УДК 004.8

АДАПТИВНОЕ ПОВЕДЕНИЕ ПРОГРАММНЫХ АГЕНТОВ В МУЛЬТИАГЕНТНОЙ КОМПЬЮТЕРНОЙ ИГРЕ

И. Ю. Сотников, И. В. Григорьева

ADAPTIVE BEHAVIOUR OF SOFTWARE AGENTS IN MULTI-AGENT COMPUTER GAME

I. Yu. Sotnikov, I. V. Grigorieva

Работа выполнена на основании Государственного задания № 2014/64.

В работе рассмотрены мультиагентные игры, их структура, программное представление агентов. Игры представляют разные жанры, имеют различные правила и структуру мира. В качестве примеров использования игр как площадки для изучения методов искусственного интеллекта рассмотрены агенты с использованием алгоритмов поиска пути и обучающиеся агенты. В качестве алгоритмов поиска пути были взяты алгоритм A* и алгоритм Ли. Для реализации обучающихся агентов были выбраны такие методы машинного обучения, как обучение с подкреплением и искусственные нейронные сети, обучаемые с помощью генетического алгоритма.

The paper consider multi-agent games, their structure, software presentation of agents. Games present different genres, have different rules and structure of the world. As examples of use of games as a platform for the study of artificial intelligence, agents are considered with the use of search algorithms path and learner agents. Algorithm A* and Lee algorithm were taken as path search algorithms. To implement learner agents machine-learning techniques were selected such as reinforcement learning and artificial neural network taught by a genetic algorithm.

Ключевые слова: искусственный интеллект, компьютерные игры, агентно-ориентированный подход, поиск пути, машинное обучение.

Keywords: artificial intelligence, computer games, agent-oriented approach, search path, machine learning.

Введение

Данная работа посвящена адаптивному поведению игровых программных агентов. Существует класс игр, в которых в качестве игроков выступают подпрограммы, называемые игровыми ботами, или игровыми программными агентами, именно такие игры и рассматриваются в данной работе. Такие игры представляют собой полигон для творчества исследователей в области искусственного интеллекта. Для таких исследователей проводятся чемпионаты по реализации ботов для игр. Например, в России это Russian AI Cup (*http://russianaicup.ru*), соревнования от компании Google – Google AI Challenge (*http://aichallenge.org*), соревнования по игре Super Mario Bros – Mario AI Championship (*http://www.marioai.org*). Такие соревнования привлекают программистов разного уровня. Для людей с небольшим опытом, таких как студенты, это прекрасная возможность расширить и углубить свои знания в области искусственного интеллекта.

Искусственный интеллект имеет множество приложений в различных областях. Компьютерные игры стали объектом исследования, в том числе и в контексте искусственного интеллекта, еще в начале 50-х годов. Игровой искусственный интеллект использует многие техники классического искусственного интеллекта, которые также применяются и в других областях: поиск в пространстве состояний, нейронные сети, нечеткая логика и др. В настоящее время в разработке игр все больше применяются методы классического искусственного интеллекта, растет интерес академического сообщества к технологиям, используемым в компьютерных играх [4].

Доминирующим подходом к разработке интеллектуальных систем является агентно-ориентированный подход. Основополагающими его концепциями являются понятия агента и среды, в которой агент находится. Агентом является всё, что может рассматриваться как воспринимающее свою среду с помощью датчиков и воздействующее на эту среду с помощью исполнительных механизмов [6]. Игровой агент взаимодействует со средой согласно правилам игры, которые имеют различную сложность, при этом свойства среды также являются определяющими для поведения агента. Описывая игры с различными правилами и различными свойствами игрового мира, создаются модели миров произвольной сложности, для которых необходимо создать эффективного и возможно обучающегося агента. Именно поэтому игры представляют прекрасный полигон для апробаций различных методов искусственного интеллекта.

В мультиагентных играх принимают участие два или более игроков-подпрограмм, которые конкурируют за некоторый ресурс в игровом поле. Игровое поле представляет собой некоторое ограниченное пространство, по которому некоторым образом распределены препятствия, угрозы в виде различных элементов, наносящих игрокам урон, а также сами ресурсы. Задача игроков заключается в том, чтобы опередив соперника получить ресурсы и при этом избежать угроз. Как правило, игрокам доступны различные действия, выбор следующего действия игроки осуществляют на основании доступной информации о среде и конкурентах, а также на основании собственной истории актов восприятия. Среда, в которой существуют игровые агенты, может быть полностью или частично наблюдаемой, когда агент видит все игровое поле или не видит каких-то его областей из-за препятствий или видит ограниченный набор квадратов около себя. Среда может быть стохастической, а

может такой казаться из-за того, что среда является частично наблюдаемой. Практически все игровые среды являются конкурентными, последовательными, при этом могут быть как дискретными, так и непрерывными. Частично наблюдаемые, стохастические, динамические среды являются особенно сложными, так как в условиях недостатка информации об изменяющемся мире агенту приходится поддерживать некоторую внутреннюю модель мира, чтобы иметь представление об его состоянии и изменениях. Реализация агентов для таких игр представляет собой сложную задачу.

В рамках данной работы была предпринята попытка пройти полный путь создателя компьютерной мультиагентной игры, начиная от описания правил и программной реализации игры до построения и апробации различных игровых агентов. В результате были созданы две игры [8; 9].

Первая игра “Worms Strategy” представляет собой стратегию в реальном времени для двух и более игроков. В ней несколько колоний червей борются в ограниченном пространстве за единственный ресурс в игре – пищу. Первая игра стала пробным пером, в ходе ее реализации постепенно формировался взгляд на применение мультиагентной модели в игровом приложении. Также были рассмотрены различные подходы в реализации движка для игры. Для этой игры были реализованы два агента с использованием алгоритмов поиска пути A* и алгоритма Ли [1 – 2; 6]. После проведенных экспериментов было решено перейти к изучению методов машинного обучения. Для данной игры апробировать эти методы оказалось сложной задачей из-за большого количества управляемых одной колонией сущностей и большого количества правил игры. Поэтому они были исследованы на примере агентов для второй игры.

Вторая игра “Collector Stars” представляет собой двумерную игру в жанре платформер для одного или нескольких игроков. Вторая игра имеет более простую структуру и правила, по сравнению с первой. Она создавалась с использованием библиотеки Qt, которая во много упростила реализацию движка. Для реализации игровых агентов использовались такие инструменты машинного обучения, как обучение с подкреплением и искусственные нейронные сети [6 – 7; 10], обучаемые с помощью генетического алгоритма [3; 7]. Так как данная работа посвящена исследованию адаптивного поведения, то далее рассматривается только вторая игра.

1. Правила игры

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 (рис. 1).

Правила игры:

– Игра будет завершена, как только будут пройдены все эпизоды.

– Переход к следующему эпизоду будет совершен, как только все Собиратели завершат текущий.

– При пересечении Собирателем ячейки со звездой она будет подобрана, количество собранных звезд для него будет увеличено. Ячейка со звездой сменит тип на проходимую ячейку.

– При пересечении Собирателем ячейки с черной дырой он погибнет. Эпизод для него считается завершенным.

– При пересечении ячейки с дверью текущий эпизод для Собирателя будет завершен.

– При пересечении несколькими Собирателями одной и той же ячейки со звездой выбор, кому ее засчитать, будет сделан случайным образом.

Цель игры – набрать суммарно за все эпизоды наибольшее количество очков. Количество очков зависит от количества собранных звезд и времени, затраченного на прохождение.

2. Программная реализация

Игра реализована с помощью языка C++ и библиотеки Qt. На рисунке 2 представлена полная диаграмма программных классов, отображающая архитектуру игры.

Классы Map и Collector представляют объекты: карту и собирателя. Класс CollectorAgent является связующим между объектами классов, представляющих агента IAgent и Собирателя. Игровой агент реализуется путем создания класса, унаследованного от базового класса IAgent. Класс GameData является заместителем классов Map и Collector. Он контролирует доступ к ним с учетом ограничений для агента. Объект этого класса создается классом CollectorAgent, который затем передает его агенту. В классе Game реализована базовая логика игры, взаимодействие с агентами.

Для визуализации использовалась модель графического представления Qt graphics model view. Графическое представление предоставляет основанный на элементах подход к программированию модель-представление. Для этого используются три основных класса: QGraphicsItem – базовый класс для графических элементов сцены, класс QGraphicsScene выступает в качестве контейнера для объектов QGraphicsItem и QGraphicsView, последний визуализирует содержимое сцены. Классы MapView и CollectorView являются наследниками класса QGraphicsItem. Они хранят указатели на объекты классов Map и Collector и используют данные, получаемые от них для визуализации карты и Собирателя, соответственно. Класс GameView – это наследник класса QGraphicsView. В нем переопределены функции для рисования заднего и переднего планов.

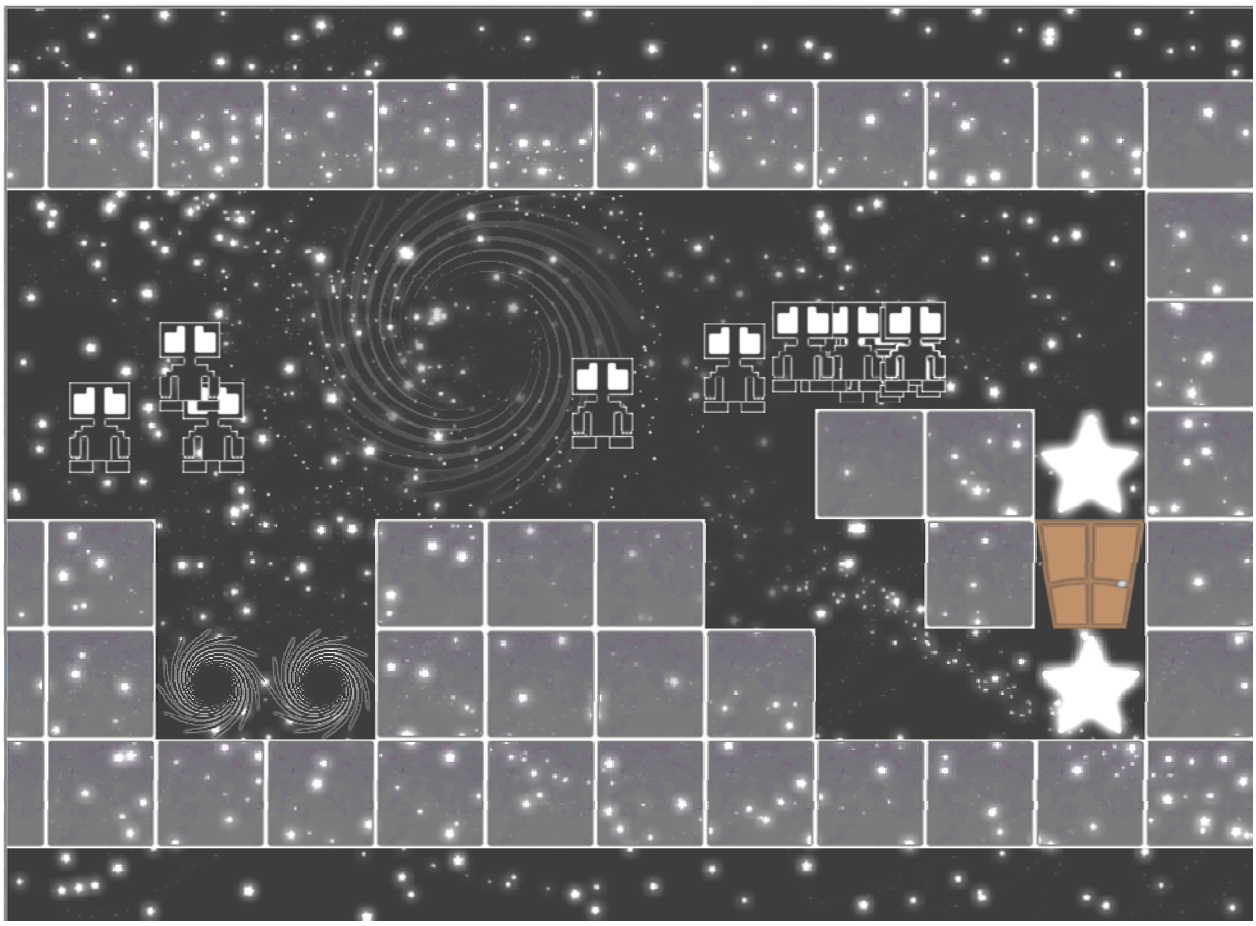

Рис. 1. ***Скриншот игрового момента***

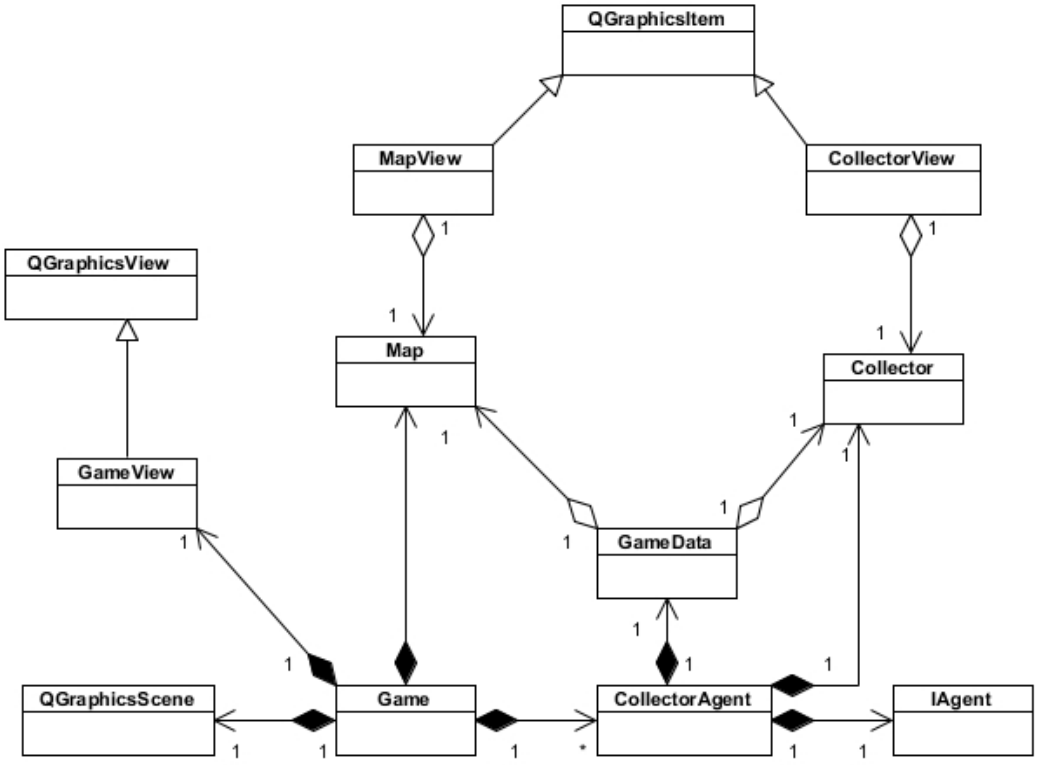

Рис. 2. ***Диаграмма классов игры***

3. Обучающиеся агенты

3.1. Обучающийся агент с использованием обучения с подкреплением

При размышлении о том, как реализовать обучающегося агента, можно провести аналогии с человеческой жизнью, с тем, как человек взаимодействует со своей средой и учится в процессе этого взаимодействия. При этом человек получает огромное количество информации, в том числе и о последствиях своих действий, она позволяет определить, что нужно сделать для достижения некоторой цели. Обучение с подкреплением позволяет определить, в каких ситуациях, какие действия предпринимать лучше всего. Агент может ничего не знать о среде, в которой находится, и лишь получать вознаграждение или наказание в виде числа за свои действия, которые привели его в текущее состояние [11].

Основными элементами обучения с подкреплением является агент и среда, в которой он находится. Кроме них, есть также три основных подэлемента: *стратегия*, *функция вознаграждения* и *функция полезности*. На каждом шаге временной последовательности агент получает текущее состояние среды. За уже выполненное действие, приведшее в некоторое состояние, агент получает вознаграждение. Далее состояние отображается в вероятность выбора каждого действия. Это отображение есть *стратегия* агента. Стратегия корректируется в процессе обучения на основе предыдущего опыта.

В игре “Collector Stars” карта представлена сеткой ячеек. Но персонаж перемещается не строго из ячейки в ячейку и может находиться на пересечении двух или четырех ячеек одновременно. Кроме того, персонаж может стоять на поверхности блока или падать вниз, могут возникнуть ситуации, когда персонажу нужно переместиться в одном направлении, а затем вернуться тем же путем и др. Поэтому для описания состояний игры был задан набор атрибутов, которые могут иметь значение ИСТИНА или ЛОЖЬ:

1. Персонаж находится на земле.
2. Наличие препятствий, звезд, черных дыр в каждой из восьми ячеек, вокруг текущей.
3. Строго положительное/отрицательное значение скорости по вертикали/горизонтали.
4. Персонаж находится с левого/правого/нижнего/верхнего края текущей ячейки.
5. Ближайший соперник находится слева/справа/снизу/сверху относительно персонажа.

Все атрибуты представляются в виде числовых масок, из которых определяется число, которое и будет состоянием.

Агенту доступно три основных действия: перемещения влево, вправо и прыжок. Падение без горизонтальных перемещений было также выделено как действие. В некоторых состояниях доступны не все действия. Так, например, если Собиратель находится в воздухе, то прыжок будет недоступным.

Основная цель агента – добраться до целевой ячейки с дверью, набрав наибольшее количество очков. После ее пересечения, уровень считается успешно пройденным.

После череды экспериментов был подобран приемлемый вариант функции вознаграждения: если агент успешно завершил уровень, то будет возвращено значение равное 100+*SCORES*, где *SCORES* – количество набранных агентом очков; иначе, если агент погиб, то будет возвращено значение равное -50; иначе агент будет наказан за повтор одних и тех же действий в тех же состояниях, а именно будет возвращено значение, вычисляемое по формуле $e^{-N(s,a)} - 1$, где $N(s,a)$ – таблица, хранящая количество выполнения действия a в состоянии s.

Для определения стратегии агента существуют различные алгоритмы. В качестве испытуемых были взяты алгоритм SARSA и модифицированный R-learning с γ.

Алгоритм SARSA является одношаговым и представлен следующим соотношением:

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

где Q представляет собой двумерную таблицу, состоящую из пар $(s,\ a)$. Для каждой она хранит оценку действия a в состоянии s. Коэффициент α – множитель обучения. Чем он больше, тем сильнее агент доверяет новой информации, чем меньше, тем больше полагается на старый опыт. Коэффициент γ - множитель дисконтирования. Чем он меньше, тем меньше агент задумывается о выгоде от будущих своих действий.

Алгоритм R-learning представлен следующим соотношением:

$$Q(s_t, a_t) += \alpha(r - \rho + \gamma Q(s_{t+1}, a_{t+1}) - \\ Q(s_t, a_t)),$$

но если

$$Q(s_t, a_t) = Q(s_{t+1}, a_{t+1}), \text{ то}$$

$$\rho += \beta(r - \rho + Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)).$$

Подходящие значения параметров для алгоритмов SARSA и R-learning определялись путем подбора. В результате тестирования алгоритмов SARSA и R-learning с наилучшими из апробированных значениями параметров по количеству очков оба алгоритма имеют схожие результаты. По времени обучения алгоритм SARSA работает быстрее.

Для определения следующего действия агента была выбрана жадная ε-стратегия, где ε представляет собой вероятность выбора случайного действия. Значение ε линейно уменьшается на промежутке до $0.8 \cdot EP$, где *EP* – количество эпизодов обучения. На оставшемся промежутке обучение происходит с нулевым значением ε.

3.2. Обучающийся агент с использованием искусственной нейронной сети

Для реализации агента на основе нейронной сети необходимо определиться с видом сети и алгоритмом обучения. На вход искусственной нейронной сети подается набор атрибутов текущего состояния. Значение первого входа зависит от того, находится ли пер-

сонаж 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. Значения для разных типов ячеек представлены в таблице 1.

Таблица 1

Значения входящих коэффициентов для различных типов ячеек

Тип ячейки	*Значение*
Пустая проходимая ячейка	0.0
Ячейка со звездой	0.25
Блок	0.5
Ячейка с черной дырой	0.75
Ячейка с дверью	1.0

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*.

Для обучения нейронной сети 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).

Ячейки около персонажа являются входами сети, значение которых зависит от их типа. Размер матрицы ячеек задается. Были апробированы матрицы ячеек размеров *3x3*, *5x5* и *9x9*. Сеть не имела скрытых слоев. Эксперименты проводились для разного количества хромосом в популяции и поколений. Так как для вычисления приспособленности каждой хромосоме требуется прохождение эпизода, то общее количество эпизодов равняется $P \cdot G$, где P – размер популяции, G – количество поколений. По среднему значению количества очков и времени обучения матрица размера *3x3* показала лучшие результаты. Количество скрытых слоев и нейронов в каждом слое также подбиралось экспериментально. По среднему значению количества очков сеть с одним скрытым слоем и четырьмя нейронами на нем показывает лучший результат.

4. Эксперименты с обучающимися агентами

Для экспериментов использовалась карта, схематично представленная на рисунке 3. В экспериментах агенты обучались без конкуренции, то есть без участия других агентов.

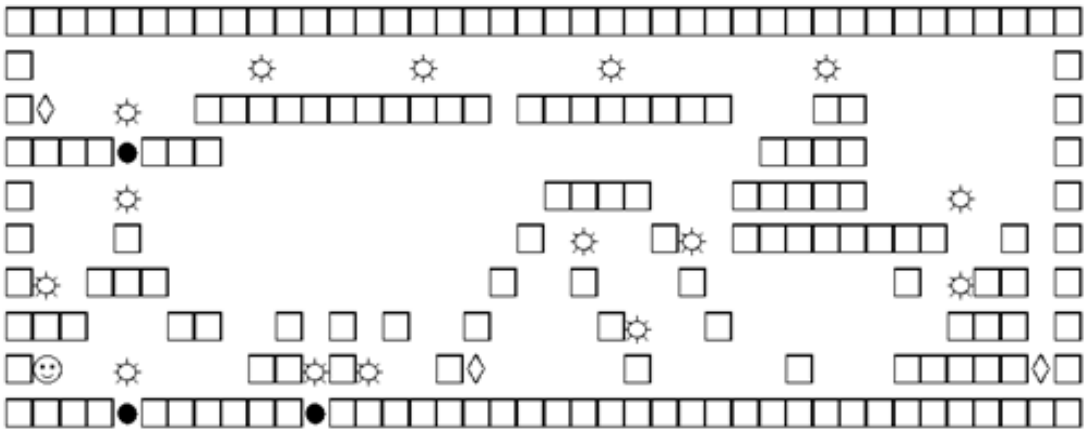

Рис. 3. ***Схема тестируемой карты.*** *□ – блок,* ☼ *– звезда,* ☺ *– Собиратель,* ◊ *– дверь,* ● *– черная дыра*

Сравнение производится по двум показателям: количеству очков и времени обучения. Положительное значение первого показателя говорит о том, что агент проходит уровень, лучший агент при этом должен набрать большее количество очков. Второй показатель – время обучения, имеет меньшее значение, тем не менее лучшим считается агент, обучающийся за более короткое время.

Чтобы сравнить насколько хорошо обучающиеся агенты проходят уровень, был добавлен еще один агент, который действует случайным образом. Действия выполняются некоторое количество итераций, определяемое случайным образом из отрезка *[0;120]*. Шестьдесят итераций – это примерно секунда игрового времени. За одну секунду Собиратель может пройти примерно четыре ячейки и/или прыгнуть на высоту одной ячейки. Если агент активизировал действие “движение влево”, то “движение вправо” уже активизировано не будет. Агенту дается заданное количество эпизодов на поиск лучшего набора действий, который приведет его к лучшему результату по количеству очков.

Графики зависимости количества очков и времени обучения от количества пройденных эпизодов представлены на рисунках 4 и 5 соответственно. Агенты были взяты с параметрами, описанными выше. Как видно из вышеприведенных графиков, по количеству очков нейросетевой агент имеет результаты лучше в среднем в *1.5* раза, чем агент с использованием алгоритма R-learning, и в *10* раз лучше результатов случайно действующего агента. По времени обучения нейросетевой агент имеет результаты лучше, чем агент с использованием алгоритма R-learning в среднем в *2.3* раза. Преимущество случайно действующего агента в данном случае во времени поиска решения.

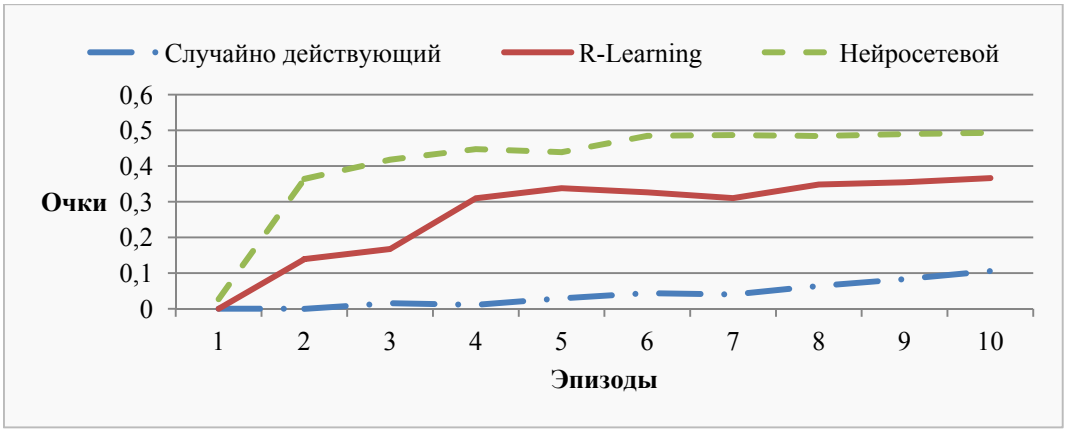

Рис. 4. ***Графики зависимости количества очков от пройденных эпизодов для различных агентов***

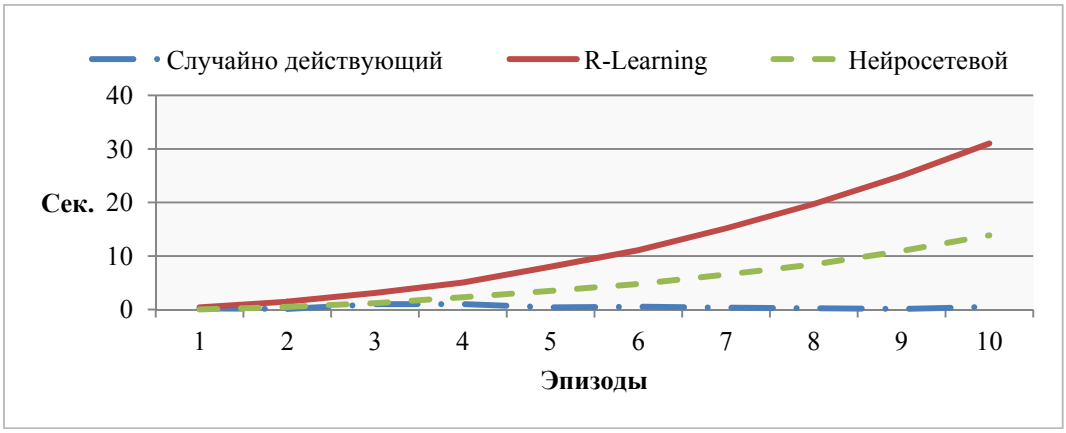

Рис. 5. ***Графики зависимости времени обучения от пройденных эпизодов для различных агентов***

Заключение

Результатом этой работы являются созданные две мультиагентные игры и два реализованных обучающихся агента для второй игры. Первая игра “Worms Strategy” представляет собой стратегию в реальном времени для двух и более игроков. В ней несколько колоний червей борются в ограниченном пространстве за единственный ресурс в игре – пищу, но построение обучающихся агентов для этой игры оказалось слишком сложной задачей из-за большого количества управляемых одной колонией сущностей и большого количества правил игры. Поэтому обучающиеся агенты были исследованы во второй, реализованной в рамках данной работы, игре “Collector Stars”, которая

представляет собой двумерную игру в жанре платформер для одного или нескольких игроков. Вторая игра имеет более простую структуру и правила, по сравнению с первой.

На примере игры “Collector Stars” изучены следующие методы машинного обучения: два алгоритма обучения с подкреплением, искусственные нейронные сети, обучаемые с помощью генетического алгоритма. Для построенных агентов были подобраны приемлемые значения параметров. Полученные в результате агенты сравнивались между собой и со случайно действующим агентом по количеству очков и времени обучения. Оба обучающихся агента продемонстрировали принципиально лучшие показатели эффективности по сравнению со случайно действующим агентом, но при этом на обучение было потрачено существенное время. По обоим показателям нейросетевой агент получил результаты лучше, чем агент, обучающийся с подкреплением.

Необходимо отметить, что все агенты обучались в условиях отсутствия конкурирующих агентов. Построение обучающихся агентов в условиях конкуренции – тема дальнейших исследований.

Литература

1. Алгоритмы поиска пути. Режим доступа: http://pmg.org.ru/ai/stout.htm#listing1
2. Алгоритм Ли. Режим доступа: http://ru.wikipedia.org/wiki/Алгоритм_Ли
3. Генетический алгоритм. Режим доступа: http://www.aiportal.ru/articles/genetic-algorithms/1/
4. Искусственный интеллект в компьютерных играх. Многоуровневое планирование и реактивное поведение агентов. Режим доступа: http://masters.donntu.edu.ua/2013/fknt/ilkun/library/ii.htm
5. Каллан Р. Основные концепции нейронных сетей / Пер. с англ. М.: Вильямс, 2001. 287 с.
6. Рассел С., Норвиг П. Искусственный интеллект: современный подход. 2-е изд. / пер. с анг. М.: Вильямс, 2006. 1408 с.
7. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы / Пер. с польск. И. Д. Рудинского. М.: Горячая линия-Телеком, 2006. 452 с.
8. Сотников И. Конкурентные мультиагентные игры для софтботов // Материалы 52-й Международной научной студенческой конференции МНСК-2014: Информационные технологии. Новосиб. гос. ун-т. Новосибирск, 2014. 265 с.
9. Сотников И. Мультиагентные компьютерные игры и обучающийся агент // Образование, наука, инновации: вклад молодых исследователей – материалы IX (XLI) Международной научно-практической конференции / сост. В. В. Поддубиков; под общ. ред. В. А. Волчека. Кемеровский государственный университет. Кемерово, 2014. Вып. 15. 1626 с.
10. Уоссермен Ф. Нейрокомпьютерная техника / пер. с англ. Ю. А. Зуев, В. А. Точенов. Режим доступа: http://www.codenet.ru/progr/alg/nks/
11. Sutton R. S., Barto A. G. Reinforcement Learning: An Introduction. The MIT Press Cambridge, Massachusetts London, England. Режим доступа: http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html

Информация об авторах:

Сотников Игорь Юрьевич – магистрант математического факультета по направлению «Прикладная математика и информатика» КемГУ, mxtfonlife@mail.ru.

Igor Yu. Sotnikov – Master’s Degree Student at the Faculty of Mathematics, Kemerovo State University.

(**Научный руководитель – *И. В. Григорьева***)

Григорьева Ирина Владимировна – кандидат физико-математических наук, доцент кафедры ЮНЕСКО по НИТ математического факультета КемГУ, igriva@list.ru.

Irina V. Grigorieva – Candidate of Physics and Mathematics, Assisatant Professor at the UNESCO Department for New Information Technologies, Kemerovo State University.

Статья поступила в редколлегию 21.10.2014 г.