«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И.Ульянова (Ленина)»
(СПбГЭТУ «ЛЭТИ»)

Направление	01.03.02 Прикладная математика и информатика
Профиль	Математическое обеспечение программно-информационных систем
Факультет	КТИ
Кафедра	МО ЭВМ

К защите допустить

И.о. зав. кафедрой А.А. Лисс

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Тема: РАЗРАБОТКА ПРОГРАММЫ ПЛАНИРОВАНИЯ ДЕЙСТВИЙ В СТРАТЕГИЧЕСКОЙ ИГРЕ

Студент		_____ *подпись*	А.А. Аухадиев
Руководитель	к.т.н., доцент	_____ *подпись*	С.А. Беляев
Консультанты	к.э.н.	_____	О.С. Артамонова
	к.т.н.	_____ *подпись*	М.М. Заславский

Санкт-Петербург

2023

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Утверждаю
И.о. зав. кафедрой МО ЭВМ
_____ А.А. Лисс
«___»_____2023 г.

Студент Аухадиев А.А. Группа 9381
Тема работы: Разработка программы планирования действий в стратегической игре
Место выполнения ВКР: Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)
Исходные данные (технические требования): ОС Windows, Java 1.8
Содержание ВКР: Постановка задачи и обзор аналогов, Описание задачи и подходов к решению, Реализация, Обеспечение качества разработки, продукции, программного продукта
Перечень отчетных материалов: пояснительная записка, иллюстративный материал
Дополнительные разделы: обеспечение качества разработки, продукции, программного продукта.

Дата выдачи задания «04» апреля 2023 г.
Дата представления ВКР к защите «06» июня 2023 г.

Студент _____ А.А. Аухадиев
Руководитель к.т.н., доцент _____ С.А. Беляев
(Уч. степень, уч. звание)

КАЛЕНДАРНЫЙ ПЛАН ВЫПОЛНЕНИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Утверждаю
И.о. зав. кафедрой МО ЭВМ
_____ А.А. Лисс
«___»_____2023 г.

Студент Аухадиев А.А. Группа 9381

Тема работы: Разработка программы планирования действий в стратегической игре

№ п/п	Наименование работ	Срок выполнения
1	Обзор литературы по теме работы	04.04–15.04
2	Разработка математической модели и архитектуры	16.04–01.05
3	Реализация программы и тестирование	02.05–10.05
4	Обеспечение качества разработки	11.05–15.05
5	Оформление пояснительной записки	16.05–20.05
6	Оформление иллюстративного материала	21.05–31.05
7	Предзащита	01.06

Студент _____ А.А. Аухадиев

Руководитель к.т.н., доцент _____ С.А. Беляев
(Уч. степень, уч. звание)

РЕФЕРАТ

Пояснительная записка 71 стр., 18 рис., 5 табл., 31 ист.

ПЛАНИРОВАНИЕ, ИГРОВОЙ ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, СТРАТЕГИИ В РЕАЛЬНОМ ВРЕМЕНИ, STARCRAFT.

Объект исследования: искусственный интеллект в компьютерных стратегических играх в реальном времени.

Предмет исследования: задача планирования действий в компьютерных стратегических играх в реальном времени.

Цель работы: разработка программы решения задачи планирования действий в стратегической игре StarCraft с учётом текущей ситуации в игре.

В данной работе были рассмотрены существующие аналоги искусственного интеллекта в играх жанра стратегии в реальном времени, в первую очередь для StarCraft[1], основными недостатками которых стали зависимость от экспертных знаний, отсутствие полноценной реализации или отсутствие поддержки принятия решений на стратегическом уровне. На основе рассмотренных аналогов были сформулированы критерии, разработаны математическая модель и архитектура, учитывающая описанные недостатки, а также реализация в виде программы.

В качестве решения был принят подход, основанный на деревьях поведения, которые решают перечисленные выше проблемы, но обладают такими недостатками, как детерминированность при выборе действий и реактивное поведение без анализа последствий. Решением этих недостатков может стать усложнение модели поведения путём добавления планировщика, способного выбирать разные стратегии поведения в зависимости от ситуации, что требует более подробного изучения в последующих исследованиях.

В главе 1 представлен обзор литературы по теме искусственного интеллекта в играх жанра стратегии в реальном времени. Обзор разбит на две части. В первой кратко описаны основные методы разработки искусственного интеллекта в компьютерных играх, во второй – существующие программы

реализации искусственного интеллекта для игр жанра RTS (далее - Программы), преимущественно для StarCraft.

В главе 2 описаны разработанные математическая модель и архитектура, основанные на деревьях поведения. Представлены основные определения, используемые деревьями поведения, UML-диаграммы компонентов и потоков данных.

В главе 3 описаны реализация Программы, проводимый эксперимент и его результаты. Представлены UML-диаграмма классов с перечислением используемых шаблонов проектирования и UML-диаграмма последовательности для основного алгоритма Программы.

Глава 4 посвящена дополнительному разделу по обеспечению качества разработки.

В заключении приведён вывод по проделанной работе, описаны достоинства и недостатки реализованной Программы, а также идеи для её улучшения и дальнейших исследований.

ABSTRACT

In this paper, the existing analogues of artificial intelligence in real-time strategy games were considered, primarily for StarCraft, the main disadvantages of which were dependence on expert knowledge, lack of full implementation or lack of support for decision-making at the strategic level. Criteria were formulated on the basis of the considered analogues, a mathematical model and an architecture based on it were developed, taking into account the disadvantages of the considered analogues, as well as implementation in the form of a program.

As a solution, we consider an approach based on behavior trees that solve the problems listed above, but have such disadvantages as determinism in the choice of actions and reactive behavior without analyzing the consequences. The solution to these shortcomings may be to complicate the behavior model by adding a scheduler capable of choosing different behavior strategies depending on the situation, which requires more detailed study in subsequent studies.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

AAAI – Association for the Advancement of Artificial Intelligence, Ассоциация по развитию искусственного интеллекта;

ABCD – Alpha-Beta Considering Durations;

AI – Artificial Intelligence, искусственный интеллект;

AIIDE – Artificial Intelligence and Interactive Digital Environment, Конференция AAAI по искусственному интеллекту и интерактивным цифровым развлечениям;

BT – Behavior Tree, дерево поведения;

BW API – Brood War Application Programming Interface, интерфейс для программного взаимодействия с игрой StarCraft: Brood War;

CBP – Case-Based Planning, планирование на основе определённых ситуаций;

DTD - Document Type Definition, определение типа документа;

FSM – Finite-States Machine, конечный автомат;

GDA – Goal-Driven Autonomy, независимое управление целью;

GOAP – Goal-Oriented Action Planning, целеориентированное планирование действий;

GSP – Generalized Sequential Patterns, обобщённая последовательность шаблонов;

HFSM – Hierarchical Finite-States Machine, иерархический конечный автомат;

HTN – Hierarchical Task Networks, иерархическая сеть задач;

RL – Reinforcement Learning, обучение с подкреплением;

RTS – Real-Time Strategy, стратегии в реальном времени;

SSCAIT – Student StarCraft AI Tournament, студенческий турнир по искусственному интеллекту StarCraft;

STRIPS – Stanford Research Institute Problem Solver;

SVS – Spatial Vision System, система пространственного видения;

ИИ – Искусственный Интеллект.

ВВЕДЕНИЕ

Игры RTS популярны среди игроков по всему миру. Игра StarCraft является соревновательной общемировой RTS-игрой, считается сбалансированной, и недавно стала испытательным полигоном для методов искусственного интеллекта. Соревновательные матчи по StarCraft проводятся не только между игроками, но и между разными реализациями внутриигрового искусственного интеллекта.

Искусственный интеллект в играх жанра RTS в настоящее время предсказуем, и игроки находят разные способы обыгрывать даже лучшие его реализации. Большинство алгоритмов, которые работают в традиционных играх, не могут быть применены к играм RTS или просто неэффективны в вычислительном отношении. Данная область исследований интересна и с практической точки зрения, потому что в ней существуют подзадачи, которые связаны с задачами в реальной жизни. Управление производством в играх RTS напоминает проблемы промышленного производства в реальном мире. Алгоритмы, изобретенные для управления движением юнитов в играх RTS, позже могут быть распространены на управление дорожным движением в реальных городах. Гораздо проще решить эти проблемы и оценивать результаты в контролируемой среде, а не в реальной жизни.

Программа, разрабатываемая в рамках ВКР, предполагает создание игрового искусственного интеллекта на основе деревьев поведения для игры StarCraft. Помимо разработки, целью ВКР является изучение сильных и слабых сторон данного метода, формирование идей для улучшений и дальнейших исследований.

Для достижения цели были поставлены следующие задачи:

1. Формализация постановки задачи;
2. Разработка математической модели;
3. Поиск и сравнение аналогов;
4. Разработка архитектуры решения;
5. Реализация программы и проведение эксперимента.

Объектом исследования стал искусственный интеллект в компьютерных играх жанра RTS.

Предметом исследования была выбрана задача планирования действий в RTS-играх.

Программа, разработанная в рамках данной ВКР, может быть впоследствии применена при разработке искусственного интеллекта для других игр жанра RTS, а также принять участие в ежегодных соревнованиях между различными реализациями искусственного интеллекта для StarCraft[2]-[3].

В первой главе пояснительной записки представлен обзор литературы по теме искусственного интеллекта в играх жанра стратегии в реальном времени. В главе 2 описаны разработанные математическая модель и архитектура. В главе 3 описаны реализация Программы, проводимый эксперимент и его результаты. Глава 4 посвящена дополнительному разделу по обеспечению качества разработки.

1 Постановка задачи и сравнение аналогов

В данной главе приводятся формализация требований, обзор существующих подходов и аналогов, их сравнение и выявление основных недостатков. В заключении главы на основе проведённого анализа строится подход к дальнейшей реализации, удовлетворяющий поставленным требованиям и решающий недостатки рассмотренных аналогов.

1.1 Постановка задачи

Формальное описание задачи планирования[4]: дана система агент-среда $\mathrm{M} = (\mathrm{Q}, \mathrm{A}, \mathrm{F})$, где

Q – множество наблюдаемых состояний;

A – множество действий;

$F: QxA => Q$ – функция перехода: для каждого состояния $q \in Q$ и действия $a \in A$ определяет следующее состояние $q' = F(q, a)$;

$I \in Q$ – начальные условия (исходное состояние);

$G \in Q$ – целевое состояние.

Требуется найти план – упорядоченное множество действий $P = \{a_1, \dots, a_n\}$ такое, что суперпозиция функций перехода $F(F(\dots F(F(q_0, a_1), a_2) \dots, a_{n-1}), a_n) \in G$ при $q_0 \in I$.

При заданной системе агент-среда $M = (Q, A, F)$ для задачи планирования $\{I, G\}$ алгоритм построения плана должен обладать следующими свойствами[5]:

1. Корректность: для любой задачи с заданными M, I, G, если алгоритм успешно генерирует план P, то выполнение шагов из P при любом $q_0 \in I$ всегда приводит к состоянию $q_n \in G$;

2. Полнота: алгоритм находит план, если тот существует.

В компьютерных играх жанра RTS и в StarCraft, в частности, состояние среды $q \in Q$ выражается такими параметрами, как:

1. Количество юнитов и зданий определённого типа;

2. Количество ресурсов для строительства и найма;

3. Уровень модернизации юнитов;

4. Количество разведанных областей карты;

5. Уровень осведомлённости о расположении, количестве и силе противников.

Для влияния на перечисленные параметры состояния доступны действия $a \in A$:

1. Строительство зданий;

2. Наём юнитов;

3. Перемещение юнитов с целью сбора ресурсов, разведки и занятия оборонительных позиций;

4. Исследование улучшений юнитов и зданий;

5. Атака противника.

1.2 Методы разработки искусственного интеллекта

Ниже представлено описание основных методов разработки, чаще всего используемых для создания искусственного интеллекта в компьютерных играх, их сильных и слабых сторон.

1.2.1 Метод, основанный на правилах (Scripting)[6]

Метод, основанный на правилах, был и остаётся популярным методом разработки искусственного интеллекта, так как он прост в реализации. Суть метода заключается в разработке правил, согласно которым действуют игровые сущности. Часто для этих целей разрабатывается высокоуровневый скриптовый язык, что ещё сильнее упрощает разработку. Помимо простоты, заранее прописанные сценарии быстры при выполнении, так как для запуска определённого поведения требуется лишь несколько игровых кадров.

Чтобы сделать достаточно сложный ИИ, требуется предусмотреть огромное число игровых ситуаций и для каждой из них выбрать поведение игровых сущностей. Таким образом, усложнение поведения приводит к быстрому увеличению сложности программного кода, поддерживать который становится всё труднее.

1.2.2 Конечные автоматы (Finite-State Machines, FSMs)[7]

Конечные автоматы рассматриваются как один из простейших подходов к реализации поведения ИИ в играх. Причина популярности конечных автоматов заключается в простоте их использования, так как они обладают весьма понятной концепцией состояний и условных переходов между ними.

Поведение искусственного интеллекта зависит от состояния, в котором находится система, переходящая при срабатывании определённых условий из одного состояния в другое. Когда сложность ИИ возрастает, количество состояний и переходов между ними быстро увеличивается, а для добавления нового состояния потребуется просмотреть все прочие, чтобы добавить все необходимые переходы.

Расширение конечных автоматов, иерархические конечные автоматы (hierarchical finite-states machines, HFSMs), решает некоторые из этих проблем. Эта модель вводит концепцию модульности, позволяя группам состояний совместно использовать переходы. Её цель состоит в том, чтобы избежать избыточных переходов и получить лучший обзор модели. Используя HFSMs, также становится проще группировать состояния и действия для формирования моделей поведения.

Таким образом, конечные автоматы представляют простой и интуитивно понятный метод создания искусственного интеллекта, но сложны при расширении и изменении, так как добавление или модификация поведения требует пересмотра и возможной корректировки всей модели. Также трудно предусмотреть все последствия, которые эти изменения могут вызвать.

1.2.3 Деревья поведения (Behavior Trees, BTs)[8]

Деревья поведения — это относительно новый подход к проектированию ИИ. BTs сочетают в себе элементы как из методов, основанных на правилах, так и из иерархических конечных автоматов, чтобы обеспечить гибкую структуру, доступную как геймдизайнерам, так и программистам. Структура BTs также предназначена для обеспечения более масштабируемого подхода, чем HFSMs, за счет снижения структурной сложности. ИИ при-

нимает решение, основываясь на указанных условиях. У него есть возможность рассматривать сложные условия без учета комбинаций состояний.

В отличие от конечных автоматов, в деревьях поведения логика принятия решений отделена от кода самого состояния. Вся логика переходов между состояниями вынесена в отдельную структуру, поэтому при добавлении нового поведения не нужно пересматривать все переходы и состояния.

BTs предоставляют интуитивно понятную структуру, которая может быть легко применена к игровому сценарию. Структура и семантика BTs позволяют легко отслеживать поток поведения, что полезно как для понимания, так и для модификации и отладки существующего поведения. Модульность, обеспечиваемая деревьями поведения, легко масштабируется по мере увеличения вариантов поведения.

Как и в случае с HFSMs, в деревьях поведения по-прежнему необходимо поддерживать обзор структуры и идентифицировать отдельные части поведения, чтобы разложить его на древовидную структуру. По этой причине BTs может демонстрировать некоторые из тех же проблем, что и HFSMs, где поведение некоторых частей дерева может стать очевидным для игрока, тем самым разрушая иллюзию интеллекта.

1.2.4 Планировщики

В классическом планировании[9] состояния представлены наборами параметров или отношений, связанных между собой действиями, позволяющими менять значения этих параметров и, непосредственно, состояние среды. Действия не представлены в виде простых функций в среде, а вместо этого описываются схемами действий. Схемы действий определяют как действия, так и последствия этих действий. То есть, выбирая действие для плана, известен соответствующий эффект этого действия, где эффект — это результирующее новое состояние. Набор предварительных условий для действия также включен в схему действия. Действие может быть предпринято только в том случае, если эти предварительные условия выполняются.

Проблему планирования последовательности действий можно рассматривать как проблему поиска. Начиная с начального состояния, можно выполнять поиск в пространстве состояний, переходя по доступным действиям из состояния в состояние, пока не будет достигнуто целевое состояние. Из-за декларативного представления схем действий существует два возможных способа сделать это: прямой поиск из начального состояния и обратный поиск из целевого состояния.

Во время выполнения плана может быть использовано перепланирование, если в результате следования плану будут достигнуты неожиданные состояния. Это приводит к некоторым накладным расходам, поскольку необходимо генерировать новый план. В зависимости от рассматриваемой задачи планирования пространство состояний может стать настолько большим, что простой поиск становится невозможным. Для преодоления этой проблемы используется эвристическая функция.

1.2.4.1 Иерархическая сеть задач (Hierarchical Task Networks, HTN)

Подобно классическому планированию, планирование иерархической сети задач использует понятие сопряженных состояний для представления схем по описанию действий. Однако, по сравнению с классическим планированием, предполагается не планирование достижения цели, а выполнение некоторого набора задач.

Задача определяется как один или несколько составных элементов последовательных упорядоченных действий или задач. Каждая задача содержит описание различных последовательностей, на которые она может быть разложена. Планирование иерархической сети задач — это процесс рекурсивной декомпозиции этих задач до тех пор, пока не останется только последовательность действий[10].

Планировщики HTN в значительной степени используют имеющиеся знания о том, как можно декомпозировать задачи. То есть всякий раз, когда при столкновении с задачей, для которой план уже был составлен путем декомпозиции, может быть использован этот план вместо того, чтобы искать

план повторно. Это может значительно сократить вычислительное время, затрачиваемое на поиск новых планов.

Специалисты по планированию HTN также могут расширить существующие знания с помощью новых декомпозиций задач. Эту базу знаний иногда называют библиотекой планов. Мощь библиотеки планов также является одной из причин, по которой планирование HTN получило более широкое применение в практических приложениях по сравнению с другими методами планирования. Разработчики игрового ИИ, например, могут заранее определить декомпозицию задач высокого уровня, чтобы ИИ действовал так, как ожидалось, при выполнении определенных задач. С другой стороны, игровой ИИ способен самостоятельно конструировать поведение для слабо определенных задач.

1.2.4.2 Целеориентированное планирование действий (Goal-Oriented Action Planning, GOAP)

GOAP является расширением STRIPS (аббревиатура от STanford Research Institute Problem Solver)[11], которое аналогично классическому планированию.

GOAP добавляет планировщику STRIPS несколько расширений. Наиболее важными из них являются включение затрат на действия и добавление предварительных условий и эффектов к схемам действий. Стоимость действий может быть использована в качестве эвристики при поиске планов в пространстве состояний.

Добавление предварительных условий расширяет формальный взгляд на мир и позволяет осуществлять дополнительную фильтрацию по состояниям. Ещё одно расширение — это добавление эффектов к схемам действий. Декларативный характер этих эффектов налагает мгновенное изменение состояния при выполнении действия. В сценариях реального мира действия не обязательно выполняются мгновенно и вместо этого требуют некоторого времени для выполнения. В GOAP используются оба этих понятия состояний. Элемент времени реализуется путем подключения системы планирова-

ния к FSMs. При выполнении действия в GOAP устанавливается состояние ожидания FSM до тех пор, пока действие не будет завершено и среда не перейдет в новое состояние.

1.2.5 Обучение с подкреплением (Reinforcement Learning, RL)

Обучение с подкреплением можно описать как изучение того, какие действия следует предпринимать в среде, чтобы максимизировать или минимизировать некоторое кумулятивное вознаграждение[12]. Однако обучение с подкреплением не определяется конкретным методом достижения этого обучения. Вместо этого он определяется как характеризующий набор проблем. Любой метод, который помогает решить эти проблемы, называется методом обучения с подкреплением.

Проблему обучения с подкреплением можно визуализировать так, как представлено на рис. 1. Здесь изображён агент, который взаимодействует со средой посредством некоторых действий a. Каждое действие приводит к некоторому изменению в среде, представленной агенту в виде состояний s. Информация об окружающей среде, представленная в состояниях, зависит от наблюдаемости окружающей среды для агента. Требуется, чтобы эта среда была, по крайней мере, частично наблюдаемой для агента.

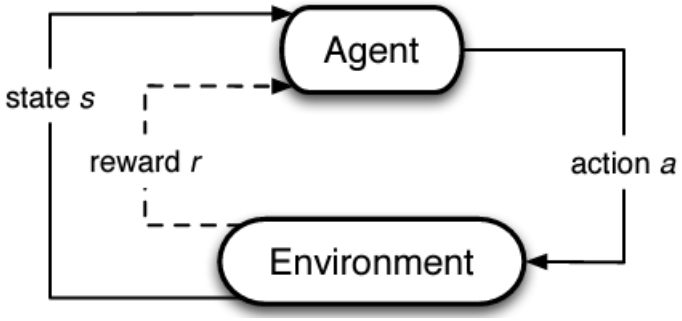

Рисунок 1 – Взаимодействие среды и агента в обучении с подкреплением

Изменения в среде могут вызвать некоторое вознаграждение r для агента, как отрицательное, так и положительное. Это вознаграждение дается функцией подкрепления, которая сопоставляет пары состояний и действий с вознаграждениями. Обучающая часть заключается в определении пар состояний и действий, которые приводят к максимальному или минимальному со-

вокупному вознаграждению, пока не будет достигнуто некоторое конечное состояние.

Вознаграждение за действия агента может быть не немедленным, а вместо этого зависеть от рассматриваемого способа обучения с подкреплением. Кроме того, цель максимизации или минимизации совокупного вознаграждения также зависит от подхода.

Поиск функции оптимального значения является ключевой проблемой обучения с подкреплением, потому что, начиная с любого состояния, будет известно не только значение этого состояния, но и значения всех достижимых состояний с помощью доступных действий. Принятие мер, приводящих к состояниям с максимумом, равносильно следованию оптимальному поведению. Таким образом, совокупное вознаграждение или подкрепление будет максимизировано, что и было целью обучения с подкреплением.

1.2.6 Заключение

В данном разделе были рассмотрены основные методы разработки искусственного интеллекта для игр.

Наиболее используемыми в индустрии остаются метод, основанный на правилах, и конечные автоматы, оба из которых сложны в поддержке и расширении поведения.

Конечные автоматы предоставляет понятную структуру для дизайнеров, которая не допускает повторного использования логики. Это приводит к созданию конструкции с увеличивающимся числом повторяющихся состояний, что делает ее излишне сложной. Проблема сложности проектирования была решена созданием иерархических конечных автоматов путем введения суперсостояний, но методу по-прежнему не хватает возможности повторного использования и повторяющихся состояний.

Деревья поведения были разработаны, чтобы компенсировать отсутствие возможности повторного использования в иерархических конечных автоматах, позволяя каждому узлу инкапсулировать свою собственную логику. Делая это, узел будет содержать некоторое поведение, основанное на нем са-

мом и его дочерних элементах. Это поведение можно рассматривать как независимое от остального контекста и может быть повторно использовано там, где это необходимо, что делает деревья поведения легко масштабируемыми по сравнению с HFSMs, и создание сложных моделей поведения будет менее затруднительным.

Все три рассмотренных метода имеют некоторые общие недостатки. Поведение, созданное на основе правил, с помощью FSMs или BTs, будет реагировать в соответствии с предопределенным поведением. Ни один из методов не предусматривает какой-либо техники для планирования наперед. Таким образом, поведение будет несколько статичным, поскольку методы не предусматривают непредсказуемых игровых событий. Если бы игрок-человек видел реализацию модели, он мог бы легко принять контрмеры, приводящие к уверенной победе.

Планирование очень важно при рассмотрении человекоподобного поведения в играх. Человеческое поведение в целом в значительной степени зависит от заблаговременного планирования не только во время игр, но и в повседневной жизни. По этой причине планирование прекрасно вписывается в концепцию стратегического мышления, необходимого в играх RTS, в которых ставятся некоторые цели и планируется способ их достижения. Для сравнения, FSMs и BTs полагаются на реактивное планирование. То есть они планируют только следующее действие, которое необходимо предпринять, исходя из текущего состояния среды.

Обучение, как и планирование, также важно при рассмотрении человекоподобного поведения. При выполнении задач человек получает опыт, будь то в результате успеха или неудачи, и этот опыт вспоминается в следующий раз, когда приходится выполнять аналогичную задачу.

Обучение с подкреплением позволяет найти оптимальную последовательность действий для достижения конечного состояния. Чтобы включить его в игру RTS, награда, получаемая за действие, должна основываться на игровом опыте, который является переменной, которую алгоритм стремится

максимизировать. Это самая большая проблема стимулирования обучения в такой игре, как StarCraft, поскольку получение вознаграждения в стратегической игре будет сильно отсрочено.

1.3 Аналоги

В отбираемые аналоги вошли существующие Программы, преимущественно для StarCraft. Многие из них используют как классические подходы, описанные выше, так и более уникальные, чаще всего совмещающие несколько подходов.

Большинство аналогов были взяты из статьи 2014 года «A review of real-time strategy game AI»[13]. В ней авторы сгруппировали используемые методы на три группы: Tactical-decision making (принятие тактических решений, микроменеджмент), Strategic-decision making (принятие стратегических решений, макроменеджмент) и Plan recognition and Learning (распознавание плана и обучение).

Перечень рассмотренных аналогов и используемых в них подходов представлен в таблице 1. Более подробные описание и сравнение приводятся далее.

1.3.1 Описание аналогов

Рассматриваемые аналоги разделены на три группы. В первую группу вошли реализации, использующие тактическое принятие решений, во вторую – стратегическое принятие решений, в третью – распознавание планов.

1.3.1.1 Тактическое принятие решений

Тактические решения принимаются при краткосрочном контроле над отдельными игровыми сущностями.

1. Sarsa[14]

Алгоритм Sarsa использует обучение с подкреплением для обучения управлению юнитами в небольших перестрелках. Авторы использовали искусственные нейронные сети, чтобы узнать ожидаемую награду за атаку или бегство с конкретным подразделением в данном состоянии, и выбрали действие с наибольшей ожидаемой наградой в игре. Система научилась побеж-

дать встроенные сценарии искусственного интеллекта StarCraft в среднем только в небольших перестрелках из трех единиц, при этом ни одна из вариаций не научилась побеждать встроенные сценарии в среднем в перестрелках из шести единиц.

Таблица 1 – Аналоги

Название	Подход
Sarsa	Обучение с подкреплением
Alpha-Beta Considering Durations	Деревья решений
Bayesian model	Байесовская вероятностная модель с использованием обучения с подкреплением
rtNEAT	Нейронные сети
Darmok	Планирование с использованием машинного обучения
Generalized Sequential Patterns	Генерация иерархической сети задач на основе игровых записей
EISBot	Планирование с использованием машинного обучения
Soar	Когнитивная архитектура
Build order optimization	Целеориентированное планирование сборки
HICOR	Дедуктивное распознавание планов в виде дерева решений

2. ABCD (Alpha-Beta Considering Durations)[15]

Алгоритм ABCD заключается в поиске по дереву решений, но с ограничением глубины поиска. В конечных узлах оценивается максимальная полезность путем вызова функции оценки, и это значение распространяется вверх по дереву, на основе чего и создаётся последовательность действий.

Основным ограничением этой системы является то, что из-за комбинаторного взрыва возможных действий и состояний по мере увеличения коли-

чества юнитов, количества возможных действий в StarCraft и ограничения по времени в 5 мс на игровой кадр поиск позволит использовать только до восьми юнитов с каждой стороны в битве двух игроков. С другой стороны, лучшие результаты могут быть достигнуты с помощью моделирования противника, поскольку поиск может включать известные действия противника вместо поиска по всем возможным действиям. Когда это было протестировано на стратегиях с идеальной моделью каждого противника, поиск смог достичь по крайней мере 95%-ного коэффициента выигрыша против каждого из сценариев в симуляции.

3. Bayesian model (Байесовская модель)[16]

Программа, основанная на Байесовской модели, учитывает цели, возможности и угрозы подразделений, чтобы решить, в каком направлении перемещать подразделения в бою. Модель рассматривает каждый из своих сенсорных входных сигналов как часть уравнения вероятности, которое может быть решено, учитывая данные (потенциально полученные с помощью RL) о распределении входных сигналов относительно направления перемещения, чтобы определить вероятность того, что единица должна двигаться в каждом возможном направлении. Можно выбрать наилучшее направление или перебрать вероятности, чтобы избежать перемещения двух единиц в одно и то же место. Эта байесовская модель работает в паре с иерархическим конечным автоматом для выбора различных наборов поведения, когда подразделения вступают в бой с вражескими силами или избегают их, или ведут разведку. Созданная Программа была очень эффективна против встроенного искусственного интеллекта StarCraft.

4. rtNEAT[17]

Нейроэволюция — это метод, который использует эволюционный алгоритм для создания или обучения искусственной нейронной сети. rtNEAT - нейроэволюционный алгоритм для разработки как топологии, так и весов соединений нейронных сетей для управления отдельными подразделениями в StarCraft. В этом алгоритме каждый юнит имеет свою собственную нейрон-

ную сеть, которая получает входные данные из источников окружающей среды (таких как близлежащие юниты или препятствия) и абстракции, определенные вручную (такие как количество, тип и “качество” близлежащих юнитов), и принимает решение атаковать, отступать или двигаться влево или вправо. Во время игры производительность юнитов оценивается с помощью разработанной вручную функции пригодности, а плохо работающие агенты юнитов заменяются комбинациями наиболее эффективных агентов.

rtNEAT тестировался в очень простых сценариях 12 против 12 юнитов, где все юниты с каждой стороны являются либо единицами рукопашного боя, либо единицами дальнего боя. В таких ситуациях он научился побеждать встроенный в StarCraft искусственный интеллект и некоторые другие Программы. Однако остается неясным, насколько хорошо он справится с большим количеством единиц или набором различных типов единиц.

1.3.1.2 Стратегическое принятие решений

Принятие стратегических решений происходит на высоком уровне с учётом долгосрочных целей. Для совершения разумных действий на стратегическом уровне многие исследователи создали системы планирования. Эти системы способны определять последовательность действий, которые необходимо предпринять в конкретной ситуации для достижения определенных целей. Это сложная проблема из-за неполноты доступной информации – «туман войны» скрывает области поля боя, которые находятся вне поля зрения дружественных подразделений, а также огромные пространства состояний и действий и множество одновременных неиерархических целей. С помощью систем планирования исследователи надеются позволить ИИ играть на человеческом уровне, одновременно сокращая затраты на разработку по сравнению с методами на основе правил.

1. Darmok[18]

Darmok использует метод Case-Based Planning (CBP, планирование на основе определённых ситуаций). Ситуации изучаются из игровых записей с комментариями человека, причём в каждом случае подробно описываются

цели, которые человек пытался достичь с помощью определенных последовательностей действий в определенном состоянии. Эти ситуации затем могут быть адаптированы и применены в игре, чтобы попытаться изменить игровое состояние. Рассуждая о дереве целей и подцелей, которые необходимо выполнить, можно выбрать ситуации и связать их вместе в план для достижения общей цели – победы в игре.

Системы CBP могут проявлять низкую реактивность на стратегическом уровне и чрезмерную реактивность на уровне действий, не реагируя на изменения ситуации на высоком уровне до тех пор, пока действие на низком уровне не завершится неудачей, или отказываясь от всего плана из-за неудачи одного действия[19].

2. Generalized Sequential Patterns[20]

Generalized Sequential Patterns (GSP) - алгоритм последовательного анализа шаблонов. GSP работает, выполняя серию сканирований последовательностей данных, каждый раз совершая поиск частых шаблонов на один элемент длиннее, чем при предыдущем сканировании. Таким образом, обучаясь на игровых записях, система извлекает шаблоны поведения в разных игровых ситуациях. Генерируемые закономерности создаются как на микро-, так и на макроуровне. Полученные шаблоны в последствии можно использовать для генерации сложных задач для модели HTN, которую никогда напрямую не использовали в области RTS AI.

3. EISBot[21]

EISBot использует модель Goal-Driven Autonomy (GDA) для одновременного анализа в нескольких масштабах. Используемый язык поведения способен формировать планы с ожиданиями относительно результата. Если возникает неожиданная ситуация или событие, система может записать это как несоответствие, сгенерировать объяснение, почему это произошло, и сформировать новую цель для пересмотра плана, что позволяет системе соответствующим образом реагировать на непредвиденные события. Изначально было невозможно изучить цели, ожидания или стратегии, поэтому эти

знания приходилось вводить и обновлять вручную, но более поздние усовершенствования позволили извлечь их из демонстрации[22].

4. Soar[23]

Soar - когнитивная архитектура, использующая систему пространственного видения Spatial Vision System (SVS) для ведения разведки и определения пути, а также рабочую память для хранения воспринимаемой и аргументированной информации о состоянии. Однако в настоящее время она ограничена частичной игрой в StarCraft, используя только базовые строения и подразделения пехоты для ведения боя и жестко заданные локации для размещения зданий.

5. Build order optimization[24]

Алгоритм оптимизации сборки использует автоматизированный подход поиска в пространстве состояний для планирования заказов на сборку в играх RTS. Алгоритм сосредоточен на одной цели: найти план строительства желаемого набора единиц и зданий за минимальное время. Домен RTS упрощается за счет абстрагирования сбора ресурсов от нормы дохода на одного работника, предполагая, что размещение зданий и перемещение юнитов занимает постоянное количество времени, и полностью игнорируя противников. Игнорирование оппонентов довольно разумно для начала игры, поскольку, как правило, взаимодействие с оппонентами невелико, и это означает, что планировщику не приходится иметь дело с неопределенностью и внешними воздействиями на состояние. Данный метод требует экспертных знаний, чтобы сформулировать параметры целевого состояния для его достижения.

Из-за вычислительных затрат на планирование позже в игре планирование было ограничено 120 секундами вперед, с перепланировкой каждые 30 секунд. Это привело к созданию более коротких или равных по длине планов для игроков-людей в начале игры и планов аналогичной длины в среднем (с большей дисперсией) позже в игре. Еще предстоит выяснить, насколько хорошо этот метод будет работать на более поздних этапах игры, поскольку

оценивались только первые 500 секунд, а во второй половине поиск занял значительно больше времени. Тем не менее, это эффективный способ получения оптимальных заказов на сборку, по крайней мере, для раннего и среднего этапов игры StarCraft.

1.3.1.3 Распознавание планов

Некоторые исследователи сосредоточились на подзадаче определения стратегии противника, которая особенно сложна в играх RTS из-за неполной информации о действиях противника, скрытой «туманом войны». Большинство методов распознавания планов используют существующую библиотеку планов для сопоставления при попытке распознавания стратегии, но некоторые методы позволяют распознавать планы без каких-либо предопределенных планов. Часто данные извлекаются из широкодоступных файлов записей опытных игроков-людей.

Одним из примеров решения задачи распознавания плана является дедуктивное распознавание плана, идентифицирующее план путем сравнения ситуации с гипотезами ожидаемого поведения для различных известных планов. Наблюдая за конкретным поведением, можно сделать вывод о предпринимаемом плане, даже если полное знание недоступно.

Система HICOR[25] выполняет распознавание плана в StarCraft путем сопоставления наблюдений за своим противником со всеми известными стратегиями, которые могли бы создать данную ситуацию. Затем она моделирует возможные планы, чтобы определить ожидаемые будущие действия своего противника, оценивая вероятность планов на основе новых наблюдений и отбрасывая планы, которые не совпадают. Используемый метод требует значительных человеческих усилий для описания всех возможных планов в структуре типа дерева решений.

1.3.2 Сравнение аналогов

Сравнение рассмотренных выше аналогов было осуществлено по следующим критериям:

1. Существует ли Программа, реализовывающая описанный метод ИИ или представлен только алгоритм.

2. Способна ли Программа провести полную игру от начала до конца, или же обеспечивает только ограниченный набор действий для определённой области игры.

3. Обучаемость: может ли рассматриваемый алгоритм обучаться на игровых записях.

4. Зависимость от экспертных знаний и многократных повторов: может ли предложенный метод создания искусственного интеллекта быть использован при отсутствии игровых записей. Например, можно ли будет применить эту технику при создании искусственного интеллекта для новой игры или не такой популярной, как StarCraft.

5. Принятие решений на стратегическом уровне: учитывает ли ИИ цели на стратегическом уровне при выборе действий или планировании.

Результат сравнения представлен в таблице 2.

Таблица 2 – Сравнение аналогов

Критерий	Аналоги				
	Sarsa	ABCD	Bayesian model	Darmok	GSP
Реализована Программа	+	+	+	+	-
Программа способна провести полную игровую сессию	-	-	-	+	-
Обучаемость	+	+	+	+	+
Отсутствие зависимости от экспертных знаний	+	+	+	-	-
Принятие решений на стратегическом уровне	-	-	-	+	+

Окончание таблицы 2

Критерий	Аналоги			
	EIS Bot	Soar	Build order optimization	HICOR
Реализована Программа	+	+	+	+
Программа способна провести полную игровую сессию	+	-	-	+
Обучаемость	+	+	+	+
Отсутствие зависимости от экспертных знаний	-	-	+	-
Принятие решений на стратегическом уровне	+	+	+	+

1.4 Заключение

Так как большинство исследователей не разрабатывало полноценную Программу, сложно судить об эффективности рассмотренных подходов.

Большое количество Программ, участвующих в соревнованиях[26] среди реализаций искусственного интеллекта для StarCraft, рассмотрено в [27]-[28]. Большинство из них использует метод, основанный на правилах, с применением машинного обучения, стремясь предусмотреть наибольшее количество игровых ситуаций. Это объясняется тем, что участники соревнований стремятся просто победить, а для этого не нужно разрабатывать новые инновационные методы. К сожалению, к Программам нет достаточно подробного описания, кроме их успехов в соревнованиях, поэтому они не рассматривались в данном обзоре.

Все рассмотренные в разделе 1.3.1 методы поддерживают машинное обучение, но многие из них зависят от наличия экспертных знаний, то есть игровых записей. Это может вызывать ряд проблем, таких как переобучение[29], невозможность внедрения ИИ при отсутствии игровых записей, например, в новых или менее популярных играх.

Таким образом, разрабатываемое решение должно удовлетворять следующим свойствам:

1. Решение должно быть представлено в виде полноценной Программы для игры жанра стратегия в реальном времени StarCraft без упрощений в виде сокращения игрового пространства, доступных для Программы и встроенного ИИ действий и прочих.

2. Программа должна воспроизводить полноценную игру без машинного обучения, чтобы разработанную систему можно было применить и для других игр на момент выпуска, а также во избежание таких проблем машинного обучения, как переобучение.

3. Программа должна принимать решения как на тактическом, так и на стратегическом уровнях.

Ниже представлен анализ методов разработки игрового ИИ из раздела 1.2 на соответствие выдвинутым критериям.

Метод, основанный на правилах, очень прост в реализации при использовании машинного обучения, но при отсутствии экспертных знаний учёт всех возможных игровых ситуаций и разработка соответствующего поведения – очень трудоёмкая задача. А с усложнением поведения также усложняется поддержка программного кода реализации.

Конечные автоматы позволяют описать поведение как на тактическом, так и на стратегическом уровне, но системы на их основе не обучаемы. Поэтому на всех этапах разработки все состояния и переходы пришлось бы описывать вручную. Также конечные автоматы становится гораздо сложнее поддерживать при добавлении или изменении поведения.

Деревья поведения обладают теми же преимуществами, что и конечные автоматы, но при этом обучаемы[30] и проще в поддержке и расширении. К тому же неправильно усвоенное поведение можно изменить, а также добавить собственное для конкретных сценариев.

Использование планировщиков требует детального представления состояния среды, что в играх жанра RTS проблематично из-за «тумана войны»

и невозможности учитывать действия соперника, пока он вне поля зрения. Среди описанных выше аналогов есть использующие планирование, но, в основном, в ограниченном масштабе (без реализации в виде Программы или только для некоторого аспекта игры, как, например, порядок сборки) и с использованием обучения.

Иерархическая сеть задач используется в методе Generalized Sequential Patterns, который на основе шаблонов способен генерировать такую сеть, но не обходится без обучения и пока не реализован в виде Программы, способной принимать на основе полученной сети задач какие-либо решения.

Похожий на целеориентированное планирование действий (GOAP) метод используется в подходе Build order optimization, который генерирует оптимальный порядок сборки построек и юнитов на ранней стадии игры. Даже будучи ограниченным лишь такими задачами стратегии в реальном времени, как постройка, наём юнитов и сбор ресурсов, системе пришлось устанавливать ограничение по времени поиска. Таким образом, метод поиска последовательности действий в пространстве состояний не подходит для создания полноценной Программы, а может использоваться только в качестве оптимизации некоторых игровых задач, так как количество возможных действий на более поздних этапах игры будет настолько большим, что система не будет успевать генерировать актуальный план.

Обучение с подкреплением используется системой, отвечающей за действия подразделений юнитов на тактическом уровне. Использование этого метода на стратегическом уровне будет проблематично из-за отсрочки получаемого вознаграждения.

Исходя из всего вышесказанного, можно сделать вывод, что наиболее подходящим решением при реализации искусственного интеллекта являются деревья поведения, которые, тем не менее, обладают большим недостатком. При выполнении определённых условий, предусмотренных разработчиком или усвоенных в процессе обучения, поведение агента всегда будет одинаковым, без учёта прочих факторов или непредвиденных событий. Поведение

агента реактивно, так как он выбирает одно действие и совершает его без анализа последствий. Тем не менее, Программа, основанная на деревьях поведения, будет отвечать всем требованиям, описанным выше.

Чтобы сделать поведение ИИ более человекоподобным, требуется использование планировщиков, но слишком большое количество доступных действий в стратегической игре не позволит генерировать актуальный план. Поэтому логичнее было бы планировать поведение не поиском среди атомарных игровых действий, а поиском среди некоторых стратегий, соединяющих эти действия в подпоследовательности. Поэтому следующим шагом после создания деревьев поведения может быть добавление планировщика, который бы сделал переходы по деревьям поведения недетерминированными.

2 Описание задачи и подходов к решению

В предыдущей главе в качестве метода реализации были выбраны деревья поведения. В данной главе описаны этот метод и разработанные на его основе математическая модель, архитектура и алгоритм.

2.1 Деревья поведения

Дерево поведения (Behavior Tree, BT) – ориентированный ациклический граф, узлами которого являются возможные варианты поведения ИИ.

Узлы BT – задачи, или поведения, имеют одно из четырёх состояний:

1. Успех – задача выполнена успешно;

2. Неудача – условие не выполнено или задача невыполнима;

3. В работе – задача запущена и ожидает завершения;

4. Ошибка – в программе возникает неизвестная ошибка.

Узлы BT можно разделить на две группы:

1. Внутренние (non-leaf nodes, рис. 2). Должны иметь любое количество дочерних узлов больше нуля (кроме корневого узла и декоратора, у которых должен быть только один дочерний узел). Внутренние узлы можно рассматривать как задачи, которые нужно выполнить. Каждая задача определяется поддеревом, состоящим из различных элементов. Существует два основных внутренних узла, которые дополняют друг друга, — это селекторы и последовательности. Также существуют декораторы, обеспечивающие большую функциональность BT, и параллельные узлы, предоставляющие возможность параллельного выполнения дочерних узлов.

2. Листовые узлы или листья (leaf nodes, рис. 3). Определяют наблюдение и взаимодействие с игровой средой. Делятся на действия, условия и ссылки. Листовые узлы часто рассматриваются как элементарные действия, которые должны быть как можно более краткими для поддержки возможности их повторного использования.

Выполнение BT происходит вглубь, начиная с корневого узла. Обычно все внутренние узлы запускают их дочерние элементы слева направо, но есть исключения. Когда узел завершает выполнение, он возвращает статус, кото-

рый может быть успешным, неудачным или исключением. Обстоятельства, при которых возвращается тот или иной статус, зависят от типа узла. Исключения возвращаются в том случае, если узел не смог вернуть ни успешного, ни неудачного статуса.

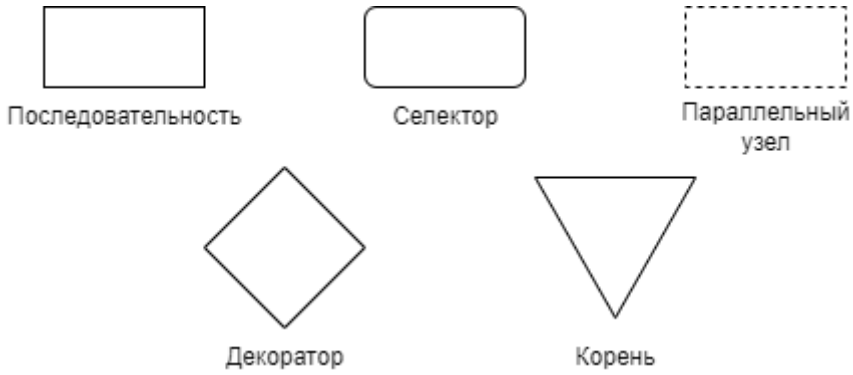

Рисунок 2 – Внутренние узлы в деревьях поведения

Внутренние узлы (рис. 2):

1. Селектор (Selector) – последовательно выполняет свои дочерние узлы слева направо, пока один из них не будет успешно завершён. Если же один из дочерних узлов вернёт неуспешный статус, будет выбран следующий дочерний элемент. Если же все дочерние элементы вернут неуспешный статус, и узел селектора вернёт неуспешный статус.

2. Вероятностный селектор (Probability selector) – узел селектора с распределением вероятности по его дочерним элементам. Вероятность указывает, насколько возможен выбор дочернего элемента. Если выбранный дочерний узел вернёт неуспешный статус, распределение вероятности будет нормализовано по оставшимся дочерним узлам, после чего будет выбран новый дочерний узел. Возвращаемое значение вероятностного селектора выбирается по тем же правилам, что и для обычного.

3. Последовательность (Sequence) – узел последовательности будет выполнять каждый из своих дочерних узлов слева направо, пока все узлы не вернут успешный статус, либо пока один из них не вернёт неуспешный статус. В первом случае узел последовательности вернёт успешный, во втором случае – неуспешный статус.

4. Декоратор (Decorator) – используется в ВТ для обеспечения большей гибкости. Декоратор позволяет добавлять новое поведения без изменения существующего кода. Чаще всего используется в качестве фильтров, чтобы действия выполнялись с некоторым условием. Например, чтобы действие выполнялось определённое количество раз, с некоторой вероятностью и т. д. Помимо фильтрации декоратор можно использовать для практически любой модификации поведения. Декоратор может иметь только один дочерний узел.

Декоратор возвращает успешный статус, если заданные условия выполнены и дочерний узел был выполнен успешно.

5. Параллельный узел (Parallel node) - параллельно выполняет все дочерние элементы. Обстоятельства, при которых узел возвращает успех или неудачу, задаются разработчиком.

Рисунок 3 – Листовые узлы (листья) в деревьях поведения

Листовые узлы (рис. 3):

1. Условие (Condition) – наблюдает за состоянием игровой среды и возвращает успех или неудачу на основе наблюдения. Это может быть сравнение значений, проверка состояния и т. д.

2. Действие (Action) – используется для взаимодействия с игровой средой посредством контролируемых действий, таких как движение сущностей и взаимодействие с объектами. Возвращает успех при удачном выполнении действия, в противном случае – неудачу.

Действие должно быть как можно более простым, то есть представлять собой отдельные действия в игровом мире.

3. Ссылка (Link) – содержит ссылку на корень другого ВТ. Выполняет узел связанного ВТ и ожидает ответа. Вводит модульность и возможность повторного использования поведений.

Для обмена информацией между узлами и деревьями используется доска объявлений (Blackboard), хранящая информацию, полученную от узлов действия или условий. Хранение такой информации позволяет получить данные в некоторой части дерева и сохранить их для дальнейшего использования.

Доска объявлений может быть общедоступной, чтобы любой узел мог считывать и записывать данные.

2.2 Математическая модель

Для решения поставленной задачи с помощью деревьев поведения была разработана следующая математическая модель:

$Q = \{q_i\}$ – множество состояний, $q =< \{p_i\}, \{v_i\} >$, где p – параметр состояния, v - его значение.

$A = \{a_i\}$ – множество действий, $a: q => q'$ – функция, для состояния q определяющая новое состояние q' путём изменения значений $\{v_i\}$ некоторых параметров из $\{p_i\}$ ($q =< \{p_i\}, \{v_i\} >, q' =< \{p_i\}, \{v'_i\} >$).

$C = \{c_i\}$ – множество условий, $c: < q, q' >=> b$ – функция проверки соответствия состояния q состоянию q' путём сравнения значений некоторых параметров этих состояний, возвращает булево значение $b \in \{True, False\}$.

Разрабатываемая математическая модель должна с помощью определённых в ней множеств действий A и условий C совершить переход из начального состояния q_0 в целевое состояние q_n. Ниже представлена модель, основанная на деревьях поведения.

M – модель дерева поведения.

$N = \{n_i\}$ – множество узлов, где $n =< r, t, \{n_i\} >$ – узел дерева поведения, где $t \in \{L, NL\}$ – тип узла, показывающий, является ли узел листом (L) или нет (NL), а множество дочерних узлов $\{n_i\}$ пусто для листовых узлов и не пусто для внутренних.

$R = \{r_i\}$ – множество исполняемых функций $r: QxN => S$, определяющих для каждого состояния из Q и узла из N статус $S \in \{s, f, w, e\}$, где s – успех, f – неудача, w – в работе, e – ошибка.

Функция r во внутренних узлах запускает дочерние узлы, а в конечных либо выполняет действие $a \in A$, совершающее переход из одного состояния в другое, либо – проверяет условие $c \in C$.

Возможная DTD-схема дерева поведения и его элементов представлена в Приложении А.

Каждое дерево поведения содержит в себе корень (root) – начальный узел, содержащий id дерева и один внутренний узел nonLeafNode, который содержит ссылку на родительский узел parentID, ссылки на один или несколько листовых или внутренних дочерних узлов childesID и статус выполнения status.

Каждый конечный узел leafNode также содержит собственный id и id родительского узла parentID, статус status и выполняемую функцию function, которая может быть трёх типов: action – изменяет значение параметров состояния values на newValues, condition – сравнивает значение параметров состояния values с expectedValues, link – содержит ссылку на корень другого дерева поведения rootAnotherTree.

Пример XML-описания с помощью разработанной DTD-схемы простого дерева поведения, включающего в себя корень, узел последовательности и два конечных узла – условие и действие (рис. 4), представлен в Приложении Б.

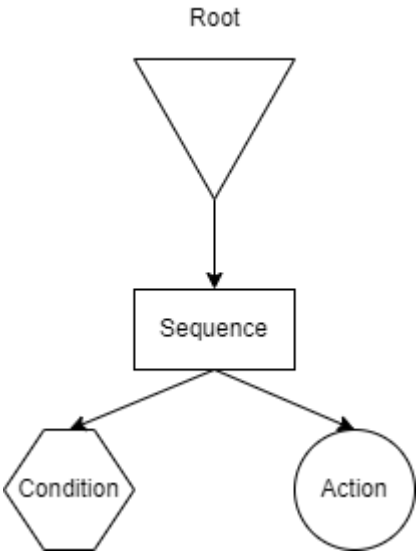

Рисунок 4 – Пример простого дерева поведения

Для упрощения данное дерево поведения взаимодействует только с двумя параметрами состояния – firstValue и secondValue.

При запуске дерева первым будет запущен узел #sequence, его статус сменится на «Run». #sequence запустит поочерёдно сначала листовой узел #condition, затем листовой узел #action.

Узел #condition проверит, равен ли параметр firstValue значению «value1». Так как условие выполняется, статус узла сменится на «success» и будет запущен следующий узел в последовательности childesID узла #sequence – узел #action.

Узел #action сменит параметры состояния firstValue и secondValue на значения newValue1 и newValue2 соответственно. Статус узла примет значение «success».

Так как все узлы из последовательности childesID узла #sequence приняли статус «success», то и сам узел примет этот статус.

Таким образом, произойдёт переход из состояния {firstValue: value1; secondValue: value2} в состояние {firstValue: newValue1; secondValue: newValue2}, и дерево поведения успешно завершит работу.

XML-описание дерева поведения в конечном состоянии представлено в Приложении В.

2.3 Архитектура

Спроектированная архитектура представлена в виде UML-диаграмм компонентов (рис. 5) и потоков данных (рис. 6).

Искусственный интеллект создаётся путём загрузки моделей поведения в память и создания на их основе деревьев поведения. Деревья поведения используют Доску объявлений для хранения и считывания информации.

Для взаимодействия деревьев поведения с игровой средой используются Утилиты, которые обрабатывают запросы от узлов деревьев поведения и выполняют действия или считывают информацию на уровне BW API.

Рисунок 5 – UML-диаграмма компонентов

В UML-диаграмме потоков данных используются понятия данных высокого и низкого уровня. Данные низкого уровня представляют собой информацию об игровых сущностях на уровне BW API, тогда как данные высокого уровня абстрактны, что позволяет разрабатывать деревья поведения, не задумываясь о детальной реализации последовательности действий в самой игре.

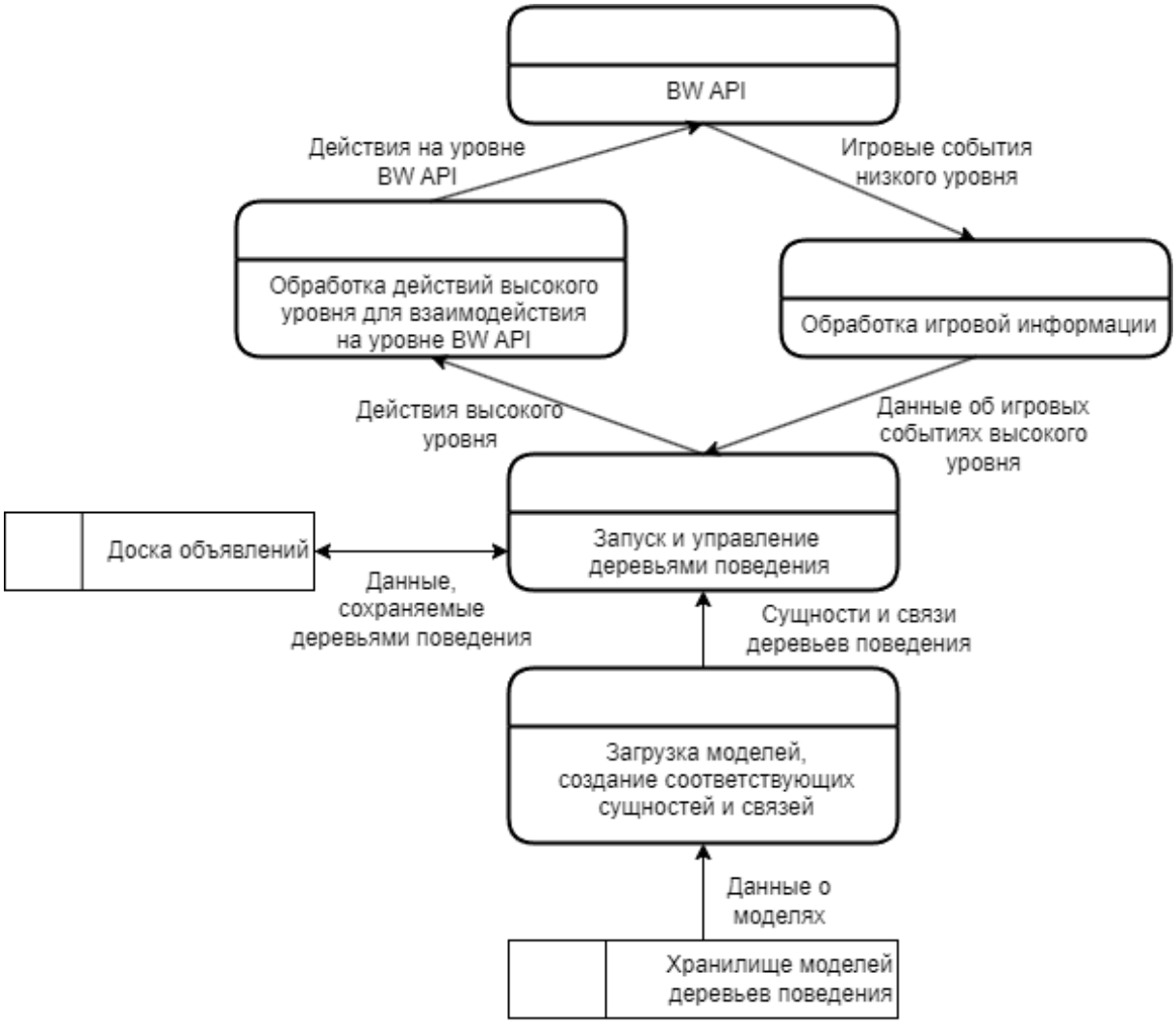

Рисунок 6 – UML-диаграмма потоков данных

Данная концепция соответствует шаблону проектирования Фасад[31], так как обработчики предоставляют упрощённый интерфейс для взаимодействия с игрой, что позволяет использовать разработанные деревья поведения для других игр жанра RTS, похожих на StarCraft. Для этого требуется только реализация подобных обработчиков для взаимодействия с API конкретной игры.

2.4 Алгоритм

Для решения задачи планирования действий с помощью деревьев поведения был разработан алгоритм, который может выбирать текущую цель в игре и наиболее подходящий способ её достижения из представленных в деревьях поведения. Для описания алгоритма используются обозначения, введённые в разделе 1.1.

Множество целей G алгоритма составляют такие задачи практически всех RTS-игр, как строительство базы, наём определённого набора боевых единиц, поддержка количества ресурсов на определённом уровне и так далее. Все цели в G упорядочены по уровню значимости, учитывая возможности игрока на разных этапах игры. Например, для найма боевых единиц прежде требуется возведение строения, способного их создавать, и сбор на эти цели определённого количества ресурсов. Для достижения каждой из целей были разработаны стратегии.

Стратегии поведения представляют собой последовательности действий (план P), которые могут также содержать в себе другие стратегии. Стратегии, направленные на достижение глобальных игровых целей (из заданных в начале игры в G) вынесены в отдельные деревья поведения, запускаемые из главного дерева игры.

Главное дерево поведения (рис. 7) использует узел последовательности, который при каждом запуске сначала выполняет последовательность стандартных действий, требующих выполнения вне зависимости от актуальной цели (например, отправки на добычу ресурсов добывающих юнитов, которые на данный момент находятся в бездействующем состоянии). Стандартное поведение вынесено в отдельное дерево, запускаемое с помощью ссылки на него.

Затем узел последовательности запускает селектор, выбирающий текущую цель и стратегию по её достижению. Каждая из стратегий представлена отдельным деревом поведения, каждое из которых может иметь свои подцели и разные способы их достижения. Все реализованные деревья, кроме основного, представлены в следующем подразделе.

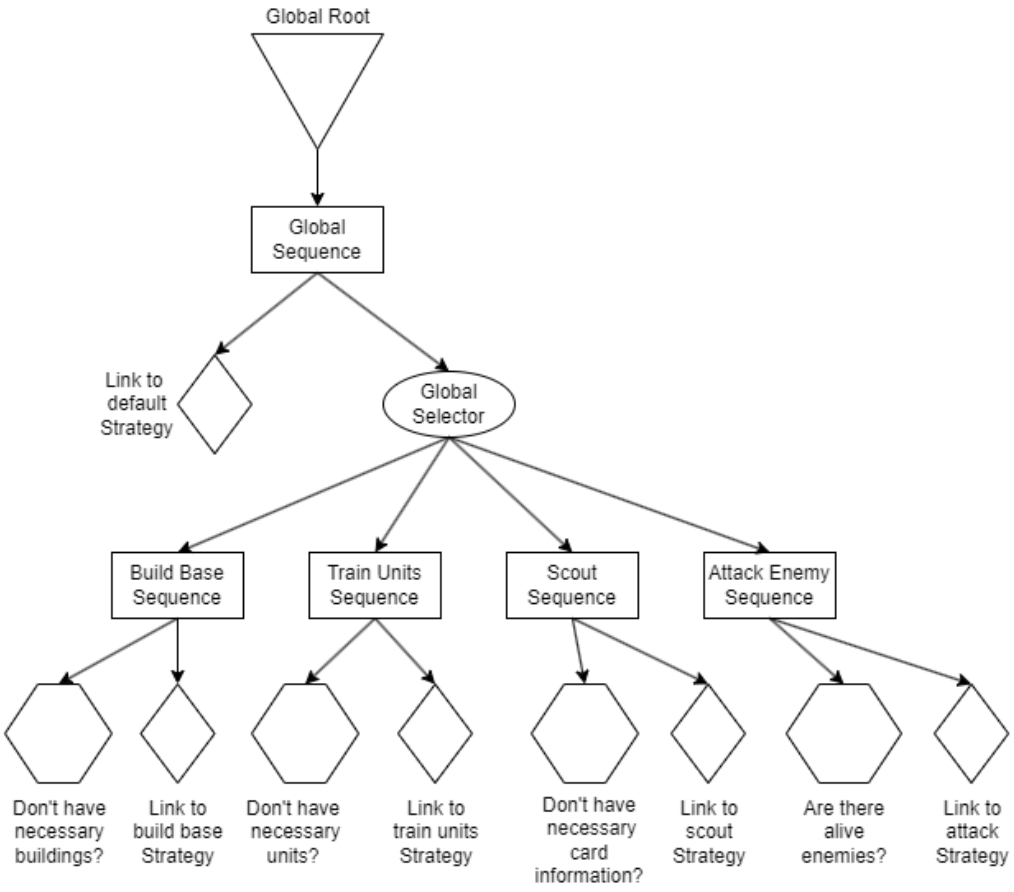

Рисунок 7 – Основное дерево поведения алгоритма

Проверка цели и запуск стратегии вынесены в отдельные узлы последовательности. Узел последовательности прекратит своё выполнение, если хотя бы один из его дочерних узлов будет выполнен неудачно, поэтому, если рассматриваемая в последовательности цель (некоторое условие, первый дочерний узел) уже достигнута, последовательность прекратит своё выполнение, а её родительский узел, селектор, перейдёт к выполнению следующей последовательности, пока хотя бы одна из них не будет успешно выполнена, либо пока они все не завершатся неудачей. После этого основное дерево игры запустится снова, выполнит стандартный набор действий и вновь запустит этот селектор.

Таким образом, все цели будут достигаться последовательно согласно плану, реализованному в одном из деревьев поведения. При выборе цели и стратегии её достижения деревья поведения изучают текущее состояние q, и, обладая планом $P = \{a_1, \dots, a_n\}$, воздействуют на состояние, приводя его в

новое состояние, пока поставленная цель из G не будет достигнута, решая тем самым задачу планирования действий.

Алгоритм удовлетворяет свойствам корректности и полноты, так как обладает планами, способными достигнуть поставленных целей.

2.5 Реализованные деревья поведения

2.5.1 Развитие базы и наём юнитов

Дерево поведения для развития базы и найма юнитов (рис. 8) определяет порядок сборки – последовательность из зданий и юнитов в порядке уменьшения приоритета.

Данное дерево является самым простым, так как задаёт только один селектор из однотипных действий.

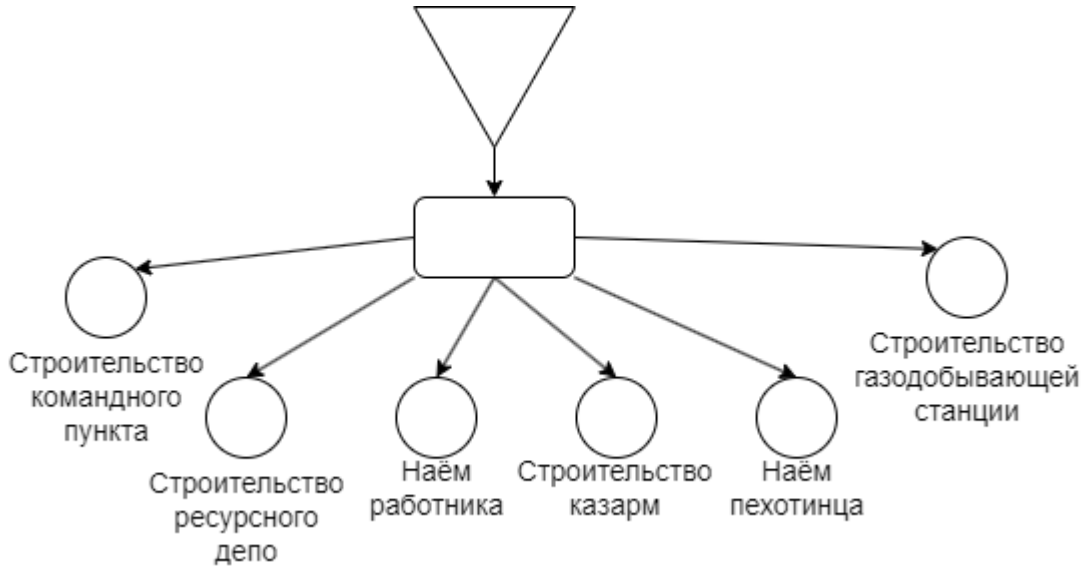

Рисунок 8 – Дерево поведения для строительства и найма

2.5.2 Стандартные действия на каждой итерации основного дерева

Дерево поведения со стандартным набором действий (рис. 9) на каждой итерации основного дерева:

1. Проверяет, угрожает ли базе опасность, и в случае необходимости запускает соответствующее поведение по её защите, сохраняя информацию об обнаруженных противниках на Доску объявлений;

2. Проверяет, требуется ли повышение лимита производимых юнитов, занося на Доску объявлений информацию о количестве необходимых ресурсных депо;

3. Проверяет, есть ли бездействующие работники, отправляя тех на добычу ресурсов, и сохраняет на Доску объявлений требуемое для данного этапа игры количество работников;

Каждое из перечисленных поведений объединено в последовательности, над которыми установлен декоратор, всегда возвращающий успешное поведение, чтобы все узлы дерева были запущены.

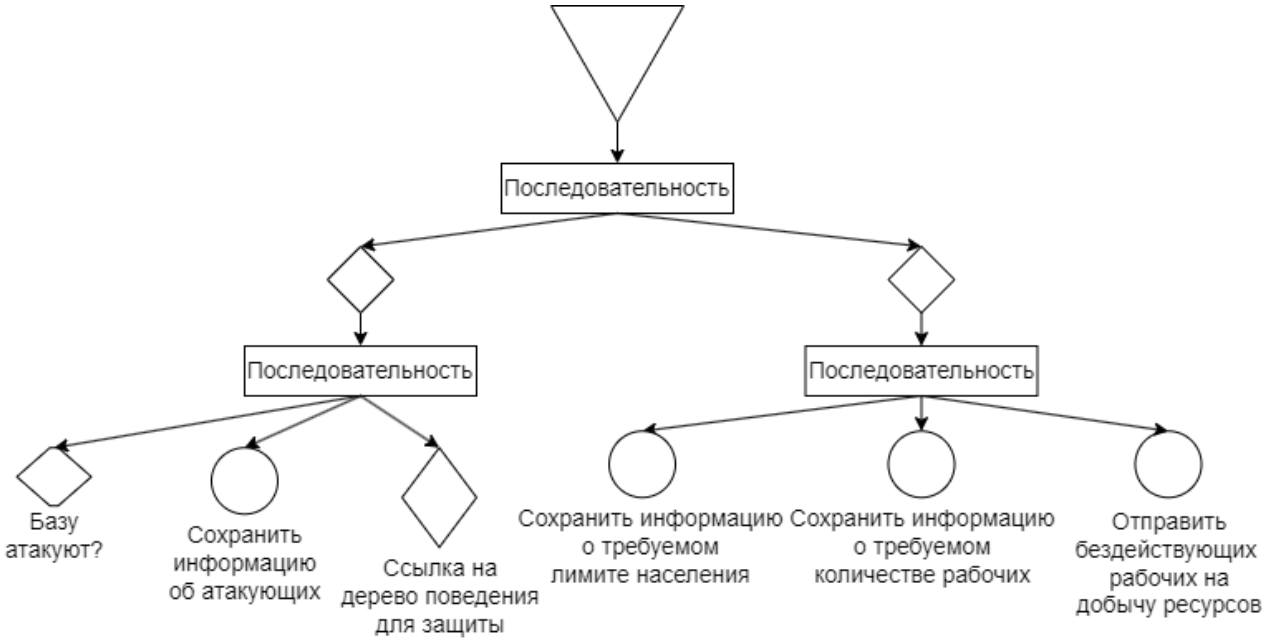

Рисунок 9 – Стандартное дерево поведения

2.5.3 Разведка

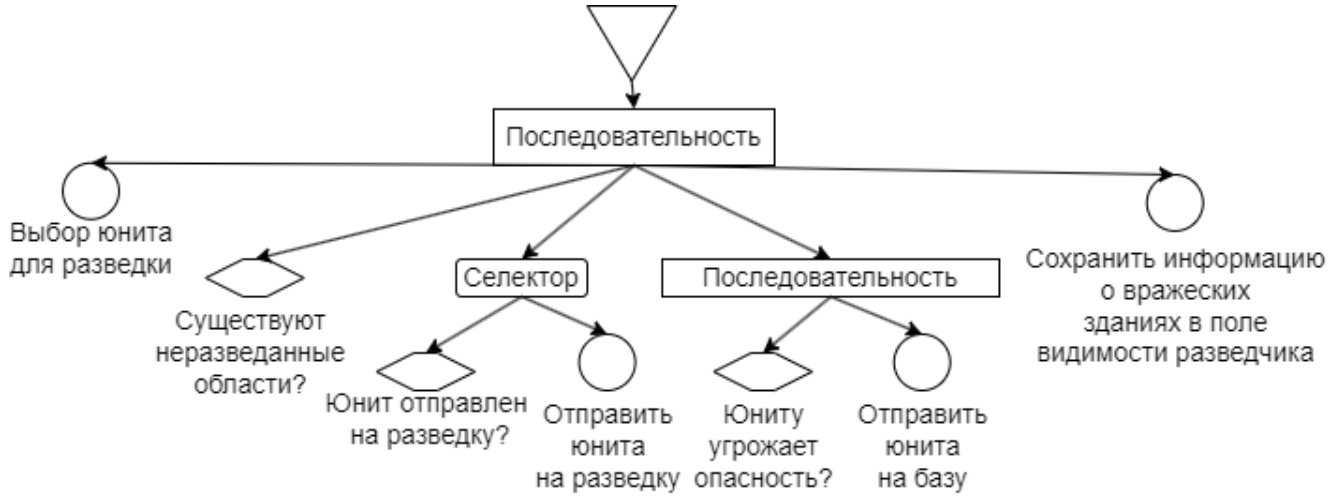

Рисунок 10 – Дерево поведения для разведки

Дерево поведения для разведки (рис. 10):

1. Выбирает юнита для разведки;

2. Проверяет, существуют ли территории, которые надо разведать;

3. Отправляет разведчика в неразведанную область;

4. В случае, если разведчику угрожает опасность, он отправляется на базу;

5. Сохраняет информацию о строениях соперника, попавших в область видимости разведчика, в виде набора координат на Доску объявлений.

2.5.4 Защита базы

Дерево поведения для защиты базы (рис. 11) выполняет всего два действия: атаку для боевых юнитов и бегство для рабочих. Информация об атакующих юнитах хранится на Доске объявлений.

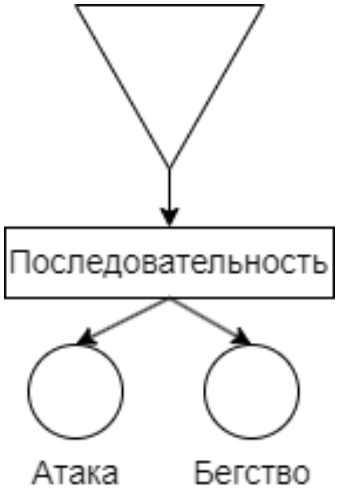

Рисунок 11 – Дерево поведения для защиты базы

2.5.5 Атака соперника

Дерево атаки соперника (рис. 12):

1. Проверяет, соблюдены ли условия для атаки: обнаружена ли вражеская база и есть ли достаточно атакующих юнитов;

2. Выбирает юнитов для атаки, сохраняя информацию о них на Доску объявлений;

3. Запускает селектор, который либо атакует вражеских юнитов, встречающихся на пути у дружественных юнитов, либо атакует вражеские здания.

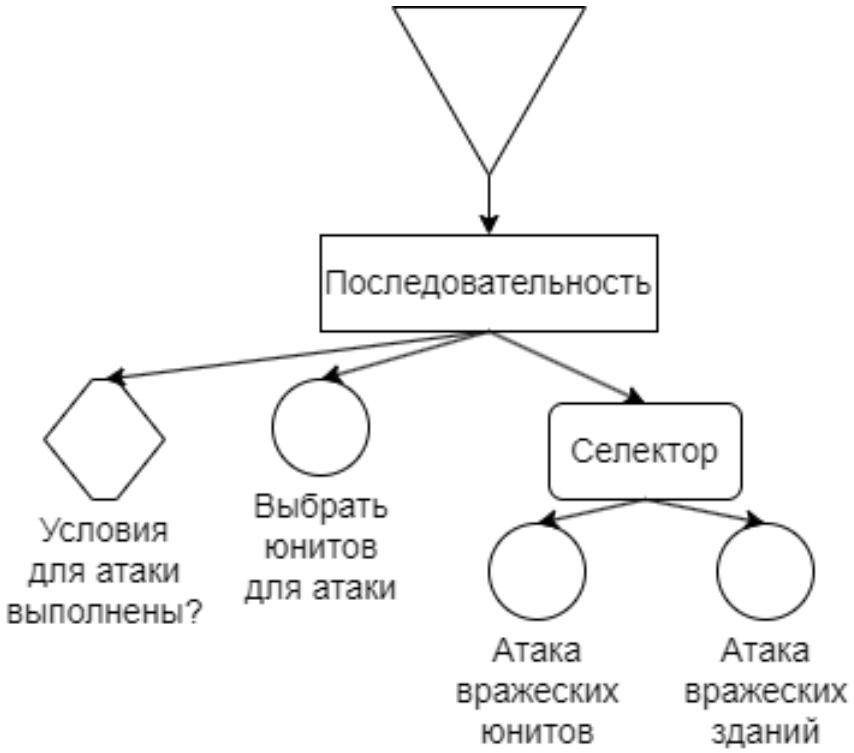

Рисунок 12 – Дерево поведения для атаки соперника

2.6 Заключение

В данной главе подробно описаны деревья поведения, разработанные математическая модель, архитектура и алгоритм решения задачи планирования действий на основе деревьев поведения.

Деревья поведения обладают несколькими типами узлов, отличающихся алгоритмом выполнения и условиями, при которых узел считается выполненным успешно или неуспешно. Выполнение BT происходит вглубь, начиная с корневого узла. Обычно все внутренние узлы запускают их дочерние элементы слева направо.

Математическая модель, разработанная на основе деревьев поведения, способна хранить текущее состояние, анализировать его и изменять. В данной главе приведён подробный пример перехода модели из одного состояния в другое.

Разработанная архитектура предполагает загрузку деревьев поведения в память, их исполнение с возможностью сохранения информации в отдельном компоненте – Доске объявлений. Согласно данной архитектуре, взаимодействие с игровой средой происходит с использованием шаблона проекти-

рования Фасад, что позволяет разрабатывать новые деревья поведения на абстрактном уровне.

Разработанный алгоритм позволяет решить задачу планирования действий с помощью деревьев поведения. Алгоритм предполагает выполнение основного дерева поведения, выбирающего цель и способ её достижения.

3 Реализация

В данной главе представлена реализация описанных в предыдущей главе архитектуры и алгоритма. Приведены UML-диаграммы классов и последовательности, перечислены используемые шаблоны проектирования. Также глава включает описание тестирования Программы, условий проводимого эксперимента и его результатов.

3.1 Диаграмма классов

В UML-диаграмме классов (рис. 14) отображены основные классы Программы, позволяющие реализовать архитектуру, описанную в разделе 2.3.

3.1.1 Дерево поведения

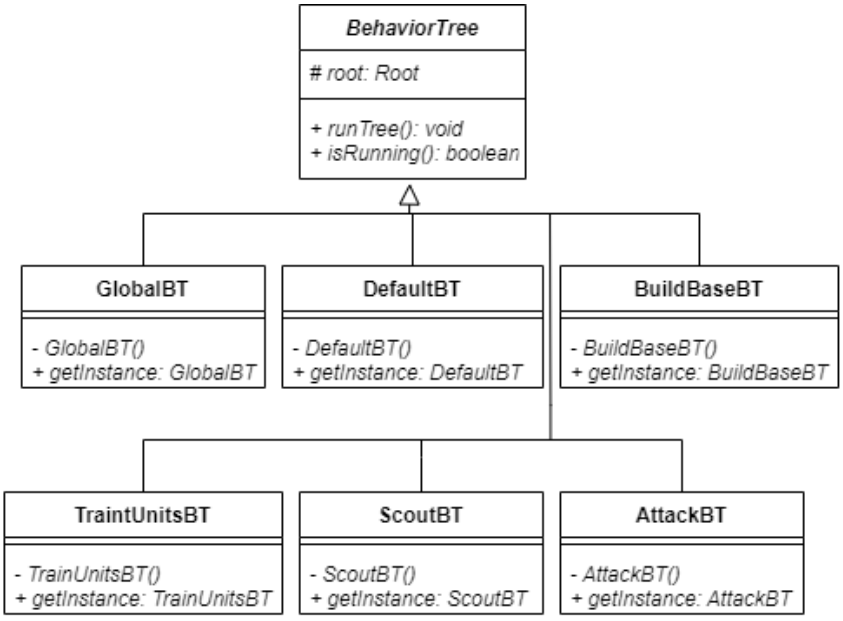

Рисунок 13 – UML-диаграмма Деревьев поведения

Каждое дерево поведения содержит корень – исполняемый узел, с запуска которого начинается выполнение стратегии данного дерева.

На рис. 13 изображены деревья поведения, описание которых приводилось в разделе 2.4: основное дерево, запускающее все остальные, дерево для выполнения стандартных действий на каждой итерации и четыре дерева для достижения основных целей игры.

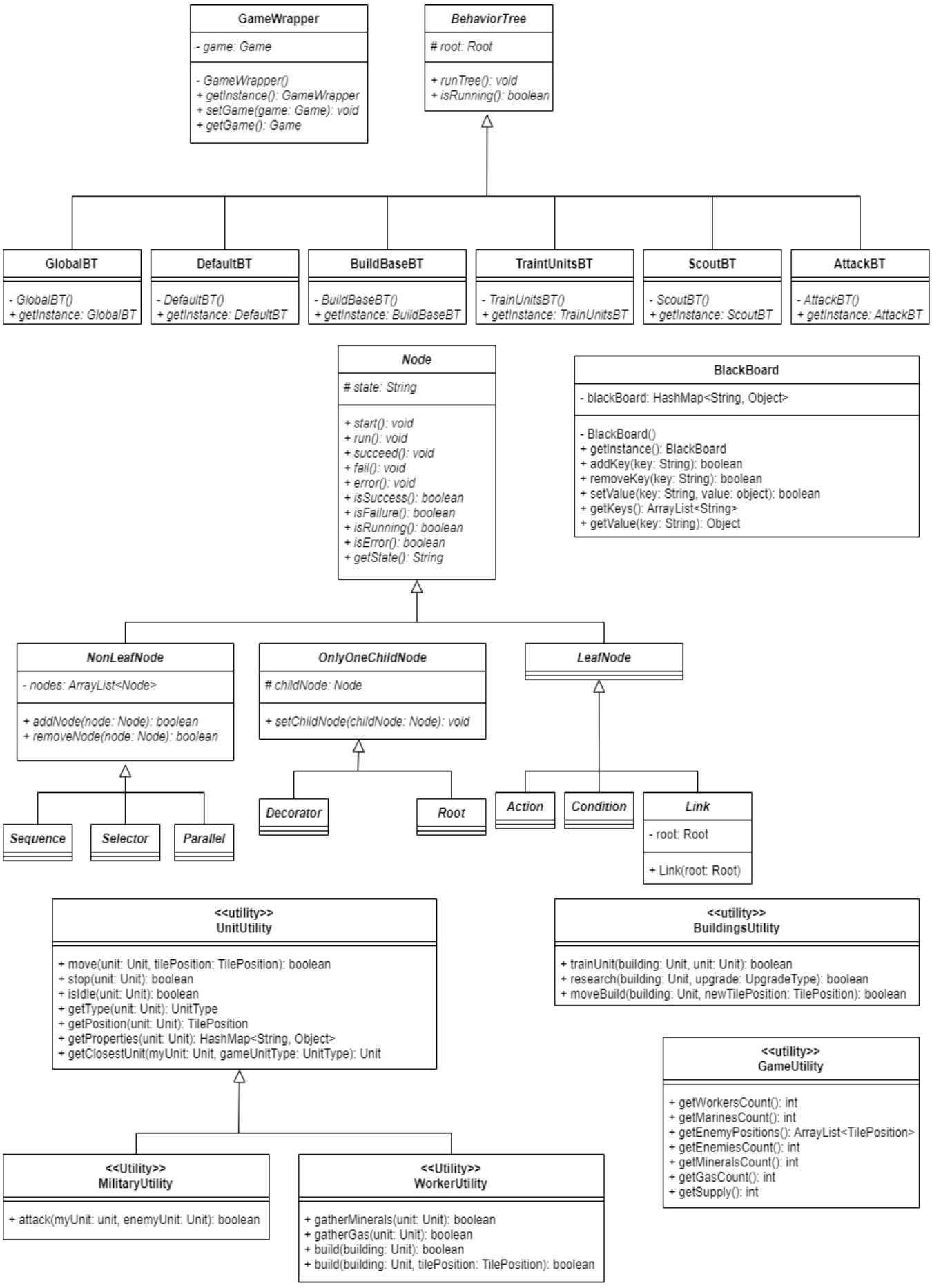

Рисунок 14 – UML-диаграмма классов

Каждое из деревьев использует шаблон проектирования Одиночка[31], позволяющий создать экземпляр класса только один раз и обратиться к нему из любой части программы. Шаблон требуется в первую очередь для того, чтобы обезопасить программу от повторного создания деревьев поведения, требующих внедрения большого количества узлов. Также это позволит сохранять состояние каждого узла на всём протяжении работы программы.

3.1.2 Узлы дерева поведения

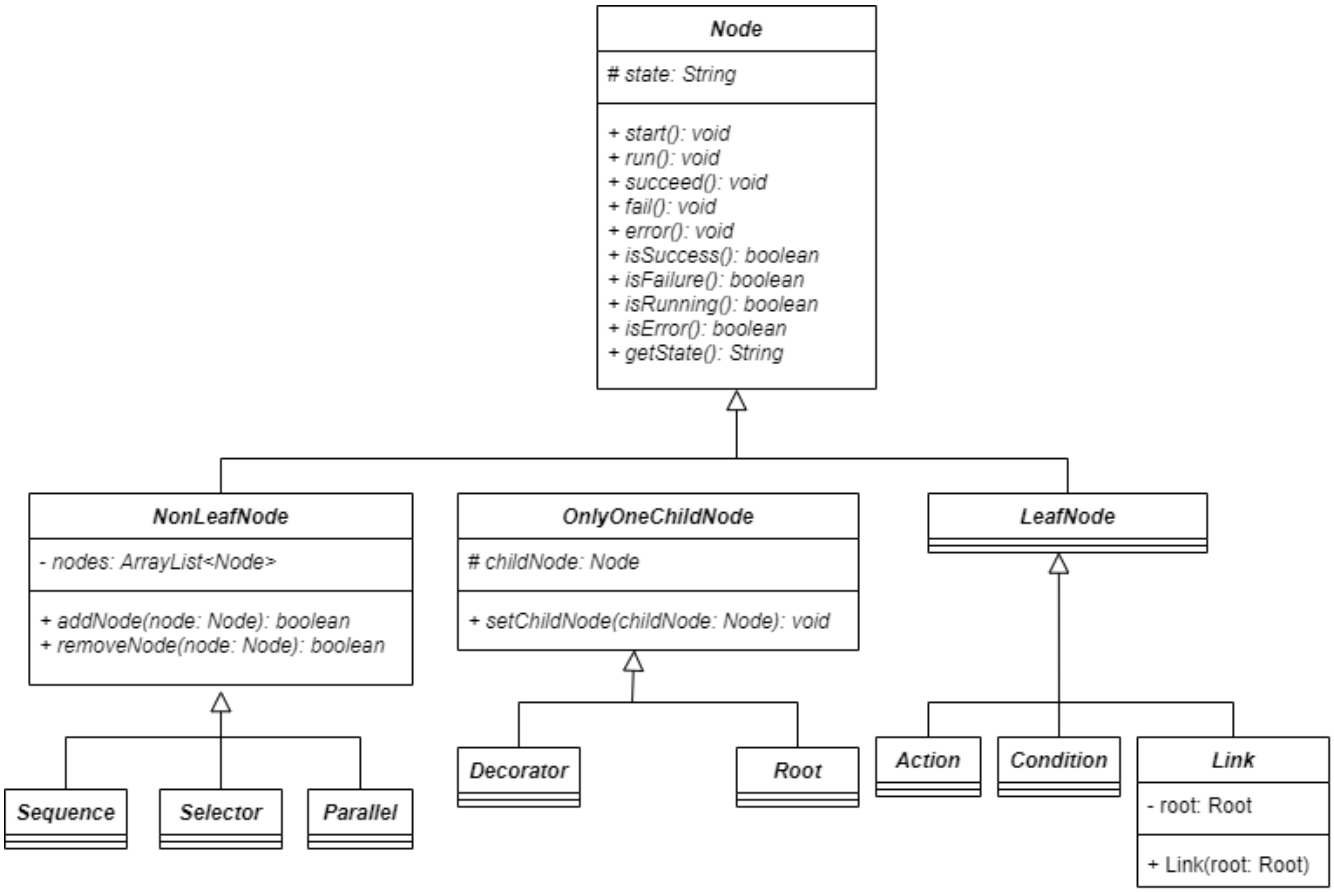

Рисунок 15 – UML-диаграмма узлов дерева поведения

Абстрактный класс Node (рис. 15) содержит текущее состояние узла, методы для его запуска, установки и проверки состояния.

Все классы, наследуемые от класса Node, требуют переопределения метода start(), что даёт возможность каждому из узлов установить уникальное поведение согласно принципам, описанным в разделе 2.1.

В классе NonLeafNode добавляется поле с массивом дочерних узлов и методы для их добавления и удаления. В каждом из наследуемых классов поразному определяется порядок запуска дочерних узлов в методе start(). Так, в

абстрактном классе Sequence все дочерние узлы запускаются до тех пор, пока либо все они не будут успешно выполнены, либо пока хотя бы один из них не завершит выполнение неудачей. Имплементация абстрактных классов Sequence, Selector и Parallel требует только инициализации дочерних узлов в необходимом порядке.

Класс OnlyOneChildNode аналогичен классу NonLeafNode, но включает в себя только один дочерний элемент. Класс Root предполагает только запуск дочернего узла, а класс Decorator – запуск дочернего узла с некоторым условием. Например, в имплементации класса Decorator можно задать логику запуска дочернего узла определённое количество раз.

Абстрактные классы, наследуемые от LeafNode отличаются реализацией метода start(). В имплементирующих классах, наследуемых от Condition, происходит проверка некоторого условия, от Action – выполнение определённого действия, от Link – запуск определённого дерева поведения. В дочерних классах Action и Condition взаимодействие с игровой средой происходит исключительно с помощью методов статических классов, отмеченных на рис. 14 как «Utility».

Описанный набор классов соответствует шаблону проектирования Компоновщик[31], так как позволяет одинаково обращаться к любому потомку класса Node, что непосредственно используется в реализации метода start() классов NonLeafNode и OnlyOneChildNode. В этих классах вызов дочерних узлов производится вне зависимости от того, к какому потомку класса Node эти узлы принадлежат. Метод start() вызывается как у листьев, так и у комбинаций разных узлов. Обращение со статусом узла (его установка и проверка) происходит также независимо от конкретного класса узла.

3.1.3 Статические классы

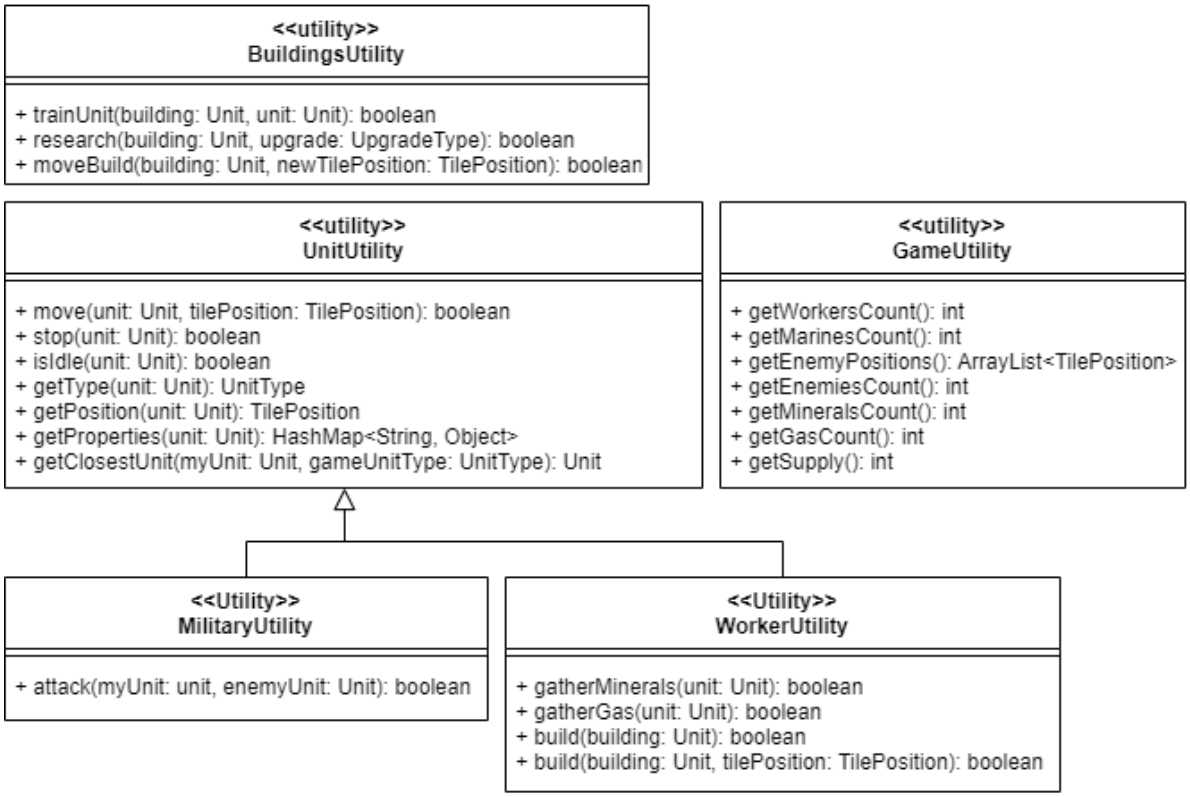

Рисунок 16 – UML-диаграмма статических классов

В Программе используется несколько статических классов (рис. 16), работающих непосредственно с функциями BW API и выполняющих простейшие игровые действия, как, например, наём юнита, его движение или строительство здания, предоставляя тем самым упрощённый интерфейс для взаимодействия с игровой средой.

Класс UnitUtility отвечает за управление движущимися игровыми сущностями – юнитами. Классы MilitaryUtility и WorkerUtility добавляют возможности для атаки у военных юнитов и для строительства, сбора ресурсов у добывающих юнитов соответственно. Класс BuildingUtility отвечает за управление зданиями, в которых происходит наём и улучшение юнитов. GameUtility предоставляет возможность получения информации об игроке и игре в целом.

Описанные статические классы удовлетворяют шаблону проектирования Фасад[31], так как упрощают интерфейсы BW API, образуя унифицированную подсистему. Паттерн Фасад предотвращает появление сильных свя-

зей между классами, отвечающими за игровую логику, и классами из BW API.

3.1.4 Доска объявлений

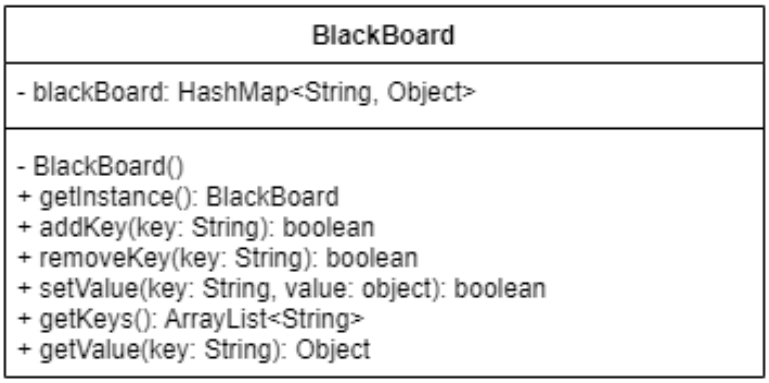

Рисунок 17 – Доска объявлений

Доска объявлений (рис. 17) предназначена для хранения и считывания информации узлами деревьев поведения. Доступ к доске объявлений осуществляется из любой части Программы, экземпляр данного класса должен быть только один, поэтому в классе BlackBoard используется шаблон проектирования Одиночка.

Главным полем класса BlackBoard является ассоциативный массив, содержащий некоторые абстрактные данные, доступ к которым осуществляется по ключу – строке. Класс предоставляет возможность добавления, удаления и получения ключей и соответствующих данных.

3.2 Диаграмма последовательности

На рис. 18 изображена диаграмма последовательности основного алгоритма Программы.

Класс Bot запускает игру, создавая тем самым объект класса Game, через который и происходит непосредственное взаимодействие с игрой. Сразу после начала игры запускаются деревья поведения, которые будут возобновлять выполнение, пока игра не закончится. Также Bot обрабатывает асинхронные события, возникающие в игре. К этим событиям, например, относятся смена игрового кадра, обрабатываемая методом onFrame(), и создание новой игровой сущности, обрабатываемое методом onUnitComplete(). Также данный класс сохраняет необходимую информацию, попавшую в обработчи-

ки, в BlackBoard. С завершением игры завершается выполнение класса Bot и всей Программы.

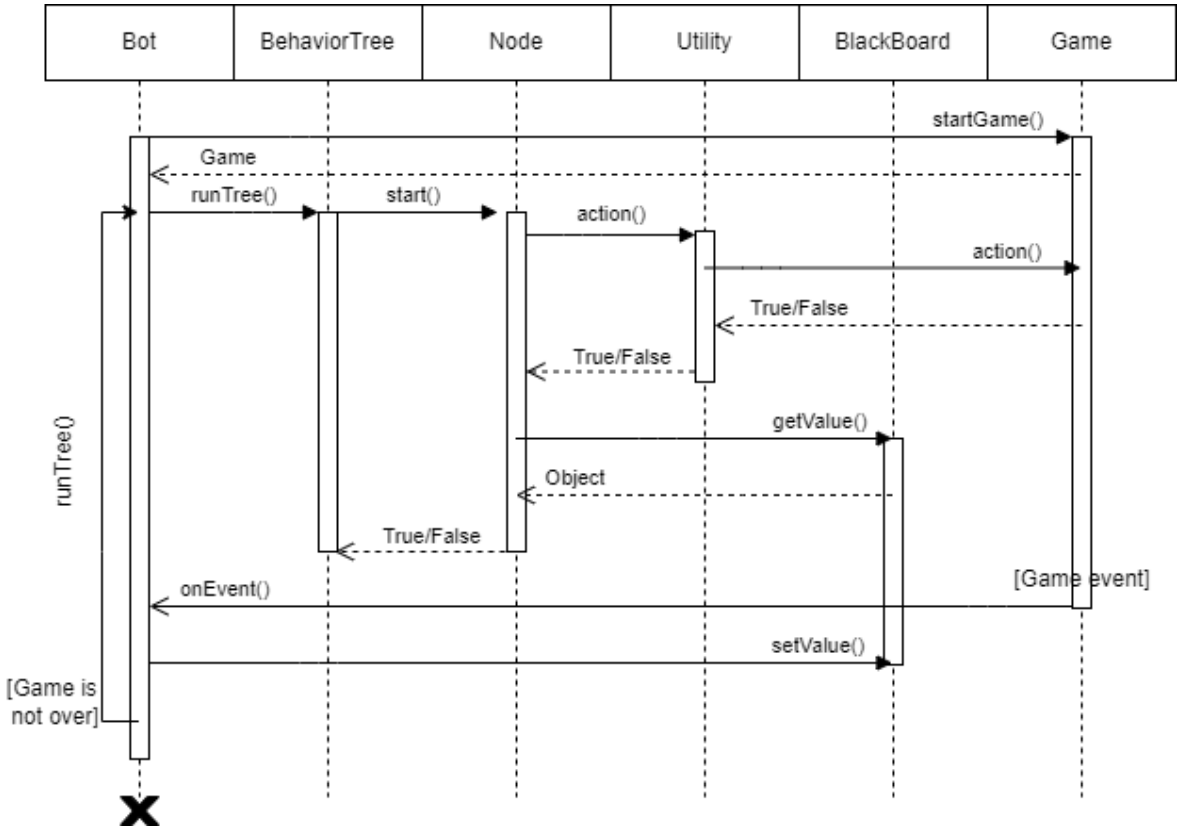

Рисунок 18 – Диаграмма последовательности

Объекты класса BehaviorTree запускают выполнение узлов – объектов класса Node, в которых реализована основная игровая логика. Узлы через классы-утилиты взаимодействуют с игрой, получая информацию об успешности/неуспешности выполненного действия. Также узлы считывают и сохраняют информацию в BlackBoard.

3.3 Эксперимент

3.3.1 Условия эксперимента

Программа разработана на языке программирования Java версии 1.8. Данная версия требуется для совместимости с BW API версии 4.4.0. Данный интерфейс используется для запуска и взаимодействия Программы с игрой StarCraft: Brood War версии 1.16.1.

В Программе реализована стратегия только для одной игровой расы. При запуске используется скорость в десять игровых кадров в секунду, что

достаточно быстро, но в то же время позволяет визуально отслеживать все игровые события.

Для проверки корректности работы Программы были выдвинуты следующие критерии оценки:

1. Программа не должна экстренно завершать своё выполнение при любых игровых событиях, не должно возникать ошибок, для которых в коде Программы не предусмотрены обработчики;

2. Поведение игровых сущностей должно быть обусловлено логикой, заложенной в Программе, полностью ей удовлетворять;

3. Программа должна решать задачу планирования действий в стратегической игре, выбирая актуальную цель и выстраивая способ её достижения.

Основным критерием из изложенных выше является третий, для проверки которого было разработано несколько сценариев поведения, которым в процессе проведения эксперимента должна удовлетворять Программа. Оценка соответствия работы Программы данным сценариям производилась на основе визуального наблюдения за поведением игровых сущностей в процессе игры, а также на основе логов, оставляемых Программой. Изначально для наглядной демонстрации работы Программы предполагалось создание пользовательского интерфейса с отображением деревьев поведения и запускаемых узлов, но их выполнение происходит на слишком высокой скорости, из-за чего невозможно визуально понять, какой стратегии на данный момент придерживается Программа. В этом случае текстовые записи предоставляют более наглядную информацию.

Разработанные сценарии и анализ соответствия им работы Программы представлены в следующем разделе.

3.3.2 Результаты эксперимента

1. Поведение по умолчанию: выполняется при каждом повторном запуске главного дерева поведения Программы.

Ожидаемое поведение:

1) Все добывающие юниты, которые на момент запуска поведения находятся в бездействующем состоянии, должны быть отправлены на добычу ресурсов;

2) В случае наличия угрозы для главной базы должны быть предприняты действия по устранению угрозы с помощью боевых юнитов и спасению небоевых юнитов путём избегания врагов;

3) Так как в игре присутствует ограничение на наём движущихся юнитов в виде лимита, его нужно увеличивать с помощью строительства зданий определённого типа. На каждой итерации работы основного дерева поведения должна происходить проверка на достижение лимита и его увеличение в случае необходимости.

Наблюдаемое поведение: первый и третий пункты рассматриваемого сценария выполнялись успешно, но в ситуации, описанной во втором пункте, при попытке спасти небоевых юнитов Программа теряла контроль над боевыми юнитами. Из-за этого они не могли в полной мере отбить атаку.

Выводы: в случае атаки на главную базу поведение боевых и небоевых юнитов должно выполняться параллельно и независимо друг от друга.

2. Развитие базы и наём юнитов: строительство необходимых зданий и наём юнитов.

Ожидаемое поведение: при наличии в множестве игровых целей требования по строительству определённого набора зданий должно запускаться поведение по их строительству по достижении требуемого для этого количества ресурсов. То же относится и к требуемым юнитам, наём которых должен происходить при наличии нужных для этого зданий и количества ресурсов.

Наблюдаемое поведение: Программа успешно справляется с данным сценарием, возводя необходимые сооружения при первой возможности.

3. Разведка: поиск вражеской базы с сохранением в Доске объявлений информации о зданиях соперника.

Ожидаемое поведение: юнит, выбранный для разведки, должен достигнуть вражеской базы, избегая при этом вражеских боевых юнитов. По достижении этой цели должно происходить сохранение информации о сопернике в Доске объявлений.

Наблюдаемое поведение: юнит-разведчик достигает вражеской базы, сохраняя информацию о ней в виде массива уникальных идентификаторов (присваиваемых каждому игровому юниту) обнаруженных зданий. Отклонением от ожидаемого поведения является немедленное отступление на главную базу при появлении в зоне видимости разведчика хотя бы одного вражеского боевого юнита, даже если этот разведчик сильнее.

Выводы: требуется добавление в дерево поведения набора узлов для анализа угрозы для дружественного подразделения.

4. Атака: нападение большинством боевых юнитов на вражескую базу с целью её ослабления или полного уничтожения.

Ожидаемое поведение: Программа должна выбрать определённое количество наиболее боеспособных юнитов и направить их на вражескую базу, предварительно до этого разведанную. По пути подразделения должны уничтожать всех встречающихся врагов, либо отступить в случае сильного снижения боеспособности (больших потерь).

Наблюдаемое поведение: Программа успешно комбинирует действия движения и атаки, вовремя реагирует на появляющуюся опасность, однако поведение боевых подразделений на тактическом уровне остаётся примитивным. Добавление более сложного поведения требует создания узлов, отвечающих за передвижение юнитов в бою, например, согласно тактике «бей и беги», чтобы юниты могли неожиданно напасть, нанести некоторый урон сопернику и отступить с минимальными потерями. Также данные узлы должны выполняться параллельно остальной логике Программы, чтобы главное дерево поведения не останавливало своё выполнение на протяжении всего боя.

Выводы: требуется создание более сложной логики тактического уровня, добавление параллелизма д ля действий тактического и стратегического уровней.

3.3.3 Заключение

Для проведения эксперимента были выдвинуты требования к разработанной Программе и созданы сценарии поведения для проверки соответствия им алгоритма Программы.

Во время проведения эксперимента Программа ни разу экстренно не завершила своё выполнение.

Во всех сценариях поведение Программы совпадало с ожидаемым, но в некоторых из них были обнаружены проблемы, требующие усложнения разработанного поведения. Так, в большинстве случаев Программе не хватало действий, которые она могла бы выполнять параллельно, что чаще всего отражалось на недостаточно проработанном поведении на тактическом уровне.

Таким образом, для решения всех обнаруженных проблем требуется усложнение разработанных деревьев поведения путём добавления в них новых узлов. При этом не требуется никаких изменений ни в алгоритме, ни в архитектуре, ни в уже реализованном поведении, что подтверждает тезис о простоте расширения деревьев поведения, выдвинутого в разделе 1.2.3. Также не требуется пересмотра игровой логики, реализованной в статических классах, описанных в разделе 3.1.3, что подтверждает отсутствие сильной связи между деревьями поведения и самой игрой.

3.4 Заключение

В данной главе были описаны результаты разработки Программы, реализующей архитектуру и алгоритм из предыдущей главы, а именно UML-диаграммы классов и последовательности, условия и результаты проводимого эксперимента.

В первом разделе главы представлены основные классы программы, их назначение и используемые шаблоны проектирования:

1. Классы BehaviorTree и BlackBoard используют шаблон Одиночка, чтобы можно было создать только один экземпляр данных классов и обратиться к нему из любой части программы;

2. Класс Node и наследуемые от него классы соответствуют шаблону Компоновщик, так как позволяют одинаково обращаться к экземплярам любого из этих классов, как к отдельным узлам, так и к их комбинациям;

3. Статические классы соответствуют шаблону Фасад, так как предоставляют упрощённый интерфейс взаимодействия с BW API, предотвращая появление сильных связей между узлами деревьев поведения и непосредственными игровыми действиями.

В представленной во втором разделе диаграмме последовательности описано выполнение основного алгоритма Программы. В диаграмме предусмотрены:

1. Запуск Программы и игровой среды;

2. Запуск деревьев поведения и их узлов для взаимодействия с игровой средой через статические классы с получением возвращаемого значения в виде True/False для определения успешности выполненного действия;

3. Взаимодействие с Доской объявлений узлов деревьев поведения для сохранения и считывания данных в виде некоторых объектов;

4. Обработка асинхронных игровых событий, возникающих в процессе игры и генерируемых BW API;

5. Циклический запуск деревьев поведения при условии, что игра ещё не окончена.

Для проведения эксперимента были выдвинуты основные критерии, которым Программа должна соответствовать. В процессе проведения эксперимента выяснилось, что Программа удовлетворяет большинству критериев, но исполняемое ею поведение отличается от ожидаемого. Был сделан вывод, что для решения данной проблемы требуется усложнение поведения Программы путём добавления новых узлов в деревья поведения.

Таким образом, разработанная Программа способна воспроизводить полноценную игру без упрощений, не требует машинного обучения, благодаря чему реализуемый ею алгоритм может применяться и для других компьютерных игр жанра RTS сразу же на момент их выпуска.

4 Обеспечение качества разработки

В данной главе описаны требования к разработанной Программе с точки зрения будущих потребителей, предложены методы выявления требований, сформулированы сами требования, для которых разработаны опциональные определения.

4.1 Определение потребителей

Потребителями разработанной Программы могут быть две группы лиц: разработчики компьютерных игр жанра RTS, рассматривающие разработку с коммерческой точки зрения, и исследователи в области искусственного интеллекта, которые могли бы использовать Программу в качестве основы для будущих исследований.

4.2 Выбор метода определения требований

Для определения требований обеих групп потребителей нужно выявить актуальные на данный момент потребности каждой из них, а именно:

1. Узнать, с какими трудностями сталкиваются разработчики искусственного интеллекта для современных RTS-игр;

2. Проанализировать, чего с точки зрения игрока не хватает в современных реализациях искусственного интеллекта;

3. Оценить, чему исследователи в области ИИ уделяют больше всего внимания.

Для этого рассматривались следующие методы:

1. Анкетирование – проведение опроса среди потребителей для подтверждения или детализации уже известных требований. Позволяет быстро получить нужную информацию, но в отсутствие каких-то начальных требований, от которых можно было бы оттолкнуться, сложно составить список вопросов для анкетирования.

2. Интервью – беседа с конкретным потребителем, разговор, целью которого является полный обзор требований с его стороны. Также позволяет быстро получить нужные сведения, но требует заранее подготовленных вопросов, хоть и не таких конкретизированных, что при анкетировании. К тому

же для сбора достаточного количества требований нужно провести интервью у нескольких представителей разных групп потребителей, мнения которых касательно рассматриваемого вопроса могут сильно отличаться.

3. Повторное использование спецификации. Спецификация – структурированный набор требований к программному обеспечению и его внешним интерфейсам. Техническое задание, подготовленное на предыдущем проекте, может быть использовано для текущего с целью сократить продолжительность сбора, анализа и разработки требований. Однако для текущей разработки сложно найти подобную спецификацию в открытом доступе, и многие из них обладают излишней детализацией, сильно отличающей их от целей данного проекта.

4. Изучение документации и литературы из открытых источников. Данный метод требует много времени для поиска, сбора и систематизации требований из открытых источников, однако позволяет собрать наибольший объём информации, глубже изучить вопрос с разных точек зрения, а также создать основу для подготовки перечня вопросов для анкетирования и интервью.

Из описанных выше методов сбора требований наиболее подходящим для разработки данной выпускной квалификационной работы является изучение документации и литературы из открытых источников, так как это не требует каких-либо начальных знаний в рассматриваемой предметной области и позволяет наиболее тщательно и с разных сторон изучить её.

4.3 Определение требований к разработке

Разрабатываемый продукт представляет из себя реализацию искусственного интеллекта для компьютерной игры жанра RTS. При определении требований и анализе актуальных проблем среди разработчиков компьютерных игр и исследователей в области ИИ выявились некоторые различия, среди которых самым значимым является определение возможностей разрабатываемого ИИ.

Для исследователей важнее всего усовершенствование ИИ для достижения им наиболее совершенного вида, во много раз превосходящего человеческие возможности. Для разработчиков игр желательно создание человекоподобного ИИ для имитации игры с реальным оппонентом. Также для разработчиков важна возможность частого изменения поведения в сторону упрощения или, наоборот, усложнения поведения ИИ, чтобы угодить наибольшему количеству игроков. Для самих же игроков наибольшую ценность представляет игровой искусственный интеллект, действия которого сложного предсказать.

Таким образом, список общих требований к продукту должен выглядеть следующим образом:

1. Разрабатываемые архитектура и алгоритм должны быть основаны на относительно новом, но перспективном подходе, чтобы привлечь внимание исследователей;

2. Программа должна содержать разработку поведения ИИ для полноценной игровой сессии игры жанра RTS, чтобы её алгоритм можно было применить при создании новой игры этого жанра;

3. Программа должна иметь гибкую структуру, чтобы любой конечный пользователь мог менять, добавлять новое или удалять ненужное поведение ИИ.

Из описанных выше базовых требований вытекают более конкретизированные функциональные требования, описанные в таблице 3.

Таблица 3 – Анализ требований к разработке

Функции по группам	Источник	Требования
Производительность	Лучшие отраслевые практики	Программа должна выполнять игровые действия, определённые реализованной в ней логикой, на скорости свыше 10 игровых кадров в секунду

Окончание таблицы 3

Функции по группам	Источник	Требования
Удовлетво-рённость	Лучшие отраслевые практики	Программа должна предоставлять функционал для реализации всех возможностей игр жанра RTS
Гибкость	Лучшие отраслевые практики	Программа должна быть модифицируема конечным пользователем без дополнительных разработок и изменений существующего кода
Обучаемость	Лучшие отраслевые практики	Искусственный интеллект должен иметь возможность обучаться на игровых записях

Требования, описанные в таблице 3, кроме последнего, удовлетворяют общим запросам всех групп конечных пользователей. Обучаемость же требуется для создания ИИ со сверхчеловеческим поведением, что не нужно разработчикам, но нужно исследователям.

4.4 Операциональные определения требований

В таблице 4 представлено операциональное определение описанных выше требований для их конкретизации.

Таблица 4 – Операциональное определение требований к разработке

Требование	Тест		Решение (правило)		
	Характеристика	Метод измерения	Соответствие	Частичное соответствие	Несоответствие
Производительность	Скорость	Программное измерение количества игровых действий за один игровой кадр	Совершение не менее пяти действий за один игровой кадр при скорости 10 кадров в секунду	Совершение одного действия за игровой кадр при той же скорости игры	Совершение менее одного действия за один игровой кадр
Удовлетворённость	Корректность	Проверка выполнения основных игровых сценариев	Воспроизведение полноценной игровой сессии с использованием всех возможностей, предоставляемых игрой	Воспроизведение полноценной игровой сессии с использованием только базовых возможностей игры	Программа не способна воспроизвести хотя бы один игровой сценарий

Окончание таблицы 4

Гибкость	Удобство использования	Анализ удобства использования продукта с точки зрения конечного пользователя	Разработан удобный интерфейс (или API) для создания собственной реализации ИИ	Предоставляемый в открытом доступе исходный код структурирован и понятен, имеет подробное описание	Исходный код требует доработок даже для простейших реализаций
Обучаемость	Удобство применения	Анализ возможностей для применения	Программа способна обучаться на предоставляемых пользователем игровых записях, находящихся в открытом доступе	Программа способна обучаться только на данных, прошедших специальную обработку	К Программе неприменимо машинное обучение

4.5 Предложения по улучшению

Разработанная Программа полностью соответствует требованию к производительности, но лишь частично – требованиям удовлетворённости и гибкости, и полностью не соответствует требованию о возможности применения машинного обучения.

Предложения по улучшению представлены в таблице 5.

Таблица 5 – Предложения по улучшению разработки

Требование	Анализ текущего состояния	Причины	Способ улучшения
Удовлетворённость	Программа не способна использовать все возможности, предоставляемые игрой, в полной мере	Недостаточная сложность ИИ	Усложнение поведения ИИ путём добавления новых сценариев поведения
Гибкость	Программа не предоставляет пользовательский интерфейс или API для создания пользователем собственной реализации, но имеет исходный код, не требующий изменений	Отсутствует необходимый функционал	Разработка пользовательского интерфейса или API
Обучаемость	Программа не способна обучаться на игровых записях	Отсутствует необходимый функционал	Разработка функционала для обработки данных из открытых источников и применения на их основе методов машинного обучения

ЗАКЛЮЧЕНИЕ

Задача планирования действий заключается в достижении из начального состояния конечного, или целевого, состояния путём воздействия на среду с помощью некоторой последовательности действий – плана, генерируемого Программой.

Для решения поставленной задачи в стратегической игре был проведён обзор распространённых методов создания игрового ИИ и существующих реализаций. Были подробно изучены сильные и слабые стороны рассмотренных методов и аналогов, выявлены основные недостатки, устранение которых стало одним из требований для разрабатываемой Программы.

В качестве главного метода решения были выбраны деревья поведения, так как они предоставляют удобную и наглядную структуру для создания игрового ИИ, обеспечивают масштабируемость при усложнении поведения, поддерживают машинное обучение. Главным недостатком деревьев поведения является детерминированность принимаемых в процессе игры решений, выбор единственного подходящего действия без анализа последствий. Для устранения этого недостатка было предложено использование планировщиков, которые могли бы генерировать последовательность действий из разных стратегий, предоставляемых деревьями поведения, что может стать темой для дальнейших исследований.

На основе деревьев поведения были разработаны математическая модель и архитектура. Математическая модель позволяет воздействовать на среду на основе некоторых условий и действий, описанных с помощью деревьев поведения. На основе математической модели была разработана DTD-схема дерева поведения, а также пример перехода модели из одного состояния в другое с помощью XML-описания.

В разработанной архитектуре помимо создания, запуска и управления деревьями поведения предусмотрен дополнительный уровень обработки информации для взаимодействия с игровой средой. Были разработаны UML-диаграммы компонентов и потоков данных.

Представленный в данной работе алгоритм решает задачу планирования действий с помощью деревьев поведения, выбирая цель, способ её достижения и выполняя составленный план. Алгоритм удовлетворяет свойствам корректности и полноты, заданным в постановке задачи.

Описанные алгоритм и архитектура были реализованы в виде Программы, для которой были разработаны диаграммы классов и последовательности. Программа включает в себя пять статических классов, более десяти абстрактных классов, на основе которых и создаются деревья поведения, а также несколько классов, которые предполагают создание единственного экземпляра. При создании диаграммы были использованы такие шаблоны проектирования, как Одиночка, Фасад и Компоновщик.

Для проверки Программы на соответствие выдвинутым требованиям был проведён эксперимент, для которого были составлены сценарии ожидаемого поведения Программы в разных игровых ситуациях. По итогам эксперимента можно сделать вывод, что Программа удовлетворяет основным требованиям, выдвинутым при постановке задачи, анализе аналогов и составлении условий эксперимента, но при реализации некоторых сценариев поведение игровых сущностей отличается от ожидаемого. Для устранения этих проблем потребуется усложнение игровой логики, заложенной в разработанных деревьях поведения.

Таким образом, в данной работе были выполнены все поставленные цели и задачи, проведён анализ предметной области, выявлены существующие проблемы в реализации ИИ для игр жанра RTS, описаны идеи для дальнейших исследований.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. StarCraft. Официальный сайт [Электронный ресурс]. URL: https://starcraft.com/en-gb/ (дата обращения: 24.12.2022).
2. AIIDE StarCraft AI Competition. Официальный сайт [Электронный ресурс]. URL: https://www.cs.mun.ca/~dchurchill/starcraftaicomp/ (дата обращения: 24.12.2022).
3. SSCAIT. Официальный сайт [Электронный ресурс]. URL: https://sscaitournament.com (дата обращения: 24.12.2022).
4. Safra S., Tennenholtz M. On planning while learning //Journal of Artificial Intelligence Research. – 1994. – Т. 2. – С. 111-129.
5. Hanks S., Weld D. S. A domain-independent algorithm for plan adaptation //Journal of Artificial Intelligence Research. – 1994. – Т. 2. – С. 319-360.
6. Sweetser P., Wiles J. Scripting versus emergence: issues for game developers and players in game environment design //International Journal of Intelligent Games and Simulations. – 2005. – Т. 4. – №. 1. – С. 1-9.
7. Frutos-Pascual M., Zapirain B. G. Review of the use of AI techniques in serious games: Decision making and machine learning //IEEE Transactions on Computational Intelligence and AI in Games. – 2015. – Т. 9. – №. 2. – С. 133-152.
8. Weber B.G., Mateas M., Jhala A. Building human-level AI for real-time strategy games, Proc. AIIDE Fall Symp. Adv. Cogn. Syst. - 2011. - С. 329-336.
9. Muise C. et al. Planning over multi-agent epistemic states: A classical planning approach //Proceedings of the AAAI Conference on Artificial Intelligence. – 2015. – Т. 29. – №. 1.
10. Sun L. et al. Modified adversarial hierarchical task network planning in real-time strategy games //Applied Sciences. – 2017. – Т. 7. – №. 9. – С. 872.
11. Fikes R. E., Nilsson N. J. STRIPS: A new approach to the application of theorem proving to problem solving //Artificial intelligence. – 1971. – Т. 2. – №. 3-4. – С. 189-208.

12. Barto A. G., Sutton R. S. Reinforcement learning in artificial intelligence //Advances in Psychology. – North-Holland, 1997. – Т. 121. – С. 358-386.
13. Robertson G., Watson I. A review of real-time strategy game AI //Ai Magazine. – 2014. – Т. 35. – №. 4. – С. 75-104.
14. Shantia A., Begue E., Wiering M. Connectionist reinforcement learning for intelligent unit micro management in starcraft //The 2011 international joint conference on neural networks. – IEEE, 2011. – С. 1794-1801.
15. Churchill D., Saffidine A., Buro M. Fast heuristic search for RTS game combat scenarios //Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment. – 2012. – Т. 8. – №. 1. – С. 112-117.
16. Synnaeve G., Bessiere P. A Bayesian model for RTS units control applied to StarCraft //2011 IEEE Conference on Computational Intelligence and Games (CIG'11). – IEEE, 2011. – С. 190-196.
17. Gabriel I., Negru V., Zaharie D. Neuroevolution based multi-agent system for micromanagement in real-time strategy games //Proceedings of the fifth balkan conference in informatics. – 2012. – С. 32-39.
18. Ontanón S. et al. Case-based planning and execution for real-time strategy games //International Conference on Case-Based Reasoning. – Springer, Berlin, Heidelberg, 2007. – С. 164-178.
19. Palma R. et al. Combining expert knowledge and learning from demonstration in real-time strategy games //Case-Based Reasoning Research and Development: 19th International Conference on Case-Based Reasoning, ICCBR 2011, London, UK, September 12-15, 2011. Proceedings 19. – Springer Berlin Heidelberg, 2011. – С. 181-195.
20. Leece M., Jhala A. Sequential pattern mining in Starcraft: Brood War for short and long-term goals //Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. – 2014. – Т. 10. – №. 2. – С. 8-13.

21. Weber B., Mateas M., Jhala A. Applying goal-driven autonomy to starcraft //Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. – 2010. – Т. 6. – №. 1. – С. 101-106.
22. Weber B., Mateas M., Jhala A. Learning from demonstration for goal-driven autonomy //Proceedings of the AAAI Conference on Artificial Intelligence. – 2012. – Т. 26. – №. 1. – С. 1176-1182.
23. Описание и исходный код Soar-SC на Github [Электронный ресурс]. URL: https://github.com/bluechill/Soar-SC (Дата обращения: 30.11.2022).
24. Churchill D., Buro M. Build order optimization in starcraft //Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. – 2011. – Т. 7. – №. 1. – С. 14-19.
25. Baikadi A. et al. Improving goal recognition in interactive narratives with models of narrative discovery events //Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. – 2013. – Т. 9. – №. 4. – С. 2-8.
26. Buro M., Churchill D. Real-time strategy game competitions //Ai Magazine. – 2012. – Т. 33. – №. 3. – С. 106-106.
27. Čertický M. et al. Starcraft AI competitions, bots, and tournament manager software //IEEE Transactions on Games. – 2018. – Т. 11. – №. 3. – С. 227-237.
28. Ontanón S. et al. A survey of real-time strategy game AI research and competition in StarCraft //IEEE Transactions on Computational Intelligence and AI in games. – 2013. – Т. 5. – №. 4. – С. 293-311.
29. Dietterich T. Overfitting and undercomputing in machine learning //ACM computing surveys (CSUR). – 1995. – Т. 27. – №. 3. – С. 326-327.
30. Robertson G., Watson I. Building behavior trees from observations in real-time strategy games //2015 International symposium on innovations in intelligent systems and applications (INISTA). – IEEE, 2015. – С. 1-7.
31. Zimmer W. et al. Relationships between design patterns //Pattern languages of program design. – 1995. – Т. 1. – С. 345-364.

ПРИЛОЖЕНИЕ А

DTD-схема дерева поведения:

```
<!DOCTYPE behaviorTree[
<!ELEMENT behaviorTree (state, root)>
<!ATTLIST behaviorTree manager (macro|micro) “micro”>

<!ELEMENT state EMPTY>
<!ATTLIST state
values CDATA #REQUIRED>
<!ELEMENT root (nonLeafNode)>
<!ATTLIST root id ID #REQUIRED>

<!ELEMENT nonLeafNode EMPTY>
<!ATTLIST nonLeafNode
id ID #REQUIRED
parentID IDREF #REQUIRED
childesID IDREFS #REQUIRED
status (noRun|success|failure|inWork|error) “noRun”>

<!ELEMENT leafNode (action|condition|link)>
<!ATTLIST leafNode
id ID #REQUIRED
parentID ID #REQUIRED
status (noRun|success|failure|inWork|error) “noRun”>

<!ELEMENT action EMPTY>
<!ATTLIST action
values CDATA #REQUIRED
newValues CDATA #REQUIRED>
```

```
<!ELEMENT condition EMPTY>
<!ATTLIST condition
values CDATA #REQUIRED
expectedValues CDATA #REQUIRED>

<!ELEMENT link EMPTY>
<!ATTLIST link
rootAnotherTree IDREF #REQUIRED>
  ]>
```

ПРИЛОЖЕНИЕ Б

XML-описание простого дерева поведения в начальном состоянии:

```
<behaviorTree manager="macro">
<state firstValue="value1" secondValue="value2"/>
        <root id="#root">
                <nonLeafNode id="#sequence" parentID="#root"
                childesID="#condition #action" status="noRun"/>
        </root>
</behaviorTree>

<leafNode id="#condition" parentID="#sequence" status="noRun">
        <condition values="firstValue" expectedValues="value1"/>
</leafNode>

<leafNode id="#action" parentID="#sequence" status="noRun">
        <action values="firstValue secondValue"
        newValues="newValue1 newValue2"/>
</leafNode>
```

ПРИЛОЖЕНИЕ В

XML-описание простого дерева поведения в конечном состоянии:

```
<behaviorTree manager="macro">
<state firstValue="newValue1" secondValue="newValue2"/>
        <root id="#root">
                <nonLeafNode id="#sequence" parentID="#root"
                childesID="#condition #action" status="success"/>
        </root>
</behaviorTree>

<leafNode id="#condition" parentID="#sequence" status="success">
        <condition values="firstValue" expectedValues="value1"/>
</leafNode>

<leafNode id="#action" parentID="#sequence" status="success">
        <action values="firstValue secondValue"
        newValues="newValue1 newValue2"/>
 </leafNode>
```