Содержание

Введение

Актуальность темы

На данный момент основными способами разработки искусственного интеллекта для различных игр является конечный автомат и дерево вариантов [10]. С развитием науки и увеличением мощности вычислительной техники стало популярным машинное обучение, в частности нейронные сети. Стандартные способы обучения плохо подходят для разработки искусственного интеллекта из-за необходимости составлять обучающую выборку, что является трудоёмкой задачей для больших игр. Для таких задач отлично подходит метод обучения с подкреплением.

Создание самообучающихся алгоритмов на основе Reinforcement Learning (Обучение с подкреплением) является актуальной задачей в современном мире, поскольку данный метод работает без заранее подготовленных ответов. Эта способность данного метода открывает возможности для решения нового вида задач. В частности, этот метод используется для обучения компьютеров прохождению игр. Создание подобных самообучающихся алгоритмов имеет важное практическое и научное значение, поскольку такие алгоритмы могут послужить основой для дальнейших исследований и совершенствований для создания алгоритмов, решающих реальные задачи.

Цели и задачи работы

Целью данной работы является разработка игры «Монополия» с использованием нейросетевых технологий.

Основные задачами работы следующие:

1. Изучение методов обучения нейронных сетей
2. Обзор используемых технологий
3. Разработка игры «Монополия»
4. Реализация нейронной сети для разработанной игры
5. Тестирование полученной программы

1 Анализ предметной области

1.1 Анализ аналогичных проектов

В основе нейросетевых технологий лежит принцип искусственных нейронных сетей. Искусственная нейронная сеть - математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма. Нейронные сети не программируются в привычном смысле этого слова, они обучаются. Возможность обучения — одно из главных преимуществ нейронных сетей перед традиционными алгоритмами. Технически обучение заключается в нахождении коэффициентов связей между нейронами[8]. Однако существует множество способов обучения нейронных сетей.

Так, в одной статье рассматривается модель нейронной сети, имитирующей логику противника в игре «крестики-нолики»[1]. Структура разработанной сети состоит из одного входного нейрона, промежуточного слоя, состоящего из девяти нейронов, каждый из которых отвечает за клетку на поле, и выходного нейрона. Каждый промежуточный нейрон имеет свое весовое значение в пределах от - 1 до 1, от которого зависит выбор клетки для хода. Сеть оценивает каждую свободную клетку и выбирает наиболее перспективную. Обучающая выборка состоит из наборов заранее записанных ходов против сети. Нейронная сеть играет против каждого набора до тех пор, пока не выиграет. Процесс обучения состоит в том, что перед началом матча нейронная сеть генерирует значения весов. Исходя из этих значений и свободных клеток, сеть делает свои ходы, если она проигрывает игру значения весов, которые соответствуют произведенным ходам, уменьшаются. Если сеть выигрывает игру, значения весов увеличиваются, и игра записывается в память. Если сеть играет в ничью, то значения весов не изменяются, игра записывается в память, но дополнительно играет еще три раза с новыми значениями весов, для проверки возможности других комбинаций ходов. Процесс выбора связан с обращением сети к ее памяти значений весов. Находится игра из памяти, где противником были сделаны ходы, сделанные в текущей иг-

ре, но сеть при этом выиграла. Выбирается наиболее приближенное к единице значение весов, которое затем конвертируется в клетку. Автор отмечает что, если оппонент специально не поддаётся сети, то исходом матчей будет являться ничья, как и в случае, если нейронная сеть будет играть против такой же нейронной сети.

В другой статье рассматривается реализация поведения агента в мультиагентной компьютерной игре[4]. Игра “Collector Stars” представляет собой двумерную игру в жанре платформер для одного или нескольких игроков. Игра представлена двумерной сеткой ячеек разного типа: проходимая, непроходимый блок, звезда, черная дыра и дверь. Персонажу Собирателю нужно найти выход из уровня, попутно собирая звезды и избегая черных дыр. Агент-собиратель может перемещаться влево или вправо и прыгать на высоту, чуть выше, чем высота одной ячейки карты. Имеет меньшие размеры по ширине и высоте, чем размер ячейки.

На вход искусственной нейронной сети подается набор атрибутов текущего состояния. Значение первого входа зависит от того, находится ли персонаж на блоке, и может иметь значение ноль или единица. Второй и третий входы представляют значения компонент нормированного вектора скорости Собирателя. Третий и четвертый входы представляют значения компонент нормированного вектора направления ближайшего противника. Пятый и шестой - координаты центра Собирателя относительно ячейки, в которую входит точка центра. Координаты нормируются с учетом размера ячейки. Далее следуют входы, представляющие ячейки вокруг собирателя, их количество определялось при тестировании. Выходом сети будут три значения, исходя из которых будут определены действия Собирателя: движение влево, вправо и прыжок, соответственно. Действие будет активизировано, если полученное значение больше 0.5.

1.2 Обзор методов обучения нейронной сети

Обучение на основе учебной выборки для игры в монополию невозможно, ввиду огромного количества вариантов развития. Однако суще-

ствуют другие варианты обучения нейронной сети. Так в статье[4] был выбран эволюционный подход с помощью генетического алгоритма. Для обучения таким способом гены хромосом определяются как вес всех связей нейронной сети. На этапе инициализации хромосомы заполняются случайными вещественными числами из интервала [-1,1]. Агент использует значения хромосом для задания весов сети. Для определения приспособленности каждой хромосомы агенту дается некоторое время на прохождение уровня. Агент будет проходить уровень столько раз, сколько хромосом в популяции. После окончания времени, гибели или пересечения двери, приспособленность определяется как количество набранных очков. Причем если Собиратель погиб, очки все равно будут вычисляться по этой формуле, но будет использоваться количество итераций, даваемое на прохождение. После вычисления приспособленности, выполняются остальные шаги алгоритма.

Селекция или выбор родителей: метод рулетки. Каждой хромосоме сопоставляется значение, величина которого устанавливается пропорционально значению приспособленности данной хромосомы. Чем больше значение приспособленности, тем больше вероятность ее выбора.

Скрещивание и мутация: скрещивание выполняется одноточечным обменом, мутация для каждого гена происходит с вероятностью 0.01. Значение гена меняется путем прибавления случайного числа из отрезка [-0.3; 0.3] и ограничивается отрезком [-1;1].

Выбор следующего поколения: из старого поколения берется 0.6 лучших по приспособленности особей. Они объединяются с полученными потомками, но с сохранением размера популяции (т. е. часть потомков не попадет в новое поколение).

Автор приводит сравнение с другими алгоритмами, из которых видно, что нейронная сеть набирает наибольшее количество очков, однако обучается и проходит уровень чуть дольше чем алгоритм R-Learning.

Помимо лучшей скорости обучения, алгоритм Reinforcement Learning превосходно подходит для случаев, где рассматриваемая система является марковским случайным процессом, то есть процессом «будущее» которого не зависит от «прошлого» при известном «настоящем». Основная

идея reinforcement learning описана в статье[6]. Авторы рассматривают применение алгоритма обучения с подкреплением на примере игр приставки Atari, каждая из игр которой подходит под определение марковского случайного процесса. То есть находясь в состоянии s и выполняя действие a, которое входит в набор возможных действий, игра может выдать reward или сообщить об проигрыше. Вводится понятие cumulative return over time – общее количество reward, которое алгоритм может получить от текущего момента до конца игры. Помимо этого, reward с каждым шагом системы уменьшается на константу γ. Оптимальное поведение при данных обстоятельствах можно описать функцией Q*(s,a), цель которой достигнуть максимального cumulative return over time. Данная функция называется Bellman Equation и выглядит данным образом:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s',a') \middle| s,a \right]$$

Рис. 1: Bellman Equation - функция поведения игрока

Помимо этого, с определённой вероятностью, которая уменьшается после каждой игры, вместо наиболее выгодного действия выбирается случайное из возможных. После каждой игры выбираются случайные наборы действий для корректировки весов нейронной сети, то есть основная задача нейронной сети предсказать какое действие приведёт к максимальному значению Q функции в следующий момент времени. Данный метод называется Q-Learning. После реализации алгоритма авторы статьи сравнили полученные результаты с результатами других способов обучения искусственного интеллекта. Q-Learning в большинстве игр оказался лучшим.

1.3 Обзор существующих технологий для реализации проекта

В связи с развитием машинного обучения в последние годы появилось множество вспомогательных библиотек, которые инкапсулируют реализацию моделей нейронной сети, позволяя очень быстро протестировать идею без постоянного написания основного каркаса нейронной сети.

Одной из самых популярных библиотек машинного обучения является TensorFlow[9], разрабатываемая компанией Google. TensorFlow – это программная библиотека с открытым исходным кодом для численных вычислений с использованием графов потоков данных (data flow graphs). Узлы в таком графе представляют математические операции, в то время, как грани представляют передачу данных (многомерных массивов) между узлами. Гибкая архитектура, на основе которой построен TensorFlow, позволяет разворачивать вычисления на одном или нескольких CPU или GPU (т.е. на центральном или графическом процессоре) на обычном персональном компьютере, сервере или даже мобильном устройстве используя единый программный интерфейс (API). В одной из статей рассматриваются основные возможности данного программного решения[2]. Автор реализует логистическую регрессию и сравнивает время выполнения с результатом библиотеки scikit-learn. TensorFlow оказывается гораздо быстрее – 17с против 3 мин., и точнее в предсказании на тестовой выборке – 92

Далее автор реализует feed-forward нейронную сеть, отмечая, что основная сложность нейронных сетей заключена в обратном распространении ошибки, однако, работая с TF об этом не приходится задумываться, потому что в нем реализован обобщенный алгоритм обратного распространения ошибки, т.е. такой алгоритм, который применим к любым доступным операциям (будь то умножение матриц или операция свертки для свёрточных нейронных сетей). Для обучения используется 55000 изображений, для валидации 5000, в качестве тестовой выборки 10000 изображений. В результате выполнения отмечается, что TensorFlow выполняется в 6 раз быстрее библиотеки Theano. Помимо этого, TensorFlow

предоставляет встроенные средства визуализации, которые повышают наглядность обучаемых моделей.

Несмотря на то, что TensorFlow подходит как для широкого набора техник машинного обучения, так и для глубинного обучения, библиотека может быть довольно сложна в понимании особенно для тех разработчиков, которые только начинают знакомство с данными областями исследования. Альтернативой является «надстройка» Keras [10], которая предоставляет удобный и простой в использовании программный интерфейс для обучения глубоких нейронных сетей. Keras не является самостоятельной системой, а работает поверх Theano, TensorFlow или CNTK. В 2016 году Keras включили в состав TensorFlow[10]. Помимо этого, все классы являются максимально расширяемыми и модульными, благодаря этому библиотека представляет широкий функционал возможностей по решению сложных практических задач.

Выводы

В результате обзора литературы были изучены различные реализации нейронных сетей для построения логики игрового противника. Игра в монополию является марковским случайным процессом [7], следовательно, для обучения искусственного интеллекта был выбран алгоритм Q-Learning.

Были рассмотрены основные средства реализации нейронных сетей в языке Python, в результате чего была выбрана реализация с применением библиотеки Keras.

2 Требования к системе

2.1 Основные правила игры «Монополия»

Для правильного определения требований к разрабатываемой системе необходимо сначала выделить основные правила разрабатываемой игры.

«Монополия» - это экономическая пошаговая стратегия. Главная цель игры – рационально использовать полученный вначале капитал, который выдаётся всем игрокам в равном объёме, и привести к банкротству других участников. Пошаговый игровой процесс разбивается на ходы, каждый игрок ходит по очереди. Игрок перемещается по игровому полю в зависимости от количества очков, выпавшему на кубиках. Если выпал дубль, то игрок повторяет свой ход. В зависимости от клетки, на которую игрок попадает совершаются разнообразные действия.

Помимо перемещения и выполнения действий, приписанных этой клетке игрок вправе:

1. Участвовать в аукционах недвижимости, которую отказывались покупать другие игроки
2. Улучшать свои владения посредством покупки домов для увеличения арендной платы, которая платится другими игроками при попадании на клетку недвижимости
3. Продавать дома по необходимости
4. Закладывать участки недвижимости при отсутствии денег
5. Выкупать свои участки из залога

Объекты игры

На игровом поле расположены клетки. В начале игры каждый игрок располагается на начальной клетке «Вперёд». Всего клетки могут быть разных типов:

1. Улица – поле доступное для покупки, улицы делятся на группы по цвету, завладев одним цветом, игрок вправе купить дом на данной улице. Если игрок попадает на данное поле, и оно не принадлежит никому, то он вправе его купить за указанную стоимость. Если игрок попадает на данном поле, и оно принадлежит другому игроку, то

игрок платит арендную плату этому игроку в зависимости от уровня развития данной улицы. Аналогичная логика используется на полях «Коммунальное предприятие» и «Железнодорожная станция».

2. Коммунальное предприятие – поле доступное для покупки, арендная плата на нем считается в зависимости от количества выпавших очков

3. Железнодорожная станция – поле доступное для покупки, арендная плата на нём считается в зависимости от количества купленных полей данного типа у владельца поля.

4. «Шанс» и «Общественная казна» - поля, на которых игрок должен выполнить действие, которое случайно выберет игра.

5. Бесплатная стоянка – поле, на котором ничего не происходит

6. «Налог» - игрок обязан выплатить определенную сумму налога

Тюрьма

Игрок отправляется в Тюрьму, если:

1. Остановится на поле «Отправляйтесь в Тюрьму»

2. Игрок взял карточку «Шанс» или «Общественная казна», на которой написано «Отправляйтесь в Тюрьму»

3. У игрока выпало одинаковое число очков на обоих кубиках три раза подряд за один ход

Ход игрока оканчивается, когда его отправляют в Тюрьму.

Чтобы выйти из Тюрьмы, игроку необходимо:

1. Заплатить штраф в размере 50 рублей и продолжить игру

2. Попытаться выбросить дубль

После того, как игрок пропустит три хода, находясь в Тюрьме, он должен выйти из неё и уплатить 50 тыс. рублей, прежде чем сможет переместиться на такое число полей, которое выпало на кубиках.

Находясь в Тюрьме, игрок может получать арендную плату за недвижимость, если она не заложена. Если игрок не был «отправлен в Тюрьму», а просто остановился на поле «Тюрьма» в ходе игры, то он не платит никакого штрафа.

Залог

Если у игрока не хватает денег, то он вправе заложить определенное поле. Для этого, сначала продаются все дома, расположенные на данной клетке, если такие имеются и затем игроку передаётся половина от стоимости клетки.

После этого, если другой игрок окажется на данной клетке, то арендная плата взыматься не будет. Однако с других клеток плата взыматься будет.

Для того, чтобы выкупить данную клетку, игроку необходимо внести половину стоимости клетки + 10%.

Покупка домов

Если игрок купил всю цветовую группу клеток, то он монополизировал данную цветовую группу и вправе строить дома на ней. Однако, если хоть одна клетка из цветовой группы заложена, то игрок теряет данную возможность.

Постройка домов увеличивает арендную плату. На одной клетке можно построить максимум 4 дома

Банкротство

Если игрок должен Банку или другому игроку больше денег, чем может получить по своим активам, то он объявляется банкротом, и выбывает из игры

Если игрок должен Банку, Банк получает все деньги игрока. Затем вся недвижимость обанкротившегося игрока продаётся с аукциона.

Если игрок должен другому игроку, все дома банкрота продаются, полученные средства передаются игроку, обанкротившему его, как и вся недвижимость, включая заложенную.

2.2 Функциональные требования

1. Система игра «Монополия» должна реализовывать основные правила настольной игры
2. Система игра «Монополия» должна поддерживать локальный многопользовательский режим
3. Система игра «Монополия» должна правильно определять победителя партии.

2.3 Нефункциональные требования

1. Система игра «Монополия» должна быть написана на языке Python
2. Система игра «Монополия» должна поддерживать как реальных игроков, так и ботов
3. Система игра «Монополия» должна реализовывать текстовый интерфейс

2.4 Диаграмма вариантов использования

Система представляет собой программу, реализующую игру «Монополия».

Единственным актёром системы является игрок.

Рис. 2: Диаграмма вариантов использования

Определим основные варианты использования системы игра «Монополия». Пользователю доступны следующие функции:

1. *Продать улучшение владения* – игрок может продать ненужные улучшения его владений во время своего хода для получения дополнительного заработка

2. *Продать владение* – игрок может продать ранее приобретённые владения

3. *Улучшить владение* – игрок может улучшить приобретённое владение, если цветовая группа владения находится под контролём его монополии

4. *Нахождение в тюрьме* – игрок может находиться в тюрьме определённое количество ходов

5. *Участие в аукционе* – игрок может участвовать в уникальном событии – аукционе

6. *Бросить кубики* – игрок бросает кубики для определения количества клеток для перемещения

7. *Переместиться* – игрок перемещается на выпавшее количество очков

8. *Купить владение* – игрок может купить владение, на которое он попал, если оно не принадлежит никому

2.5 Спецификация основных вариантов использования

Таблица 1: Спецификация прецедента «Нахождение в тюрьме»

UseCase: Нахождение в тюрьме
Аннотация: Нахождение игрока в тюрьме
Главные актеры: Игрок
Второстепенные актеры: Нет
Предусловия: Игрок находится в тюрьме
Основной поток: 1)Система проверяет количество ходов, проведенных в тюрьме игроком 2)Игроку предлагается купить выход из тюрьмы 3)Игрок выплачивает штраф за выход из тюрьмы 4)Игрок выходит из тюрьмы
Постусловия: Игрок вышел из тюрьмы
Альтернативные потоки: 2А Игрок отказался покупать выход из тюрьмы 1)Игрок остаётся в тюрьме 1А Игрок слишком долго находится в тюрьме 1)Игрок выплачивает штраф за выход из тюрьмы 2)Игрок выходит из тюрьмы 4А Игрок не в состоянии оплатить штраф за выход из тюрьмы 1)Система предлагает игроку продать владения 2)Игрок продаёт владения 3)Игрок выплачивает штраф за выход из тюрьмы 4)Игрок выходит из тюрьмы 4А 1А Игрок не имеет владений, игрок отказался продавать владения 1)Игрок становится банкротом 2)Все владения игрока продаются на аукционе

Таблица 2: Спецификация прецедента «Купить владение»

UseCase: Купить владение
Аннотация: Нахождение игрока в тюрьме
Главные актеры: Игрок
Второстепенные актеры: Нет
Предусловия: Владение никому не принадлежит и доступно для покупки
Основной поток: 1)Система предлагает купить владение 2)Игрок решает купить владение 3)Система проверяет возможность покупки владения игроком 4)Игрок завладевает ячейкой
Постусловия: Ход передаётся следующему игроку
Альтернативные потоки: 2A Игрок отказывается от покупки, 3A Игроку не хватает денег для покупки 1)Ячейка выставляется на аукционе 2)Все игроки оповещаются о начале аукциона

Выводы

В процессе анализа требований был определен основной актер системы, варианты использования системы, сформулированы основные функциональные и нефункциональные требования, предъявляемые к разрабатываемой системе.

3 Архитектура системы

Опишем архитектуры системы, основанную на требованиях, сформулированных в предыдущей главе

3.1 Диаграмма классов

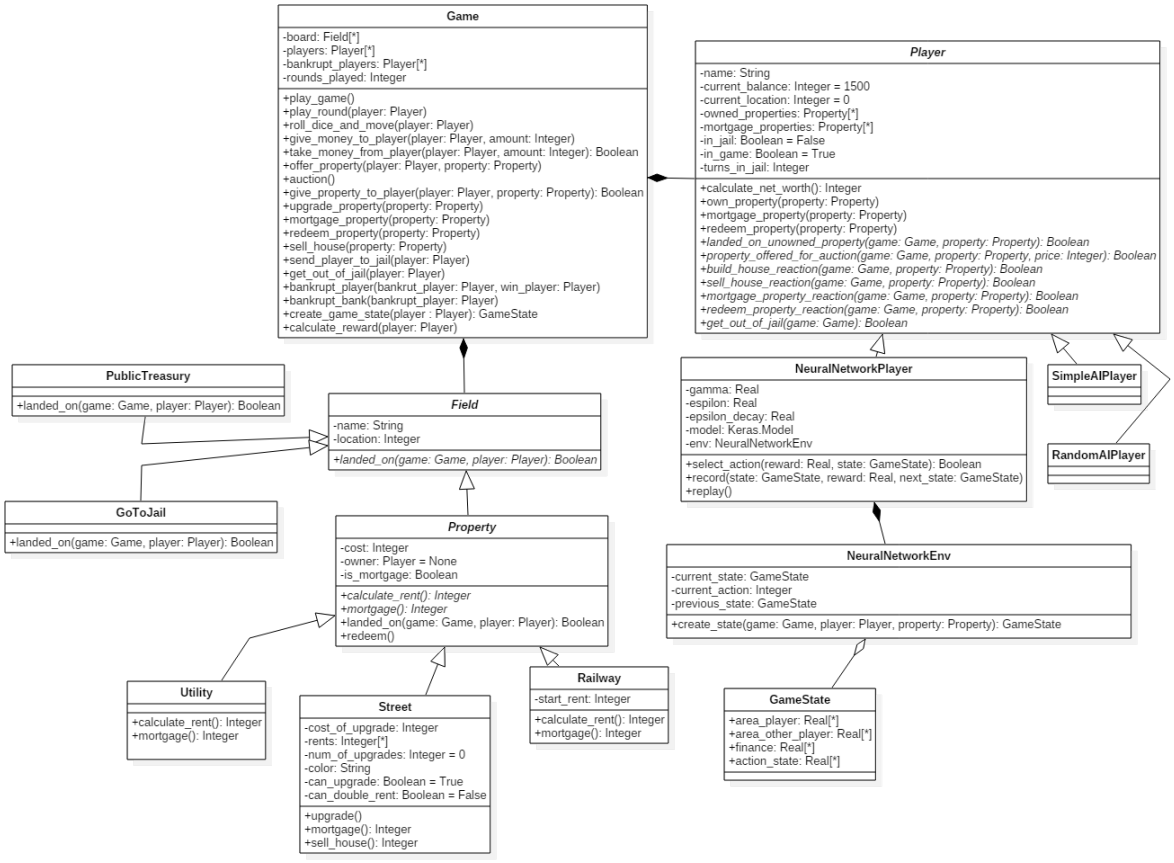

Рис. 3: Диаграмма классов

Далее будет описана семантика основных классов программной части и перечислены их функции.

1. Game - основной класс, в котором реализована логика игры
2. Field – абстрактный класс поля
3. Property – абстрактный класс поля, которым может владеть игрок
4. Chance, Forward, FreeParking, GoToJail, Jail, Tax – поля, которыми нельзя владеть, но которые реализуют различные игровые механики
5. Utility – класс, реализующий поля «Коммунальное предприятие»

6. Railway – класс, реализующий поля «Железная дорога»
7. Street – класс, реализующий поля улиц
8. Player – абстрактный класс, реализующий интерфейс для разработки ботов, путём реализации логики ответа на события, происходящие в игре
9. NeuralNetworkPlayer – класс, реализующий нейронную сеть
10. NeuralNetworkEnv – класс, сохраняющий и формирующий состояния для нейронной сети
11. RandomAIPlayer – класс, реализующий случайное реагирование на события в игре
12. SimpleAIPlayer – класс, реализующий простейшую логику бота
13. GameState – класс, реализующий структуру игрового состояния

3.2 Блок-схема алгоритма аукциона

Механика аукциона является одной из важнейших в «Монополии», однако полноценная реализация данной механики и удобное использование её средствами нейронной сети является затруднительной и ресурсоёмкой задачей. Для удобного взаимодействия ботов с механикой аукциона был разработан алгоритм, блок-схема которого представлена на рис 4.

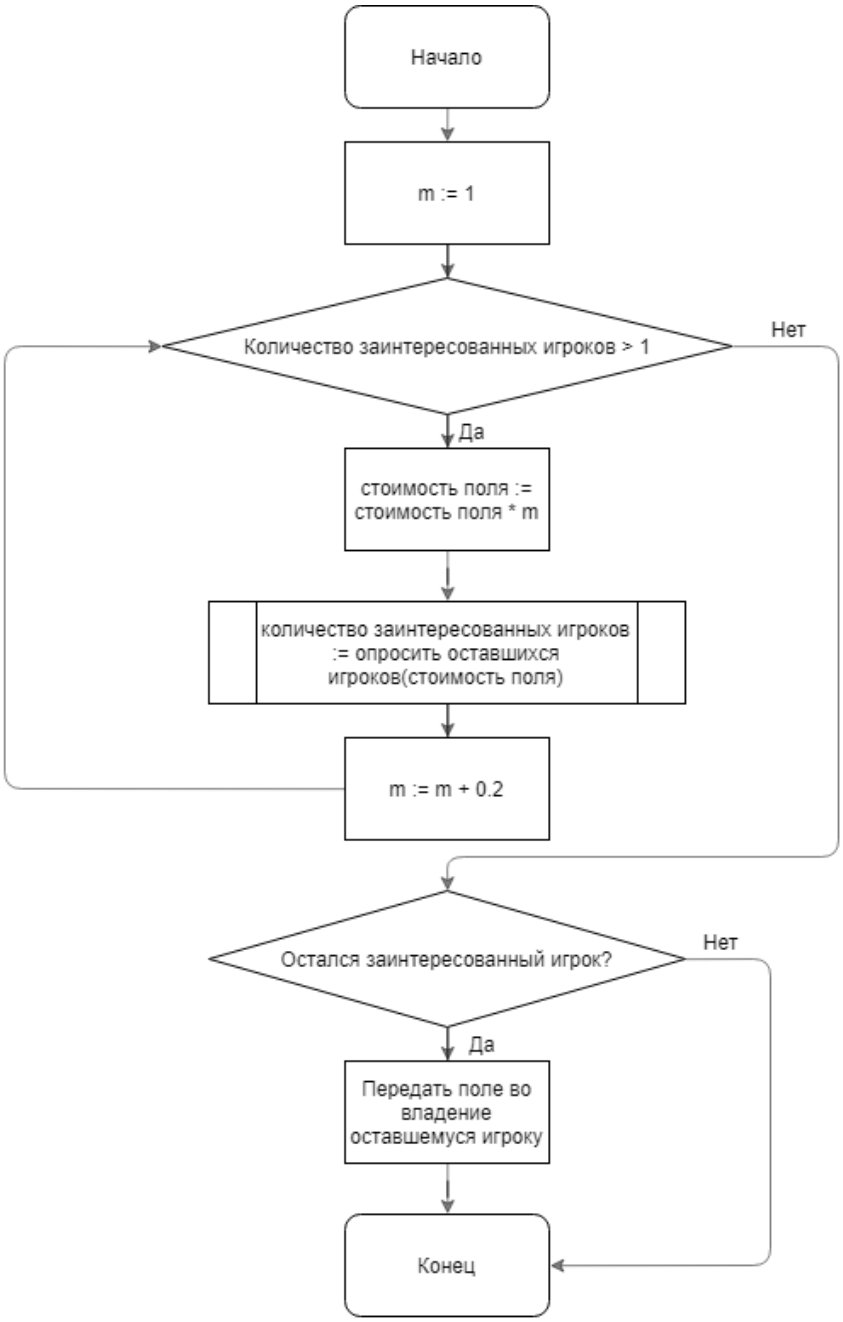

Рис. 4: Блок-схема аукциона

3.3 Определение характеристик состояния игры и действий игрока

Для успешного проектирования нейронной сети нужно сформулировать данные, которые будут формировать текущее состояние игры.

Первые 10 значений характеризуют процент владения недвижимостью игровым агентом, представляющим нейронную сеть (8 различных цветов улиц, коммунальные предприятия и железные дороги)

Для улиц значения можно описать таким образом:

1. 0 – игрок не владеет данным цветом
2. 0 < xi < 0.6 – игрок владеет хотя бы одним полем данного цвета
3. 0.6 – игрок монополизировал данное поле
4. 0.6 < xi < 1 – игрок построил хотя бы один дом на поле данного
5. 1 – игрок полностью застроил владения данного цвета

Для коммунального предприятия значение описываются так:

1. 0 – игрок не владеет ни одним полем данного типа
2. 0.5 – игрок владеет одним полем данного типа
3. 1 – игрок монополизировал поля данного типа

Для железной дороги значения описываются так:

1. 0 – игрок не владеет ни одним полем данного типа
2. 0.25 – игрок владеет одним полем
3. 0.5 – игрок владеет двумя полями
4. 0.75 – игрок владеет тремя полями
5. 1 – игрок монополизировал данный тип поля

Следующие 10 значений характеризуют процент владения недвижимостью другими игроками, который рассчитывается по принципу, описанному выше

Следующие 4 значения характеризуют финансовую составляющую агента:

1. Отношение активов, принадлежащих агенту к общим активам рассчитываются по формуле

$$x_i = \frac{b_1 + n_1}{\sum_{i=1}^{num\_players} b_i + \sum_{i=1}^{num\_players} n_i} \tag{1}$$

где b – баланс игрока, n – текущая ценность купленной недвижимости

2. Отношение текущего баланса к начальному, которое рассчитывается по формуле

$$x_i = \frac{\frac{b}{1500}}{1 + \frac{b}{1500}} \tag{2}$$

где b - текущий баланс игрока

3. Отношение максимальной возможной растраты на аренду/налог к текущему балансу, которое рассчитывается по формуле

$$x_i = \frac{\frac{maxrent-b}{b}}{1 + |\frac{maxrent-b}{b}|} \tag{3}$$

где maxrent – максимально возможная растрата, b – текущий баланс

4. Отношение купленной недвижимости к общей купленной недвижимости, которое рассчитывается по формуле

$$x_i = \frac{p_1}{\sum_{i=1}^{num\_players} p_i} \tag{4}$$

Последнее значение характеризует позицию недвижимости по которому требуется решение, которое формируется в значениях [0, 1]. Например, для поля синего цвета значение будет 0.2.

Для определения размера выходного слоя нейронной сети необходимо определить возможные действия в игре, они будут формировать вектор Action. Предполагается 3 действия:

1. Купить – купить владение, купить дом во владении, выкупить дом из залога, купить владение на аукционе
2. Продать – продать дом, заложить владение
3. Ничего не делать

3.4 Архитектура проектируемой нейронной сети

Была выбрана следующая архитектура нейронной сети (рис. 4), позволяющая оптимально сочетать в себе скорость обучения и точность решения предоставленных задач.

Нейронная сеть имеет 4 слоя и выглядит следующим образом:

1. Входной слой определяется размерностью вектора State, описанному в главе ранее.
2. Первый скрытый слой содержит 200 нейронов. У данного слоя используется функция активации ReLU.
3. Второй скрытый слой содержит 75 нейронов. У данного слоя используется функция активации ReLU.
4. Выходной слой определяется размерностью вектора Action. В каждом нейроне на выходе будет значение предполагаемой награды при совершении подобного действия.

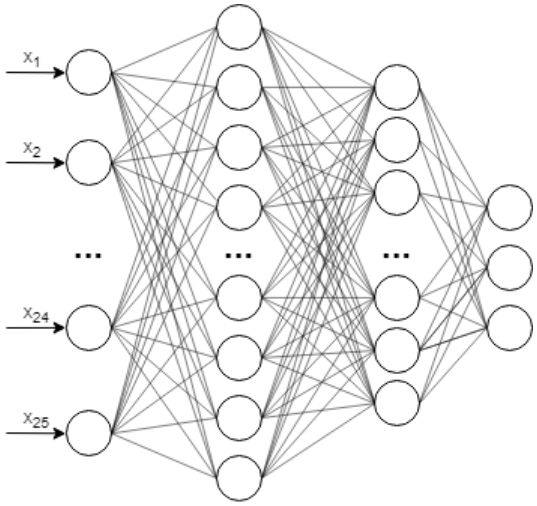

Рис. 5: Архитектура нейронной сети

3.5 Функция награды

Основной задачей игры является завладеть большим количеством полей чем противники и при этом иметь достаточно количество денег для непредвиденных растрат. Для успешного обучения нейронной сети методом Q-learning необходимо сформулировать функцию награды, которая будет выдаваться нейронной сети после каждого выполненного хода.

где v – разница между приобретенными владениями игрока и приобретенными владениями других игроков, p – количество игроков, m – отношение баланса игрока к балансу всех игроков. Награда игрока будет находиться в значениях [-1;1].

$$R = \begin{cases} \dfrac{\dfrac{v}{p^2 * 6}}{1 + \left|\dfrac{v}{p^2 * 6}\right|} + \dfrac{1.1 * m}{p}, & \text{баланс игрока} > maxrent \\ \dfrac{\dfrac{v}{p^2 * 6}}{1 + \left|\dfrac{v}{p^2 * 6}\right|} + \dfrac{1.1 * m}{p} - 1.5, & \text{баланс игрока} < maxrent \\ -1, & \text{игрок не имеет владений} \end{cases}$$

Рис. 6: Функция награды

Выводы

В соответствии с требованиями была спроектирована структура разрабатываемой системы, были определены характеристики состояния системы и выделены возможные действия игрока.

В соответствии с характеристиками состояния и возможных действий игрока была спроектирована нейронная сеть и определена функция награды

4 Реализация системы

4.1 Обзор инструментов для разработки

Для создания игры «Монополия» был выбран язык Python, который на данный момент является передовым решением для машинного обучения, помимо этого он достаточно прост в освоении и позволяет быстро реализовать необходимые алгоритмы. Для реализации нейронной сети была выбрана библиотека Keras, которая представляет собой простой и удобный интерфейс для обучения нейронных сетей. В качестве среды разработки была выбрана PyCharm 2018.1

4.2 Реализация интерфейса для создания ботов

Игра «Монополия» включает в себя интерфейс для создания ботов. Для того, чтобы разработанный бот являлся полноценным актёром системы он должен реализовывать следующие методы:

1. *landed_on_unowned_property (self, game, field)* – метод, вызываемый, когда игрок попадает на поле, не принадлежащее никому. Возвращает true – если покупка одобрена, false – покупка не одобрена

2. *property_offered_for_auction (self, game, field, price)* - метод, вызываемый, когда игрок участвует в аукционе. Возвращает true – если цена устраивает, false – если цена не устраивает, и игрок хочет выйти из аукциона

3. *build_house (self, game, field)* – метод, вызываемый, когда игроку предлагается купить дом на поле. Возвращает true – если покупка одобрена, false – покупка не одобрена

4. *sell_house (self, game, field)* – метод, вызываемый, когда игроку предлагается продать дом на поле. Возвращает true – если продажа одобрена, false – продажа не одобрена

5. *mortgage_property (self, game, field)* – метод, вызываемый, когда игроку предлагается заложить своё поле. Возвращает true – если залог

одобрен, false – залог не одобрен

6. *redeem_property (self, game, field)* – метод, вызываемый, когда игроку предлагается выкупить своё поле из залога. Возвращает true – если выкуп одобрен, false – выкуп не одобрен

7. *get_out_of_jail (self, game)* – метод, вызываемый, когда игрок находится в тюрьме и ему предлагается выйти из тюрьмы, оплатив выкуп. Возвращает true – если выкуп одобен, false – выкуп не одобрен.

4.3 Реализация Q-обучения нейронной сети

Для реализации обучения нейронной сети были сделаны несколько методов:

1. *select_action (self, state)* – выбор следующего действия нейронной сети, исходя из игрового состояния

2. *record (self, state,action, reward, next_state, done)* – запись текущих действий нейронной сети

3. *replay(self)* – функция обучения нейронной сети

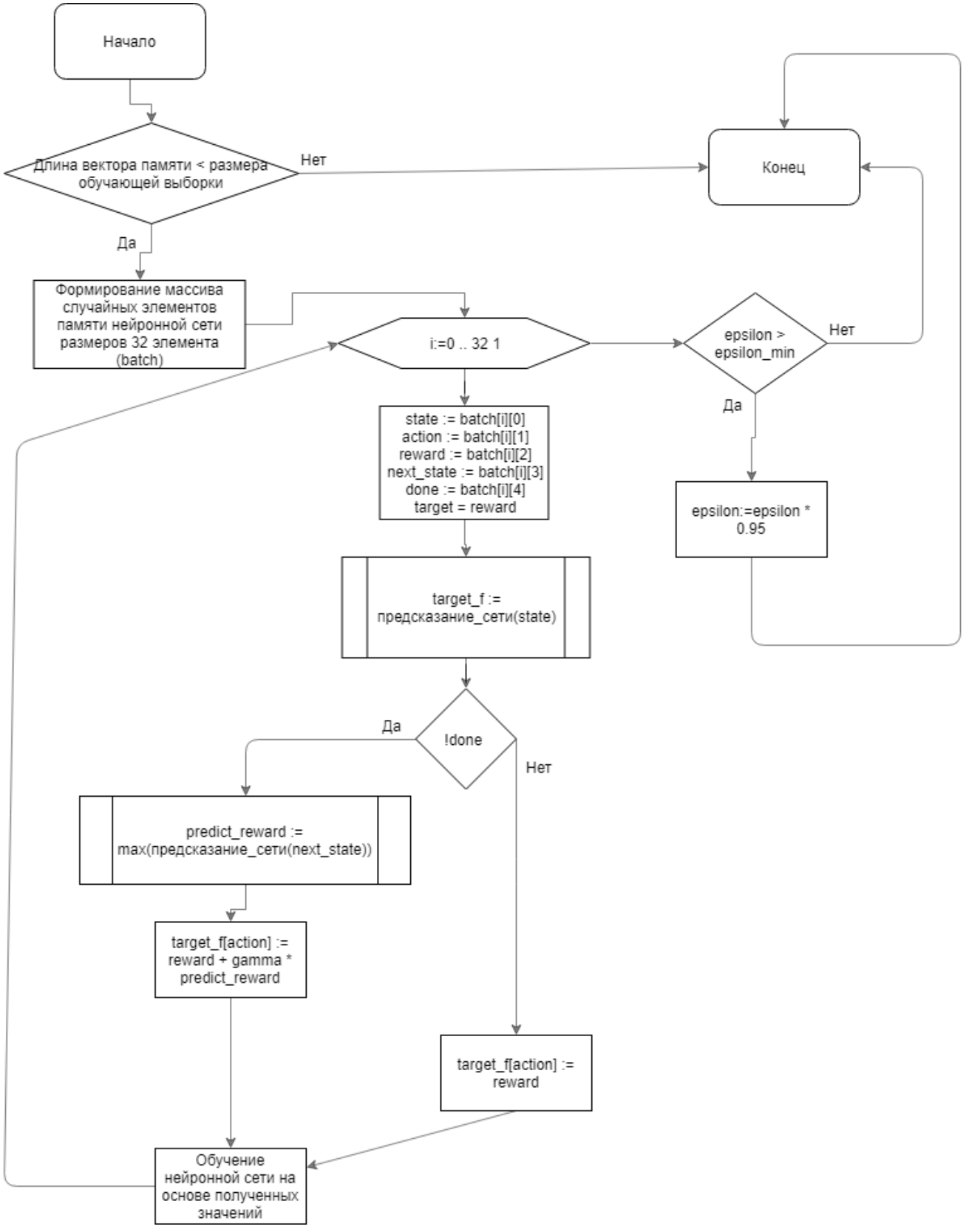

Рис. 7: Блок-схема обучения нейронной сети

4.4 Реализация игровых ботов

Для удобного обучения и тестирования нейронной сети была реализована простая логика противников.

Первый тип противника работает по принципу случайного поведения игрока, то есть на любое предлагаемое действие он реагирует случайным образом.

Второй тип противника реализует простое поведение игрока. Основные правила, которых он придерживается:

1. Согласиться с покупкой, если баланс игрока – стоимость покупки> 350
2. Согласиться с продажей, если баланс игрока меньше 150
3. Ничего не делать, если баланс находится в промежутке между 150 и 350.

Выводы

На основе разработанной архитектуры было реализована система игра «Монополия». Выполнена реализация интерфейса для создания ботов и обучения нейронной сети.

5 Тестирование

5.1 Функциональное тестирование

Таблица 3: Функциональное тестирование

№	Название теста	Действия тестировщика	Ожидаемый результат	Прохождение теста
1	Покупка недвижимости	Соглашается с предложением о покупке недвижимости	Недвижимость переходит во владение игрока. Со счёта игрока снимается стоимость недвижимость	Да
2	Отказ от покупки недвижимости	Отказывается от предложения о покупке недвижимости	Недвижимость выставляется на аукцион	Да
3	Залог недвижимости	Соглашается с предложением о залоге недвижимости	Недвижимость становится заложенной. Игроку перечисляются средства на счёт	Да
4	Выкуп недвижимости	Соглашается с предложение о выкупе недвижимости	Недвижимость снова становится активной. Со счёта игрока снимается стоимость выкупа	Да

5.2 Тестирование обучения нейронной сети

Перед стартом обучения нейронной сети было протестировано преимущество противника, реализующего простейшую логику, перед противником, реализующим случайное поведение. Отношение побед оказалось 97 на 3 в пользу простой логики.

Нейронная сеть обучалась при параметрах:

1. learning_rate = 0.01
2. gamma = 0.95
3. размер памяти нейронной сети = 100
4. размер обучающей выборки = 32
5. epsilon = 1, epsilon_min = 0.1, epsilon_decay = 0.995

Обучение нейронной сети в игре против случайного поведения противника

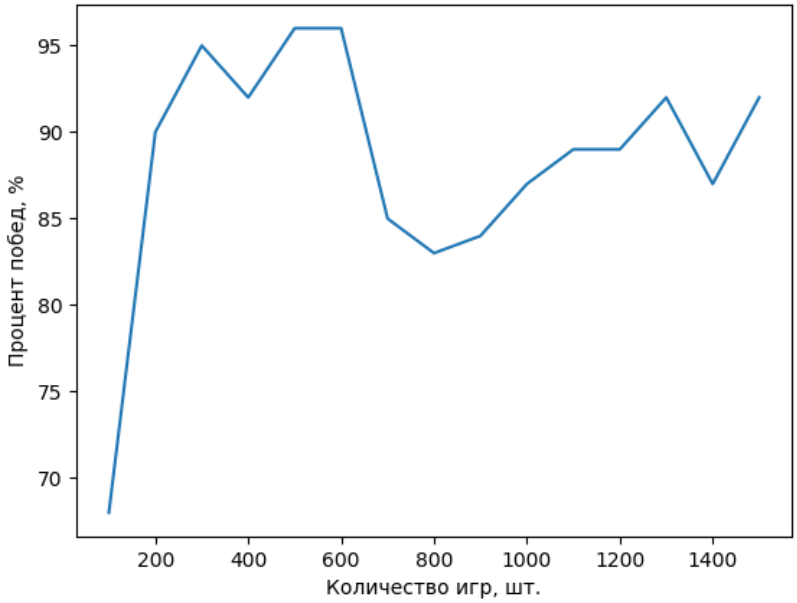

Рис. 8: Изменение процента побед в игре против случайного поведения противника

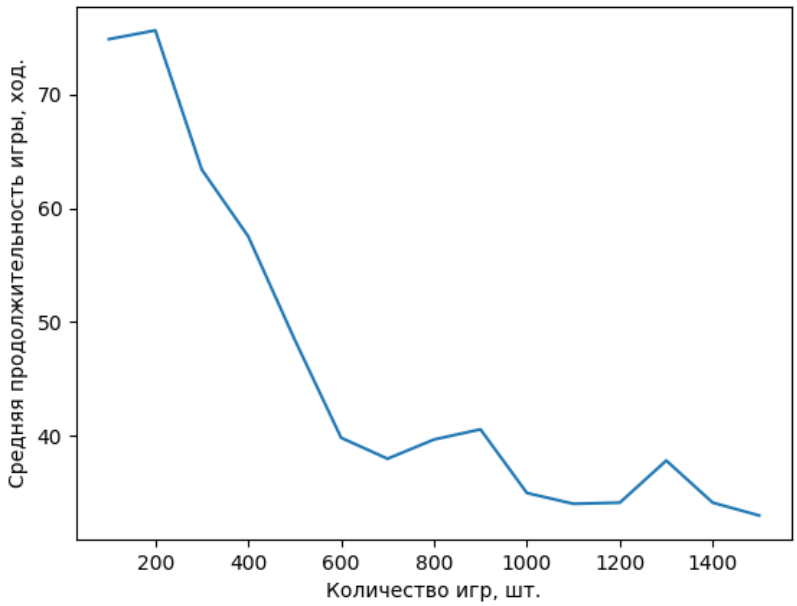

Рис. 9: Изменение средней продолжительности игры в игре против случайного поведения противника

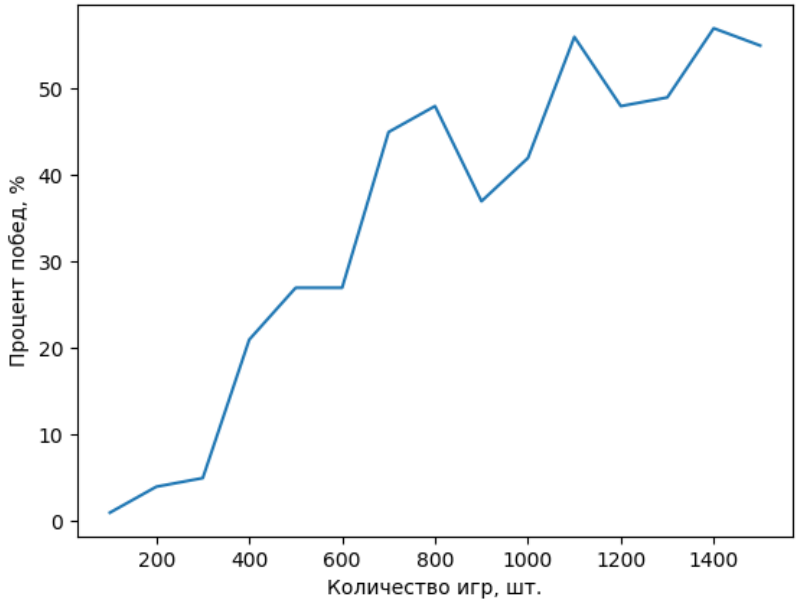

Рис. 10: Изменение процента побед в игре против простого поведения противника

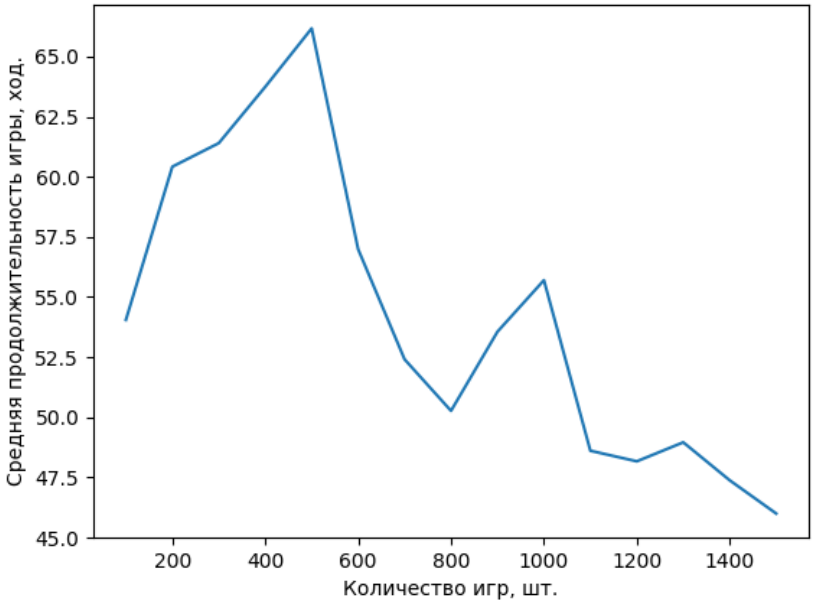

Рис. 11: Изменение средней продолжительности игры в игре против простого поведения противника

Обучение нейронной сети против простого поведения противника

Выводы

В ходе тестирования были проведены несколько экспериментов, которые обнаружили достоинства и недостатки алгоритма Q-обучения.

Достоинства:

1. Отсутствие необходимости формирования обучающей выборки перед началом обучения
2. Возможность формирования стратегии игрока в функции награды

Недостатки:

1. Возможное переобучение
2. Сложность формирования действий нейронной сети, подлежащих обучению
3. Сложность определения функции награды

Для улучшения процента побед предлагается:

1. Изменение функции награды
2. Улучшение алгоритма выбора совершенных действий нейронной сети, подлежащих обучению.

Заключение

В ходе проделанной работы были решены следующие задачи:

1. Изучены основные методы обучения нейронной сети для игр
2. Определены требования к системе игра «Монополия»
3. Разработана структура игры и архитектура нейронной сети
4. Реализована игра «Монополия»
5. Реализована нейронная сеть
6. Протестированы функционал игры и реализация нейронной сети

Литература

1. Коваленко А.С., Ковалевский В.Н. Построение логики игрового противника путём использования машинного обучения // Новые задачи технических наук и пути их решения. Сборник статей Международной научно-практической конференции, 2017. С. 140-144.

2. Латкин И.И. Обзор возможностей TensorFlow для решения задач машинного обучения // Молодежный научно-технический вестник, 2016.

3. Созыкин А. В. Обзор методов обучения глубоких нейронных сетей // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика, 2017. С. 28-49

4. Сотников И.Ю., Григорьева И.В. Адаптивное поведение программных агентов в мультиагентной компьютерной игре // Вестник Кемеровского государственного универститета, 2014. С. 65-71

5. Шампандар А. Искусственный интеллект в компьютерных играх. М.: Вильямс, 2007 - 768 с

6. Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing atari with deep reinforcement learning // NIPS Deep Learning Workshop. 2013. pp. 1-9

7. Robert B Ash, Richard L Bishop, 'Monopoly as a markov process', Mathematics Magazine, 45(1), 1972, pp. 26–29

8. Tariq Rashid. Make your own neural network, 2015. 240 p

9. Официальный сайт TensorFlow. [Электронный ресурс] URL: https://www.tensorflow.org (дата обращения 21.03.2018).

10. Keras: The Python Deep Learning library [Электронный ресурс] URL: http://keras.io (дата обращения 21.03.2018)