Известия Гомельского государственного университета
имени Ф. Скорины, № 6 (129), 2021

УДК 004.41

Использование методов машинного обучения при управлении персонажем в экшен игре

А.А. Кадетова, Н.Б. Осипенко

Статья посвящена описанию прототипа мобильной игры жанра экшен с использованием искусственного интеллекта. Процесс управления персонажем игры обучается на основе генетического алгоритма на языке программирования Lua в платформе Corona SDK. Приведены разработанные проектные диаграммы приложения в нотации UML: вариантов использования, классов, состояний объектов и популяции. Кратко описан игровой процесс, особенности реализации игрового приложения и идея использования искусственного интеллекта при управлении персонажем. Приведены результаты апробации.
Ключевые слова: мобильная игра, искусственный интеллект, нейронная сеть, генетический алгоритм.

The article is devoted to the description of a prototype of a mobile game of the action genre using artificial intelligence. The process of controlling a game character is trained based on a genetic algorithm in the Lua programming language in the Corona SDK platform. The developed design diagrams of the application in UML notation are presented: use cases, classes, states of objects and populations. The gameplay, the features of the implementation of the game application and the idea of using artificial intelligence to control the character are briefly described. The results of the approbation are presented.
Keywords: mobile game, artificial intelligence, neural network, genetic algorithm.

Введение. В настоящей статье предлагается описание прототипа мобильной игры на языке программирования Lua [1] в платформе Corona SDK в жанре экшен с использованием искусственного интеллекта для управления персонажем. Машинное обучение [2] всё глубже проникает во все сферы, в частности, в постоянно развивающуюся индустрию компьютерных игр, в которой мобильные игры занимают наибольшую долю рынка. Мечта любого геймера о полной персонализации игры становится всё более реальной. Искусственный интеллект (ИИ) и машинное обучение помогают оптимизировать сами игры. ИИ может самонастроиться так, что он будет играть в игры, как это делает обычный пользователь. Это поможет найти уязвимые места и слабые точки в игровых процессах. Широкое развитие за последнее десятилетие получили разные виды нейронных сетей (НС) и методы их обучения. Многие современные подходы приводят к предопределенным, статичным и предсказуемым реакциям игрового агента, не имея возможности приспособиться во время игры к поведению или стилю игры пользователя, а методы машинного обучения позволяют улучшить поведенческую динамику управляемых компьютером игровых агентов, облегчая автоматизированную генерацию и отбор моделей поведения, тем самым расширяя возможности цифрового игрового искусственного интеллекта и предоставляя возможность создавать более увлекательные и занимательные компьютерные игры [3].

Проектирование игрового приложения. В процессе проектирования приложения разработаны диаграммы в нотации UML: вариантов использования, классов, состояний объектов и популяции. Диаграмма вариантов использования, приведенная на рисунке 1, иллюстрирует отношения между актёром (игроком) и вариантами использования (просмотр информации, повышение уровня, выбор типа игры). Диаграмма классов UML, приведенная на рисунке 2, иллюстрирует логическую структуру системы, описывая классы, их атрибуты, операции и отношения между объектами. В проекте присутствуют следующие классы: популяция, НС, сцена игры, объект, объект-враг, объект-лезвие, объект-персонаж. Диаграмма состояний показывает, как объект переходит из одного состояния в другое, и полезна также при моделировании жизненного цикла объекта; она описывает процесс изменения состояний только одного экземпляра определенного класса. В приложении такие сущности как объект,

объект-враг, объект-лезвие, объект-персонаж имеют 3 состояния: «Новый», «Существует», «Не существует». Их жизненный цикл можно увидеть на рисунке 3. Объект популяция также может попадать в различные состояния, его диаграмма состояний приведена на рисунке 4.

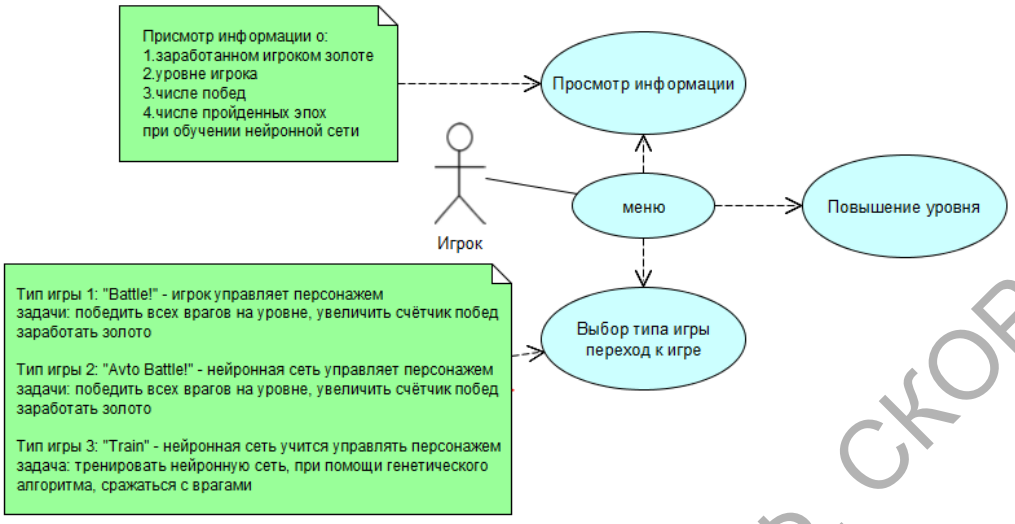

Рисунок 1 – Диаграмма вариантов использования

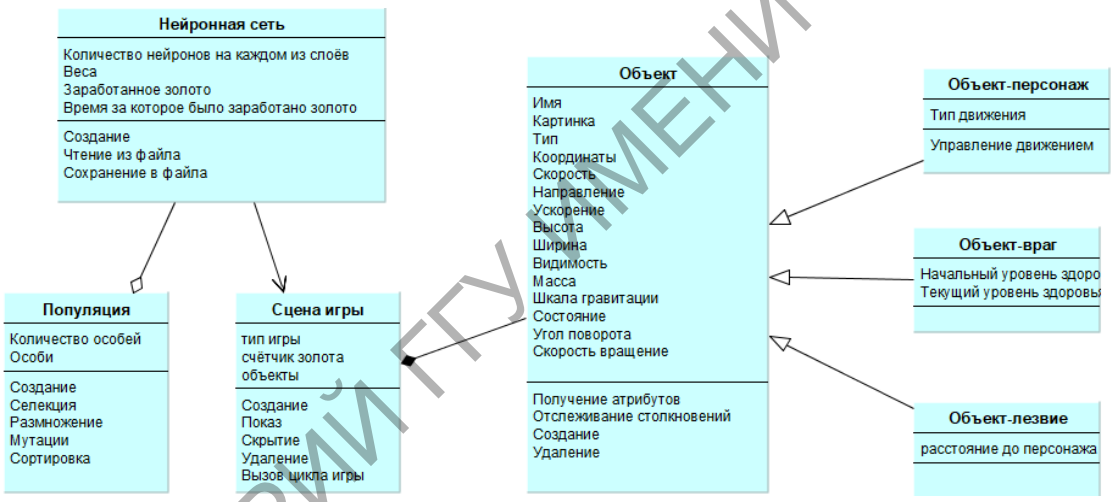

Рисунок 2 – Диаграмма классов

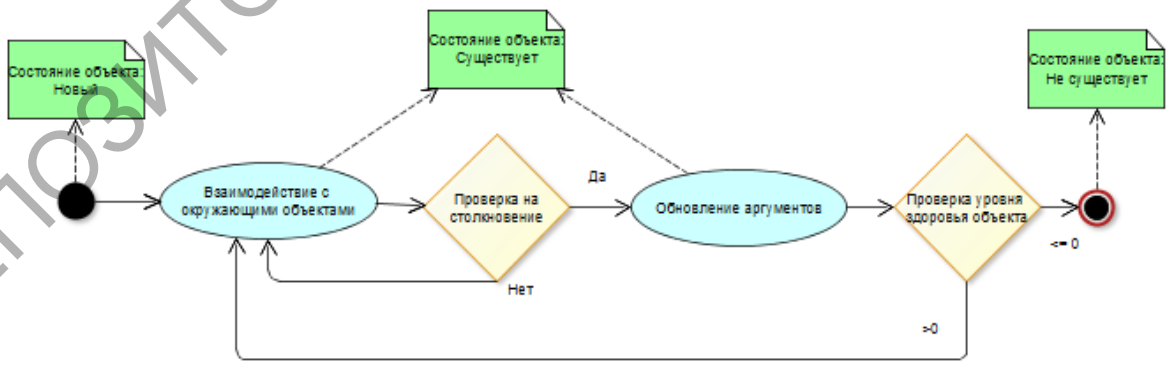

Рисунок 3 – Жизненный цикл объекта

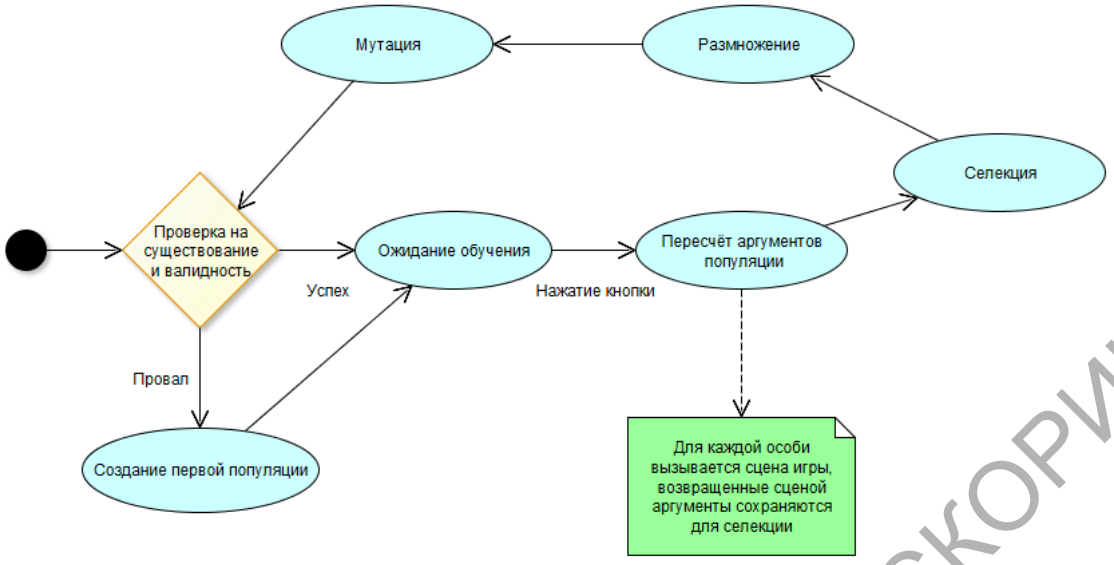

Рисунок 4 – Жизненный цикл популяции

Суть игрового процесса. Имеется персонаж, у него есть инструмент защиты себя от врагов (лезвие), есть цель (увеличение значения счётчика побед при передвижении), есть враги, имеется схема поведения. Необходимо реализовать три типа игры: «Battie!», «AvtoBattle!», «Train!». В игре «Battle!» игрок управляет персонажем, который должен победить всех врагов на уровне, увеличив счетчик побед, заработанное на уровне золото сохраняется, при этом игра ставится на паузу всякий раз, когда персонажем не управляют более одной секунды. В игре «Avto Battle!» искусственный интеллект управляет движением персонажа таким образом, чтобы победить всех врагов на уровне, увеличив счетчик побед, заработанное на уровне золото сохраняется. Тип игры «Train» используется игроком для тренировки искусственного интеллекта; здесь нейронная сеть учится управлять персонажем и ее задача: обучение нейронной сети с использованием генетического алгоритма сражаться с врагом.

Игровой процесс заключается в увеличении значения счётчика побед. Значение на счётчике увеличивается на единицу каждый раз при прохождении игроком уровня игры. На каждом из них присутствует 50 врагов, уровень жизни которых имеет равномерное распределение на промежутке $[2; 6+2\cdot m]$, где m – число побед, что постепенно увеличивает сложность игры. Для победы над врагом необходимо снизить его уровень жизни до 0. При победе над всеми врагами уровень считается пройденным. Разработанные изображения врага, лезвия (меча), персонажа приведены на рисунках 5, 6, 7.

Рисунок 5 – Иконка врага

Рисунок 6 – Иконка меча

Рисунок 7 – Иконка персонажа

Уровень жизни врага снижается при столкновении с одним из мечей. Мечи вращаются вокруг персонажа на некотором расстоянии, при этом расстояние и скорость их вращения зависят от уровня игрока. Наносимый мечом урон начинается с 1 и увеличивается на 1 за каждые 4 уровня игрока. Уровень игрока можно повысить, тратя золото. Золото выпадает после победы над врагом и его количество равно уровню жизни врага. Цена за повышение уровня игрока вычисляется по формуле: $\mathrm{P} = -75\cdot L + 200\cdot 7^{L}/5$, где P – цена; L – уровень игрока.

Особенности реализации игрового приложения. Для описания и обучения небольших НС удобно использовать встраиваемые языки программирования, например, Lua. Они упрощают и ускоряют разработку ПО, а также обеспечивают быстрое и эффективное

обучение НС. Существенно облегчающая разработку мобильных приложений на языке Lua платформа Corona SDK использует все его преимущества: плотная интеграция с языками C/C++; динамическая типизация; большой набор библиотек расширений и др.

Для удобства разработки и последующего обслуживания проект приложения хранится в файлах: Main.lua; Game.lua; Config.lua; Menu.lua. Файл Config.lua содержит параметры приложения; Main.lua – код приложения, из него вызывается основная сцена игры, описание которой содержится в файле Menu.lua. Эта сцена отображает информацию об игроке и позволяет запустить игру. При запуске игры вызывается сцена, хранящаяся в файле Game.lua, в ней происходит игровой процесс, заключающийся в увеличении значения счётчика побед на единицу при прохождении на более высокий уровень игры.

Задача основной сцены игры заключается в отображении информации о прогрессе игрока. Так как эта информация должна сохраняться при выходе из приложения, то были реализованы функции выгрузки и загрузки информации в таблицу playerInfo в json-файл. Чтение и запись из файла осуществляется при помощи библиотеки ввода-вывода io, а преобразование данных к формату json – библиотекой json. Приведем некоторые особенности реализации функций выгрузки из файла и сохранения в файл. Вначале происходит открытие файла на чтение: если файл был открыт успешно, его содержимое считывается в переменную contents, а сам файл закрывается; содержимое переменной contents декодируется и заносится в таблицу playerInfo. Затем проводится несколько проверок на случай какого-либо повреждения файла. Функция сохранения в файл устроена ещё проще: файл открывается на запись, при этом всё предыдущее содержимое файла удаляется; если файл открыт успешно, то в него записывается закодированное содержимое таблицы playerInfo и файл закрывается. Помимо этого в файле с основной сценой игры для реализации отображения информации об игре есть функции: calculateGoldToLvlup() – возвращает цену на поднятие уровня игрока; lvlup (event) – поднимает уровень игрока, если было накоплено достаточно золота (эта функция вызовется при нажатии на надпись «level up»); updateText() – обновляет информацию об игроке на сцене.

Идея использования искусственного интеллекта для управления персонажем. В игре типа «Avto Battle!» обучение НС управлению персонажем при сражении с врагом осуществляется на основе генетического алгоритма. Суть генетического алгоритма заключается в том, что случайным образом задаётся множество генотипов (или хромосом) начальной популяции. Генотипы оцениваются при помощи функции приспособленности и среди них выбираются несколько лучших особей. К ним применяются какие-либо генетические операторы, в результате чего получается новое множество генотипов. Далее генетические операторы применяются уже к лучшим особям из нового множества генотипов до тех пор, пока не будет достигнут результат.

Итак, на вход НС поступают координаты ближайших врагов, выходом НС является направление движения центральной точки персонажа (x, y), где $x, y \in [-1, 1]$. НС имеет произвольную архитектуру, для которой количества скрытых слоев (countHideLayers), нейронов на входном слое (coutInputNeuron) и нейронов на скрытых слоях (coutHideNeuron) задаются параметрически в основной сцене игры. В качестве функции активации используется гиперболический тангенс. Выходные данные НС используются для управления движением персонажем, движение происходит дискретным образом каждые 50 миллисекунд. Генами для генетического алгоритма являются веса НС, в гене содержатся два параметра a и b, при создании первой популяции a и b имеют нормальное распределение на отрезке [-5, 5]. Хромосомой для генетического алгоритма является набор весов нейронной сети. Количество особей в популяции (countIndivid) также задается в основной сцене игры.

После создания первой популяции генетический алгоритм оценивает эффективность особей на основании взаимодействия особей с моделью окружающего мира. Параметрами для оценки являются заработанное золото (gold) и прошедшее время (time). Два наиболее эффективных генотипа выживают и дают потомство. Оператор мутации применяется к половине полученных особей, а каждый ген из генотипа мутирует с вероятностью (mutationChance). Две лучшие (по приспособленности) НС сохраняются в файлы и запоминаются при выходе из приложения.

При реализации генетического алгоритма для сохранения прогресса обучения в приложении реализованы функции чтения и записи лучших результатов по обучению НС в файлы, они сходны с описанными выше функциями чтения и записи информации о персонаже. Если прочесть файл не получается или если данные не валидны, НС создаются заново, тем самым процесс обучения перезапускается.

Функция createNeuralNetwork(...) отвечает за создание НС. Если на функцию не поступает параметров, то возвращается НС со случайными весами, согласно параметрам, указанным выше. Если на функцию поступают две НС, то возвращается НС, являющаяся потомком этих НС. Функция createFirstPopulation() отвечает за создание первой популяции при запуске приложения.

Первыми двумя особями в популяции становятся две лучшие сохранённые особи. При этом, если чтение из файла сохранения не было успешным, геномы остальных особей генерируются случайным образом, а прогресс по обучению обнуляется. В противном случае вызывается функция createPopulation(), отвечающая за размножение особей и мутацию половины особей с шансом мутации в 1/mutationChance.

Функция selection() реализует механизм, при котором «гибнут» все особи, кроме двух лучших из них, а затем эти две выжившие особи размножаются.

За весь генетический алгоритм отвечает функция stady(). Таймер «studyTimer» запускает функцию stady столько раз, сколько генотипов находится в популяции и ещё один раз для реализации шагов генетического алгоритма. Если запуск функции не последний, то для текущей популяции вызывается сцена game2 с передачей в неё генотипа и его номера. В противном случае происходит селекция мутации, обновление информации персонажа. Если при этом в обучении прошло меньше эпох, чем задано на один запуск обучения, то вызывающий функцию stady таймер перезапускается.

Помимо этого в файле со сценой для реализации перехода к другим сценам присутствуют следующие функции: gotoGame() – вызывается при нажатии кнопки «playButton» и осуществляет переход к сцене game; startStudyTimer() – вызывается при нажатии кнопки «studyButton» и осуществляет запуск таймера, по которому выполняется цикличный запуск функции stady(); gotoAutoGame() – вызывается при нажатии кнопки «playButtonAvt» и осуществляет переход к сцене game2.

Также файл со сценой содержит в себе переопределение четырех основных методов сцены: create – создать сцену; show – показать сцену; hide – скрыть сцену; destroy – уничтожить сцену. Для этих методов существуют две фазы выполнения: will – описывает то, что необходимо осуществить перед выполнением метода; did – описывает то, что необходимо осуществить после выполнения метода.

Схематично опишем метод show. В фазе will непосредственно перед запуском метода show происходит следующее: загружается информация об игроке из файла; из сцены game загружаются переменные gold, time и win, переменная gold отражает количество золота, которое было заработано на предыдущем уровне игры, а переменная win является показателем того, был ли пройден уровень, переменная time отражает, сколько времени существовала предыдущая сцена; обновляются значения в таблице playerInfo, если не идёт обучение; информация об игроке сохраняется в файл; обновляется текст с информацией об игроке на экране; в переменную population заносятся показатели приспособленности, если идёт обучение; возобновляется таймер, вызывающий функцию stady(), если идёт обучение.

В фазе did метода show при создании сцены происходит следующее: загружается картинка на задний фон игры; картинка заднего фона выравнивается; в верхнем левом углу экрана размещается текст, отображающий информацию об игроке; для текста «level up» переопределяется метод, ожидающий событие «tap», теперь по нажатию на текст будет вызываться функция lvlup; создаётся кнопка с надписью «Battle!»; для кнопки переопределяется метод ожидающий событие «tap», теперь по нажатию на текст будет вызываться функция gotoGame; создаётся кнопка с надписью «Avto Battle!!»; для кнопки переопределяется метод, ожидающий событие «tap», теперь по нажатию на текст будет вызываться функция gotoAutoGame; создаётся кнопка с надписью «Train»; для кнопки переопределяется метод, ожидающий событие «tap», теперь по нажатию на текст будет вызываться функция startStudyTimer.

Результаты апробации. Для запуска игры на android необходимо установить собранный файл, после чего игра полностью готова к использованию. Если игра собирается под Windows, то не требуется даже установка, достаточно запустить файл с расширением exe и установка дополнительного окружения не требуется. После первого запуска открывается меню игры с информацией: количество золота и число побед равны 0, уровень персонажа равен 1 (для поднятия уровня персонажа необходимо 205 единиц золота), в обучении было пройдено 0 эпох. Персонаж на экране управляется пальцем, враги двигаются к персонажу с 3-х сторон экрана. Также на экране изображён счётчик золота, заработанного персонажем на данном уровне игры. При столкновении врага с персонажем уровень считается проваленным, игрок получает уведомление о проигрыше. При уничтожении всех врагов уровень игры считается пройденным и игроку демонстрируется уведомление о победе, после чего осуществляется переход в меню игры.

При нажатии на «Train!» начинается обучение НС. Персонаж на экране управляется согласно выходу нейронной сети. Счётчик золота на экране показывает количество заработанного персонажем золота, счётчик особей – номер текущей особи в популяции. При столкновении врага с персонажем уровень считается проваленным, игроку показывается уведомление о проигрыше. Заработанное при обучении золото не сохраняется. При нажатии на «AvtoBattle!» по аналогии с типом игры «Train!» запускается сеанс, но с автоматическим управлением. Для автоматического управления при уведомлении о проигрыше заработанное золото сохраняется, после чего следует переход в меню игры. При победе над врагом счётчик золота увеличивается на первоначальный уровень здоровья врага.

Так как НС имеет произвольную архитектуру, где количества скрытых слоев, нейронов на входном слое и нейронов на скрытых слоях задаётся параметрически в сцене menu, то были проверены различные архитектуры; полученные результаты приведены в таблице. Из нее следует, что при увеличении числа входных параметров скорость обучения уменьшается, но невозможно на данный момент проверить ухудшается или улучшается качество обучения. Остальные параметры влияют на скорость обучения значительно меньше, и их влияние сложно отследить из-за самой сути генетического алгоритма: первая популяция создаётся случайно, и случайно выбираются гены из двух лучших генотипов.

Таблица – Результаты апробации различных архитектур нейронной сети

К-во нейронов на входном слое	К-во скрытых слоев	К-во нейронов на скрытых слоях	Прогресс в обучении начался с эпохи	Первая победа произошла в эпохе
5	1	5	5	26
5	1	6	16	61
5	1	7	5	20
5	1	8	4	22
5	2	5	5	26
5	2	6	6	22
5	2	7	4	26
5	2	8	5	24
5	3	5	20	58
5	3	6	30	164
5	3	7	10	50
5	3	8	4	30
5	4	5	2	26
5	4	6	5	38
5	4	7	3	50
5	4	8	25	238
10	1	5	8	27
10	1	6	20	171
10	1	7	10	297
10	1	8	3	264
10	2	5	7	216
10	2	6	15	270
10	2	7	10	117
10	2	8	30	690

Окончание таблицы

10	3	5	4	75
10	3	6	3	180
10	3	7	7	90
10	3	8	1	63
10	4	5	40	145
10	4	6	2	128
10	4	7	4	139
10	4	8	60	570

Полученная игра была протестирована. В целом прогресс в обучении НС для игры типа «Train!» виден после 20-й эпохи, а первые победы при автоматическом управлении для игры типа «AvtoBattle!» – на 100-й эпохе.

Заключение. В работе рассмотрена задача построения прототипа мобильной игры в жанре экшен с использованием искусственного интеллекта. Процесс управления персонажем игры обучается на основе генетического алгоритма на языке программирования Lua в платформе Corona SDK. Для решения задачи использованы два способа управления игрой: непосредственно игроком-пользователем и обученной нейронной сетью. При этом управление персонажем обученной нейронной сетью на основе генетического алгоритма показало свою хорошую работоспособность. Полученные данные позволяют утверждать, что использование искусственного интеллекта для управления персонажем в играх подобного типа позволяет мотивировать начинающих разработчиков к изучению инструмента машинного обучения. Разработанное игровое приложение доступно по ссылке https://drive.google.com/file/d/1gChFfQzEaiv1gOg251jICwjQPGkBgjDX/view?usp=sharing.

Литература

1. Иерузалимски, Р. Программирование на языке Lua / Р. Иерузалимски, А. Боресков – СПб. : Москва, 2014. – 382 с.

2. Николенко, С. Глубокое обучение / С. Николенко, А. Кадурин, Е. Архангельская. – СПб. : Питер, 2018. – 480 с.

3. Геймификация всей страны. Какие задачи и для чего решает философия видеоигр [Электронный ресурс] // Lenta.ru. – Режим доступа : https://lenta.ru/articles/2014/01/03/gamification. – Дата доступа : 10.03.2021.

Гомельский государственный
университет имени Франциска Скорины

Поступила в редакцию 06.09.2021