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

Направление подготовки: 09.03.01 “Информатика и вычислительная техника”
Профиль: “Вычислительные машины, комплексы, системы и сети”
Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

К защите допустить:
Заведующий кафедрой
д. т. н., профессор _____ М. С. Куприянов

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Тема: “Алгоритмы управления восприятием интеллектуального агента в среде RoboCup Soccer Simulator”

Студент _____ А. В. Табаков

Руководитель
к. т. н., доцент _____ М. Г. Пантелеев

Консультант от кафедры
к. т. н., доцент, с. н. с. _____ И. С. Зуев

Консультант кафедры ИМ
к. т. н., доцент _____ В. И. Фомин

Санкт-Петербург
2018 г.

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

Направление 09.03.01 Информатика и вычислительная техника
Профиль: "Вычислительные машины, комплексы, системы и сети"
Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой ВТ
д. т. н., профессор
(М. С. Куприянов)
"___" _____ 2018 г.

ЗАДАНИЕ
на выпускную квалификационную работу

Студент Табаков Андрей Викторович Группа № 4306

1. Тема Алгоритмы управления восприятием интеллектуального агента в среде RoboCup Soccer Simulator

(утверждена приказом № _____ от _____)

Место выполнения ВКР: СПбГЭТУ "ЛЭТИ"

2. Объект и предмет исследования

Модель визуального сенсора интеллектуального агента в среде RoboCup Soccer Simulator.

3. Цель работы

Разработка алгоритма управления восприятием с использованием метода опережающего итеративного планирования. Разработка симуляционной системы для экспериментального исследования и отладки данного алгоритма.

4. Исходные данные

Научные работы и пособия на тему интеллектуальных агентов и многоагентных систем. Руководство по работе RoboCup Soccer Server.

Платформа разработки: операционная система – Ubuntu Linux, язык программирования – Kotlin, среда разработки – Intellij IDEA.

5. Содержание

Понятия и особенности интеллектуального агента, визуального восприятия, реализации восприятия в RoboCup Soccer Server. Модели и алгоритмы для управления визуальными сенсорами агента. Реализация предложенных алгоритмов и экспериментальное исследование. Разработка и стандартизация программных средств.

6. Технические требования

- Разработанные алгоритмы должны быть построены с учётом метода опережающего итеративного планирования.
- Алгоритмы должны учитывать, что агент является системой реального времени.
- Приложение должно взаимодействовать с RoboCup Soccer Server по протоколу UDP.

7. Дополнительные разделы

Дополнительным разделом представлена разработка и стандартизация программных средств.

8. Результаты

Пояснительная записка, реферат, аннотация, иллюстративный материал, исходный код приложения на языке программирования Kotlin.

Дата выдачи задания «__» _____ 2018 г.	Дата представления ВКР к защите «__» _____ 2018 г.

Руководитель к. т. н., доцент	_____	М. Г. Пантелеев
Студент	_____	А. В. Табаков

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

Направление 09.03.01 Информатика и вычислительная техника
Профиль: "Вычислительные машины, комплексы, системы и сети"
Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой ВТ
д. т. н., профессор
(М. С. Куприянов)
"___" _____ 2018 г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения выпускной квалификационной работы

Тема Алгоритмы управления восприятием интеллектуального агента в среде RoboCup Soccer Simulator

Студент Табаков Андрей Викторович Группа № 4306

№ этапа	Наименование работ	Срок выполнения
1	Освоение материалов, необходимых для выполнения выпускной квалификационной работы	01.03.2018 – 20.03.2018
2	Рассмотрение интеллектуальных агентов, среды RoboCup Soccer, модели визуального восприятия	01.04.2018 – 20.04.2018
3	Формализация и постановка задачи	25.04.2018 – 27.04.2018
4	Проектирование и разработка алгоритма управления восприятием	28.04.2018 – 12.05.2018
5	Отладка и экспериментальное исследование	13.05.2018 – 20.05.2018
6	Оформление пояснительной записки	25.05.2018 – 03.06.2018
7	Предварительная защита работы	10.06.2018
8	Представление работы к защите	19.06.2018

Руководитель
к. т. н., доцент _____ М. Г. Пантелеев

Студент _____ А. В. Табаков

РЕФЕРАТ

Пояснительная записка: 57 с., 4 ч., 16 рис., 1 табл., 2 прил.

Цель работы: Разработать алгоритм управления визуальным сенсором агента-футболиста, а также симуляционную систему, которая позволит интегрировать алгоритмы управления восприятием в поведение агентов.

В результате выполнения выпускной квалификационной работы были спроектированы и реализованы алгоритмы управления визуальным восприятием на языке программирования Kotlin. На данный момент не имеется аналогичного алгоритма управления визуальным сенсором в реальном времени, построенном на методе опережающего итеративного планирования. Реализованные алгоритмы позволяют в заданный промежуток времени, получить достаточное количество информации для совершения действия. Помимо алгоритмов, была реализована симуляционная система, которая позволяет тестировать и встраивать различные исследовательские алгоритмы в поведение агентов.

Разработанные алгоритмы могут быть использованы при построении различных агентов реального времени, которым требуется управлять визуальным сенсором.

Ключевые слова: Интеллектуальный агент, многоагентная система, алгоритм управления восприятием, визуальный сенсор, RoboCup Soccer.

ABSTRACT

Artificial intelligence has a significant value in our lives. One of the main problem for artificial intelligent system is perception management. The aim is to develop an algorithms for perception management of a real-time intelligent agent for RoboCup Soccer Server, also need to implement simulation system that will allow integrating this algorithms into agent behavior.

As a result of the performance of the final qualifying work, an algorithms of visual perception management, were designed and implemented in Kotlin. A simulation system was developed where the algorithms is integrated into the behavior of one of the agents. In the paper, models and algorithms for visual perception management of the agent player are presented. Main algorithm based on the methodology of advanced iterative planning, which allow obtaining a sufficient amount of information for the fulfillment of the action in a given period of time.

The developed algorithms can be used to construct various real-time agents that need to operate a visual sensor.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 9
ВВЕДЕНИЕ 10
1 Интеллектуальные агенты в среде Robocup soccer simulator 12
1.1 Модель интеллектуальных агентов и многоагентных систем 12
1.2 Среда RoboCup Soccer Simulator 14
1.3 Понятия визуального сенсора 16
1.4 Существующие алгоритмы управления восприятием 18
1.5 Метод опережающего итеративного планирования 20
1.6 Постановка задачи 20
2 Модели и алгоритмы управления восприятием интеллектуального агента 22
2.1 Модели управления визуальным восприятием 22
2.1.1 Модель максимального угла обзора 22
2.1.2 Модель выбора качества получаемой информации интеллектуального агента 23
2.1.3 Модель вычисления допустимого числа тактов без объекта наблюдения 24
2.2 Алгоритмы управления визуальным восприятием 25
2.2.1 Проектирование алгоритма управления восприятием 26
2.2.2 Алгоритм расчёта максимального угла обзора 28
2.2.3 Алгоритм адаптации ширины угла обзора 30
2.2.4 Алгоритм вычисления времени допустимого числа тактов без объекта наблюдения 32

2.2.5 Алгоритм извлечения визуальной информации 33
3 Разработка и отладка Программного 36
обеспечения 36
3.1 Разработка интеллектуальных агентов для симуляции 36
3.2 Проектирование архитектуры программной реализации 37
3.2.1 Sequence диаграмма 37
3.2.2 Диаграмма классов 39
3.3 Реализация алгоритма управления сенсором 42
3.3.1 Метод «calculateMaxVisibleAngle» 44
3.3.2 Метод «calculateSensorTickWithoutMainObject» 44
3.3.3 Метод «getVisualInfo» 44
3.4 Экспериментальное исследование 46
4 Разработка и стандартизация программных средств 47
4.1 Планирование работ 47
4.2 Отечественные и международные стандарты 50
4.3 Классификация программного продукта 51
4.4 Определение затрат на выполнение и внедрение проекта 52
ЗАКЛЮЧЕНИЕ 55
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 56
ПРИЛОЖЕНИЕ А Исходный код алгоритма управления 58
восприятием интеллектуального агента 58
ПРИЛОЖЕНИЕ Б Лог работы алгоритма управления восприятием
интеллектуального агента 71

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

API (англ. Application Programming Interface) – набор готовых реализаций методов, классов, структур и констант, которые предоставляются сервисом или библиотекой для использования во внешних программных продуктах.

IDE (англ. Integrated Development Environment) – среда разработки, система программных компонентов, используемая программистами для разработки программного обеспечения.

JVM (англ. Java Virtual Machine) – виртуальная машина исполняющая байт-код Java.

Kotlin – статически типизированный язык программирования, работающий поверх JVM и разрабатываемый Российской, Санкт-Петербургской компанией JetBrains.

Multi-agent system – многоагентная система, в которой отдельные интеллектуальные агенты имеют возможность коммуницировать между собой.

RoboCup Soccer Server – приложение платформа, предоставляющая окружающую среду для симуляции поведения интеллектуальных агентов в контексте виртуального футбола.

Sequence diagram – диаграмма последовательности, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл какого-либо определённого объекта, момент его создания, деятельности и уничтожения.

Ubuntu – бесплатная операционная система созданная на ядре Linux.

UDP (англ. User Datagram Protocol) – транспортный протокол для передачи данных без установления связи. Является ненадёжным сервисом, в протоколе не обеспечивается порядок и целостность пакетов, а также доставка пакетов.

ВВЕДЕНИЕ

Искусственный интеллект занимает всё большее место в современном мире. Интеллектуальные агенты являются одним из классов искусственного интеллекта, представляя собой разумную единицу, которая способна воспринимать информацию с помощью сенсоров и осуществлять влияние на окружающую среду эффекторами. При существовании более одного агента в среде, они получают возможность коммуницировать и строить планы взаимодействуя друг с другом, такой род систем, называется многоагентной.

Одной из систем моделирования среды интеллектуальных агентов является RoboCup Soccer Simulator, пакет программ предоставляющий возможность разрабатывать и исследовать интеллектуальных агентов-футболистов. Футбол является эталонной задачей с множеством различных ситуаций, решение которых помогает реализовывать и отлаживать алгоритмы поведения агентов. В футболе можно выделить некоторые виды задач: управление восприятием, оценка ситуации, принятие решений и другие. RoboCup Soccer Simulator является динамической системой реального времени, что вносит необходимость разрабатывать алгоритмы с учётом данного факта. Реализованные алгоритмы поведения агентов-футболистов, можно экстраполировать для решения задач, которые ставятся перед искусственным интеллектом в различных областях.

Исследования в области интеллектуальных агентов, в настоящее время представляют особенный интерес, вследствие развития автопилотируемых средств передвижения, автономных роботов для производства, голосовых ассистентов и прочего.

Важную роль в интеллектуальном агенте является управление сенсорами, которые позволяют получать информацию, на основе которой можно построить планы.

Цель работы: Разработать алгоритм управления визуальным сенсором агента-футболиста, а также реализовать симуляционную систему, которая позволит интегрировать алгоритмы управления восприятием в поведение агентов.

Предмет разработки: алгоритмы поведения интеллектуального агента в многоагентной среде.

Аннотации основных разделов: в первой главе представлен краткий обзор интеллектуальных агентов, а также описание проблем, связанных с визуальным восприятием. Вторая глава содержит понятия моделей и алгоритмов для управления визуальными сенсорами. В третьей главе даны описания архитектуры и особенности реализации приложения. Четвёртая глава представляет собой описание проекта с точки зрения управления проектами. В заключении представлены выводы по настоящей работе.

1 ИНТЕЛЛЕКТУАЛЬНЫЕ АГЕНТЫ В СРЕДЕ ROBOCUP SOCCER SIMULATOR

Интеллектуальные агенты представляют интерес для современной области искусственного интеллекта. В совокупности, несколько интеллектуальных агентов, представляют собой многоагентную систему.

В данном разделе рассматриваются интеллектуальные агенты, среда для моделирования поведения агентов RoboCup Soccer Simulator, проблемы, связанные с управлением визуального сенсора в данной среде, существующие методы управления визуальным сенсором и обоснование необходимости разработки общего алгоритма управления визуальным восприятием.

1.1 Модель интеллектуальных агентов и многоагентных систем

Интеллектуальный агент представляет подкласс искусственного интеллекта. Под интеллектуальным агентом понимается основанная на знаниях система реального времени, реализующая автономное целенаправленное поведение в открытых, динамических и неопределенных мирах [1]. В [2] агент описывается как система имеющая следующие свойства:

- Автономность – способность действовать (решать задачи) без прямого вмешательства человека или других агентов, самостоятельно управляя своими действиями и внутренним состоянием
- Реактивность – способность к активному восприятию внешней среды и своевременному отклику на происходящие в ней события.
- Проактивность – способность осуществлять целенаправленное поведение и проявлять инициативу, а не просто откликаться на внешние события
- Социальность – способность взаимодействовать с другими агентами и/или людьми, общаясь с ними посредством языка(ов) межагентного общения.

В [3] даётся *сильное определение* агента, как интенсиональной (intentional) системы, то есть системы, при описании внутреннего состояния и процессов переработки информации которой используются так называемые категории. К таким категориям принято относить: знания, убеждения, желания, цели, намерения, планы, обязательства и тому подобное.

Агенты существуют в окружающей среде, которая является открытым динамическим неопределённым миром [3]. Открытые миры не позволяют на этапе проектирования описать среду функционирования агентов, поскольку неизвестно количество и в каких состояниях агент обнаружит сущности в окружающим мире. Динамичность вносит фактор постоянного изменения мира, так целенаправленные динамические объекты и сущности способны изменять состояние мира и других объектов. Неопределённость означает принципиальную невозможность полно и точно идентифицировать состояния мира. При функционировании в многоагентных средах важную роль играет неопределённость относительно целей других агентов, что обуславливает ограниченную предсказуемость развития событий.

Многоагентная система определяется как совокупность интеллектуальных агентов, взаимодействующих в процессе своей целенаправленной деятельности. Среди взаимодействий между интеллектуальными агентами можно выделить основные типы отношений:

- Полная согласованность (когерентность) целей – применяется в случае решения несколькими агентами общей задачи.
- Кооперация на основе взаимной выгоды – решение задачи одним агентом предоставляет полезность для решения задачи другим агентом.
- Нейтральность – в данном случае агенты имеют непересекающиеся, независимые цели. В этом виде взаимодействия агенты проявляют нейтралитет друг к другу

- Конкуренция – возникает в случае пересечения целей агентов и необходимостью захвата недостаточного всем агентам определённого ресурса.
- Антагонизм – применим, когда агенты изначально создаются для противодействия другим агентам.

Перечисленные типы взаимоотношений индивидуальны для каждого взаимодействия между агентами, поэтому в одной многоагентной системе могут существовать все виды данных отношений, как в начальном состоянии мира, так и по истечении определённого количества времени, так как цели агента могут меняться с течением времени.

При проектировании и реализации интеллектуальных агентов необходимо учитывать природу систем реально времени, в частности ограниченности временного ресурса. Интеллектуальные агенты должны принимать решения ограниченной рациональности за заданный промежуток времени. Для решения подобного класса задач можно воспользоваться методом опережающего итеративного планирования.

1.2 Среда RoboCup Soccer Simulator

RoboCup Soccer Simulator – является набором программных средств для исследования поведения агентов в многоагентной среде, а также для проведения ежегодного чемпионата по виртуальному футболу. В составе пакета программ, присутствуют следующие компоненты [4]:

- Server – сервер, который предоставляет агентам: окружающий мир и его объекты, возможность взаимодействия с другими агентами. Также с помощью данного сервера осуществляется управление агентом и получение сенсорной информации.
- Monitor – программа, которая подключается к серверу и визуально демонстрирует происходящее на сервере.
- LogPlayer – Программа для проигрывания логов с сервера, для повторного просмотра игры с возможностями пошагового воспроизведения.

RoboCup Soccer Simulator – предлагает для рассмотрения множество проблем реального мира с точки зрения искусственного интеллекта и многоагентных систем, такие как помехи при получении информации с сенсоров и ограниченность действий эффекторов и возможности коммуникаций [5]. Интеллектуальные агенты-футболисты должны принимать решения в реальном времени. У них имеется только ограниченное знание мира в любой конкретный момент времени и многие его аспекты остаются неизвестными. В дополнение к вышесказанному, действия по получению сенсорной информации и выполнению действий имеют асинхронную природу, что позволяет использовать классическую парадигму искусственного интеллекта по задействованию эффекторов на основе только что полученной информации. Агент также имеет ограниченное представление о намерениях других агентов на поле, независимо от того являются ли другие агентами противниками или сокомандниками. Так как нет возможности рассчитать все варианты развития событий становится необходимо разрабатывать агентов таким образом, чтобы они могли грамотно реализовывать свой потенциал, подстраиваться под изменяющиеся условия и быть обученными стратегиям игры в футбол.

Агенты реализуются разработчиками самостоятельно, предоставляя возможность самостоятельно проектировать алгоритмы поведения. Вследствие данная система является отличным средством для разработки и отладки алгоритмов, которые могут быть использованы не только в агентах футболистах, но и в других видах агентов.

В совокупности, данный пакет предоставляет широкий набор возможностей для моделирования и отладки работы интеллектуальных агентов в многоагентной системе

1.3 Понятия визуального сенсора

Сенсором является датчик, способный фиксировать изменения свойств объекта или окружающей его среды. Сенсоры могут быть как внешнего состояния, так и внутреннего.

В задачи сенсора входит получение информации из окружающей среды для реализации текущего и планирования следующего действий. Критерием эффективности сенсора является охват максимально полного количества вариантов информации для текущего и планируемого действий. На качественном уровне, у сенсоров можно выделить две характеристики: 1) полнее – получение большего количества информации в заданный промежуток времени; 2) точнее – получение большего количества свойств воспринимаемых сенсором объектов.

Человек имеет два визуальных сенсора в виде глаз. С помощью них, становится возможным получения сведений о виде объектов, окружающего мира. Зрение имеет следующие ограничения:

- Дальность обзора – на каком расстоянии доступны визуальные свойства объекта.
- Ширина угла обзора – какой угол обзора доступен в один момент времени.
- Качество получаемой информации – параметр, который определяет сколько свойств объекта способен распознать визуальный сенсор.

Возможность фокусировки зрения позволяет сузить область видимости, концентрируя зрение только на одной его части, для получения лучшего качества визуальной сенсорной информации.

В среде виртуального футбола, интеллектуальный агент имеет 3 режима ширины угла обзора, широкий – 180°, нормальный – 90° и узкий – 45° [4], а также 2 режима качества получаемой информации. В зависимости от ширины

угла обзора, чем шире угол, тем меньше дальность обзора и дольше время получения информации. Режим качества получаемой информации влияет на время получаемой информации, чем лучше качество, тем больше времени между сенсорными сообщениями.

Каждый режим визуального сенсора представляет собой коэффициент с плавающей точкой. Так для ширины угла обзора используются следующие коэффициенты (*ViewWidthFactor*): WIDE – 2; NORMAL – 1; NARROW – 0.5. Режим качества информации также имеет коэффициенты (*ViewQualityFactor*): HIGH – 1; LOW – 0.5.

Используя данные коэффициенты становится возможным рассчитать ширину угла обзора:

$$ViewAngle = VisibleAngle * ViewWidthFactor$$

где:

VisibleAngle – по умолчанию равен 90°

Также можно рассчитать время одного сенсорного такта (*SensorFreq*)

$$SensorFreq = SenseStep * ViewQualityFactor * ViewWidthFactor$$

где:

SenseStep – по умолчанию равен 150 миллисекунд

На рисунке 1.1 представлена модель визуального сенсора агента-футболиста. Точки a,b,c,d,e,f,g – представляют собой некоторые объекты на поле, в данном контексте не имеет значения, что именно они олицетворяют. Обозначения visible_distance и view_angel – представляют видимое расстояние и угол обзора агента соответственно. Если объекты попадают в область видимого расстояния, но не в угол обзора, то агент может однозначно знать название объекта (мяч, флаг, игрок, ворота). Пунктирными линиями обозначены граничные состояния зрения агента, для наблюдения за другими агентами: unum_far_length – граница, где агент может видеть номер и команду другого футболиста, unum_too_far_length – в данной области видно название команды, но номер игрока только с определённой случайной вероятностью,

team_far_length – в данной области видно агент может определить только команду видимого другого агента, team_too_far_length – граница для определения команды игрока с некоторой случайной вероятностью, за ней, агент не может определить команду другого агента.

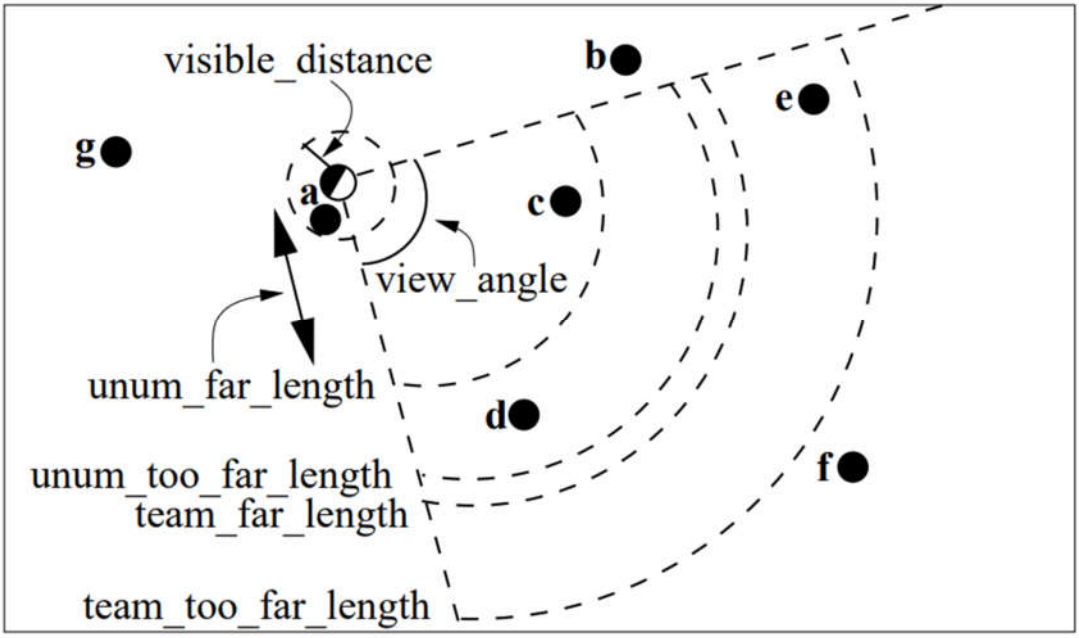

Рисунок 1.1 – Модель визуального сенсора агента-футболиста

1.4 Существующие алгоритмы управления восприятием

Проблематика управления восприятием представляет очень большую область для исследования. В настоящее время одной из самых трудных областей компьютерных наук, является обработка и рекогносцировка визуальной информации, а также непосредственное управление данным визуальным сенсором. Особенное распространение данная проблема получила в интеллектуальных агентах, имеющие визуальные сенсоры, вследствие были исследованы и разработаны различные алгоритмы управления визуальным восприятием.

В [6] предлагается рассмотреть два алгоритма для обработки визуального восприятия. В первом методе для определения обстановки мира предлагается использовать основанные на картинках представления, используется

допущение, что каждый объект имеет свой цвет, тогда при обработке изображения агент может считывать цвета объектов, вычислять список видимых объектов, а также определять отношения между объектами. Другой алгоритм основан на методе излучения разноцветных лучей, агент посылает несколько лучей и принимает информацию где цвет луча пересёкся с цветом объекта. При управлении визуальными сенсорами необходимо учитывать данную природу обработки визуальной информации, оптимизировав изменение положения сенсоров для облегчения расчётов.

В [7] предложен подход активного восприятия. Данный метод основан на принципе интеллектуальных агентов убеждение-желание-действие, которым руководствуется агент. Активное восприятие фокусируется на динамических объектах мира, направляя визуальные сенсоры в том направлении, где происходит больше изменений динамической среды. Предполагает, что данный алгоритм использует возможность переключать режим восприятия в зависимости от поступающих данных. Когда агент вычислил для себя определённую цель, он начинает фокусироваться только на тех видах объекта, которые могут помочь достичь его цель, а во время, когда агент ещё не определил чёткую цель, он пытается направлять взгляд таким образом, чтобы охватывать, как можно больше состояний мира.

[8] для управления восприятием используется модель, которая содержит базу знаний о свойствах некоторых объектах мира, на основе которых можно построить различные предположения. Так, имея достаточную базу знаний, появляется возможность комбинировать разные виды восприятия. Получив на вход информацию со слухового сенсора, можно по звуку определить класс объектов, находящихся в данном направлении, тем самым дав дополнительную информацию другим сенсорам. Как было сказано ранее, агенту необходимо фокусироваться только на необходимых для достижения целей объектах,

используя комбинации различных сенсоров можно одновременно обрабатывать разные виды информации и соответственно фокусировать различные сенсоры в необходимом направлении.

1.5 Метод опережающего итеративного планирования

Итеративное планирование от будущего состояния предполагает выполнение прогнозирования ситуации в каждом такте на момент окончания текущего действия [9]. Используемые при этом модели в соответствии с фазами планирования можно разбить на три группы:

- Генерация базового множества возможных вариантов действий
- Конкретизация и оценка полезности вариантов действий
- Окончательная конкретизация действий и выбор лучшего для выполнения

Общая длительность процесса обдумывания определяется длительностью текущего действия и в общем случае может включать любое число тактов. При этом генерация базового множества действий всегда выполняется в первом такте интервала обдумывания, а окончательная конкретизация и выбор действия в последнем. Конкретизация и оценка действий выполняются на всём интервале обдумывания решения. По мере выполнения текущего действия агент приближается к моменту его завершения и получает в каждом такте новую информацию о состоянии мира. Поскольку интервал прогнозирования ситуации на момент окончания текущего действия с каждым тактом сокращается, точность прогнозирования ситуации на момент окончания действия повышается.

1.6 Постановка задачи

Основываясь на всём вышесказанном возникает необходимость в общем алгоритме управлении восприятием, способным учитывать ограничения визу-

альных сенсоров и ориентированном на получение максимально полной информации об обстановки окружающего мира. Алгоритм необходимо проектировать с поправкой на тот факт, что агент является системой реального времени, в качестве базовой парадигмы взять метод опережающего итеративного планирования.

Контекст решаемой задачи следующий: Агент, ведущий мяч прикрывается агентом другой команды, вследствие чего, у него возникает потребность дать пас одному из напарников своей команды, которые расположены впереди него на некотором расстоянии. Агенту необходимо принять визуальную информацию перед собой, заметить напарников своей команды и получить достаточное количество информации, для оценки общей обстановки и принятия общего решения.

В качестве результата ожидается приложение на языке программирования Kotlin, взаимодействующее с сервером по протоколу UDP, а также имеющее реализацию алгоритма управления визуального восприятия интеллектуального агента.

2 МОДЕЛИ И АЛГОРИТМЫ УПРАВЛЕНИЯ ВОСПРИЯТИЕМ ИНТЕЛЛЕКТУАЛЬНОГО АГЕНТА

Данный раздел включает модели и алгоритмы, используемые при разработке общего алгоритма управления визуальным сенсором.

2.1 Модели управления визуальным восприятием

В приведённой главе рассматриваются методы управления визуальным восприятием в среде виртуального футбола.

2.1.1 Модель максимального угла обзора

Для расчёта максимального угла обзора необходимо знать следующие параметры: текущий режим ширины угла обзора агента, режим качества воспринимаемой им информации, количество тактов до совершения действия и границы максимального угла. В задаче принято, что напарники для паса находятся спереди, то есть $BoundAngle = 180°$, при котором максимальный угол обзора находится в диапазоне $MaxAngle \in [-90°..90°]$.

На основе предоставленного режима ширины угла, рассчитывается текущий угол обзора агента ($ViewAngle$) в градусах, по формуле в [4]. Количество взглядов для захвата максимального угла рассчитывается, как:

$$SightsCountForAngle = \left[\frac{BoundAngel}{2} / ViewAngle + 1\right] \quad (2.1)$$

Далее на основе режима качества воспринимаемой информации по формуле, представленной в [4] вычисляется время для принятия единицы сенсорной информации ($SensorFreq$). Также необходимо вычислить количество сенсорных тактов, которые будут затрачены на получение информации по максимально возможному углу обзора $BoundAngle$, следующим образом:

$$TickMaxAngle = SightsCountForAngle * SensorFreq \quad (2.2)$$

Также необходимо получить количество сенсорных тактов до начала совершения действия агентом. Количество тактов до совершения действия

($TickUntilAction$), рассчитывает другой алгоритм, не рассматриваемый в данной работе. Исходя из описанных выше данных, можно вывести формулу расчёта количества сенсорных тактов до совершения действия:

$$SensorTickUA = \frac{TickUntilAction * TICK}{SensorFreq} \quad (2.3)$$

Где $TICK$ параметр сервера и равен 100 миллисекунд. В конечном итоге становится достаточно данных для расчёта максимального угла обзора при текущем режиме визуального сенсора и оставшемся времени до начала совершения действия:

$$MaxAngle =$$

$$= \begin{cases} \left[\frac{SensorTickUA}{SensorFreq}\right] * viewAngle & TickMaxAngle > SensorTickUA \\ BoundAngle * 2 & TickMaxAngle \leq SensorTickUA \end{cases} \quad (2.4)$$

2.1.2 Модель выбора качества получаемой информации интеллектуального агента

Данная модель рассматривается в контексте метода опережающего итеративного планирования. В первую очередь, необходимо обеспечить алгоритм информацией минимального качества. Таким образом появляется возможность «осмотреться», выполнив при этом обзор максимально возможного угла, рассчитанного выше, за минимальное время.

На рисунке 2.1 представлена модель обзора максимально возможного угла с минимальным качеством. Каждый полученный на данном шаге кадр визуальной информации фильтруется по наличию в ней других агентов и в дальнейшем, отфильтрованные данные, подаются на вход подсистеме оценки ситуации. Рассмотрение данной подсистемы оценки также выходит за рамки настоящей работы. На основе полученной оценки формируется список, состоящих из направлений взглядов, которые представляют интерес для детализации информации.

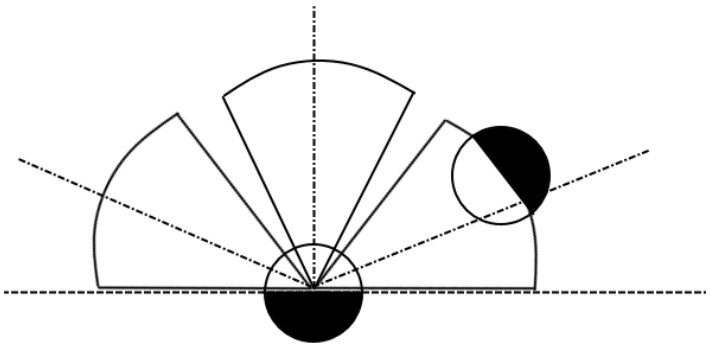

Рисунок 2.1 – Охват угла обзора агента с минимальным качеством

На рисунке 2.2 изображён обзор агентом с фокусировкой в направлении, которые отфильтровала подсистема оценки по значимости ситуации.

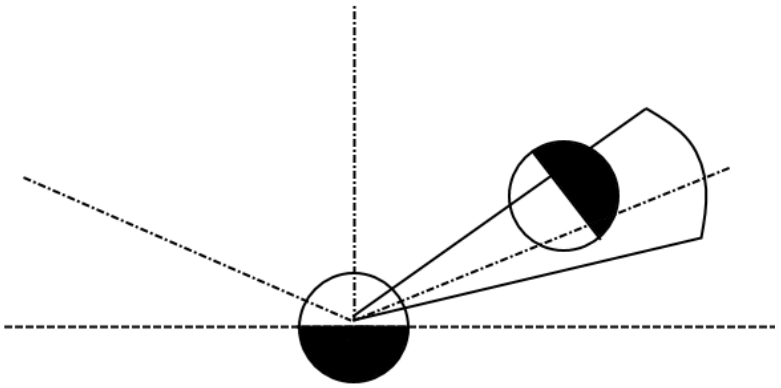

Рисунок 2.2 – Обзор агентом направления с высоким качеством

2.1.3 Модель вычисления допустимого числа тактов без объекта наблюдения

Совершение действия агентом, обычно является результатом влияния на окружающую его на среду, а если точнее, на объект, также существующий в данном мире. В контексте задачи, агенту необходимо взаимодействовать с мячом, а именно сделать удар по мячу в сторону напарника. Для этого агенту

необходимо следить за объектом наблюдения (мячом), чтобы к моменту совершения действия, объект всё ещё был на том месте, где планирует оказаться агент, вследствие появляется потребность не упустить объект из вида. В данной модели рассматривается вычисление количества тактов ($SensorTickWMO$), на которые можно оставить объект наблюдения без «присмотра», для обеспечения большего количества сенсорной информацией – агента. Для расчёта $SensorTickWMO$, необходимы следующие параметры: текущий режим ширины угла обзора, текущий режим качества получаемой информаций, максимальный угол обзора ($MaxAngle$) и количество оставшихся сенсорных тактов до совершения действия ($SensorTickUA$). Данная модель не имеет смысла, если текущий угол обзора больше или равен максимальному углу обзора. Количество направлений взгляда без объекта наблюдений вычисляется по формуле:

$$AnglesWMO = \left[\frac{MaxAngle - ViewAngle}{ViewAngle} + 1\right] \quad (2.5)$$

Формула вычисления количество сенсорных тактов без объекта наблюдения приведена в 2.6

$$SensorTickWMO = SensorFreq * AnglesWMO \quad (2.6)$$

$SensorTickWMO$, не может превышать количество сенсорных тактов до совершения действия, если такая ситуация произошла, то следует приравнять количество тактов без объекта к количеству сенсорных тактов до совершения действия.

2.2 Алгоритмы управления визуальным восприятием

Данная глава затрагивает алгоритмы для расчёта представленных выше моделей и рассматривает проектирование общего алгоритма управления визуальным сенсором интеллектуального агента.

2.2.1 Проектирование алгоритма управления восприятием

Интеллектуальный агент является системой реального времени, учитывая данный факт, необходимо разрабатывать алгоритм способный адаптироваться к миру, который постоянно изменяется. Для решения поставленной задачи, в качестве основы алгоритма, использован метод опережающего итеративного планирования [9].

В общем виде алгоритм представлен на рисунке 2.3. Данный рисунок имеет лишь ознакомительный характер, полный алгоритм управления визуальным сенсором содержит управление дополнительными параметрами, рассмотренные в алгоритмах, представленных ниже. Как было сказано ранее, основными ограничениями алгоритма является время до совершения действия, угол обзора, дальность обзора и качество воспринимаемой визуальной информации. С учётом данных ограничений и учитывая тот факт, что агент является системой реального времени, требуется проектировать алгоритм, способный в любой момент времени выдать решение ограниченной рациональности.

Представленный на рисунке 2.3 алгоритм, на каждой итерации, корректирует параметры агента в соответствии с полученной визуальной информацией от сервера, что соответствует базовому методу опережающего итеративного планирования. Количество тактов до совершения действия предоставляется отдельным алгоритмом, который не входит в рамки данной работы. Далее, необходимо определить режим угла обзора, рассчитать максимальный угол обзора и, на основе количества тактов до совершения действия, необходимо вычислить количество сенсорных тактов с минимальным качеством воспринимаемой визуальной информации. Дополнительно следует рассчитать количество допустимого числа тактов без объекта наблюдения.

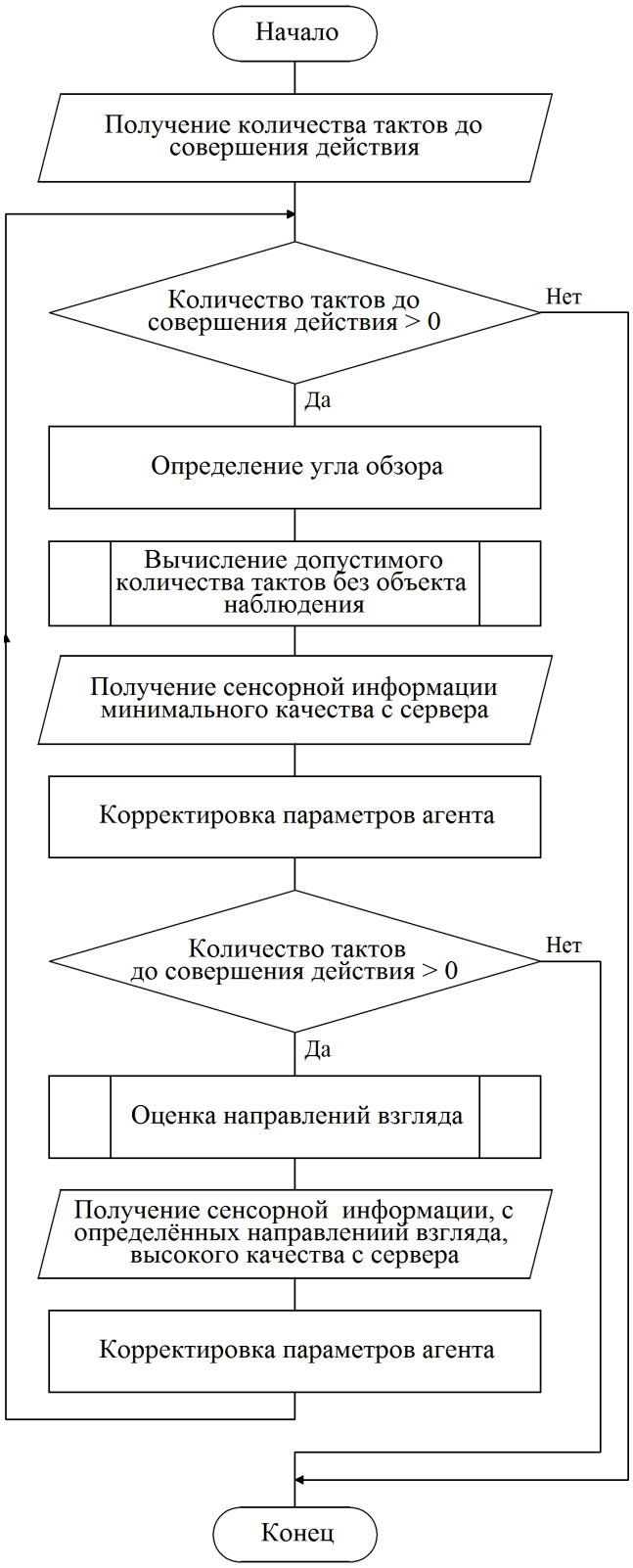

Рисунок 2.3 – Схема алгоритма управления визуальным сенсором в соответствии с методом опережающего итеративного планирования

Исходя из определённого угла обзора, рассчитанного максимального угла обзора и допустимого количества тактов без объекта наблюдения, можно приказать агенту получать информацию минимального качества в данном максимальном диапазоне угла обзора. На основе полученных данных, необходимо скорректировать параметры агента, в нашем случае параметром выступает направление для подачи паса. Следующим шагом необходимо повторно проверить количество оставшихся тактов. Если ресурс времени не исчерпан, то основываясь на полученных данных минимального качества, необходимо оценить направления взглядов, в которых есть другие агенты. Данная визуальная информация минимального качества посылается подсистеме оценки, которая возвращает список, состоящий из представляющих интерес, направлений взглядов α. Подсистема оценки также выходит за рамки настоящей работы, в данном случае она выступает чёрным ящиком, которая принимает визуальную информацию и выдаёт результат. Предпоследним этапом алгоритма является получение информации с высоким качеством, но только в направлениях α. Так как происходит увеличение времени принятия информации из-за режима высокой детализации, рассмотрение только направлений α, экономит временной ресурс агента, позволяя детализировать информацию только в необходимых для принятия решения направлениях. На последнем этапе происходит повторная корректировка параметров агента и переход в начало алгоритма для следующей итерации.

В результате спроектирован алгоритм, который: адаптируется к изменениям в окружающей его среде, рационально расходует отведённый ему ресурс, предоставляет решение ограниченной рациональности в любой момент времени.

2.2.2 Алгоритм расчёта максимального угла обзора

Как было описано в модели, на вход алгоритму необходимо подать границу диапазона максимального угла, текущий режим ширины угла обзора агента и качество получаемой информации.

В представленной, на рисунке 2.4, схеме показан алгоритм вычисления максимального угла в общем виде, основываясь на аргументах, полученных на входе.

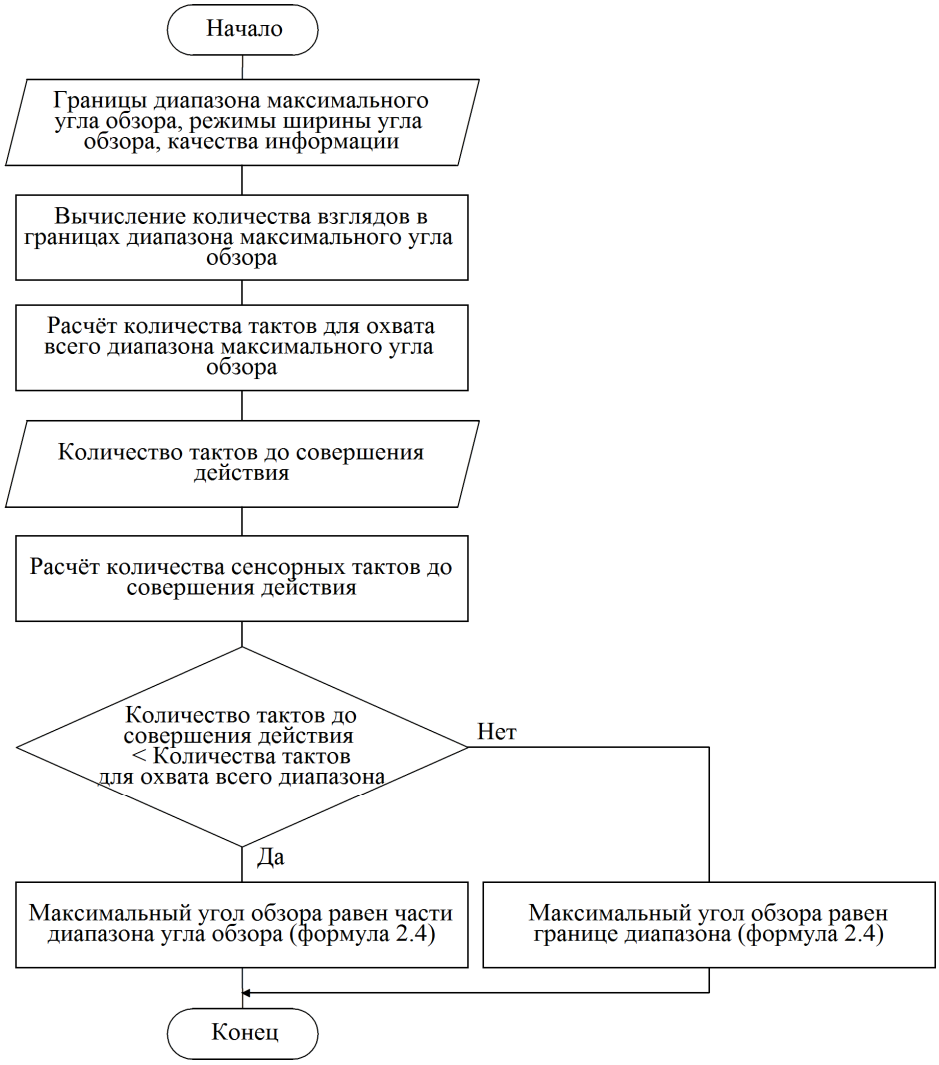

Рисунок 2.4 – Схема алгоритма расчёта максимального угла обзора

Вначале вычисляется количество взглядов, требуемых для полного обзора в границах заданного диапазона. Затем, используя количество взглядов, необходимо рассчитать количество тактов, затрачиваемые на полный обзор всего диапазона. На основе тактов до совершения действий, можно рассчитать количество сенсорных тактов, до совершения действия при заданном режиме

визуального сенсора. На следующем этапе выполняется проверка, если времени достаточно, чтобы охватить весь диапазон максимального угла обзора, дальнейшие действия выполняются в соответствии с формулой 2.4. При достаточном оставшемся времени, происходит полный обзора диапазона максимального угла, в ином случае вычисляется доступные для обзора границы.

2.2.3 Алгоритм адаптации ширины угла обзора

Как было описано в 1.3, у агента имеется 3 режима ширины угла обзора: широкий, нормальный и узкий. Ширина угла обзора, непосредственно влияет на дальность обзора. Данный параметр дальности играет важную роль при обеспечении агента, визуальной информацией низкого качества, когда агенту необходимо выяснить в каких направлениях присутствуют другие объекты на поле.

Исходя из описанной ситуации, возникает необходимость разработать алгоритм, способный адаптировать ширину угла обзора, в том случае, если в текущей конфигурации, при просмотре максимального угла обзора и оставшемся времени, не было получено информации о других объектах.

На первой итерации получения визуальной информации, агенту задаётся режим широкого угла обзора, если агент получил информацию о каком-либо объекте, то общий алгоритм управления восприятием отправляет полученные данные на подсистему оценки. В ином случае, если агент не получил информации, то общий алгоритм управления восприятием не имеет достаточных сведений об окружающем мире, чтобы продолжить работу. Вследствие данного факта, требуется начать алгоритм сначала, но при этом уменьшить ширину угла обзора, чтобы повысить дальность.

Как показано на рисунке 2.5, алгоритм адаптации имеет следующие этапы: выставляется режим ширины угла, вызывается алгоритм получения информации низкого качества и если в поле зрения попали какие-либо объекты, то управление передаётся общему алгоритму управлением восприятия, иначе

требуется выполняется ещё одна итерация алгоритма с другим режимом ширины угла обзора.

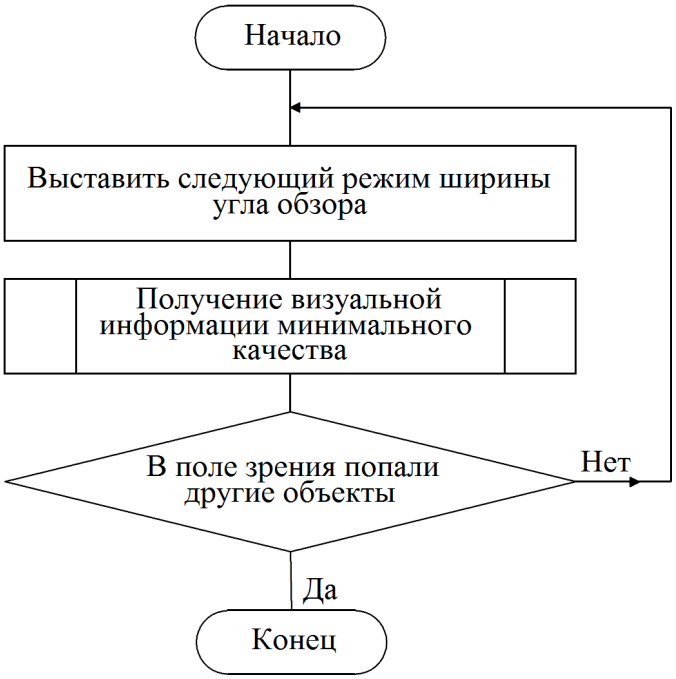

Рисунок 2.5 – Схема алгоритма адаптации ширины угла обзора

Под выражением «следующий режим ширины угла обзора», подразумевается, что режимы упорядочены по ширине от большей к меньшей. В самой первой итерации данного алгоритма выставляется режим с самым широким углом обзора, на следующей итерации он сужается до нормального, если в нормальном режиме также не было получено информации, то выставляется режим с узким углом обзора. В последнем варианте, даже при узком угле обзора, существует вероятность, что агент не получит информацию, тогда на следующей итерации будет выставлено значение нормального угла обзора.

Следует вопрос, почему после узкого угла обзора был выбран режим с нормальным, а не широким углом. В реализации сервера, широкий угол обзора имеет очень маленькую дальность обзора и большое время получения информации, фактически, данный режим выгоден только в том случае, если объекты находятся на очень близком расстоянии, от агента, который осматривает поле.

Если в первой итерации таких объектов не было обнаружено, то за время работы алгоритма, агент получит ту же информацию при режиме с более узким углом обзора. Более того данный режим расходует ресурс времени вдвое больше, чем нормальный режим. На подобранных эвристическим путём, выше указанных основаниях, широкий режим применяется только в первой итерации алгоритма адаптации.

2.2.4 Алгоритм вычисления времени допустимого числа тактов без объекта наблюдения

Объект наблюдения может быть, как статический, так и динамический. При работе со статическими объектами, не имеет смысла постоянно наблюдать их положение на поле, так как оно неизменно. Проблема возникает, когда объект может изменять своё положение в пространстве, в этом случае возникает необходимость постоянно проверять состояние данного объекта, для возможности дальнейшего взаимодействия с ним. Агенту необходим результат данного алгоритма, для оценки, на сколько тактов можно упустить объект из поля зрения, при этом сохранив возможность взаимодействия с данным объектом в рассчитанный момент времени.

На рисунке 2.6 обозначена схема алгоритма, который вычисляет количество сенсорных тактов без объекта наблюдения. На вход алгоритму поступают текущие режимы взгляда агента, рассчитанный ранее максимальный угол обзора и количество тактов до совершения действия. Если текущий угол обзора агента, охватывает весь угол, который необходимо рассмотреть (максимальный угол обзора), то предполагается, что объект всегда находится в поле видимости и количество сенсорных тактов без него равняется нулю. В ином случае происходит вычисление количество тактов в соответствии с формулами, представленными в главе 2.1.3 настоящей работы.

Рисунок 2.6 – Схема алгоритма вычисления количества сенсорных тактов без объекта наблюдения

2.2.5 Алгоритм извлечения визуальной информации

Следующий алгоритм агрегирует и использует вычисленные ранее параметры для непосредственного получения визуальной информации от сервера. Алгоритм спроектирован таким образом, чтобы обобщить принятие информации высокого и низкого качества. Направления взгляда сформированы в виде

списка значений, которые следует просмотреть, в случае получения информации высокого качества, данный список сформирован заранее на основе полученной информации низкого качества, а в случае получения информации низкого качества список формируется на основе текущей ширины угла обзора и рассчитанного максимального угла обзора, который можно просмотреть за отведённое время. Так как агент является системой реального времени и находится в динамическом мире, некоторые положения данного мира могут вносить помехи в получение информации, поэтому каждую просмотренную итерацию алгоритм проверяет осталось ли время для получения следующей единицы сенсорной информации, в случае если он не успеет получить пакет данных, он заканчивает своё выполнение передав полученную информацию для обработки

Рисунок 2.7, даёт представление о том, как работает алгоритм извлечения визуальной информации. В качестве аргументов алгоритму поступают все вычисленные ранее параметры, а также качество получаемой визуальной информации в текущей конфигурации и направления для обзора. Далее вычисляется следующее направление взгляда с поправкой на тот факт, что необходимо «присматривать» за объектом наблюдения, то есть время от времени переводить направление взгляда на него. После вычисления направления взгляда, изменяется конфигурация агента, в частности направление поворота визуального сенсора, а затем происходит извлечение информации с сервера в данном направлении, с заданной шириной угла обзора и качеством воспринимаемой информацией. Далее извлечённые данные преобразовываются в вид, понятный разработанной системе, и затем записываются в структуру данных для отправки этих данных подсистеме оценки. Если были просмотрены не все направления взгляда и осталось время для их просмотра, алгоритм начинается сначала, в ином случае он заканчивается, а записанные данные отправляются подсистеме оценки для вычисления корректировки параметров агента.

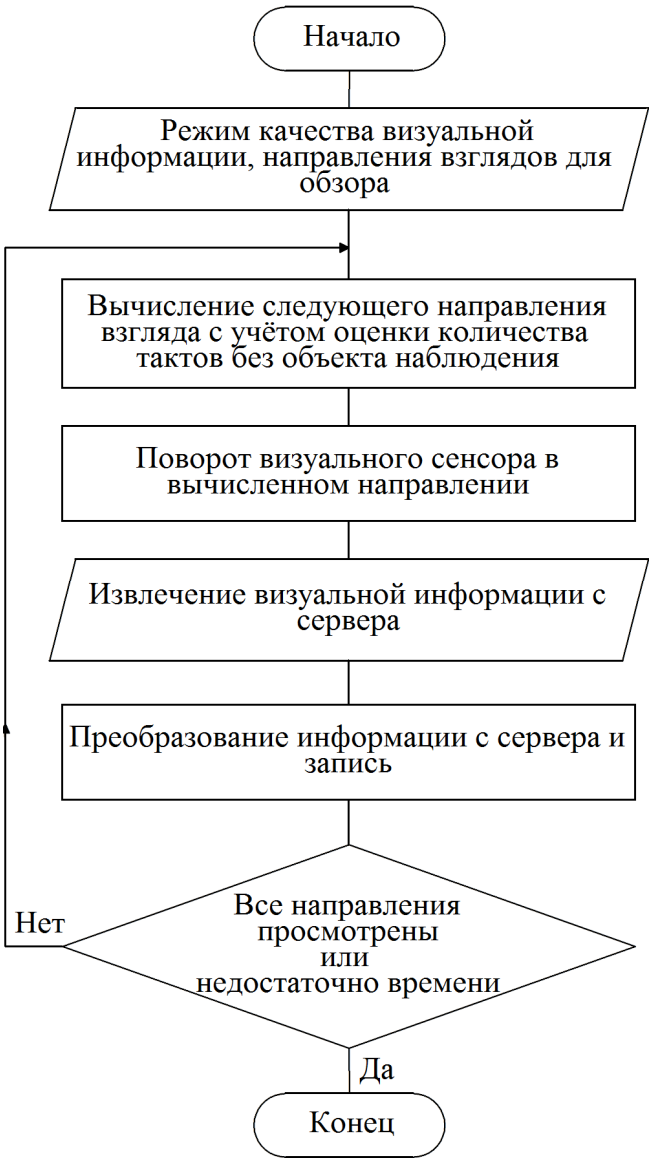

Рисунок 2.7 – Схема алгоритма извлечения визуальной информации с сервера

3 РАЗРАБОТКА И ОТЛАДКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Дипломный проект помимо теоретической части, подразумевает реализацию данных алгоритмов на языке программирования. Ниже представлены особенности реализации классов, интерфейсов и методов. Реализация данного проекта написана на языке программирования Kotlin, который в свою очередь является строготипизированым, способный совмещать объектно-ориентированный и функциональный подход к программированию, вследствие программы имеет меньшую жёсткость связей классов и методов.

3.1 Разработка интеллектуальных агентов для симуляции

В качестве исходной системы, имеется RoboCup Soccer Server, работающий на основе протокола UDP. Для осуществления симуляции необходимо спроектировать и реализовать интеллектуальных агентов таким образом, чтобы каждый из них имел собственное, независимое от других, UDP подключение для отправки команд на сервер и получения сенсорной информации. Все агенты могут отправлять только один набор команд и принимать только один набор сенсорных данных. Реализация всех агентов будет осуществляться с помощью отдельных потоков в одном приложении, так как в этом случае легко переиспользовать код, а также организовывать поведение агентов. При разработке методов отправки некоторых команд на сервер, необходимо учесть временные ограничения на принятие команды сервером и включить данные ограничения в реализацию методов.

Задача симуляционных агентов, обеспечить тестовое окружение для алгоритма управления восприятием. В данной работе рассматривается ситуация, когда агент ведущий мяч, за которым следует противник, желает отдать пас напарнику. Необходимо создание четырёх агентов, два агента будут бегать вокруг и ожидать паса, один агент подающий пас и также один агент противопо-

ложной команды, прикрывающий агента, отдающего пас. Каждый агент должен иметь свою конфигурацию, отвечающую за совершения действий, а также алгоритм выполнения команд. Необходимо предусмотреть возможность одновременного запуска всех агентов.

Помимо реализации самих агентов и классов взаимодействия с сервером, существует потребность в разборе данных пришедших с сервера и создания на их основе классов приложения, в настоящей работе необходим класс, представляющий объект видимого игрока, для проведения оценки ситуации на поле.

3.2 Проектирование архитектуры программной реализации

Используя при написании язык Kotlin, существует возможность совмещать парадигму объектно-ориентированного программирования и функционального. Объектно-ориентированный подход имеет большое преимущество над функциональным, предлагая инкапсуляцию, наследование и полиморфизм, однако функциональный подход также имеет свои преимущества в виде меньшего написания, не несущего смысловой нагрузки, кода. При разработке архитектуры программы с использованием языка Kotlin, следует придерживаться особенностей двух парадигм одновременно. Например, прямое взаимодействие с сервером удобно выделить в отдельный файл с функциями, не создавая при этом класс, т.к. у него нет полей и скрытых методов, вследствие исчезает необходимость создания экземпляра класса сервер, только для использования его методов. В том время, как алгоритм управления восприятием удобно выделить в отдельный класс, так как у него имеются поля и состояния, тем самым возникает необходимость инкапсуляции данных и методов.

3.2.1 Sequence диаграмма

На рисунке 3.1 изображена работа симуляционной системы и агента футболиста, ведущего мяч.

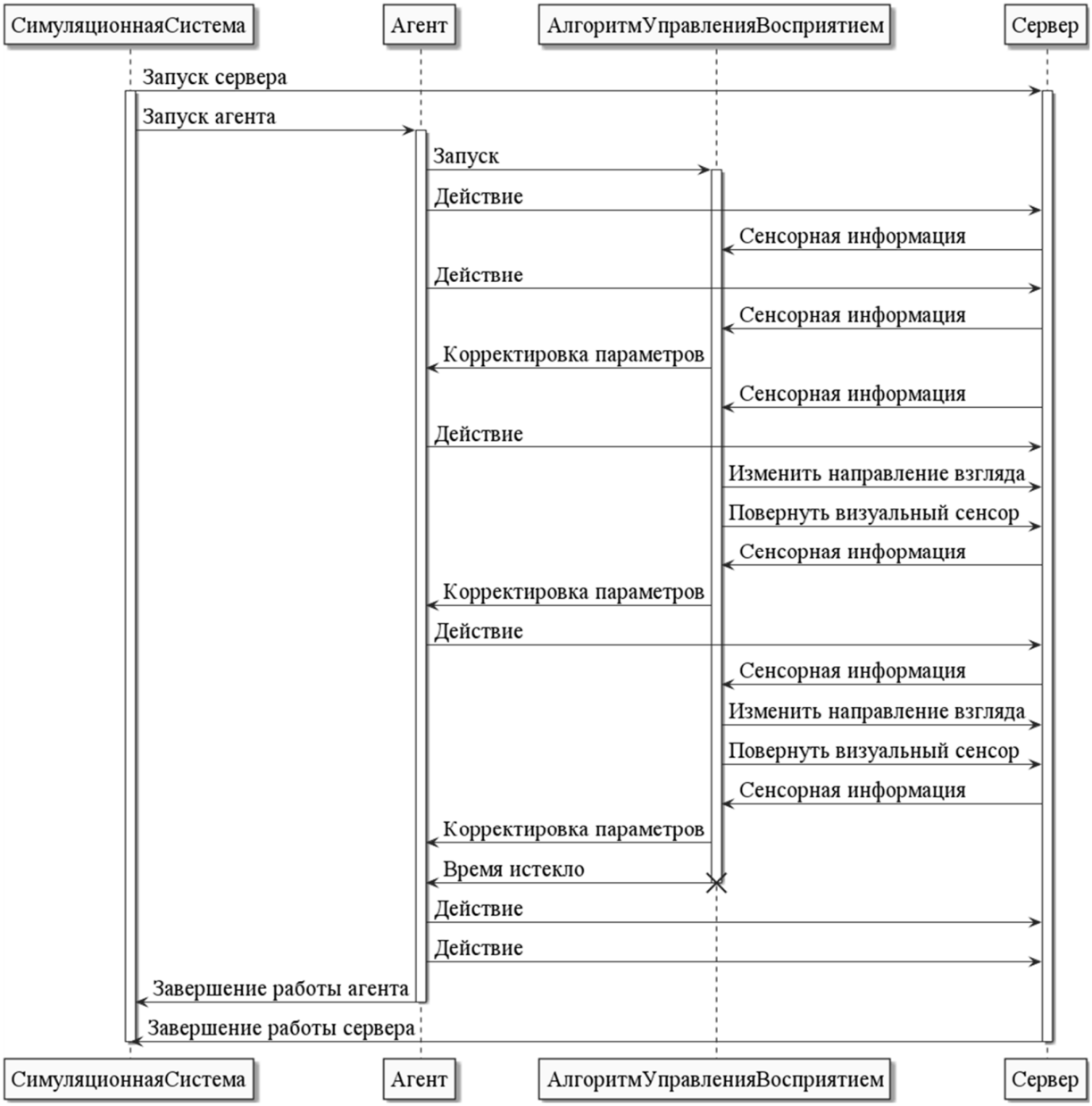

Рисунок 3.1 – Sequence диаграмма проекта

Исходя из представленной диаграммы видно, что симуляционная система запускает сервер, агент стартует после инициализации сервера, а затем агент включает алгоритм управления восприятием, который занимается непосредственно управлением визуальным сенсором агента. Агент параллельно с

алгоритмом управления восприятия может отправлять команды на сервер, однако сенсорная информация поступает в алгоритм управления восприятием и корректирует поведение агента.

3.2.2 Диаграмма классов

Ниже, на рисунке 3.2, представлена диаграмма классов всего проекта.

На диаграмме опущены классы реализаций: AngleServiceImpl, EstimateSubSystemsImp, TickServiceImpl; также были опущены классы констант для экономии места и фокусировки на более важных классах проекта. Некоторые элементы диаграммы имеют в названии суффикс «Kt», он означает, что данный элемент не является классом, а представляет из себя файл с набором функций.

Запускает данный проект класс Main, в нём инициализируется запуск сервера, а затем все агенты. Взаимодействие с сервером происходит с помощью функций, описанных в файле Server.

Классы Def и Attack – содержат конфигурации агентов футболистов, DefActors и AttackActors инициализируют простых независимых агентов на основе конфигурации, которые не отличаются сообразительностью. Kicker – класс агента ведущего мяч, он имеет особое поведение, поэтому он выделен в отдельный класс, в его составе имеется метод возвращающий поведение с включенным в него алгоритмом управления восприятием.

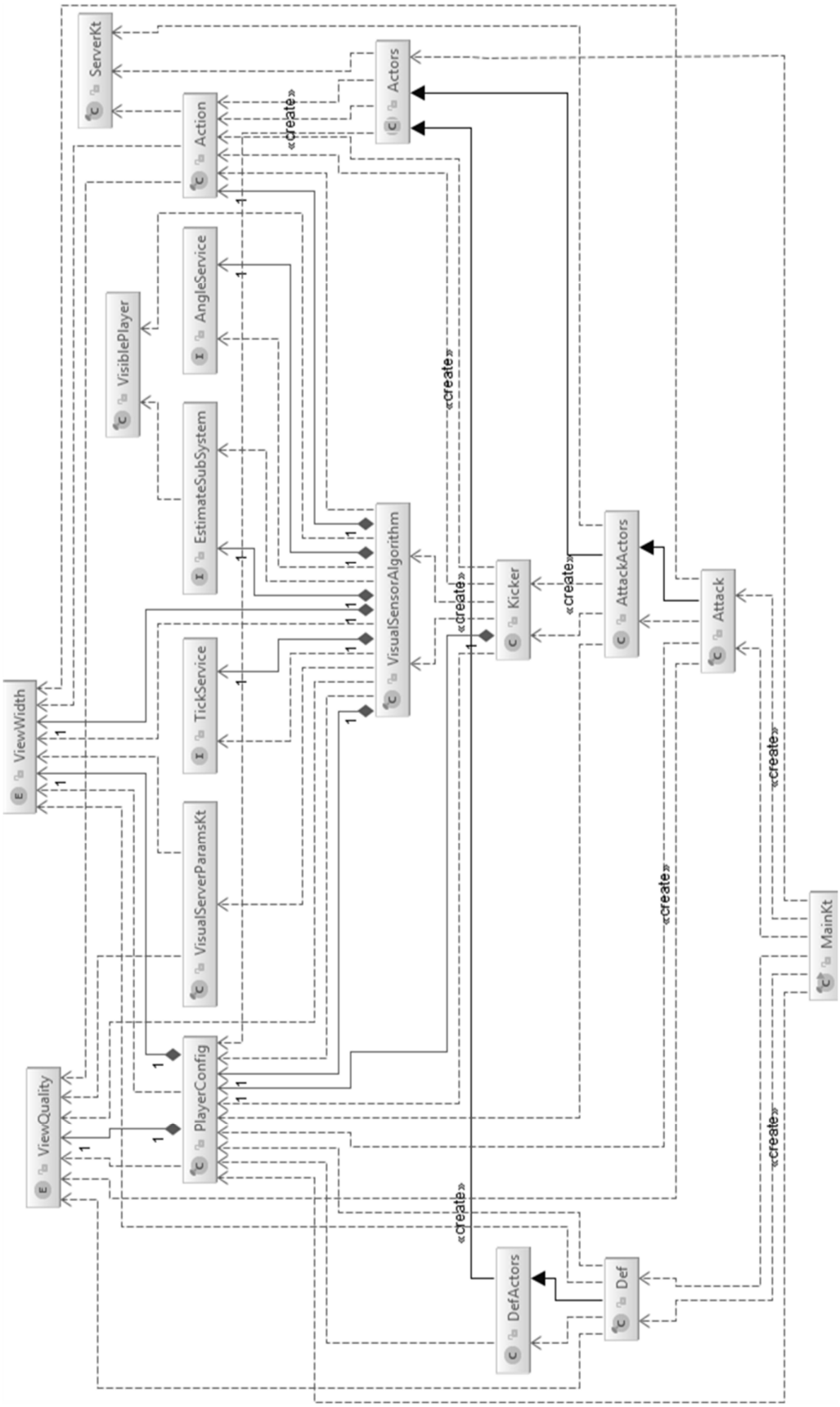

Рисунок 3.2 – Диаграмма классов проекта

3.2.3 Описание интерфейсов

Алгоритму требуются дополнительные сведения и взаимодействия, которые должны быть реализованы либо другим алгоритмом подсистемы восприятия, либо другой подсистемой, например, планирования или оценки. Использованные в работе реализации интерфейсов требуют проведения дополнительных исследований и проектирования алгоритмов, которые выходят за рамки настоящей работы.

В спроектированном алгоритме используется три интерфейса:

- AngleService – в любой момент времени позволяет узнать угол между необходимым наблюдаемым объектом и направлением туловища интеллектуального агента. Метод getBodyObjectAngle непосредственно возвращает целочисленное значение угла в градусах.

Рисунок 3.3 – Интерфейс AngleService

- EstimateSubSystem – интерфейс подсистемы оценки интеллектуального агента. В данной реализации, был необходим метод, оценки ситуации, где на вход подаётся попадавший в поле зрения агент-футболист, а в качестве результата число с плавающей точкой двойной точности, которое является оценкой полезности взгляда в данном направлении.

Рисунок 3.4 – Интерфейс EstimateSubSystem

- TickService – данный интерфейс предоставляет следующие методы: получить количество тактов до начала совершения действия, проверить остались ли такты до начала совершения действия и технический метод позволяющий запустить отсчёт тактов до начала совершения действия.

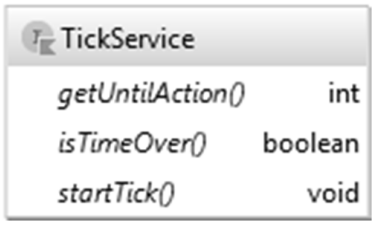

Рисунок 3.5 – Интерфейс TickService

3.3 Реализация алгоритма управления сенсором

На рисунке 3.6 показаны поля и методы реализации класса VisualSensorAlgorithm. Представленный класс на вход получает объект конфигурацию агента, а также объект управления данным агентом, на основе представленных аргументов можно рассчитывать параметры для управления самим агентом, однако данная информация используется только в целях управления восприятием, такие как изменения режимов взгляда, направление взгляда. Для реализации, данный класс использует все выше представленные модели, алгоритмы и реализации интерфейсов. Запуск алгоритма осуществляется с помощью метода *start*, который инициализирует поля начальными параметрами, вычисляет значения полей, по текущим параметрам и начинает отправлять команды на сервер, а также получать сенсорную информацию.

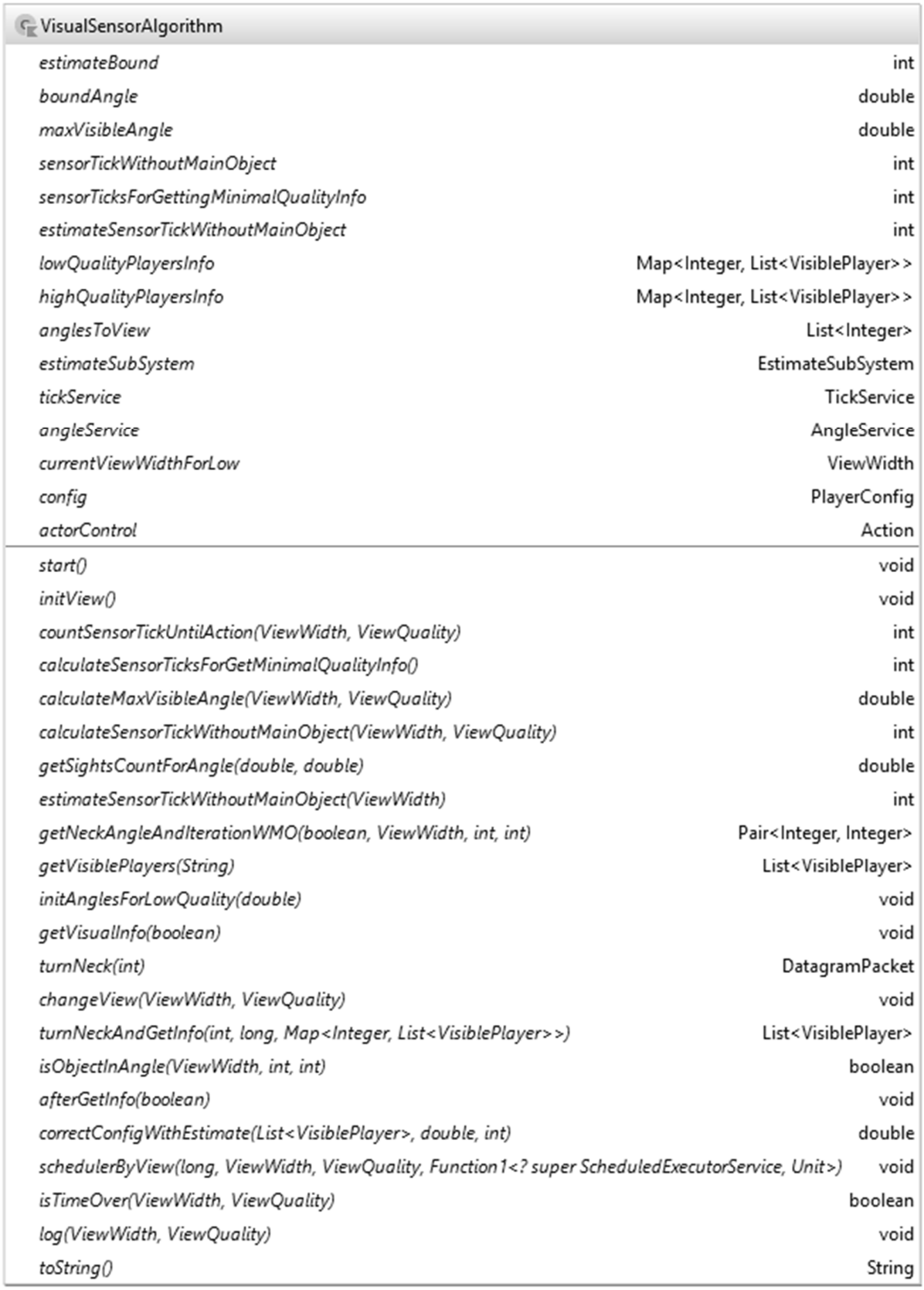

Рисунок 3.6 – Класс VisualSensorAlgorithm

3.3.1 Метод «calculateMaxVisibleAngle»

С помощью данного метода вычисляется максимальный угол обзора агентом, учитывая количество оставшихся сенсорных тактов. На вход данного метода необходимо подать текущий режим ширины угла обзора и качество воспринимаемой информации. В качестве результата, данный алгоритм устанавливает значение максимального угла обзора (*maxVisibleAngle*), доступный данному агенту при заданных условиях. Для вычисления, используется поле класса *boundAngle*, представляющее из себя эвристическое ограничение ширины угла обзора в одну сторону. *boundAngle* обязателен для вычисления количества необходимых взглядов для осмотра максимального угла обзора, при текущем режиме ширины угла обзора. Если времени оказывается достаточно, то в результате используется удвоенное значение данного ограничения.

3.3.2 Метод «calculateSensorTickWithoutMainObject»

Данный метод необходим для вычисления количества тактов, в которые объект наблюдения не попадает в поле зрения агента. Входными параметрами являются режимы ширины угла обзора и качества информации, а также используется поле *maxVisibleAngle*, в котором установлено значение максимального угла обзора доступного агенту. В результате выполнения данного метода устанавливается значение *sensorTickWithoutMainObject*, которое представляет количество сенсорных тактов без объекта наблюдения. Данное значение должно быть скорректировано, так как в зависимости от ситуации необходимо принимать конкретное решение в отношении перевода взгляда на объект наблюдения.

3.3.3 Метод «getVisualInfo»

Описываемый метод используется для непосредственного взаимодействия с сервером. В качестве аргумента поступает логическое значение, обозначающее режим принимаемой информации: *true* – режим высокого качества,

false – низкого. Если выставлен режим высокого качества, то в качестве режима взгляда будет выставлен узкий угол обзора и высокий режим получения качества, в ином случае ширина угла обзора зависит от текущего параметра режима ширины обзора агента, выставленный алгоритмом адаптации ширины угла обзора. Следом за инициализацией режима взгляда, отправляется команда на сервер для изменения параметров агента. Также алгоритм использует список направлений взглядов для получения ситуации на поле. Если режим низкого качества получения информации, то список направлений формируется на основе ограничений максимального угла обзора и текущего режима ширины обзора агента.

Для отправки команд на сервер и получения сенсорной информации, с заданным интервалом и необходимым прерыванием, используется исполнитель *ScheduledThreadPool*, интервал отправки команды вычисляется на основе частоты сенсорных тактов, которые вычисляются по параметрам ширины угла обзора и качества воспринимаемой визуальной информации. Используя параметры, вычисленные до получения сенсорной информации, можно точно вычислить в каких направлениях агенту необходимо смотреть. Исполнитель, отправляющий команды на сервер, заканчивает свою работу только в том случае, если все направления взгляда были просмотрены, с поправкой необходимой для возврата направления взгляда на объект наблюдения, чтобы не упустить его из виду, или в случае если такты для получения сенсорной информации закончились. На протяжении всей работы исполнителя, формируется список с направлениями взглядов и информацией на поле. По завершении работы исполнителя, данный список обрабатывается с использованием подсистемы оценки, на основе данной оценки корректируются параметры агента, а также при низком качества получаемой информации формируется список направлений, которые следует рассмотреть детально.

3.4 Экспериментальное исследование

Алгоритмы были протестированы в рамках поставленной задачи. В ходе проведения исследования выявлены слабые места, такие как неоднозначность в работе сервера, в следствие принимаемой сенсорной информации, а также сильная ограниченность дальности обзора интеллектуального агента.

На рисунке 3.7 представлен пример лога, который выводит алгоритм управления восприятием. Пример полного лога см. приложение Б.

```
VisualSensorAlgorithm(config=PlayerConfig(initialX=-13, initialY=0, initialTurnNeck=0,
                                          kickPower=20, kickDirection=0, dashPower=90,
                                          turnNeck=90, viewWidth=normal, viewQuality=low),
                                    estimateBound=10, boundAngle=90.0, maxVisibleAngle=180.0,
                                    sensorTickWithoutMainObject=0,
                                    sensorTicksForGettingMinimalQualityInfo=3,
                                    estimateSensorTickWithoutMainObject=0, lowQualityPlayersInfo={},
                                    highQualityPlayersInfo={}, anglesToView=[-45, 45],
                                    tickService=TickServiceImpl(tickUntilAction=13))
Frequency for normal low: 75.0
Angle for normal: 90.0
{-45=[VisiblePlayer(teamName=Attack, playerNumber=1, direction=-10, distance=null,
                                    ext=VisiblePlayerExtInfo(distChange=0, dirChange=0,
                                    bodyFacingDir=0, headFacingDir=0), tick=3),
            VisiblePlayer(teamName=Attack, playerNumber=2, direction=17, distance=null,
                                    ext=VisiblePlayerExtInfo(distChange=0, dirChange=0,
                                    bodyFacingDir=0, headFacingDir=0), tick=3)]}
```

Рисунок 3.7 – Пример логгирования информации о текущей ситуации на поле

Несмотря на выявленные слабости, алгоритм демонстрирует должное поведение, предоставляя всю необходимую информацию, для оценки обстановки на поле и принятия решения агентом.

4 РАЗРАБОТКА И СТАНДАРТИЗАЦИЯ ПРОГРАММНЫХ СРЕДСТВ

Дипломный проект является разработкой программного средства и требует дополнительных сведений по части управления им.

4.1 Планирование работ

Перед началом непосредственного проектирования и разработки программного продукта, необходимо провести его планирование. На данном этапе выявляются: общий объём работ, последовательность их выполнения, исполнители, необходимые ресурсы и календарные сроки выполнения работ по проекту [10].

Для формализованного представления планируемых работ воспользуемся диаграммой Ганта.

На рисунке 4.1 представлен план-график выполнения работ по разработке и реализации алгоритмов управления восприятием интеллектуального агента. Данный график наглядно демонстрирует сроки выполнения работ, общий срок выполнения проекта и исполнителей.

Работы	Начало	Конец	Временные периоды																				Исполнители
			Февраль				Март				Апрель				Май				Июнь				
Выдача материалов по интеллектуальным агентам	25.02.2018	01.03.2018																					М. Г. Пантелеев
Освоение материалов, изучение особенностей	01.03.2018	20.03.2018																					А. В. Табаков
Проектирование симуляционной системы	01.03.2018	05.04.2018																					А. В. Табаков
Реализация симуляционной системы	01.04.2018	20.04.2018																					А. В. Табаков
Формализация и постановка задачи	25.04.2018	27.05.2018																					М. Г. Пантелеев, А.В. Табаков
Проектирование моделей управления восприятием	28.04.2018	01.05.2018																					М. Г. Пантелеев, А.В. Табаков
Разработка алгоритмов	01.05.2018	05.05.2018																					А. В. Табаков
Реализация алгоритмов на языке программирования	06.05.2018	09.05.2018																					А. В. Табаков
Интеграция алгоритмов в симуляционную систему	10.05.2018	12.05.2018																					А. В. Табаков
Отладка	13.05.2018	15.05.2018																					А. В. Табаков
Экспериментальное исследование	16.05.2018	20.05.2018																					А. В. Табаков
Разработка технической документации	25.05.2018	03.06.2018																					А. В. Табаков

Рисунок 4.1 – Диаграмма Ганта выполнения работ по проекту

Таблица 4.1 – Планирование работ проекта

Работы	Предшествующие работы	Длительность работ (дни)	События	
			Начальное	Конечное
Выдача материалов по интеллектуальным агентам	-	5	-	Материалы
Освоение материалов, изучение особенностей	Выдача материалов по интеллектуальным агентам	20	Материалы	Понятие области
Проектирование симуляционной системы	Освоение материалов, изучение особенностей	15	Понятие области	Проект системы
Реализация симуляционной системы	Освоение материалов, проектирование системы	20	Проект системы	Реализация системы
Формализация и постановка задачи	Освоение материалов, изучение особенностей	3	Понятие области	Задача
Проектирование моделей управления восприятием	Освоение материалов, постановка задачи	4	Задача	Модели
Разработка алгоритмов	Проектирование моделей управления восприятием	5	Модели	Алгоритмы
Реализация алгоритмов на языке программирования	Разработка алгоритмов	4	Алгоритмы	Реализация алгоритмов
Разработка интерфейсов для алгоритмов	Реализация системы и алгоритмов на языке программирования	1	Реализация системы	Система с алгоритмами
Интеграция алгоритмов в симуляционную систему	Реализация системы и алгоритмов на языке Kotlin	2	Реализация алгоритмов	Система с алгоритмами
Отладка	Интеграция алгоритмов	3	Система с алгоритмами	Отлаженная система
Экспериментальное исследование	Интеграция и отладка алгоритмов	5	Отлаженная система	Результаты исследования
Разработка технической документации	Проектирование, разработка, реализация алгоритмов	10	Результаты исследования	Документация

4.2 Отечественные и международные стандарты

При разработке проекта, одним из важных критериев является выполнение общепринятых требований к оформлению программной документации. Стандарты требований определяют внешний вид таким образом, который позволяет быстро и чётко понимать намерения другого разработчика без непосредственного общения с ним на тему данного проекта.

В следствии данного факта при разработке программной документации и продукта, необходимо руководствоваться принятыми стандартами ГОСТ, ISO, IEC и другими. К программным продуктам и документации относятся ГОСТ серий: 19, 24, 34.

Во время проектирования и разработки программного продукта следует придерживаться стандартов:

- ГОСТ Р ИСО/МЭК 12207-2010 – Процессы жизненного цикла программных средств. Данный стандарт предназначен для представления определённой совокупности процессов, облегчающих связи между приобретающими сторонами, поставщиками и другими правообладателями в течение жизненного цикла программных продуктов.
- ГОСТ Р ИСО/МЭК 15271-2002 – Руководство по применению ГОСТ Р ИСО/МЭК 12207. Стандарт содержит рекомендации по использованию ГОСТ Р ИСО/МЭК 12207.

При написании программной документации необходимо руководствоваться следующими стандартами:

- ГОСТ 19.709-90 – Схемы алгоритмов, программ, данных, систем. В стандарте приведено руководство по условным обозначениям для применения их в схемах данных, программ, работы систем, взаимодействия программ и ресурсов.

- ГОСТ 19.781-90 – Программное обеспечение систем обработки информации. Устанавливает термины и определения понятий в области программного обеспечения систем обработки информации.
- ГОСТ 19.102 – Стадии разработки. Стандарт устанавливает стадии разработки программ и программной документации для вычислительных машин, комплексов и систем независимо от их назначения и области применения.
- ГОСТ 34.003 – Комплекс стандартов на автоматизированные системы. Вводит определение автоматизированная система.
- ГОСТ 34.602-89 – Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы. Распространяется на автоматизированные системы различных видов деятельности. Определяет вид технического задания.
- ГОСТ 24.205-80 – Требования к содержанию документов по информационному обеспечению. Устанавливает требования к содержанию технической документации.
- ГОСТ 24.204-80 – Требования к содержанию документа «Описание постановки задачи».

4.3 Классификация программного продукта

Полной, разработанной системе управления визуальным восприятием агента футболиста, могут быть присвоены общероссийские классификаторы продукции (ОКП и ОКПД). ОКП является устаревшим и в настоящий момент используется классификатор ОКПД, однако имеет смысл указывать оба кода, с целью сравнения данного продукта с продуктами, которым присвоены коды до 2017 года.

Так дипломный проект является программным средством, общего назначения для систем искусственного интеллекта. В соответствии с ОКП, ему присваивается классификатор: (фыва, фыва) 50 2890 Программные средства для

систем искусственного интеллекта прочие. В соответствии с ОКПД, присваивается: 58.29.29.000 Обеспечение программное прикладное прочее на электронном носителе.

4.4 Определение затрат на выполнение и внедрение проекта

Разработка программного средства имеет затраты. Для расчёта затрат на проект, воспользуемся формулами из [10]:

$$К_{ПР} = К_{ПЕРС} + К_{СВТ} + К_{ПРОЧ} \quad (4.1)$$

где:

$К_{ПЕРС}$ – затраты на оплату труда разработчиков;

$К_{СВТ}$ – затраты на средства вычислительной техники;

$К_{ПРОЧ}$ – прочие расходы;

Затраты на оплату труда вычисляются по следующей формуле:

$$К_{ПЕРС} = \sum_{i=1}^{m} Z_{ЗП_i} * (1 + Н + Ф) * d_{ЗАГР_i} * n_i \quad (4.2)$$

где:

m – количество разработчиков;

i – индекс разработчика;

$Z_{ЗПi}$ – заработная плата i-го разработчика;

Н – процент накладных расходов, исчисляемых к сумме зарплаты разработчиков;

Ф – процент отчислений в фонды;

$d_{ЗАГРi}$ – доля загрузки разработчика работой по проекту;

n_i – количество недель в течение которых разработчик занят проектом;

Таким образом, в нашем проекте данные параметры имеют следующие значения: m = 2, Н – 41%, Ф – 30%. Значения заработной платы, доли загрузки и количества недель вычисляются персонально для каждого из разработчиков. В случае руководителя: $Z_{ЗП1}$ = 40000 р./н., $d_{ЗАГР1}$ = 5%, n_1 = 4 недели. В случае

разработчика: $Z_{ЗП2}$ = 30000 р./н., $d_{ЗАГР2}$ = 100%, n_2 = 10 недель. Тогда затраты на заработную плату исполнителей по формуле 4.2 составляют:

$$К_{ПЕРС} = (40000 * 1.71 * 0.05 * 4) + (30000 * 1.71 * 10) = 526680$$

Далее вычислим затраты на средства вычислительной техники и инструментальные программные средства. Для разработки программного обеспечения подобного рода требуется: компьютер, принтер, операционная система Ubuntu Linux, среда разработки, приложение RoboCup Soccer Server. Компьютер и принтер является собственностью разработчика. Операционная система Ubuntu Linux и приложение RoboCup Soccer Server являются бесплатным и открытым программным обеспечением и не требуют затрат. Затраты на компьютер, принтер и среду разработки рассчитываются по следующей формуле:

$$К_{СВТ} = \sum_{j=1}^{k} T_{мвj} * С_{ардj} \text{ [руб.]} \quad (4.3)$$

где:

$Т_{мвj}$ – объём арендуемого машинного времени j-ого СВТ [дни];

$С_{ардj}$ – стоимость машино-дня j-ого СВТ [руб./день];

Объём машинного времени составляет 50 дней, по плану.

Стоимость машино-дня j-ого СВТ может быть рассчитана по следующей формуле:

$$С_{ардj} = \frac{S_{балj}}{T_{слj} * R_{годj}} \text{ [руб./день]} \quad (4.4)$$

где:

$S_{БАЛj}$ – балансовая стоимость j-го СВТ

$Т_{слj}$ – нормативный срок службы

$R_{годj}$ – число рабочих дней в году

В нашем проекте используется:

- компьютер $S_{БАЛ1}$ = 34000 р.
- принтер $S_{БАЛ2}$ = 7000 р.
- среда разработки $S_{БАЛ3}$ = 25000 р.

Сроки службы: $Т_{сл0} = Т_{сл1} = 5$ лет; $Т_{сл1} = 1$ год.

Рабочих дней в году: $R_{год0} = R_{год1} = R_{год2} = 250$.

Тогда арендная плата составляет: $С_{ард0} = 34000/1250 = 27.2$ [руб./день]; $С_{ард1} = 7000/1250 = 5.6$ [руб./день]; $С_{ард2} = 25000/250 = 100$ [руб./день]; Итого, с учётом аренды СВТ независимо от времени реально использования, стоимость составляет 132.8 [руб./день];

$$К_{СВТ} = 50 * 132.8 = 6640 \text{ р.}$$

Прочие расходы отсутствуют.

С помощью формулы 4.1, вычислим себестоимость всего проекта.

$$К_{ПР} = 526680 + 6640 = 533320 \text{ р.}$$

Допустим проект разрабатывается с целью продажи заказчику, тогда можно вычислить цену договора по формуле 4.5

$$Ц_{ДОГ} = К_{пр} * (1 + П) \quad (4.5)$$

где П – доля прибыли. Допустим П = 25%, тогда проект заказчику обойдётся в $Ц_{ДОГ} = 533320 * 1.25 = 666650$ рублей.

Если же проект разрабатывался для тиражирования, то цена экземпляра вычислялась бы по формуле 4.6

$$Ц_{ЭКЗ} = Ц_{ДОГ}/М_{ТИР} \quad (4.6)$$

где $М_{ТИР}$ – гарантируемый тираж экземпляров, предположим, что $М_{ТИР} = 1000$, тогда $Ц_{ЭКЗ} = 666650/1000 = 666.65$ р.

ЗАКЛЮЧЕНИЕ

Исследования интеллектуальных агентов показывает, что существует достаточно много нерешённых задач в области искусственного интеллекта. RoboCup Soccer Simulator позволяет рассмотреть некоторые виды ситуаций, возникающие в реальном времени, а также промоделировать поведение агентов в многоагентной системе.

Разработаны модели и алгоритмы вычисления необходимых параметров для визуального сенсора, а также представлена реализация симуляционной системы с интегрированным, в одного из агентов-футболистов, алгоритма управления визуального восприятия.

Предложенный в настоящей работе общий алгоритм управления визуальным сенсором решает одну из задач получения необходимого количества визуальной информации для принятия решения, если необходимая информация находится впереди агента. Данный алгоритм может быть применён в различных ситуациях, однако в данной работе он рассматривался только в контексте задачи дать пас в направлении агента той же команды.

Дальнейшая разработка данного алгоритма может быть продолжена в нескольких направлениях: увеличение угла обзора, при достаточном времени, до 360°; добавление расчёта параметров по пространственной оси *z*; доработка алгоритма оценки количества сенсорных тактов без определённого объекта наблюдения; абстрагирование данного алгоритма от среды RoboCup Soccer Simulator.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Пантелеев М. Г. Концепция построения интеллектуальных агентов реального времени на основе модели опережающего итеративного планирования // Труды 13-ой Нац. Конф. по ИИ с международным участием КИИ-2012. Т 3. – Белгород: Изд-во БГТУ. – 2012. – С. 25-33.

2. Wooldridge M., Jennings N. R. Intelligent agents: Theory and practice //The knowledge engineering review. – 1995. – Т. 10. – №. 2. – С. 115-152.

3. Пантелеев М. Г., Пузанков Д.В. Интеллектуальные агенты и многоагентные системы. – СПб: Изд-во СПбГЭТУ «ЛЭТИ», 2015, 216 с.

4. Mao Chen, Klaus Dorer. User's Manual: RoboCup Soccer Server, 2003.

5. Remco de Boer, Jelle Kok. The Incremental Development of a Synthetic Multi-Agent System: TheUvA Trilearn 2001 Robotic SoccerSimulation Team, 2002.

6. Vosinakis S., Panayiotopoulos T. Programmable agent perception in intelligent virtual environments //International Workshop on Intelligent Virtual Agents. – Springer, Berlin, Heidelberg, 2003. – С. 202-206.

7. So R., Sonenberg L. The roles of active perception in intelligent agent systems //Pacific Rim International Workshop on Multi-Agents. – Springer, Berlin, Heidelberg, 2005. – С. 139-152.

8. Kuiper D. M., Wenkstern R. Z. Virtual agent perception combination in multi agent based systems //Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems. – International Foundation for Autonomous Agents and Multiagent Systems, 2013. – С. 611-618.

9. Пантелеев М. Г. Формальная модель опережающего итеративного планирования действий интеллектуальных агентов реального времени //Труды. – 2014. – С. 323-334.

10. Фомин В. И. Методические указания по выполнению дополнительного раздела «Разработка и стандартизация программных средств». – СПб: Изд-во СПбГЭТУ «ЛЭТИ», 2017, 41 с.

11. Рассел С., Норвиг П. Искусственный интеллект: современный подход, 2-е изд. // М.: Издат. Дом ≪Вильямс≫, 2006. – 1408 с.

12. Kotlin References - https://kotlinlang.org/docs/reference/

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД АЛГОРИТМА УПРАВЛЕНИЯ ВОСПРИЯТИЕМ ИНТЕЛЛЕКТУАЛЬНОГО АГЕНТА

```
/**
 * Алгоритм управления восприятием на базе метода опережающего итеративного планирования.
 * Содержит вспомогательные функции для расчёта параметров визуального сенсора.
 * @param config - текущая конфигурация агента-футболиста
 * @param actorControl - объект управления агентом
 * */
class VisualSensorAlgorithm(private val config: PlayerConfig, private val actorControl: Action) {
    //Граничная оценка для определения полезности направления обзора
    private val estimateBound = 10
    //Граница угла обзора, используется для определения максимального угла обзора
    private val boundAngle: Double = 90.0
    //Максимальный угол обзора, рассчитывается в calculateMaxVisibleAngle
    private var maxVisibleAngle: Double = 0.0
    //Количество сенсорных тактов без объекта наблюдения,
    //рассчитывается в calculateSensorTickWithoutMainObject
    private var sensorTickWithoutMainObject: Int = 0
    //Количество сенсорных тактов для получения информации низкого качества,
    //рассчитывается в calculateSensorTicksForGetMinimalQualityInfo
    private var sensorTicksForGettingMinimalQualityInfo: Int = 0
    //Количество допустимого количество сенсорных тактов без объекта наблюдения,
    //рассчитывается в estimateSensorTickWithoutMainObject
    private var estimateSensorTickWithoutMainObject: Int = 0
    //Карта для обозначения в каком направлении видны игроки с наблюдением низкого качества.
    //Ключом является направление взгляда агента, а значением -
    //список видимых игроков и их свойства
    private val lowQualityPlayersInfo: MutableMap<Int, List<VisiblePlayer>> = HashMap()
    //Карта для обозначения в каком направлении видны игроки с наблюдением высокого качества.
    //Ключом является направление взгляда агента, а значением -
    //список видимых игроков и их свойства
    private val highQualityPlayersInfo: MutableMap<Int, List<VisiblePlayer>> = HashMap()
    //Текущий список, необходимых для просмотра, направлений взгляда
    private val anglesToView: MutableList<Int> = ArrayList()
```

```
//Текущий режим ширины угла обзора, для получения информации низкого качества.
//Данный параметр изменяется в цикле, если при просмотре поле зрения не было выявлено
//необходимых объектов
private var currentViewWidthForLow: ViewWidth = ViewWidth.NORMAL

//Объект взаимодействия с подсистемой оценки
private val estimateSubSystem: EstimateSubSystem = EstimateSubSystemImpl()
//Объект взаимодействия с системой расчёта тактов
private val tickService: TickService = TickServiceImpl()
//Объект взаимодействия с системой расчёта углов обзора
private val angleService: AngleService = AngleServiceImpl()

/**
 * Начало отсчёта тактов до совершения действия и запуск работы алгоритма управления
 * восприятием.
 */
fun start() {
    tickService.startTick()
    initView()
}

/**
 * Инициализация необходимых для алгоритма переменных, в зависимости
 * от текущей конфигурации алгоритма.
 * Вызывается каждый раз при смене конфигурации алгоритма.
 */
fun initView() {
    calculateMaxVisibleAngle(currentViewWidthForLow, ViewQuality.LOW)
    calculateSensorTicksForGetMinimalQualityInfo()
    calculateSensorTickWithoutMainObject(currentViewWidthForLow, ViewQuality.LOW)
    estimateSensorTickWithoutMainObject(currentViewWidthForLow)
    getVisualInfo(false)
}
```

```
 /**
    * Расчёт количества сенсорных тактов до совершения действия, в зависимости от текущей
    * конфигруации алгоритма.
    *
    * @param viewWidth - режим ширины угла обзора
    * @param viewQuality - режим качества получаемой информации
    * @return количество сенсорных тактов до совершения действия
    */
fun countSensorTickUntilAction(viewWidth: ViewWidth, viewQuality: ViewQuality): Int =
        floor(tickService.getUntilAction() * TICK / getViewFrequency(viewWidth, viewQuality))
            .toInt()

/**
 * Расчёт количества взглядов для охвата полного угла обзора.
 *
 * @param maxAngle - необходимый для обзора угол
 * @param viewAngle - текущий угол обзора агента за один сенсорный такт
 * @return количество необходимых взглядов
 */
private fun getSightsCountForAngle(maxAngle: Double, viewAngle: Double)
        = ceil((maxAngle * 2) / viewAngle)

/**
 * Вычисление максимального угла обзора в зависимости от колчиества сенсорных тактов
 * до совершения действия.
 *
 * @param viewWidth - режим ширины угла обзора
 * @param viewQuality - режим качества получаемой информации
 * @return максимальный угол обзора в градусах
 */
fun calculateMaxVisibleAngle(viewWidth: ViewWidth, viewQuality: ViewQuality): Double {
    val viewAngle = getViewAngle(viewWidth)
    val sightsCountForMaxAngle = getSightsCountForAngle(boundAngle, viewAngle)
    val sensorFrequency = getViewFrequency(viewWidth, viewQuality) / TICK
    //Количество тактов для принятия единицы сенсорной информации в текущем режиме
    //виузального сенсора
    val ticksForSeeMaxAngle = sightsCountForMaxAngle * sensorFrequency
    val sensorTickUntilAction = countSensorTickUntilAction(viewWidth, viewQuality)
    //Если времени недостаточно, для охвата максимально-возможного угла обзора, то
    //угол ширины обзора сужается до допустимого
```

```
    maxVisibleAngle = if (ticksForSeeMaxAngle > sensorTickUntilAction) {
        floor(sensorTickUntilAction / sensorFrequency) * viewAngle
    } else {
        boundAngle * 2.0 // Максимально-возможный угол обзора без поворота туловища
    }
    return maxVisibleAngle
}

/**
 * Вычисление количества сенсорных тактов для получения информации минимального качества,
 * с текущим режимом угла обзора и рассчитанным максимальным углом обзора
 *
 * @return количество сенсорных тактов для получения информации низкого качества
 */
fun calculateSensorTicksForGetMinimalQualityInfo(): Int {
    sensorTicksForGettingMinimalQualityInfo =
            ceil(maxVisibleAngle / getViewAngle(currentViewWidthForLow) *
                    getViewFrequency(currentViewWidthForLow, ViewQuality.LOW) / TICK).toInt() + 1
    return sensorTicksForGettingMinimalQualityInfo
}

/**
 * Вычисление количества тактов без объекта наблюдения для просмотра полного угла обзора.
 *
 * @param viewWidth - режим ширины угла обзора
 * @param viewQuality - режим качества получаемой информации
 * @return количество сенсорных тактов, в которые не попадает объект наблюдения
 */
fun calculateSensorTickWithoutMainObject(viewWidth: ViewWidth, viewQuality: ViewQuality): Int {
    val viewAngle = getViewAngle(viewWidth)
    sensorTickWithoutMainObject = 0
    //Если текущий угол обзора агента больше, чем необходимый полный угол обзора,
    //то количество сенсорных тактов без объекта наблюдения равно 0,
    //иначе происходит вычисление
    if (viewAngle < maxVisibleAngle) {
        //Вычисление количество направлений угла обзора, в которые не попадает объект наблюдения
        val angleWithoutMainObject = ceil((maxVisibleAngle - viewAngle) / viewAngle)
        val sensorFrequency = getViewFrequency(viewWidth, viewQuality) / TICK
        //Вычисление количества тактов без объекта наблюдения
        sensorTickWithoutMainObject = (sensorFrequency * angleWithoutMainObject).toInt()
```

```
        val sensorTickUntilAction = countSensorTickUntilAction(viewWidth, viewQuality)
        //Дополнительная проверка, если количество тактов без объекта наблюдения выходит
        //за рамки отведённого времени, то количество сенсорных тактов без объекта наблюдения
        //становится равным количеству времени, до совершения действия
        if (sensorTickWithoutMainObject > sensorTickUntilAction) {
            sensorTickWithoutMainObject = sensorTickUntilAction
        }
    }
    return sensorTickWithoutMainObject
}

/**
 * Оценка количества сенсорных тактов без объекта наблюдения.
 *
 * @param viewWidth - режим ширины угла обзора
 * @return количество допустимых сенсорных тактов,
 * в которые может не попадать объект наблюдения
 */
fun estimateSensorTickWithoutMainObject(viewWidth: ViewWidth): Int {
    estimateSensorTickWithoutMainObject = 0
    //Оценка вычисляется, только если существует необходимость отводить взгляд от объекта
    //наблюдения
    if (sensorTickWithoutMainObject > 0) {
        val viewAngle = getViewAngle(viewWidth)
        estimateSensorTickWithoutMainObject = floor(maxVisibleAngle / viewAngle / 2).toInt()
    }
    return estimateSensorTickWithoutMainObject
}

/**
 * Определение попадает ли объект наблюдения в текущий угол обзора.
 *
 * @param viewWidth - режим ширины угла обзора
 * @param objectBodyAngle - угол между направлением туловища агента и объектом наблюдения
 * @param direction - направление визуального сенсора в градусах относительно туловища
 *
 * @return true - если объект попадает в поле зрения агента, false - в ином случае.
 */
private fun isObjectInAngle(viewWidth: ViewWidth, objectBodyAngle: Int, direction: Int):
        Boolean {
```

```
    val halfSeeAngle = getViewAngle(viewWidth) / 2 //Вычисление половины угла обзора агента
    return objectBodyAngle > (halfSeeAngle - direction)
            && objectBodyAngle < (halfSeeAngle + direction)
}

/**
 * Вычисление необходимого следующего угла поворота визуального сенсора агента,
 * в соответствии с количеством тактов необходимых для присмотра за наблюдаемым объектом.
 * @param viewWidth - режим ширины угла обзора
 * @param turnNeckOnce - определяет требуется ли единственный поворот сенсора
 * @param neckAngle - текущий поворот визуального сенсора в градусах
 * @param iterationWithoutMainObject - количество тактов без объекта наблюдения
 * @return угол поворота сенсора в градусах и количество итераций без объекта наблюдения
 */
private fun getNeckAngleAndIterationWMO(turnNeckOnce: Boolean, viewWidth: ViewWidth,
                                        neckAngle: Int, iterationWithoutMainObject: Int):
        Pair<Int, Int> {
    var localNeckAngle = neckAngle
    var localIterationWithoutMainObject = iterationWithoutMainObject
    //Если поворот сенсоров не единственный и есть необходимость отводить направление взгляда
    //от объекта наблюдения, то выполняется расчёт
    if (!turnNeckOnce && estimateSensorTickWithoutMainObject > 0) {
        //Вычисление только в том случае, если объект наблюдения не попадает в поле зрения агента
        if (!isObjectInAngle(viewWidth, angleService.getBodyObjectAngle(), localNeckAngle)) {
            if (estimateSensorTickWithoutMainObject == localIterationWithoutMainObject) {
                //направить взгляд в сторону объекта
                localNeckAngle = angleService.getBodyObjectAngle()
                localIterationWithoutMainObject = 0
            } else {
                ++localIterationWithoutMainObject
            }
        } else {
            localIterationWithoutMainObject = 0 //объект попадает в поле зрения
        }
    }
    return Pair(localNeckAngle, localIterationWithoutMainObject)
}

/**
 * Вспомогательный метод для парсинга сообщений с сервера.
```

```
 *
 * @param serverMessage - Сообщение от сервера
 */
private fun getVisiblePlayers(serverMessage: String): List<VisiblePlayer> {
    var vp: List<VisiblePlayer> = ArrayList()
    if (!serverMessage.contains("warning", true)
            && !serverMessage.contains("error", true)) {
        if (serverMessage.contains("(p \"", true)) {
            vp = parseVisiblePlayers(serverMessage)
        }
    }
    return vp
}

/**
 * Инициализация углов обзора для получения информации низкого качества
 * на основе максимального угла обзора.
 *
 * @param viewAngle - текущий угол обзора, по умолчанию равен граничному углу обзора
 */
private fun initAnglesForLowQuality(viewAngle: Double = boundAngle) {
    anglesToView.clear() //Очистка текущих направлений взгляда
    var neckAngle = (-(maxVisibleAngle / 2) + (viewAngle / 2)).toInt() //Начальное направление
    //Добавление всех направлений в пределах максимального угла обзора
    while (neckAngle < (maxVisibleAngle / 2)) {
        anglesToView.add(neckAngle)
        neckAngle += viewAngle.toInt()
    }
}

/**
 * Изменение направления сенсора
 *
 * @param turnNeck - угол для поворота сенсора в градусах
 */
private fun turnNeck(turnNeck: Int): DatagramPacket {
    config.turnNeck = turnNeck
    return actorControl.turnNeck(turnNeck)
}
```

```
/**
 * Изменение режима взгляда агента.
 *
 * @param viewWidth - режим ширины угла обзора
 * @param viewQuality - режим качества получаемой информации
 */
private fun changeView(viewWidth: ViewWidth, viewQuality: ViewQuality) {
    actorControl.changeView(viewWidth, viewQuality)
    config.viewWidth = viewWidth
    config.viewQuality = viewQuality
}

/**
 * Агрегирующий метод. В нём выполняется поворот сенсора и парсинг информации с сервера
 *
 * @param angle - угол направления взгляда
 * @param scheduleFrequency - время в мс. необходимое для принятия информации с сервера
 * @param qualityMap - структура для хранения направления взгляда и списка видимых агентов
 * @return список видимых агентов на поле
 */
private fun turnNeckAndGetInfo(angle: Int, scheduleFrequency: Long,
                               qualityMap: MutableMap<Int, List<VisiblePlayer>>):
        List<VisiblePlayer> {
    var vp: List<VisiblePlayer> = ArrayList()

    var message = getServerMessage(turnNeck(angle))
    var isSee = isMessageSee(message)

    if ((isSee && vp.isEmpty())) {
        Thread.sleep(scheduleFrequency)
        message = getServerMessage(actorControl.receive())
        isSee = isMessageSee(message)
    }

    if (isSee) {
        vp = getVisiblePlayers(message)
    }

    if (vp.isNotEmpty()) {
        qualityMap[angle] = vp
```

```
    }
    return vp
}

/**
 * Извлечение сенсорной информации агента, по рассчитанный параметрам.
 *
 * @param isHigh - режим качества получаемой информации: true - высокое, false - низкое
 */
private fun getVisualInfo(isHigh: Boolean) {
    val viewWidth: ViewWidth
    val viewQuality: ViewQuality
    if (isHigh) { //Выставление параметров извелчение информации в зависимости от режима
        viewWidth = ViewWidth.NARROW
        viewQuality = ViewQuality.HIGH
    } else {
        //Текущий режим ширины угла обзора, изменяется при работе алгоритма
        viewWidth = currentViewWidthForLow
        viewQuality = ViewQuality.LOW
        initAnglesForLowQuality(getViewAngle(viewWidth))
    }
    changeView(viewWidth, viewQuality) //Отправка команды серверу об изменении режима взгляда

    val turnNeckOnce = anglesToView.size == 1 //Требуется ли повернуть сенсор только один раз

    val scheduleFrequency = getViewFrequency(viewWidth, viewQuality).toLong()
    var sensorTickCounter = 0
    //Количество просмотренных направлений взгляда,
    //является индексом в списке направлений для просмотра
    var anglesCounter = 0
    var allAnglesChecked = false //Определяет все ли направления взгляда просмотрены
    var iterationWithoutMainObject = 0 //Количество итераций без объекта наблюдения
    schedulerByView(0, viewWidth, viewQuality) { scheduler ->
        var neckAngle = anglesToView[anglesCounter] //Текущий угол поворота сенсора

        val (angle, iteration) = getNeckAngleAndIterationWMO(turnNeckOnce, viewWidth, neckAngle,
                iterationWithoutMainObject)
        iterationWithoutMainObject = iteration
        neckAngle = angle
```

```
        //Если потребовалось повернуть сенсор в сторону объекта наблюдения
        if (neckAngle != anglesToView[anglesCounter]) {
            --anglesCounter
        }

        //Поворот сенсора и извлечение информации
        turnNeckAndGetInfo(neckAngle, scheduleFrequency,
                if (isHigh) highQualityPlayersInfo else lowQualityPlayersInfo)

        //Если все направления просмотрены или время для просмотра истекло
        if (allAnglesChecked ||
                (!isHigh && sensorTickCounter == sensorTicksForGettingMinimalQualityInfo)) {
            afterGetInfo(isHigh) //Обработка полученных результатов
            scheduler.shutdown() //Завершение цикла просмотра
        }

        anglesCounter = (anglesCounter + 1) % anglesToView.size //Выбор следующего угла обзора
        if (anglesCounter == 0) {
            allAnglesChecked = true
        }
        ++sensorTickCounter //Количество потраченных сенсорных тактов
    }
}

/**
 * Обработка и оценка полученной информации с сервера.
 *
 * @param isHigh - режим качества получаемой информации: true - высокое, false - низкое
 */
private fun afterGetInfo(isHigh: Boolean) {
    println(if (isHigh) highQualityPlayersInfo else lowQualityPlayersInfo)
    //Структура для обработки информации
    val mapForEstimate = if (isHigh) highQualityPlayersInfo else lowQualityPlayersInfo

    //Если объекты в данном режиме не были найдены
    if (mapForEstimate.isEmpty()) {
        if (!isHigh) { //Если режим извлечения информации выставлен в низкое качество
            //Определение следующего режима ширины угла обзора
            currentViewWidthForLow = when (currentViewWidthForLow) {
                ViewWidth.WIDE -> ViewWidth.NORMAL
```

```
                ViewWidth.NORMAL -> ViewWidth.NARROW
                ViewWidth.NARROW -> ViewWidth.NORMAL
            }

            initView() //Пересчёт параметров алгоритма с новой конфигурацией
        } else {
            getVisualInfo(false)
        }

        return
    }

    anglesToView.clear() //Очистка списка направлений взгляда для просмотра
    var averageEstimate = 0.0
    //Оценка полученной информации с помощью подсистемы оценки
    mapForEstimate.forEach { angle, visiblePlayers ->
        averageEstimate = correctConfigWithEstimate(visiblePlayers, averageEstimate, angle)
        //Если данное направление представляет интерес для дальнейшего просмотра,
        //то добавить в список направлений для следующего просмотра
        if (!isHigh && averageEstimate > estimateBound) {
            anglesToView.add(angle)
        }
    }

    getVisualInfo(!isHigh)
}

/**
 * Корректировка параметров агента в соответствии с полученной информацией
 * об обстановке на поле.
 *
 * @param visiblePlayers - список видимых агентов
 * @param estimateIn - текущая оценка лучшего результата
 * @param angle - направление взгляда
 * @return оценка лучшего результата
 */
private fun correctConfigWithEstimate(visiblePlayers: List<VisiblePlayer>,
                                      estimateIn: Double, angle: Int): Double {
    var estimate = estimateIn
    val averageEstimate = visiblePlayers.map { estimateSubSystem.forVisiblePlayer(it) }
```

```
            .average() //Средняя оценка для всех видимых агентов в заданном направлении
    //Если расчитанная средняя оценка лучше текущей, то корректировка параметров агента
    if (averageEstimate > estimate) {
        config.kickDirection = angle
        estimate = averageEstimate
    }
    println("Angle: $angle, Estimate: $averageEstimate")
    return estimate
}

/**
 * Инициализация планировщика задач для просмотра в заданных направлениях.
 *
 * @param initialDelay - начальная задержка на запуск планировщика
 * @param viewWidth - режим ширины угла обзора
 * @param viewQuality - режим качества получаемой информации
 * @param run - задача для выполнения
 */
private fun schedulerByView(initialDelay: Long, viewWidth: ViewWidth,
                            viewQuality: ViewQuality,
                            run: (scheduler: ScheduledExecutorService) -> Unit) {
    //Если время закончилось, то планировщик не требуется
    if (isTimeOver(viewWidth, viewQuality)) {
        return
    }
    log(viewWidth, viewQuality) //Логгирование текущего режима просмотра
    val scheduler = Executors.newScheduledThreadPool(1)
    val scheduleFreq = getViewFrequency(viewWidth, viewQuality).toLong()
    scheduler.scheduleAtFixedRate({
        run(scheduler)
        //Завершение цикла просмотра если время закончилось
        if (isTimeOver(viewWidth, viewQuality)) {
            scheduler.shutdown()
        }
    }, initialDelay, scheduleFreq, TimeUnit.MILLISECONDS)
}

/**
 * Определение осталось ли время для извлечения информации с сервера
 *
```

```
     * @param viewWidth - режим ширины угла обзора
     * @param viewQuality - режим качества получаемой информации
     * @return true - время вышло, false - время для просмотра достаточно
     */
    private fun isTimeOver(viewWidth: ViewWidth, viewQuality: ViewQuality): Boolean
            = countSensorTickUntilAction(viewWidth, viewQuality) < 1

    /**
     * Логгирование режима просмотра
     */
    private fun log(viewWidth: ViewWidth, viewQuality: ViewQuality) {
        println(this)
        println("Frequency for $viewWidth $viewQuality: " +
                "${getViewFrequency(viewWidth, viewQuality)}")
        println("Angle for $viewWidth: ${getViewAngle(viewWidth)}")
    }

    /**
     * Вспомогательная функция для логгирования режима просмотра
     */
    override fun toString(): String {
        return "VisualSensorAlgorithm(config=$config, estimateBound=$estimateBound, " +
                "boundAngle=$boundAngle, maxVisibleAngle=$maxVisibleAngle, " +
                "sensorTickWithoutMainObject=$sensorTickWithoutMainObject, " +
                "sensorTicksForGettingMinimalQualityInfo=$sensorTicksForGettingMinimalQualityInfo, " +
                "estimateSensorTickWithoutMainObject=$estimateSensorTickWithoutMainObject, " +
                "lowQualityPlayersInfo=$lowQualityPlayersInfo, " +
                "highQualityPlayersInfo=$highQualityPlayersInfo, " +
                "anglesToView=$anglesToView, tickService=$tickService"
    }
}
```

ПРИЛОЖЕНИЕ Б

ЛОГ РАБОТЫ АЛГОРИТМА УПРАВЛЕНИЯ ВОСПРИЯТИЕМ ИНТЕЛЛЕКТУАЛЬНОГО АГЕНТА

```
VisualSensorAlgorithm(config=PlayerConfig(initialX=-13, initialY=0, initialTurnNeck=0,
                                          kickPower=20, kickDirection=0, dashPower=90,
                                          turnNeck=90, viewWidth=normal, viewQuality=low),
                      estimateBound=10, boundAngle=90.0, maxVisibleAngle=180.0,
                      sensorTickWithoutMainObject=0,
                      sensorTicksForGettingMinimalQualityInfo=3,
                      estimateSensorTickWithoutMainObject=0, lowQualityPlayersInfo={},
                      highQualityPlayersInfo={}, anglesToView=[-45, 45],
                      tickService=TickServiceImpl(tickUntilAction=13))
Frequency for normal low: 75.0
Angle for normal: 90.0
{-45=[VisiblePlayer(teamName=Attack, playerNumber=1, direction=-10, distance=null,
                      ext=VisiblePlayerExtInfo(distChange=0, dirChange=0,
                      bodyFacingDir=0, headFacingDir=0), tick=3),
        VisiblePlayer(teamName=Attack, playerNumber=2, direction=17, distance=null,
                      ext=VisiblePlayerExtInfo(distChange=0, dirChange=0,
                      bodyFacingDir=0, headFacingDir=0), tick=3)]}
Angle: -45, Estimate: 22.0

VisualSensorAlgorithm(config=PlayerConfig(initialX=-13, initialY=0, initialTurnNeck=0,
                                          kickPower=20, kickDirection=-45, dashPower=90,
                                          turnNeck=-45, viewWidth=narrow, viewQuality=high),
                      estimateBound=10, boundAngle=90.0, maxVisibleAngle=180.0,
                      sensorTickWithoutMainObject=0,
                      sensorTicksForGettingMinimalQualityInfo=3,
                      estimateSensorTickWithoutMainObject=0,
                      lowQualityPlayersInfo={-45=[VisiblePlayer(teamName=Attack, playerNumber=1,
                                                  direction=-10, distance=null,
                                                  ext=VisiblePlayerExtInfo(distChange=0,
                                                                           dirChange=0,
                                                                           bodyFacingDir=0,
                                                                           headFacingDir=0),
                                                  tick=3),
                                                VisiblePlayer(teamName=Attack, playerNumber=2,
                                                  direction=17, distance=null,
                                                  ext=VisiblePlayerExtInfo(distChange=0,
                                                                           dirChange=0,
                                                                           bodyFacingDir=0,
```

```
                                                                headFacingDir=0),
                                                        tick=3)]},
                        highQualityPlayersInfo={}, anglesToView=[-45],
                        tickService=TickServiceImpl(tickUntilAction=10))
Frequency for narrow high: 75.0
Angle for narrow: 45.0
{}

VisualSensorAlgorithm(config=PlayerConfig(initialX=-13, initialY=0, initialTurnNeck=0,
                                          kickPower=20, kickDirection=-45, dashPower=90,
                                          turnNeck=-45, viewWidth=normal, viewQuality=low),
                        estimateBound=10, boundAngle=90.0, maxVisibleAngle=180.0,
                        sensorTickWithoutMainObject=0,
                        sensorTicksForGettingMinimalQualityInfo=3,
                        estimateSensorTickWithoutMainObject=0,
                        lowQualityPlayersInfo={-45=[VisiblePlayer(teamName=Attack, playerNumber=1,
                                                        direction=-10, distance=null,
                                                        ext=VisiblePlayerExtInfo(distChange=0,
                                                                dirChange=0,
                                                                bodyFacingDir=0,
                                                                headFacingDir=0),
                                                        tick=3),
                                                VisiblePlayer(teamName=Attack, playerNumber=2,
                                                        direction=17, distance=null,
                                                        ext=VisiblePlayerExtInfo(distChange=0,
                                                                dirChange=0,
                                                                bodyFacingDir=0,
                                                                headFacingDir=0),
                                                        tick=3)]},
                        highQualityPlayersInfo={}, anglesToView=[-45, 45],
                        tickService=TickServiceImpl(tickUntilAction=9))
Frequency for normal low: 75.0
Angle for normal: 90.0
{-45=[VisiblePlayer(teamName=Attack, playerNumber=1, direction=-10, distance=null,
                        ext=VisiblePlayerExtInfo(distChange=0, dirChange=0,
                        bodyFacingDir=0, headFacingDir=0), tick=3),
        VisiblePlayer(teamName=Attack, playerNumber=2, direction=17, distance=null,
                        ext=VisiblePlayerExtInfo(distChange=0, dirChange=0,
                        bodyFacingDir=0, headFacingDir=0), tick=3)]}
Angle: -45, Estimate: 22.0

Turn Neck Moment: -45 | Kick Direction: -45
```