0; i < core.goodsKeep.Count; i++)
            if (core.goodsKeep[i].cost < minCost && core.goodsKeep[i].count<core.goodsKeep[i].maxCount)
               minCost = core.goodsKeep[i].cost;
         if (minCost > m)    return;   int breaker = 0;
         while (minCost < m) {
            for (int i = 0; i < core.goodsKeep.Count; i++)
               if (core.goodsKeep[i].count < core.goodsKeep[i].maxCount && m > core.goodsKeep[i].cost) {
                  core.goodsKeep[i].count++;
                  m -= core.goodsKeep[i].cost;
                  todayNeededMoney -= core.goodsKeep[i].cost;                    }
            //снова ищем минимальную цену товара
            for (int i = 0; i < core.goodsKeep.Count; i++)
               if (core.goodsKeep[i].cost < minCost && core.goodsKeep[i].count < core.goodsKeep[i].maxCount)
                  minCost = core.goodsKeep[i].cost;
            breaker++;
            if (breaker > core.goodsKeep.Count)  break;  }   }  }
   public static int returnMoneyEqualGoods(int needMoney)    {
      int maxReturn = 0;        int calcRetMoney = 0;
      while(calcRetMoney<needMoney){
         int maxCost = 0;    int minCost = 0;
         for (int i = 0; i < core.goodsKeep.Count; i++){
            if (core.goodsKeep[maxCost].cost < core.goodsKeep[i].cost && core.goodsKeep[maxCost].count>0)
               maxCost = i;
            if (core.goodsKeep[minCost].cost < core.goodsKeep[i].cost && core.goodsKeep[minCost].count > 0)
               minCost = i;           }
        if (calcRetMoney + core.goodsKeep[maxCost].cost < needMoney && core.goodsKeep[maxCost].count > 0){
            calcRetMoney += core.goodsKeep[maxCost].cost;
            core.goodsKeep[maxCost].count--;           }           else
            if (core.goodsKeep[minCost].count > 0)  {  calcRetMoney += core.goodsKeep[minCost].cost;
      core.goodsKeep[minCost].count--; } else  return calcRetMoney; maxReturn++; if (maxReturn > 50) break;  }
      Debug.Log("Кладовщик : Вернуть товар вместо денег " + calcRetMoney); return calcRetMoney; }
   public static int returnNeededMoneyForFood() { return todayNeededMoney; }
   void checkGoodsNeeded() { todayNeededMoney = 0; for (int i = 0; i < core.goodsKeep.Count; i++)
         if (core.goodsKeep[i].count <= core.goodsKeep[i].maxCount / 2)
            todayNeededMoney += (core.goodsKeep[i].maxCount - core.goodsKeep[i].count) * core.goodsKeep[i].cost; }
   public override void beginWorkDay() { checkGoodsNeeded(); }
   public          override          void          endWorkDay()                                    {                                    }}
```

Приложение В

Код агента работника

```
using UnityEngine;
using System.Collections;
public class Employee : MonoBehaviour {
    public int salary = 30;
    public int finedThisMonth = 0;
    public bool workingToday = true;
    public bool workingNow = true;
    public string position = "";
    public bool initPosition = false;
    public int indexWorkers = 0;
    float timeWorking = 0.2f;
    public Employee(int sal, string pos, int indx)
    {
        salary = sal;
        position = pos;
        indexWorkers = indx;
    }
            void Start () {       }
            void Update () {   }
    public void UpdateClass() {
        if (!workingNow && initPosition)
            return;
        if(initPosition)
            switch (position) {
                case "cashier":
                    timeWorking -= (Time.deltaTime + core.timeKoeff);
                    if (timeWorking <= 0.0f)
                    {
                        core.popGuests(indexWorkers);
                        timeWorking = 0.15f;
                    }
                    break;
                case "hallworker":
                    timeWorking -= (Time.deltaTime + core.timeKoeff);
                    if (timeWorking <= 0.0f)
                    {
                        core.pushFreeGoods();
                        timeWorking = 0.1f;
                    }
                    break;
            }
    }
    public virtual void endWorkDay()
    {
        workingNow = false;
        initPosition = false;
    }
    public virtual void beginWorkDay()
    {
        if (UnityEngine.Random.Range(0, 100) > 90)        {
            workingNow = false;            Debug.Log("Я прогулял " + indexWorkers+" "+position);        }
        else                                                                workingNow        =        true;                        }}
```

Приложение Г

Код ядра виртуального мира

```
using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine.UI;
public class core : MonoBehaviour {
    //stats
    public TextMesh time;
    public TextMesh balanceText;
    public Text goodsText;
    public TextMesh countOfCashiers;
    public TextMesh guestsToCashiers;
    public TextMesh workersInHall;
    public TextMesh guestsInHall;
    public TextMesh koeff;
    public static float timeKoeff = 0.0f;
    public static bool updKoeff = false;
    float timeLimit = 0.0f;
    int day = 0;
    public static bool initCashiersAndGuests = false;
    public static bool generateGuests = false;
    public static int guestsAtOneDay = 0;
    public static int guestsBuyAtOneDay = 0;
    public static bool once = true;
    public static string dayOrNight = "day";
    public static List<string> dayLog = new List<string>();
    public class goods {
        public string name;
        public int countAtHall;
        public int count;
        public int maxCount;
        public int cost;
        public goods(string n, int c, int cst) {
            name = n;
            count = c;
            maxCount = c;
            cost = cst;
            countAtHall = c / 2; } }
    public class Position {
        public string namePosition;
        public int indxEmpl = 0;
        public bool free = true;
        public Position(string pos, int indx, bool isFree) {
            namePosition = pos;
            indxEmpl = indx;
            free = isFree; } }
    public static int balance = 1000;
    public static List<goods> goodsKeep = new List<goods>();
    public static List<Employee> employees = new List<Employee>();
    public static List<Position> positions = new List<Position>();
    public static List<guests> buyers = new List<guests>();
    public static List<List<int>> cashierRows = new List<List<int>>();
            void Start () {
```

```
        initPositionsAndEmpl();
        initGoods(20, 100, 200);
        initStartedPos();
        initCashierRows();
        for (int i = 0; i < employees.Count; i++)
            employees[i].beginWorkDay();          }
    void OnGUI() {
        if (GUI.Button(new Rect(255, 5, 90, 24), "Быстрее"))
            incrementKoeff();
        if (GUI.Button(new Rect(255, 30, 90, 24), "Медленнее"))
            decrementKoeff();    }
    void Update() {
        timeLimit += Time.deltaTime + timeKoeff;
        if (timeLimit > 8.0f && once)        {
            dayOrNight = "night";
            generateGuests = false;
            for (int i = 0; i < employees.Count; i++)
                employees[i].endWorkDay();
            buyers.Clear();
            breakPosition();
            dayLog.Add("Гостей : вошло " + guestsAtOneDay + " вышло " + guestsBuyAtOneDay);
            once = false;
            guestsAtOneDay = 0;
            guestsBuyAtOneDay = 0;
            dayLogUpdate();        }
        if (timeLimit > 24.0f){
            timeLimit = 0;
            dayOrNight = "day";
            day++;
            if (day == 30)            {
                int moneyForSalary = 0;
                for (int i = 0; i < employees.Count; i++)
                    moneyForSalary += employees[i].salary;
                manager.moneyForSalary = moneyForSalary;
                manager.todayIsPayDay = true;
                day = 0;            }
            for (int i = 0; i < employees.Count; i++) employees[i].beginWorkDay();
            once = true;        }
        if (initCashiersAndGuests) {
            initCashierRows();
            addRandomGuests(3, 3);
            initCashiersAndGuests = false;
            generateGuests = true;    }
        if (dayOrNight=="day" && generateGuests && Math.Round(UnityEngine.Random.Range(1, 20) / (1 + (timeKoeff *
10)*4 )) == 1)
            addRandomGuests(1, 4);
        time.text = "Дней " + day.ToString() + " Время " + (Math.Round(timeLimit, 1)).ToString();
        balanceText.text = "Баланс " + balance.ToString();
        if (updKoeff) {
            koeff.text = (Math.Round(timeKoeff,2) ).ToString();
            updKoeff = false; }
        updateCashiers();
        updateHallWorkers();
        updateGuests();
        updateInfo();  }
    void dayLogUpdate() {
        Debug.Log("_____");
        Debug.Log("День " + day);
        for (int i = 0; i < dayLog.Count; i++) {
```

```
            Debug.Log(dayLog[i]); }
        dayLog.Clear(); }
    void updateCashiers() {
        for (int i = 0; i < employees.Count; i++)
            if (employees[i].position == "cashier")
                employees[i].UpdateClass();}
    void updateHallWorkers() {
        for (int i = 0; i < employees.Count; i++)
            if (employees[i].position == "hallworker")
                employees[i].UpdateClass();}
    void updateGuests() {
        for (int i = 0; i < buyers.Count; i++)
            buyers[i].Update(); }
    private void initGoods(int count, int maxCountG, int maxCosts) {
        for (int i = 0; i < count; i++ )
            goodsKeep.Add(new goods("Продукт " + i.ToString(), UnityEngine.Random.Range(1, maxCountG),
UnityEngine.Random.Range(5, maxCosts)));}
    private void updateInfo() {
        int gs= 0, gh=0;
        for(int i=0; i<buyers.Count; i++)
            switch(buyers[i].status){
                case "hall":
                    gh++;
                    break;
                default :
                    gs++;
                    break;  }
        guestsToCashiers.text = gs.ToString();
        guestsInHall.text = gh.ToString();
        int wc = 0, wh=0;
        for(int i=0; i<employees.Count; i++)
            if (employees[i].workingNow && employees[i].initPosition)
                switch (positions[i].namePosition) {
                    case "cashier":
                        wc++;
                        break;
                    case "hallworker":
                        wh++;
                        break; }
        countOfCashiers.text = wc.ToString();
        workersInHall.text = wh.ToString();
        goodsText.text="";
        for (int i = 0; i < goodsKeep.Count; i++)
            goodsText.text += "\n" + goodsKeep[i].name +"; Количество в зале: "+goodsKeep[i].countAtHall + "; Количество
на складе: " + goodsKeep[i].count+"; Максимальное количество: "+goodsKeep[i].maxCount+"; Цена:
"+goodsKeep[i].cost; }
    private void addRandomGuests(int count, int countBags) {
        guestsAtOneDay += count;
        for (int i = 0; i < count; i++)
            buyers.Add(new guests(goodsKeep.Count, countBags, buyers.Count)); }
    public static void initPositionsAndEmpl() {
        employees.Add(GameObject.Find("manager").GetComponent<manager>());
        employees.Add(GameObject.Find("storekeeper").GetComponent<storekeeper>());
        for (int i = 0; i < 2; i++ )
            employees.Add(new Employee(UnityEngine.Random.Range(20, 30), "hallworker", employees.Count));
        for (int i = 0; i < 2; i++)
            employees.Add(new Employee(UnityEngine.Random.Range(20, 30), "cashier", employees.Count));
        for (int i = 0; i < 4; i++)
            positions.Add(new Position("hallworker", 0, true));
```

```
        for (int i = 0; i < 4; i++)
            positions.Add(new Position("cashier", 0, true));
        positions.Add(new Position("manager", 0, false));
        positions.Add(new Position("storekeeper",0, false));    }
    public static void initCashierRows() {
        for (int i = 0; i < positions.Count; i++)
            if (positions[i].namePosition == "cashier" && !positions[i].free) cashierRows.Add(new List<int>());}
    public static void initStartedPos() {
        for (int i = 0; i < positions.Count ; i++)
            for (int j = 0; j < employees.Count ; j++)
                if (!employees[j].initPosition && positions[i].free && employees[j].position == positions[i].namePosition &&
employees[j].workingNow) {
                    employees[j].initPosition = true;
                    positions[i].indxEmpl = j;
                    positions[i].free = false;  }    }
    public static void initMiddlePos() {
        int cash = 0;
        for (int i = 0; i < employees.Count; i++)
            if (employees[i].position == "cashier" && !employees[i].workingNow)
                cash++;
        for(int i=0; i<employees.Count; i++)
            if (cash > 0) {
         if (employees[i].position == "cashier" && !employees[i].workingNow) employees[i].position = "hallworker";
                if (employees[i].position == "hallworker" && employees[i].workingNow) {
                    employees[i].position = "cashier";
                    cash--;  }           }
        int hall = 0;
        for (int i = 0; i < employees.Count; i++)
            if (employees[i].position == "hallworker" && employees[i].workingNow) hall++;
        if (hall == 0)
            for (int i = 0; i < employees.Count; i++ )
                if (employees[i].position == "cashier" && employees[i].workingNow && hall>0) {
                    employees[i].position = "hallworker";
                    hall--;
                    break;   }
        initStartedPos();    }
    public static void popGuests(int indexCashier)    {
        if (cashierRows.Count == 0)  return;
        int k = 0; //переменная нужна, чтобы найти индекс кассы, к которой привязан текущий кассир
        //индекс кассы ищется порядковым номером вхождения индекса кассира в positions, где имя позиции равен
кассир
        //пока индекс в positions не равен нашему indexCashier - инкрементируем
        for (int i = 0; i < positions.Count; i++)
            if (positions[i].namePosition == "cashier" && !positions[i].free)           {
                if (positions[i].indxEmpl == indexCashier)               {
//-------------------------------------------------------------------------------------------------------
                    try                  {
                        if (cashierRows[k].Count != 0)                      {
                            //ищем индекс покупателя который стоит ближе к кассе текущего кассира
                            //по номеру кассы, чтобы удалить покупателя из магазина и из кассы
                            int ind = 0;
                            for (int l = 0; l < buyers.Count; l++)
                                if (buyers[l].index == cashierRows[k][0]) ind = l;
                            balance += buyers[ind].moneyForFood;
                            //Debug.Log(buyers[ind].name+" "+buyers[ind].moneyForFood + " "+buyers[ind].money);
                            buyers.RemoveAt(ind);
                            cashierRows[k].RemoveAt(0);
                            guestsBuyAtOneDay++;                    }                }
                    catch {
```

```
                Debug.Log("ERROR index");
                Debug.Log("k = "+k);
                Debug.Log("cashierRows = "+cashierRows.Count);
                Debug.Log("index cashier = " + indexCashier);
                for (int s = 0; s < employees.Count; s++)
                   if (employees[s].position == "cashier")
                      Debug.Log("cashier N"+s+" is "+employees[s].workingNow);
                for (int s = 0; s < positions.Count; s++)
                   if (positions[s].namePosition == "cashier")
                      Debug.Log("position ="+s+" empl="+positions[s].indxEmpl+" is "+positions[s].free);
             } } else  k++;    }    }
public static void pushFreeGoods() {
   for(int i=0; i<goodsKeep.Count; i++)
      if (goodsKeep[i].countAtHall < goodsKeep[i].maxCount && goodsKeep[i].count>0) {
         goodsKeep[i].countAtHall++;
         goodsKeep[i].count--;
         return;           }    }
public static void breakPosition() {
   for (int i = 0; i < positions.Count; i++)
      positions[i].free = true;
   for (int i = 0; i < employees.Count; i++)
      employees[i].initPosition = false;
   cashierRows.Clear();    }
public static void incrementKoeff() {
   timeKoeff += 0.1f;
   updKoeff = true; }
public static void decrementKoeff() {
   if (timeKoeff >= 0.1f && timeKoeff<0.2f)
      timeKoeff = 0.0f;
   else if(timeKoeff>0.1f) timeKoeff -= 0.1f;
   updKoeff = true;    }
public void updateTime(int newValue) {
   balance = newValue;
   balanceText.text = balance.ToString();}
public void updateGoods(int indx, int newValue) {
   goodsKeep[indx].count = newValue;    }
public static int ReturnNumRow(int indx) {
   if (cashierRows.Count==0)  return 50000;
   int min = cashierRows[0].Count;
   int countCash = 0;
   for (int i = 0; i < positions.Count; i++)
      if (positions[i].namePosition == "cashier" && !positions[i].free) countCash++;
   int ind=0;
   for (int i = 0; i < countCash; i++)
      if (cashierRows[i].Count < min) {
         min = cashierRows[i].Count;
         ind = i;           }
   cashierRows[ind].Add(indx);
   return ind;    }
public static int returnCountOfWorkers()    {
   //1 - good 2 - middle 3 - bad
   int cash = 0;       int hw = 0;       int allw=0;
   for (int i = 0; i < employees.Count; i++)       {
      if (employees[i].workingNow && employees[i].position == "cashier") cash++;
      if (employees[i].workingNow && employees[i].position == "hallworker") hw++;
      if (employees[i].position == "cashier" || employees[i].position == "hallworker") allw++; }
   int res = 3;
   if (cash + hw == allw)           res = 1;
   if (cash + hw < 2 )          res = 3;
```

```
    if (cash + hw > 1 && cash + hw < allw)            res = 2;
    return res;    }
public static void manageThisWorkersAsUsual()    {        initStartedPos();    }
public static void closeTodayWork() {
    for (int i = 0; i < employees.Count; i++ ) {
        employees[i].workingNow = false;
        employees[i].workingToday = false;        }    }
public static void fineEmployee() {
    for (int i = 0; i < employees.Count; i++)
        if (employees[i].finedThisMonth > 2)            {
            dayLog.Add("Был уволен сотрудник " + employees[i].position);
            employees.Remove(employees[i]);            }
    for (int i = 0; i < employees.Count; i++)
        if (employees[i].workingToday && !employees[i].workingNow)
            employees[i].finedThisMonth++;    }
public static int reportForNextMonth()    {        return 1;    }
public static void needDismissEmployee()    {
    for (int i = 0; i < employees.Count; i++)
        if (employees[i].finedThisMonth > 1)
            employees.Remove(employees[i]);    }
public static void hirePeopleIfNeeded()    {
    int c = 0;
    for (int i = 0; i < positions.Count; i++)
        if (positions[i].namePosition == "cashier")
            c++;
    int ec = 0;
    for (int i = 0; i < employees.Count; i++)
        if (employees[i].position == "cashier")                ec++;
    for (int i = 0; i < c - ec; i++)        {
        employees.Add(new Employee(UnityEngine.Random.Range(20, 30), "cashier", employees.Count));
        dayLog.Add("Был нанят кассир");        }
    int h = 0;
    for (int i = 0; i < positions.Count; i++)
        if (positions[i].namePosition == "hallworker")                h++;
    int eh = 0;
    for (int i = 0; i < employees.Count; i++)
        if (employees[i].position == "hallworker")                eh++;
    for (int i = 0; i < h - eh; i++)        {
        employees.Add(new Employee(UnityEngine.Random.Range(20, 30), "hallworker", employees.Count));
        dayLog.Add("Был нанят рабочий зала");        }    }    }
```

Приложение Д

Код агента покупателя

```
using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;
public class guests  {
   public List<int> goodsNeeded = new List<int>();
   public int patience; // 0 - good 1 - bad
   public int money;
   public int moneyForFood;
   public string name;
   public int index;
   public string status; // hall , 1-row , 2-row
   public float timeLimit = 0.0f;
   public guests(int countGoods, int countBags, int n)    {
      for (int i = 0; i < countBags; i++)
         goodsNeeded.Add(UnityEngine.Random.Range(1, countGoods));
      money = UnityEngine.Random.Range(200, 700);
      patience = UnityEngine.Random.Range(0, 2);
      name = "Guest N-" + n.ToString();
      index = n;
      status = "hall";    }
   public void Update()    {
      timeLimit += Time.deltaTime + core.timeKoeff;
      if (core.dayOrNight == "day")       {
         if (status == "hall")
            if (timeLimit > 0.4f)              {
               for (int i = 0; i < goodsNeeded.Count; i++)                  {
                  int    mFwithPercent=core.goodsKeep[goodsNeeded[i]].cost+(int)(core.goodsKeep[goodsNeeded[i]].cost    *
0.2f);
                  if (money > mFwithPercent && core.goodsKeep[goodsNeeded[i]].countAtHall > 0)                    {
                     moneyForFood += mFwithPercent;
                     core.goodsKeep[goodsNeeded[i]].countAtHall--;
                     money -= mFwithPercent;                    }                }
               if (moneyForFood != 0)
                  status = core.ReturnNumRow(index).ToString() + "-row";
               else                {
                  /*Debug.Log(name);
                  Debug.Log("money :" +money);
                  Debug.Log("goodsNeeded.count :"+goodsNeeded.Count);
                  for (int i = 0; i < goodsNeeded.Count; i++)                     {
                     Debug.Log("i :" + i );
                     Debug.Log("name :" + core.goodsKeep[goodsNeeded[i]].name);
                     Debug.Log("cost :" + core.goodsKeep[goodsNeeded[i]].cost);
                     Debug.Log("count :"+ core.goodsKeep[goodsNeeded[i]].countAtHall);                    }*/   }
               if (status == "50000-row")                {
                  Debug.Log(name + " " + status);
                  status = "hall";
                  money += moneyForFood;
                  moneyForFood = 0;}   }   }  }
}
```

Приложение Е
Техническое задание.

1. ВВЕДЕНИЕ

1.1.Наименование программы

«Разработка алгоритмов взаимодействия программных агентов в виртуальной среде».

1.2.Краткая характеристика области применения программы

Разработка программных агентов в виртуальной среде в условиях текущего прогресса набирают обороты. Многие люди стараются найти способ оптимизировать время и качество услуг в таких сферах как менеджмент. Для того, чтобы задачи, поставленные директором, выполнялись на производстве с наиболее благоприятным исходом – люди выделили профессию, которая несет ответственность за ходом выполнения. Я выбрал профессию менеджера торгового зала.

Для специалистов подобного рода в обязанности подходит, прежде всего, руководство и контроль над группой кассиров, продавцов-консультантов, продавцов. Различный опыт предприятий торговли демонстрирует, что приемлимый состав такой группы – более 10 человек.

Используя различные способы создания торгового процесса на отличающихся организациях торговли, администратор (менеджер) может работать складированием и учетом товара, актуальной выкладкой товара в торговый зал, заниматься оформлением соответствующих документов. В обязанности товароведа обычно входит приемка товара от заказчика и его проверка по количеству и качеству, но иногда этим может заниматься менеджер торгового зала.

Товаровед также занимается переоценкой, списанием, резервированием товара, ведет классификатор товара, участвует в инвентаризации. На нем лежит обязанность за наличие ценников на товаре, реализации за пополнение ассортимента торгового зала, за соблюдение сроков, своевременный заказ товара и за многое другое. Оформление торгового зала входит в обязанности товароведа. В

магазинах оптовой торговли администратор лично работает с наиболее важными клиентами для улучшения репутации всего магазина.

В программировании агентом может быть всё, что может распознаваться как воспринимающее свое окружение с помощью датчиков и воздействующее на эту среду с помощью своего инструментария.

Поэтому актуальной можно признать цель данного выпускного проекта – повышение автоматизации качества обработки задач менеджера торгового зала, используя виртуально запрограммированный мир.

2. ОСНОВАНИЕ ДЛЯ РАЗРАБОТКИ

2.1. Основание для проведения разработки

Основанием для проведения разработки является выполнение учебного плана по специальности «Математическое обеспечение и администрирование информационных систем».

2.2. Наименование и условное обозначение темы разработки

Разработка алгоритмов программных агентов, отвечающих за выполнение задач виртуальной среде.

3. НАЗНАЧЕНИЕ РАЗРАБОТКИ

3.1. Функциональное назначение программы

Автоматизация принятия решений и приоритезации задач за счет применения интеллектуальных вычислений при обработке данных.

3.2. Эксплуатационное назначение программы

Программа может эксплуатироваться на предприятиях в сфере торговли. Программа может использоваться высшим руководством фирм в качестве инструмента поддержки принятия решения.

4. ТРЕБОВАНИЯ К ПРОГРАММЕ

4.1. Требования к функциональным характеристикам

4.1.1. Требования к составу выполняемых функций

Программа должна обеспечивать возможность выполнения следующих функций:

- анализ данных состояния торгового зала;
- состояние торгового зала зависит от программных агентов, в том числе работников торгового зала, а также покупателей;
- формирование лога, который описывает все действия произошедшие за промежуток времени с начала работы торгового зала и его окончания;
- Сохранение результатов работы программы в формат txt.

4.1.2. Требования к организации входных данных

Данные, вводимые вручную, проверяются на корректность программной частью. Входные данные программы должны быть организованы в виде вводимого в специальную форму текста, соответствующего определенному шаблону.

4.1.3. Требования к организации выходных данных

Выходные данные программы должны быть организованы в виде записей, в которой будет отображена информация с учетом выбранных параметров.

4.1.4. Требования к временным характеристикам

Требования к временным характеристикам отсутствует.

4.2. Требования к надежности

4.2.1. Требования к обеспечению надежного функционирования программы

Надежное работоспособность системы должно быть обеспечено выполнением организационно-технических мероприятий, перечень которых приведен ниже:

- организацией бесперебойного питания технических средств;

- выполнением требований ГОСТ Р 51583-2000. Защита информации. Порядок создания автоматизированных систем в защищенном исполнении.

4.2.2. Время восстановления после отказа

Время для восстановления после отказа системы, вызванного сбоем электропитания технических средств (иными внешними факторами), не фатальным сбоем (не крахом) операционной системы, не должно превышать времени, необходимого на перезагрузку операционной системы и запуск программы, при условии соблюдения условий эксплуатации технических и программных средств.

Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем (крахом) операционной системы, не должно превышать времени, требуемого на устранение неисправностей технических средств и переустановки программных средств.

4.2.3. Отказы из-за некорректных действий оператора

Работоспособность программы не должна зависеть от данных, вводимым пользователем.

4.3. Условия эксплуатации

4.3.1. Климатические условия эксплуатации

Климатические условия эксплуатации системы, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям, предъявляемым к техническим средствам в части условий их эксплуатации – ГОСТ 15150-69.

4.4. Требования к информационной и программной совместимости

4.4.1. Требования к информационным структурам и методам решения

Пользовательский интерфейс должен быть интуитивно понятным и содержать подсказки.

4.4.2. Требования к исходным кодам и языкам программирования

Требования определяются в процессе предпроектного анализа системы.

4.4.3. Требования к защите информации и программ

В Системе должен быть обеспечен надлежащий уровень защиты информации в соответствии с законом о защите персональной информации и программного комплекса в целом от несанкционированного доступа - “ Об информации, информатизации и о защите информации” РФ N 149-ФЗ от 27.07.2006.

4.5. Специальные требования

Специальные требования не предъявляются.

5. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

5.1. Предварительный состав программной документации

Состав программной документации должен включать в себя:

1. Техническое задание.
2. Текст программы.
3. Описание программы.
4. Программу.
5. Описание применения в описание программы

6. СТАДИИ И ЭТАПЫ РАЗРАБОТКИ

6.1. Стадии разработки

Разработка должна быть проведена в следующих стадиях:

- разработка технического задания;
- постановка задач;
- определение и уточнение требований к техническим средствам;
- определение требований к программе;
- определение стадий, этапов и сроков разработки ПО;

- выбор языков программирования;
- согласование и утверждение технического задания;
- разработка программы;
- разработка программной документации;
- испытания программы.
- внедрение.

6.2. Этапы разработки

На стадии разработки технического задания должен быть выполнен этап разработки, согласования и утверждения требований и архитектуры разрабатываемого ПО.

На стадии рабочего проектирования должны быть выполнены перечисленные ниже этапы работ:

- разработка программы;
- разработка программной документации;
- испытания программы.

На стадии внедрения должен быть выполнен этап разработки - подготовка и передача программы.

6.3. Содержание работ по этапам

На этапе разработки технического задания должны быть выполнены перечисленные ниже работы:

- постановка задачи;
- определение и уточнение требований к техническим средствам;
- определение требований к программе;
- определение стадий, этапов и сроков разработки ПО;
- выбор языков программирования;
- согласование и утверждение технического задания;

На этапе разработки ПО должна быть выполнена работа по программированию и отладке программы.

На этапе испытаний программы должны быть выполнены перечисленные ниже виды работ:

- разработка, согласование и утверждение программы и методики испытаний;
- проведение приемо-сдаточных испытаний;
- корректировка ПО по результатам испытаний.

6.4. Исполнители

Руководитель
Преподаватель Лиманова Н.И.
Исполнитель
Студент группы МОб-1201 Демидович А.В.