МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет Информационных технологий и программирования

Направление (специальность) Прикладная математика и информатика

Квалификация (степень) бакалавр прикладной математики и информатики

Специализация _____

Кафедра компьютерных технологий Группа 4539

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Применение машинного обучения для создания управляющих автоматов на примере игры «Robocode»

Автор квалификационной работы Чернявский И.И. (подпись)
(Фамилия, И., О.)

Руководитель Шалыто А. А. (подпись)
(Фамилия, И., О.)

Консультанты:

а) По экономике и организации производства _____ (подпись)
(Фамилия, И., О.)

б) По безопасности жизнедеятельности и экологии _____ (подпись)
(Фамилия, И., О.)

в) _____ (подпись)
(Фамилия, И., О.)

К защите допустить
Зав. Кафедрой Васильев В.Н. (подпись)
(Фамилия, И., О.)

“___”_____ 2010 г.

Санкт-Петербург, 2010 г.

Аннотация

В настоящей работе исследуется вопрос применимости методов машинного обучения к созданию управляющих автоматов. Рассматривается игра «Robocode», представляющая собой соревнование танков, управляемых программами, написанными на языке *Java* (или языках *.NET*), и ставится задача построения робота для данной игры – танка, способного к успешной игре против других танков. Рассматриваются предыдущие подходы к задаче построения танков и предлагается подход, основанный на использовании конечных автоматов и обучения с подкреплением. В работе описан способ построения роботов-танков с помощью обучения с подкреплением, а также проведено сравнение построенного робота с другими танками.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

С давних пор перед людьми стоит задача создания разумных машин. Решением этой задачи занимается один из разделов информатики – *искусственный интеллект* [1]. Важным понятием для данной дисциплины является понятие *рационального агента* [1] – автономной сущности, характеризующейся:

- способностью воспринимать окружающую среду и воздействовать на неё;
- способностью самостоятельно принимать решения и выполнять действия, направленные на достижение поставленных целей, получение наилучших результатов;
- способностью накапливать опыт и обучаться, взаимодействуя со средой.

Основным предметом искусственного интеллекта как дисциплины является изучение рациональных агентов, способов их построения, решение возникающих при этом задач.

Тесно связанным с искусственным интеллектом является *машинное обучение.*

В настоящей работе изучается задача построения робота-танка в компьютерной игре «Robocode» [2]. Данная игра представляет собой соревнование роботов-танков между собой. Каждый танк является программой, написанной на языке программирования *Java* или языках платформы *.NET*. Соревнование проводится в несколько раундов путем сражения танков друг с другом.

Задачу можно сформулировать в терминах искусственного интеллекта – создать рационального агента (робота-танка),

взаимодействующего со средой, самостоятельно принимающего решения и побеждающего заданного противника.

Для решения данной задачи выбрана группа методов машинного обучения – *методы обучения с подкреплением* [3].

Рационально действующий робот является примером системы со сложным поведением. В ряде задач [4 – 6] было показано, что логику поведения таких систем удобно представлять с помощью *конечных автоматов* [7].

Управляющие конечные автоматы могут иметь сложное поведение и, как следствие, их проектирование вручную может быть трудоемким, а также быть подверженным ошибкам. Поэтому исследователей сразу привлек вопрос о поиске альтернативных методов построения управляющих автоматов.

В работах [5, 6] для построения автоматов было успешно применено генетическое программирование [8].

В настоящей работе изучается применимость методов обучения с подкреплением к созданию управляющих автоматов на примере задачи построения робота-танка для игры «Robocode», а также проводится сравнение результатов, полученных с помощью обучения с подкреплением, с результатами, полученными при использовании генетического программирования.

ГЛАВА 1. ОСНОВЫ ОБУЧЕНИЯ С ПОДКРЕПЛЕНИЕМ

1.1. О методах обучения с подкреплением

Методы обучения с подкреплением (*reinforcement learning* [3]) являются обширным классом методов машинного обучения.

В отличие от другого класса методов – методов обучения с учителем, обучение с подкреплением предполагает отсутствие учителя и в общем случае отсутствие каких-либо начальных знаний о среде.

Обучение с учителем предполагает наличие источника готовых знаний, базы примеров правильного поведения, которые обучаемый должен перенять и сформировать свое поведение на их основе. Данные методы отлично работают для таких задач как, например, классификация образов.

В то же время существует множество задач, в которых создание набора обучающих примеров затруднительно. Примером такой задачи может быть, например, обучение игре в нарды. В данном случае целью обучения являются навыки успешной игры. База примеров с позициями на доске и правильными ходами не будет эффективным способом обучения вследствие огромного числа позиций в игре. Обучение с подкреплением, в свою очередь, хорошо зарекомендовало себя в данной задаче и в обучении играм в целом [11].

Идея обучения с подкреплением состоит в обучении посредством взаимодействия со средой. Данная идея является результатом непосредственных наблюдений за поведением человека и животных. С первых дней своей жизни человек начинает изучать окружающий мир путем проб и ошибок, запоминая результаты своих действий, вырабатывая правила, которым он будет следовать в дальнейшем. Возвращаясь к играм,

важным компонентом обучения является практика, т.е. непосредственно сам процесс игры, позволяющий накопить необходимый игровой опыт.

Методы обучения с подкреплением были разработаны в 80-х годах Р. Саттоном, К. Уоткинсом и другими.

Один из самых ярких результатов в области машинного обучения был получен Дж. Тесауро, который в начале 90-х создал программу *TD-Gammon* [11], играющую в нарды. Играя с чемпионами мира практически на равных, *TD-Gammon* смог добиться впечатляющих результатов. Некоторые общепринятые способы разыгрывания партий были изменены после анализа игры данной программы.

Отличием *TD-Gammon* от программ-предшественников явилось использование метода *TD(λ)* – одного из методов обучения с подкреплением.

1.2. Формулировка задачи обучения с подкреплением

Методы обучения с подкреплением направлены на обучение агента, взаимодействующего с окружающей средой.

Общая схема работы агента изображена на рис. 1.

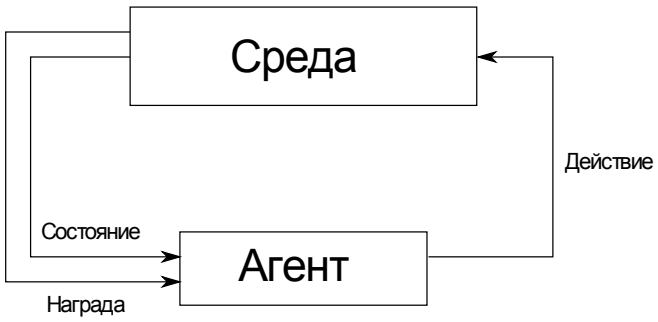

Рис. 1. Схема работы агента

Агент взаимодействует со средой в дискретные моменты времени

$$t = 0, 1, 2 \ldots$$

На каждом шаге t агент получает информацию о среде – *состояние* среды:

$$s_t \in S,$$

где S – множество состояний среды. Состояние может включать в себя как низкоуровневые параметры (например, электрические напряжения), так и высокоуровневые (например, время суток). Более того, в состояние могут входить и внутренние параметры робота-агента. Таким образом, граница между средой и агентом не обязательно совпадает с физической границей между средой и роботом.

На основе состояния агент выбирает *действие*:

$$a_t \in A(s),$$

где $A(s)$ – множество возможных действий, доступных их состояния s.

На следующем шаге агент получает новое состояние s_{t+1} и *награду* r_{t+1}.

Награда – это число, являющееся непосредственной оценкой средой действия, совершенного агентом. В общем случае награда может включать в себя шум – быть случайной величиной. Награда может быть отрицательной, то есть быть наказанием.

Важно отметить, что награда является оценкой действий агента только в ближайшей перспективе.

Целью агента является максимизация суммы наград в *долгосрочной перспективе*.

Заметим, что жадная стратегия – стратегия выбора действий, в которой агент на каждом шаге выбирает только те действия, которые приносят максимальную награду, в общем случае не является оптимальной. В качестве примера можно привести агента, обучающегося

играть в шахматы. Известно, что в некоторых случаях бывает выгодно жертвовать своими фигурами. Агент, реализующий жадную стратегию, не будет способен действовать подобным образом, так как в ближайшей перспективе потеря фигур не является выгодной.

Приведем несколько примеров наград. В задаче обучения агента игре в крестики-нолики, шашки или любой другой игре, в качестве награды можно взять ноль за все шаги, кроме последнего, а за последний шаг давать положительную награду в случае выигрыша, отрицательную – в случае проигрыша и нулевую – в случае ничьи. В задаче обучения робота поиску выхода из лабиринта можно давать отрицательную награду за каждый шаг проведенный внутри лабиринта.

Существует несколько подходов к формализации цели агента – максимизации суммы наград в долгосрочной перспективе.

В *модели конечного горизонта* под целью агента подразумевается максимизация величины:

$$E\left\{\sum_{t=0}^{h} r_t\right\}.$$

Следуя данной модели, агент будет стремиться максимизировать математическое ожидание суммы последующих h наград. Данная модель удобна, когда число шагов ограничено.

В случае, когда нет ограничения на число шагов, для учета всех последующих наград удобно использовать *модель бесконечного горизонта*, в которой агент максимизирует величину:

$$E\left\{\sum_{t=0}^{\infty} \gamma^t r_t\right\},$$

где $0 \le \gamma \le 1$ – коэффициент (*discount factor).*

Коэффициент γ позволяет давать ближайшим наградам больший вес. В условии $0 \le \gamma < 1$ и ограниченности наград ряд в формуле гарантированно сходится.

Заметим, что при $\gamma = 0$ агент реализует жадную стратегию.

Еще одной моделью является *модель среднего вознаграждения*, выражающаяся в максимизации предела среднего вознаграждения:

$$\lim_{h \to \infty} E\left\{ \frac{1}{h} \sum_{t=0}^{h} r_t \right\}.$$

Данная модель не делает различий между ближайшей и отдаленной наградами.

1.3. Стратегии и функции выигрыша

1.3.1. Понятие стратегии и функции выигрыша

На каждом шаге агент решает задачу выбора действия. Решение этой задачи агент производит на основе состояния среды. После совершения действия агент получает новое состояние и оценку от среды. Затем агент снова выбирает действие и т.д.

Модель взаимодействия агента и среды удобно представлять в виде *марковского процесса принятия решений* [3].

Марковский процесс принятия решений состоит из:

- конечного множества состояний S;
- множеств действий. $A(s)$ – действия, доступные из состояния s;
- вероятностей переходов между состояниями:

  $P(s, a, s')$ – вероятность перехода в состояние s' из состояния s при выполнении действия $a \in A(s)$;

- математических ожиданий наград: $R(s, a, s')$ – ожидание награды при переходе в состояние s' из состояния s после выполнения действия $a \in A(s)$.

Марковский процесс удобно представлять в виде *диаграммы переходов*, пример [3] которой представлен на рис. 2.

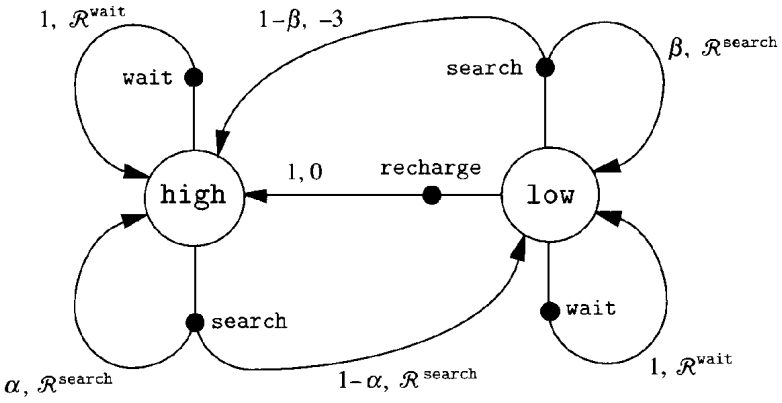

Рис. 2. Диаграмма переходов марковского процесса

На этом рисунке изображена диаграмма переходов марковского процесса, описывающего робота, собирающего мусор. На диаграмме можно выделить узлы состояний – вершины с названием состояния внутри, и узлы действий – черные круги. Узлы состояний соединены с узлами действий, которые агент может выполнить. Узлы действий соединены с узлами состояний, в которые агент может перейти, выполнив данное действие. Дуги помечены вероятностью перехода и ожиданием награды. В данном примере робот имеет два состояния – состояние низкой заряженности батарей и высокой, а также три действия – поиск мусора, ожидание и подзарядка. Из примера видно, что из состояния с низким зарядом батарей робот может выбрать подзарядку и перейти в состояние с высоким уровнем батарей и не получить награды. С другой стороны, робот может начать искать мусор. Поиск мусора может привести робота к нахождению мусора с вероятностью β и получению награды или к полной разрядке батарей, что приведет к отрицательной награде – наказанию.

Заметим, что используя модель марковского процесса принятия решений, подразумевается *марковское свойство* – независимость вероятностей перехода и ожиданий выигрыша от истории предыдущих переходов. Вероятности перехода в следующие состояния и награды зависят только от текущего состояния.

Марковское свойство позволяет агенту учитывать в своих действиях только текущее состояние.

Стратегией агента будем называть функцию $\pi : S \times A \to [0,1]$, где $\pi(s,a)$ – вероятность выбора действия $a \in A(s)$ в состоянии s.

В случае детерминированной стратегии – стратегии, для которой в каждом состоянии существует действие, которое агент выбирает с вероятностью единица, выбираемое действие обозначается как $\pi(s)$.

Функция ожидаемого выигрыша (*value function*) $V^{\pi}(s)$ – это функция математического ожидания суммы наград при условии, что агент придерживается стратегии π и начинает работу в состоянии s:

$$V^{\pi}(s) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \mid s_t = s\right\}$$

Также можно рассмотреть другую функцию выигрыша (*action-value function*):

$$Q^{\pi}(s,a) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \mid s_t = s, a_t = a\right\},$$

для которой известно первое действие a, выбираемое агентом, находящимся в состоянии s.

Функция выигрыша для любой стратегии π и состояния s удовлетворяет следующему рекурсивному соотношению:

$$V^{\pi}(s) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \mid s_t = s\right\} = E_{\pi}\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+2} \mid s_t = s\right\} =$$

$$= \sum_{a\in A(s)}\pi(s,a)\sum_{s'}P(s,a,s')\left[R(s,a,s') + \gamma\, E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+2} \mid s_{t+1} = s'\right\}\right] =$$

$$= \sum_{a\in A(s)}\pi(s,a)\sum_{s'}P(s,a,s')\left[R(s,a,s') + \gamma\, V^{\pi}(s')\right]$$

Таким образом, значение функции выигрыша для состояния s определяется через значения функции выигрыша в состояниях, в которые может перейти агент из состояния s, и параметры марковского процесса.

Данное соотношение называется *уравнением Беллмана*.

Выписав рекурсивные отношения для каждого состояния s, можно составить систему из $|S|$ линейных уравнений с $|S|$ неизвестными – значениями функции выигрыша. Известно, что при условии $0 \le \gamma < 1$ система имеет единственное решение.

Таким образом, зная стратегию π и параметры марковского процесса, значения функции выигрыша можно получить путем решения системы линейных уравнений.

На практике получение значений функции выигрыша таким способом часто бывает затруднительным в силу ряда причин:

- параметры марковского процесса часто неизвестны;
- число состояний может быть так велико, что решение системы на практике невозможно.

1.3.2. Улучшение стратегий

Рассмотрим стратегию π и функцию выигрыша V^{π}.

$V^{\pi}(s)$ является оценкой выигрыша при следовании стратегии π из состояния *s*. Рассмотрим выигрыш, который получается, если в состоянии *s* выбрать действие *a* и затем следовать стратегии π:

$$Q^{\pi}(s, a) = \sum_{s'} P(s, a, s') \left(R(s, a, s') + \gamma V^{\pi}(s')\right).$$

Пусть $Q^{\pi}(s, a) \geq V^{\pi}(s)$ – в состоянии *s* выгоднее выбрать действие *a* и затем следовать π, нежели сразу следовать π.

Тогда стратегия θ, выбирающая действие a в состоянии s и совпадающая с π во всех остальных состояниях, *лучше*, чем стратегия π:

$$\forall s : V^{\pi}(s) \leq V^{\theta}(s).$$

Данный результат является следствием теоремы – *policy improvement theorem* [3].

Пусть для детерминированных стратегий θ и π для каждого состояния *s* выполняется неравенство:

$$Q^{\pi}(s, \theta(s)) \geq V^{\pi}(s),$$

тогда стратегия θ лучше, чем стратегия π.

Докажем данную теорему.

Рассмотрим функцию выигрыша стратегии π:

$$\begin{aligned}
&V^{\pi}(s) \leq Q^{\pi}(s, \theta(s)) = E_{\theta}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) \mid s_t = s\right\} \leq \\
&\leq E_{\theta}\left\{ r_{t+1} + \gamma Q^{\pi}(s_{t+1}, \theta(s_{t+1})) \mid s_t = s \right\} = \\
&= E_{\theta}\left\{ r_{t+1} + \gamma E_{\theta}\left\{r_{t+2} + \gamma V^{\pi}(s_{t+2})\right\} \mid s_t = s \right\} =
\end{aligned}$$

$$= E_\theta \left\{ r_{t+1} + \gamma r_{t+2} + \gamma^2 V^\pi(s_{t+2}) \mid s_t = s \right\} \le$$
$$\le E_\theta \left\{ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V^\pi(s_{t+3}) \mid s_t = s \right\} \le$$
$$\le \ldots \le V^\theta(s)$$

Таким образом, стратегия θ лучше, чем стратегия π.

1.3.3. Оптимальные стратегии

Стратегии π можно сопоставить функции выигрыша V^π и Q^π.

Стратегия θ называется *оптимальной*, если для любой другой стратегии π и любого состояния s выполняется неравенство:

$$V^\pi(s) \le V^\theta(s)$$

Функции выигрыша оптимальной стратегии обозначим, как V^* и Q^*. Функции V^* и Q^* связаны соотношением:

$$Q^*(s, a) = E\left\{ r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a \right\}.$$

Можно показать, что оптимальная стратегия θ существует всегда.

Действительно, вследствие уравнения Беллмана и условия оптимальности, для оптимальной функции выигрыша V^* верны следующие равенства (*уравнение оптимальности Беллмана*):

$$V^*(s) = \max_{a \in A(s)} Q^\theta(s, a) = \max_{a \in A(s)} E_\theta \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right\} =$$
$$= \max_{a \in A(s)} E_\theta \left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s, a_t = a \right\} =$$
$$= \max_{a \in A(s)} E \left\{ r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a \right\} =$$
$$= \max_{a \in A(s)} \sum_{s'} P(s, a, s') \left(R(s, a, s') + \gamma V^*(s') \right)$$

Выписав уравнения для всех $V^*(s)$, можно составить систему из $|S|$ уравнений с $|S|$ неизвестными. Известно, что данная система имеет единственное решение для любого конечного марковского процесса при условии $0 \le \gamma < 1$.

Таким образом, значения $V^*(s)$ можно получить как решение системы уравнений. Заметим, что данная система уравнений не зависит от какой-либо стратегии и зависит только от параметров марковского процесса.

Несложно видеть, что зная $V^*(s)$, можно построить оптимальную стратегию π с функцией выигрыша равной V^*.

Рассмотрим состояние s и действие $a \in A(s)$. Действие a является *оптимальным* для состояния s, если совершение данного действия из состояния s максимизирует ожидаемый выигрыш, который определяется формулой:

$$\sum_{s'} P(s, a, s') \, (R(s, a, s') + \gamma V^*(s')).$$

Тогда достаточно в качестве оптимальной стратегии взять такую стратегию π, которая будет выбирать с ненулевой вероятностью только оптимальные действия.

Заметим, что оптимальных стратегий с функцией выигрыша V^* может быть бесконечно много.

1.4. Некоторые методы обучения с подкреплением

Методы, изложенные в данном разделе, направлены на вычисление оптимальной функции выигрыша. Как было показано в разд. 1.3.3, знание оптимальной функции выигрыша позволяет построить оптимальную стратегию.

Рассмотрим три группы методов – методы динамического программирования, методы Монте-Карло и методы темпоральных разностей.

1.4.1. Методы динамического программирования

В методах данной группы предполагается, что агент имеет полную информацию о среде – знает все параметры марковского процесса.

Основной идеей данной группы методов является итеративное построение последовательности стратегий, приводящей к оптимальной стратегии.

Вычисление функции выигрыша

Рассмотрим сначала задачу вычисления функции выигрыша для заданной стратегии. Одним из простых способов ее решения является итеративный способ.

Известно, что значения функции выигрыша V^{π} удовлетворяют уравнению Беллмана:

$$V^{\pi}(s) = \sum_{a \in A(s)} \pi(s,a) \sum_{s'} P(s,a,s') \left(R(s,a,s') + \gamma V^{\pi}(s')\right).$$

Для конечного марковского процесса функция V^{π} существует и единственна при условии $0 \le \gamma < 1$.

Итеративный метод строит последовательность функций $\left\{V^k\right\}$, вычисляя значения очередной функции по следующей формуле:

$$V^{k+1}(s) = \sum_{a \in A(s)} \pi(s,a) \sum_{s'} P(s,a,s') \left(R(s,a,s') + \gamma V^{k}(s')\right).$$

Известно, что в условии существования V^{π}, последовательность $\left\{V^k\right\}$ сходится к V^{π}.

V^0 определяется как функция с нулевыми значениями. Итеративный процесс останавливают при выполнении условия:

$$\max_s |V^{k+1}(s) - V^k(s)| \le \Delta,$$

где Δ – малый параметр.

Метод итераций по стратегиям

Данный метод строит последовательность стратегий, в которой каждая следующая стратегия лучше предыдущей.

В качестве начальной стратегии достаточно взять любую стратегию.

На каждой итерации, метод вычисляет функцию V^{π_k} для текущей стратегии π_k и пытается улучшить ее.

В соответствии с теоремой, описанной в разд. 1.3.2, стратегию π можно заменить стратегией π_{k+1}, которая будет лучше стратегии π_k:

$$\pi_{k+1}(s) = \arg\max_a Q^{\pi_k}(s, a) = \arg\max_a \sum_{s'} P(s, a, s')(R(s, a, s') + \gamma V^{\pi_k}(s'))$$

Если стратегия π_{k+1} отличается от стратегии π_k, то алгоритм переходит к следующей итерации – вычисляет значения функции выигрыша для стратегии π_{k+1}, строит улучшенную стратегию и т.д.

Заметим, что на каждой итерации метод строит стратегию, которая лучше предыдущей, и останавливается, когда выполняется равенство:

$$\pi^*(s) = \arg\max_a Q^{\pi^*}(s, a).$$

Данное равенство является условием оптимальности для стратегии, поэтому метод сходится к оптимальной стратегии.

Принцип, используемый в данном подходе, широко применяется и в других методах. В общем случае *принцип итераций по стратегиям* состоит в постоянной оценке значений функции выигрыша для текущей

стратегии и постоянном улучшении текущей стратегии на основе значений ее функции выигрыша.

Модификации метода итераций по стратегиям

Метод итераций может быть модифицирован несколькими способами.

Шаги вычисления функции выигрыша и получения новой стратегии можно объединить, заменив итеративный процесс вычисления функции выигрыша только первым шагом. Тогда каждая итерация сводится к применению уравнения оптимальности Беллмана для обновления значений $V^k(s)$:

$$V^{k+1}(s) = \max_a \sum_{s'} P(s, a, s')\,(R(s, a, s') + \gamma V^k(s')).$$

Заметим, что обновление значений функции выигрыша производится для *каждого* состояния. В ряде задач такое обновление невозможно вследствие большого числа состояний.

Для преодоления этой проблемы метод итераций по стратегиям можно применять *во время* работы агента. В качестве состояния s, для которого производится обновление $V^k(s)$, достаточно выбирать состояние, которое агент непосредственно наблюдает, а в качестве стратегии агента – стратегию, построенную на основе текущей аппроксимации функции выигрыша.

1.4.2. Методы Монте-Карло

В отличие от методов динамического программирования методы Монте-Карло не требуют каких-либо знаний об окружающей среде. Отсутствие данного требования значительно расширяет сферу применения данной группы методов, так как в большинстве практических задач параметры марковского процесса не известны.

Заметим, что при отсутствии информации об окружающей среде, построение оптимальной стратегии по оптимальной функции выигрыша V^* затруднительно. Поэтому методы данной группы направлены на вычисление оптимальной функции выигрыша Q^*.

Основная идея методов Монте-Карло – получать оценки значений функции выигрыша $Q^\pi(s, a)$ непосредственно из эксперимента. Для получения оценок проводится ряд экспериментов, состоящих в записи состояний, действий и полученных наград при взаимодействии агента-робота, использующего стратегию π, со средой. Полученные экспериментальные данные могут быть использованы для вычисления суммы последующих наград для каждой пары состояние-действие. Повторяя эксперимент и усредняя суммы наград, можно получить достаточно точные оценки.

Заметим, что некоторые пары состояние-действие могут не встречаться или встречаться редко в процессе взаимодействия агента и среды. В этом случае оценку значения функции выигрыша для пары (s, a) можно получить, начав эксперимент из состояния s и первого действия агента a.

Для построения оптимальной стратегии используется принцип итерации по стратегиям. На каждой итерации путем проведения эксперимента вычисляется функция выигрыша для текущей стратегии и строится улучшенная стратегия на основе вычисленной функции.

1.4.3. Методы темпоральных разностей

Методы данной группы имеют свойства присущие как методам Монте-Карло, так и методам динамического программирования.

Подобно методам Монте-Карло, методы темпоральных разностей (*temporal difference methods, TD-methods*) не требуют полной информации

об окружающей среде и используют результаты непосредственного взаимодействия агента и среды.

В то же время, методы темпоральных разностей, как и методы динамического программирования, используют имеющиеся оценки для вычисления новых оценок.

Вычисление функции выигрыша

Также как и методы Монте-Карло, методы темпоральных разностей обновляют оценки значений функции выигрыша на основе полученных наград.

Но, в отличие от методов Монте-Карло, методам темпоральных разностей не требуется ждать окончания эксперимента для того, чтобы получить сумму наград. Вместо этого методы данной группы используют уже имеющиеся оценки $V^{\pi}(s)$.

Пусть, находясь в состоянии s, агент совершил действие a и перешел в состояние s', получив при этом награду r. Тогда оценку $V(s)$ значения $V^{\pi}(s)$ можно обновить следующим образом:

$$V(s) = V(s) + \alpha\,(r + \gamma V(s') - V(s)).$$

Значение $V^{\pi}(s')$ неизвестно, поэтому для оценки общей суммы наград, которые получит агент, используется имеющаяся оценка для состояния s':

$$R = r + \gamma V(s').$$

Таким образом, оценка $V(s)$ сдвигается в сторону новой оценки на долю α от разности двух оценок.

Начав с нулевых значений $V(s)$, агент на каждом шаге применяет указанное правило для обновления значений $V(s)$, получая при этом все более точные оценки значений функции выигрыша.

Заметим, что в отличие от методов Монте-Карло, где оценки значений функции выигрыша получаются только после проведения эксперимента, в методах темпоральных разностей оценки значений функции выигрыша доступны на каждом шаге и уточняются со временем.

Q-обучение

Q-обучение (*Q-learning*) – метод обучения из группы *TD*-методов. Для нахождения оптимальной стратегии метод аппроксимирует значения оптимальной функции выигрыша Q^*. Получение оценок для значений Q^* основано на принципе итерации стратегий.

Во время работы метода согласно принципу итерации стратегий происходит два процесса – оценка значений функции выигрыша текущей стратегии и улучшение текущей стратегии на основе полученных оценок. Данные процессы происходят на каждом шаге.

Подобно обновлению оценок $V(s)$, метод Q-обучения на каждом шаге обновляет оценки значений функции выигрыша Q^π текущей стратегии π, используя следующую формулу:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)),$$

где s_t – состояние, в котором находился робот на шаге t, a_t – действие, которое совершил робот на шаге t, r_{t+1} – полученная награда, s_{t+1} – состояние, в которое перешел робот.

Как видно из данной формулы, для оценки последующего выигрыша (суммы последующих наград) применяется формула:

$$R_t = r_{t+1} + \gamma \max_a Q(s_{t+1}, a).$$

Улучшение текущей стратегии проводится путем выбора из данного состояния s действия, максимизирующего оценку Q:

$$\arg\max_a Q(s, a).$$

Таким образом, агент в процессе своей работы улучшает свою текущую стратегию, выбирая действия, приводящие к наибольшему выигрышу, и, в то же время, на каждом шаге улучшает оценки, используемые при выборе действий.

При применении жадной относительно значений Q стратегии выбора действий возникает определенное противоречие.

С одной стороны, оценки Q являются оценками суммы последующих наград, и имеет смысл использовать жадную стратегию, так как агент стремится к получению наибольшего выигрыша.

С другой стороны, агент должен улучшать оценки Q и поэтому должен «попробовать» как можно больше различных действий из данного состояния.

Эта проблема, известная под названием «*exploration vs. exploitation»,* присуща не только методу Q-обучения, но и большинству методов обучения с подкреплением.

Одним из решений данной проблемы является использование ε-*жадной* стратегии.

Следуя данной стратегии, агент с вероятностью ε выбирает случайное действие, доступное из данного состояния, и с вероятностью $1-\varepsilon$ выбирает действие, максимизирующее оценку Q.

Q(λ)-обучение

Модификацией метода Q-обучения является $Q(\lambda)$-обучение. Идея, которая лежит в основе $Q(\lambda)$-обучения, состоит в изменении на данном шаге не только оценки для текущего состояния и выбранного действия, но

и для всех предыдущих пар состояние-действие, оказавших влияние на выбор данного действия.

Для осуществления данной идеи, каждой паре (s, a) ставится в соответствие число $0 \le e(s, a) \le 1$, использующееся для обновления значений $Q(s, a)$.

Первоначально $e(s, a) = 0$ для любой пары (s, a).

На каждом шаге агент, находясь в состоянии s, совершает действие a, переходит в состояние s' и получает награду r. Затем агент обновляет $e(s, a)$:

$$e(s, a) = 1,$$

вычисляет разницу:

$$\delta = r + \gamma \max_a Q(s', a) - Q(s, a)$$

и обновляет для каждой пары (s, a) оценку Q:

$$Q(s, a) = Q(s, a) + \alpha e(s, a)\delta .$$

Затем для всех пар (s, a) агент корректирует $e(s, a)$. Если выбор последнего действия агентом носил не случайный характер, то предыдущая история работы агента имеет значение и для всех пар (s, a) агент корректирует $e(s, a)$ следующим образом:

$$e(s, a) = \lambda e(s, a),$$

где $0 \le \lambda \le 1$. Если действие было выбрано случайно, то все $e(s, a)$ обнуляются.

Заметим, что на практике на каждом шаге достаточно работать не со всеми парами (s, a), а только с теми, в которых побывал агент.

Данная модификация *Q*-обучения полезна, когда число наград, полученных агентом, не велико. В частности, Q(λ)-обучение эффективно в случаях, когда награда дается только на последнем шаге.

1.4.4. Аппроксимация с использованием нейронных сетей

Во всех описанных методах предполагалось, что агент хранит оценки значений функции выигрыша для каждого состояния или пары состояние-действие.

Самый простой способ хранения данной информации – в таблице. Для ряда задач такой способ хранения невозможен вследствие большого числа состояний и, как следствие, большого размера таблицы в памяти. Также при большом числе состояний в процессе обучения агента могут возникать проблемы с обобщением полученного опыта.

Для решения данных проблем значения функции выигрыша можно выбирать не из таблицы, а вычислять с помощью нейронной сети [12]. На входы нейронной сети подаются числа, описывающие состояние (или пару состояние-действие) и значение на выходе используется в качестве оценки выигрыша для данного состояния (или пары состояние-действие).

Для обновления значений оценок применяется обучение нейронной сети. В качестве алгоритма обучения можно использовать алгоритм обратного распространения ошибки [12]. При этом примером для обучения сети является пара из входных данных и новой оценки.

Выводы по главе 1

1. Методы обучения с подкреплением являются обширным классом методов машинного обучения, основанных на обучении агента через непосредственное взаимодействие с окружающей средой.

2. Методы обучения с подкреплением зарекомендовали себя как эффективные методы создания успешных агентов для различных игр.

ГЛАВА 2. ЗАДАЧА ПОСТРОЕНИЯ ТАНКА ДЛЯ ИГРЫ «ROBOCODE»

2.1. Описание игры «Robocode»

«Robocode» – это популярная компьютерная игра, разработанная М. Нельсоном в 2001 году. Игра представляет собой соревнование роботов-танков на игровом поле. Пример игрового поля с элементами пользовательского интерфейса представлен на рис. 3.

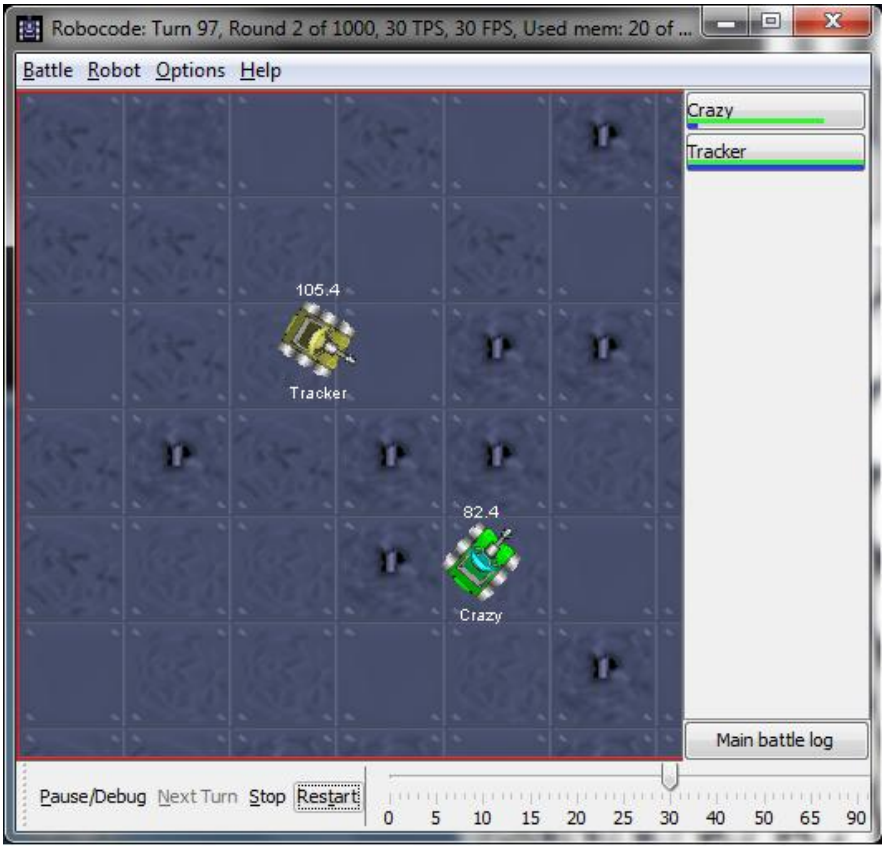

Рис. 3. Игра «Robocode»

Танк состоит из корпуса, способного перемещаться по полю, радара, способного определять позиции других танков на поле, и пушки, с помощью которой танк стреляет в своих соперников. Схема танка изображена на рис. 4.

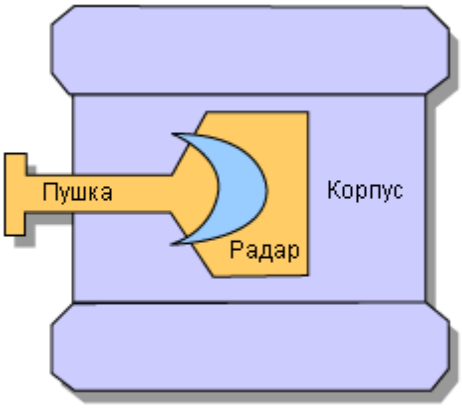

Рис. 4. Схема танка в игре «Robocode»

Программно танк является классом, написанным на языке *Java* или языках *.NET*. Данный класс управляет движениями всех частей танка – корпуса, радара и пушки, стрельбой, а также занимается обработкой поступающих событий. К числу обрабатываемых событий относятся события от радара, события, связанные с попаданием снарядов в танк, поражением других танков, столкновения и т.д.

Каждый танк в начале игры имеет 100 единиц энергии. За каждое попадание в противника, танк получает дополнительную энергию. При попадании снаряда в танк часть энергии теряется. Также танк теряет часть энергии при столкновениях со стенами и с другими танками.

Танк проигрывает в поединке (*раунде*), если энергия танка уменьшается до нуля. Игра состоит из последовательности раундов. По завершении всех раундов победитель определяется по числу набранных за игру баллов. Число баллов зависит от того, насколько успешно танк поражал другие танки и избегал попадания снарядов в себя.

Цель игры – создать танк, побеждающий другие танки.

2.2. Постановка задачи

Задача, решаемая в данной работе, состоит в разработке метода *автоматического* построения танка для игры «Robocode», побеждающего

заданного противника – стандартный танк из поставки игры (`sample.Tracker`, `sample.Fire` и `sample.Walls`).

2.3. Известные подходы к решению задачи построения танка

Наиболее распространенный подход для создания роботов-танков – написание их кода вручную. Данный подход может быть очень трудоемким – число строк кода в известных танках измеряется десятками тысяч. Поэтому исследователей привлек вопрос об автоматическом создании танков для игры «Robocode».

В работах [9, 14] предложен способ представления логики робота-танка в виде конечных автоматов.

В работе [10] исследуется задача автоматического создания управляющих автоматов для роботов-танков при помощи генетического программирования, а в работе [6] предлагается метод двухэтапного генетического программирования для построения автомата. Суть метода состоит в разбиении задачи построения робота-танка на два этапа. На первом этапе с помощью генетического программирования строятся состояния управляющего автомата, а на втором – сам автомат.

Выводы по главе 2

1. В игре «Robocode» ставится задача построения робота-танка, побеждающего другие танки.

2. Существует ряд подходов к автоматическому построению роботов-танков. Подход, основанный на применении генетического программирования, позволяет строить управляющие автоматы для роботов-танков.

ГЛАВА 3. ПРЕДЛАГАЕМЫЙ МЕТОД ПОСТРОЕНИЯ ТАНКА

3.1. Общая идея метода

Предлагаемый в данной работе подход к решению задачи построения танка основан на идеях из работ [6, 9, 10]:

1. Логика танка представляется в виде конечного автомата.
2. Построение автомата производится в два этапа – построение состояний автомата и создание автомата.

В качестве методов, используемых на первом и втором этапах, предлагается использовать методы обучения с подкреплением [3].

На первом этапе робот обучается выполнению следующих действий:

- наведение пушки на цель;
- преследование цели;
- уклонение («убегание») от цели.

Обучение каждому действию производится с помощью методов обучения с подкреплением независимо от других действий. Данные действия соответствуют трем состояниям управляющего автомата.

Для представления общей стратегии игры робота используется управляющий конечный автомат, схема которого изображена на рис. 5.

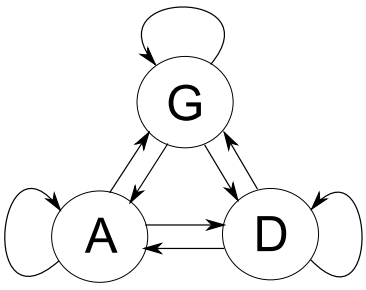

Рис. 5. Схема управляющего автомата

Состоянию A соответствует преследование цели, D – уклонение от цели, G – наведение пушки и стрельба. Находясь в каком-либо состоянии, робот выполняет соответствующее данному состоянию действие. В процессе игры робот переходит из состояния в состояние на основе информации о положении противника и других параметров.

Цель второго этапа – найти функцию переходов управляющего автомата, и, тем самым, получить общую стратегию игры робота, приводящую к победе над противником.

3.2. Построение состояний управляющего автомата

Состояния автомата соответствуют простым действиям, используемым роботом:

- наведение пушки на цель и стрельба;
- преследование цели;
- уклонение от цели.

На первом этапе состояния строятся независимо друг от друга. Для построения состояний используется *Q*-обучение.

Результатом работы *Q*-обучения являются найденные значения *Q*-функции, которая может быть представлена в виде таблицы или в виде нейронной сети. Таким образом, состояние автомата представляется либо в виде таблицы (набор всех значений *Q-функции*), либо в виде нейронной сети, вычисляющей *Q-функцию*. Схема нейронной сети, использованной в данной работе, изображена на рис. 6.

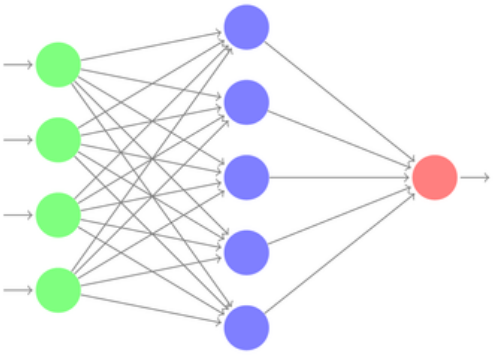

Рис. 6. Схема нейронной сети

Нейронная сеть имеет *входной слой* нейронов (выделен зеленым), *скрытый слой* (выделен синим) и *выходной нейрон* (выделен красным).

3.2.1. Наведение пушки

В терминах обучения с подкреплением задача наведения пушки на цель формулируется следующим образом:

- состояние среды: угол между пушкой и целью;
- действия: поворот пушки вправо/влево на 10 градусов и на два градуса, сохранение текущего направления пушки;
- награда: +2 за наведение пушки на цель.

3.2.2. Преследование цели

Формулировка задачи обучения с подкреплением:

- состояния: угол между направлением движения и направлением на цель и расстояние до цели;
- действия: сохранение направления движения или поворот влево/вправо на 10 градусов;
- награда: +4 за приближение к цели.

3.2.3. Уклонение от цели

В данном эксперименте производится обучение робота «убеганию» от противника.

Формулировка задачи обучения с подкреплением:

- состояния: относительное положение и относительное положение ближайшего препятствия (стены);
- действия: сохранение направления движения или поворот влево/вправо на 10 градусов;
- награда: -5 за приближение к противнику, -10 за столкновение со стеной.

3.3. Построение управляющего автомата

Управляющий автомат (рис. 5) состоит из трех состояний: A – состояние, в котором робот преследует цель, D – состояние, в котором робот уклоняется от цели, G – состояние, в котором робот наводит пушку на цель и стреляет.

Все три состояния были построены на предыдущем этапе. Цель второго этапа – построить функцию переходов автомата.

Задача поиска функции переходов автомата формулируется как задача обучения с подкреплением:

- состояния: номер текущего состояния и расстояние до цели;
- действия: переход в состояние A, D, G;
- награда: +5 за победу в раунде, -5 за поражение.

В качестве метода обучения с подкреплением используется $Q(\lambda)$-обучение.

Выводы по главе 3

1. Предлагаемый метод основан на идее двухэтапного построения управляющего автомата [6].

2. Для построения состояний автомата и функции переходов предлагается использовать обучение с подкреплением.

ГЛАВА 4. РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТОВ

4.1. Схема эксперимента

Робот-танк строится в два этапа.

На первом этапе путем решения соответствующих задач обучения с подкреплением (разд. 3.2) строятся состояния автомата. Результатом обучения является таблица или нейронная сеть. Полученный результат можно оценить численно – оценкой является средняя награда, получаемая роботом, использующим данную таблицу или сеть, за один шаг.

Состояния автомата также строятся с помощью генетического программирования. В данном случае особью является нейронная сеть, вычисляющая *Q*-функцию. Оценка особей производится путем сравнения среднего значения награды за один шаг.

Результаты, полученные с помощью обучения с подкреплением, сравниваются с результатами, полученными применением генетического программирования.

На втором этапе эксперимента находится функция переходов автомата. Задача решается методом обучения с подкреплением (разд. 3.3), а также генетическим программированием – особью является тройка нейронных сетей, соответствующих каждому состоянию автомата. Находясь в некотором состоянии автомата, робот использует соответствующую данному состоянию нейронную сеть для выбора следующего состояния автомата. Вычисляются выходные значения сети для трех входов, кодирующих возможное следующее состояние и расстояние до цели. Следующее состояние определяется наибольшим полученным значением.

Роботы обучаются (выращиваются) против конкретных роботов-противников (`sample.Tracker`, `sample.Fire` и `sample.Walls`). Полученные роботы сравниваются друг с другом по результатам игры с роботами-противниками.

Схема построенного автомата изображена на рис. 7.

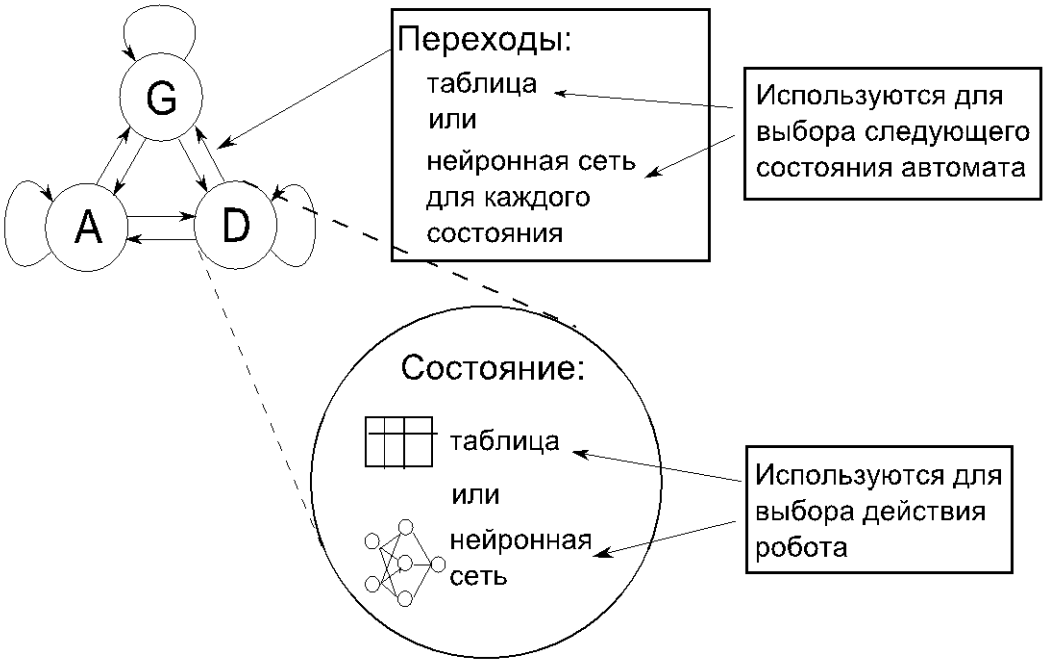

Рис. 7. Схема построенного автомата

4.2. Построение состояний управляющего автомата

4.2.1. Наведение пушки

Обучение проводилось с использованием нейронной сети. На рис. 8 красным и зеленым изображены графики значений средней награды за один шаг при использовании обучения (красный график) и генетического программирования (зеленый график). По горизонтали на графиках отложено время, прошедшее с начала эксперимента, по вертикали – средняя награда. Робот обучился наведению пушки – значение средней награды 1,35 означает, что на большинстве шагов пушка была направлена на цель. При использовании генетического программирования получено решение со средней наградой 0,7.

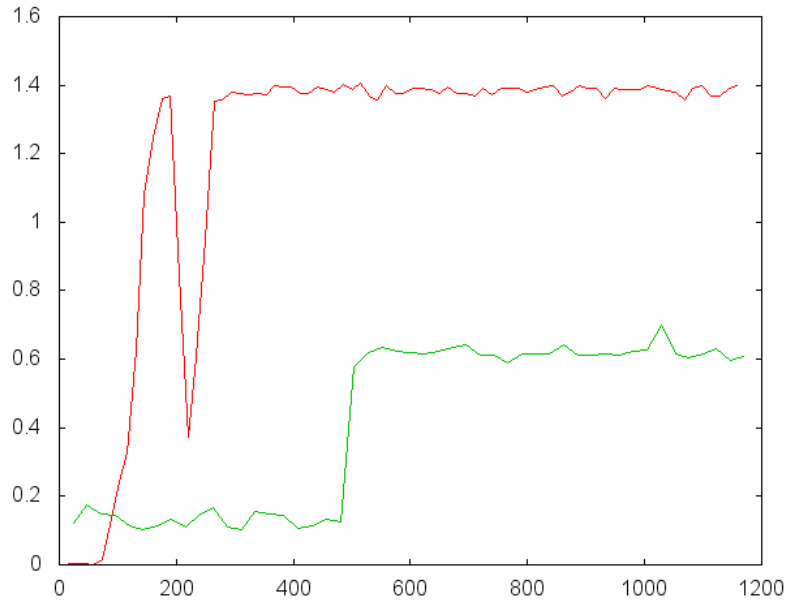

Рис. 8. Графики средней награды в задаче наведения пушки

4.2.2. Преследование цели

На рис. 9 приведены результаты обучения и генетического программирования. Обучение проводилось с использованием таблиц.

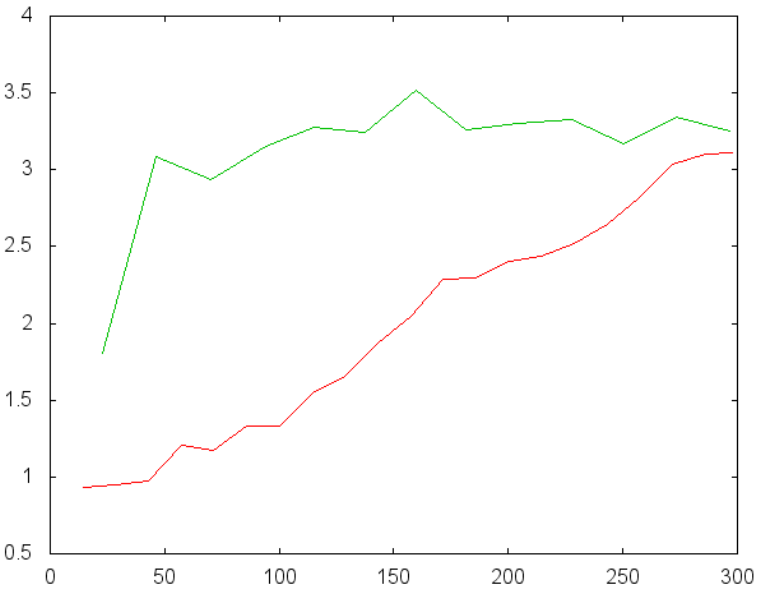

Рис. 9. Графики средней награды в задаче преследования цели

С помощью обучения была достигнута средняя награда за шаг 3.13. Метод генетического программирования быстро находит решение со средней наградой за шаг – 3,51.

4.2.3. Уклонение от цели

Результаты эксперимента приведены рис. 10.

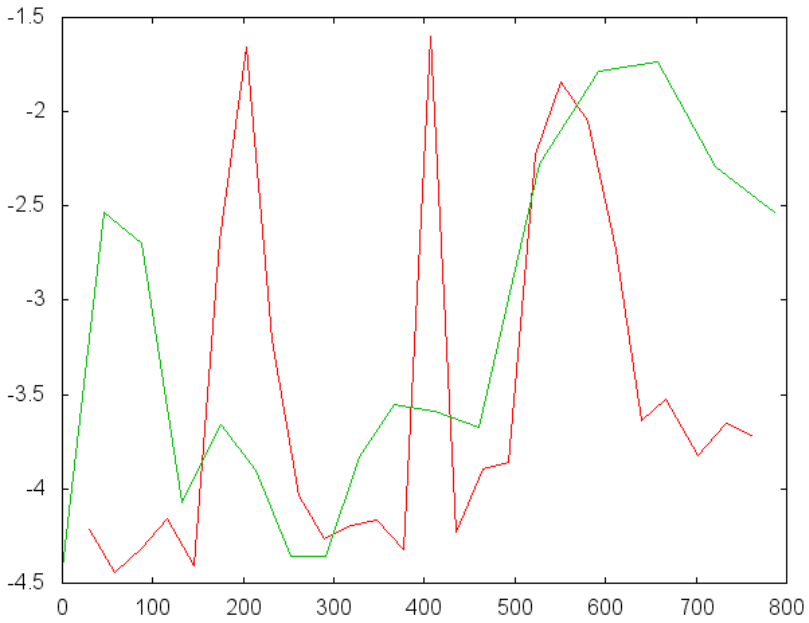

Рис. 10. График средней награды в задаче уклонения от противника

Наблюдаются сильные колебания средней награды в процессе обучения. Через 406 с получена нейронная сеть, соответствующая средней награде -1.6. Робот, использующий данную сеть, двигается в сторону от противника, почти не сталкиваясь со стенами.

С помощью генетического программирования получено решение со значением средней награды -1.7.

4.3. Построение робота-танка

Роботы `sample.Tracker` и `sample.Fire` являются стандартными роботами поставки и реализуют простые стратегии игры: `sample.Tracker` сначала пытается приблизиться к цели, затем наводит пушку и стреляет, `sample.Fire` – не перемещается по полю, вращает пушку и стреляет из нее при обнаружении цели.

Оба метода (генетическое программирование и обучение с подкреплением) быстро справляются с построением роботов, побеждающих `sample.Tracker` и `sample.Fire`. С помощью

генетического программирования уже на втором поколении строится робот, побеждающий `sample.Tracker` с отношением полученных баллов к общей сумме баллов 0,7. Робот наводит пушку, стреляет в цель и не перемещается по полю. Обучение с подкреплением за 500 раундов строит робота, демонстрирующего такую же стратегию игры. Результаты обучения против `sample.Tracker` и `sample.Fire` приведены в таблице

Таблица. Сравнительные результаты

Робот	RL	GP
sample.Tracker	8694:21471 (0.71)	9042:21253 (0.70)
sample.Fire	7512:22378 (0.75)	3264:18774 (0.85)

В таблице указаны баллы, полученные роботами после 100 раундов игры, в скобках указано отношение баллов, полученных построенным роботом, к общей сумме баллов.

Робот `sample.Walls` имеет более сложное поведение – двигаясь вдоль стен, робот направляет пушку в сторону поля и стреляет в цель. С помощью генетического программирования было получено решение с отношением баллов (к общей сумме) 0,62. С помощью *Q*-обучения за 500 раундов построен робот, побеждающий `sample.Walls` с отношением полученных баллов 0,71.

Выводы по главе 4

1. Методы обучения с подкреплением и генетического программирования успешно справляются с задачей построения управляющего автомата для робота-танка.

2. Результаты, полученные с помощью обучения с подкреплением, и результаты, полученные применением генетического программирования, отличаются незначительно.

ЗАКЛЮЧЕНИЕ

В работе была рассмотрена задача построения робота-танка для компьютерной игры «Robocode».

Основными требованиями к танку были автоматическое построение, успешная игра против стандартных танков.

Задача была успешно решена – построен танк, удовлетворяющий данным требованиям.

Для решения задачи был предложен подход, основанный на двухэтапном построении управляющего автомата с помощью обучения с подкреплением.

Сравнение результатов применения *Q*-обучения с результатами, полученными методом генетического программирования, показывает, что на обоих этапах результаты работы методов не сильно отличаются друг от друга. В задаче наведения пушки *Q*-обучение показало лучший результат, в то время как в задаче преследования цели лучший результат показывает метод генетического программирования. В задаче построения автомата против `sample.Fire` лучший результат показало генетическое программирование, а при построении автомата против `sample.Walls` – *Q(λ)*-обучение.

Полученные данные позволяют утверждать, что методы обучения с подкреплением применимы к построению управляющих автоматов для роботов-танков и не уступают в эффективности генетическому программированию.

ИСТОЧНИКИ

1. *Рассел С., Норвиг П.* Искусственный интеллект. Современный подход. М.: Вильямс. 2006.
2. Компьютерная игра «Robocode». Официальный сайт. (http://robocode.sourceforge.net)
3. *Sutton R.S., Barto A.G.* Reinforcement Learning. An Introduction. The MIT Press, 1998.
4. *Царев Ф. Н., Шалыто А. А.* Применение генетического программирования для генерации автомата в задаче об «умном муравье» /Сборник трудов IV-ой Международной научно-практической конференции «Интегрированные модели и мягкие вычисления в искусственном интеллекте». Т. 2. 2007, с. 590–597. http://is.ifmo.ru/genalg/_ant_ga.pdf
5. *Царев Ф. Н.* Разработка метода совместного применения генетического программирования и конечных автоматов. Бакалаврская работа. СПбГУ ИТМО, 2007. http://is.ifmo.ru/papers/_2010_03_03_tsarev.pdf
6. *Соколов Д. О.* Применение двухэтапного генетического программирования для построения автомата, управляющего моделью танка в игре «Robocode». Бакалаврская работа. СПбГУ ИТМО, 2009. http://is.ifmo.ru/papers/_sokolov_bachelor.pdf
7. *Хопкрофт Д., Мотвани Р., Ульман Д.* Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2008.
8. *Mitchell M.* An Introduction to Genetic Algorithms. The MIT Press, 1996.
9. *Туккель Н. И., Шалыто А. А.* Система управления танком для игры Robocode. Объектно-ориентированное программирование с явным

выделением состояний. Программная документация. http://is.ifmo.ru/projects/tanks/

10. *Бедный Ю.Д.* Применение генетических алгоритмов для генерации автоматов при построении модели максимального правдоподобия и в задачах управления. Магистерская диссертация. СПбГУ ИТМО, 2008. http://is.ifmo.ru/papers/bednij/masters.pdf
11. *Tesauro G.* TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play //Neural Computation. 1994. 6(2): 215–219.
12. *Хайкин С.* Нейронные сети. Полный курс. М.: Вильямс, 2006.
13. *Koza J.R.* Genetic programming: on the programming of computers by means of natural selection. MIT Press, 1992.
14. *Кузнецов Д. В., Шалыто А. А.* Система управления танком для игры «Robocode». Курсовая работа. СПбГУ ИТМО. http://is.ifmo.ru/projects/robocode2/