УДК 004.855.6

РАЗРАБОТКА ПЛАГИНА ПОВЕДЕНИЯ NPC ДЛЯ ИГРОВОГО ДВИЖКА UNITY

Л. Н. Паренюк[1], В. В. Кугуракова[2]

[1,2]Казанский (Приволжский) федеральный университет

[1]parenyukln@gmail.com, [2]vlada.kugurakova@gmail.com

Аннотация

Существуют различные подходы для создания искусственного интеллекта в играх, и каждый имеет как и плюсы, так и недостатки. В настоящем исследовании описана собственная реализация задания поведения NPC с использованием алгоритмов машинного обучения, которые будут связаны со средой Unity в режиме реального времени. Такой подход может быть применен при разработке игр.

Ключевые слова: *Unity, python, machine learning, AI, искусственный интеллект в играх, поведение агентов, NPC, разработка игр, scikit-learn.*

ВВЕДЕНИЕ

Индустрия игр стремительно растет и развивается. Игровой процесс сейчас – это не только развлечение и досуг. Сфер применения игр достаточно много, начиная от образования, заканчивая экономикой и бизнесом. В связи с этим возникает потребность разрабатывать игры, которые способны максимально завлечь игрока и сохранять это состояние на протяжении всего игрового процесса.

В компьютерных играх термином NPC[1] обозначают персонажи, общающиеся с игроком, независимо от их отношения к игровому персонажу. NPC могут быть дружественными, нейтральными и враждебными. Неигровые персонажи служат важным средством создания игровой атмосферы, мотивируют игроков совершать те или иные действия и являются основным источником информации об игровом мире и сюжете игры. Традиционно проектирование поведения NPC (иначе агентов) в играх всегда приводит к описанию фиксированного поведения

[1] NPC – англ. Non-player controller — «персонаж, управляемый не игроком», персонаж в играх, который не находится под контролем игрока. В компьютерных играх термином «NPC» обозначаются персонажи, общающиеся с игроком, независимо от их отношения к игровому персонажу.

[1]. Но если агент имеет фиксированное поведение, игра становится предсказуемой и реиграбельность понижается. Важно найти способ обогатить интеллект агента, чтобы поведение компьютерного персонажа стало более разнообразным и рациональным. В [4] предложен метод дублирования человеческого поведения, чтобы сделать агента более интеллектуальным.

Другой распространенный способ управления NPC – использование конечных автоматов. Однако количество состояний (и переходов состояний) увеличивается со сложностью игры. Дерево поведения – это вариант решения этой проблемы [12], поскольку оно может определять основное поведение игры и, когда сложность игр возрастает, используются связующие деревья. Подход для нахождения этих деревьев представлен в [12], где генетическое программирование используется для развития контроллеров, которые могут выступать в качестве противника или замены игрока.

В [8] разработан искусственный интеллект (ИИ) игры на основе преобразованного дерева поведения ID3 с применением его в системе принятия решений – этот метод наследует структуру и метод традиционного дерева поведения, но его адаптивность улучшена. Такая схема избавляет от оков заранее подготовленных логических параметров, используя реальную игровую среду для динамической настройки системы принятия решений в игре, что делает поведение NPC непредсказуемым, в то же время обеспечивая стабильность принятия решений и улучшая пользовательский опыт, получаемый в игровом процессе.

В [9] для разработки поведения агентов применено грамматическое генетическое программирование (GGP), которое обычно использует контекстно-свободную грамматику для создания допустимых программ.

СТЕК ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ

В настоящей работе представлен собственный подход к реализации задания поведения NPC в игровой среде Unity. С помощью такого подхода агенты получают возможность «принимать решения» о своих дальнейших действиях. В основе работы лежат алгоритмы машинного обучения, а также алгоритм оптимизации набора данных, сформированного из состояний агента на протяжении всего обучения.

Демонстрируемый подход в задании поведения NPC может быть использован при разработке игр. Анализ существующих решений показал, что предложенный подход может оказаться менее трудозатратным и более гибким.

Разработчики Unity в 2017 году представили библиотеку *ml-agents* [2], которая призвана внедрить ИИ в игровую среду через движок Unity. Типичный подход к решению подобных задач заключается в реализации цикла обучения (см. рис. 1).

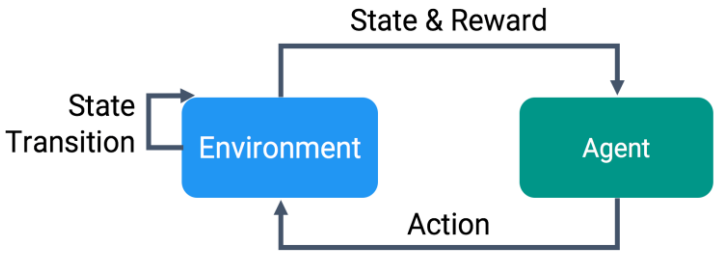

Рисунок 1. Цикл обучения.

Характерным сценарием обучения агентов в виртуальных средах является наличие единой среды и агента, которые тесно связаны между собой. Действия агента изменяют состояние среды и предоставляют агенту награды (или наказания). Этот метод используется для изучения поведения практически всего, что возможно: от промышленных роботов, дронов и автономных транспортных средств до игровых персонажей и противников. Этот же подход использован и в *ml-agents*, позволяя просто и гибко внедрять подходы ИИ в игровую среду (см. рис. 2).

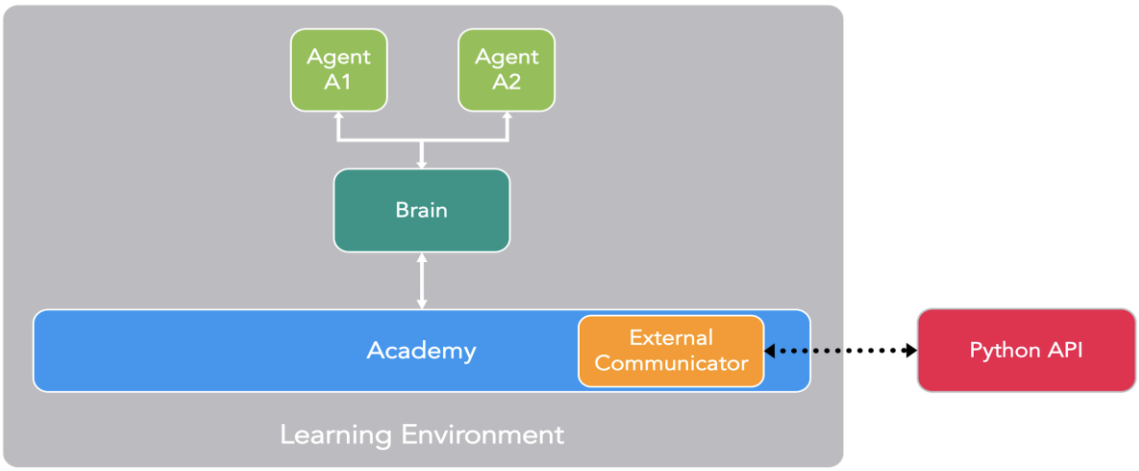

Рисунок 2. Схема взаимодействия интеллектуальной системы с компонентами машинного обучения.

Взаимодействие библиотеки *ml-agents* и игровой среды:

- *Learning Environment* – сцена игры и все NPC;
- *Python API* – алгоритмы машинного обучения, которые находятся вне среды Unity, но общаются с ней посредством коммуникатора;
- *External Communicator* – коммуникатор, который объединяет учебную среду и машинное обучение.

Суть подхода в библиотеке *ml-agents* – обучение с подкреплением [3]. В классе агента описано его поведение: указано, что правильно, а что нет (традиционные награда/наказание). В итоге в каждом скрипте поведения агента жёстко прописано, за какие действия и какое количество награды получает агент (см. рис. 3).

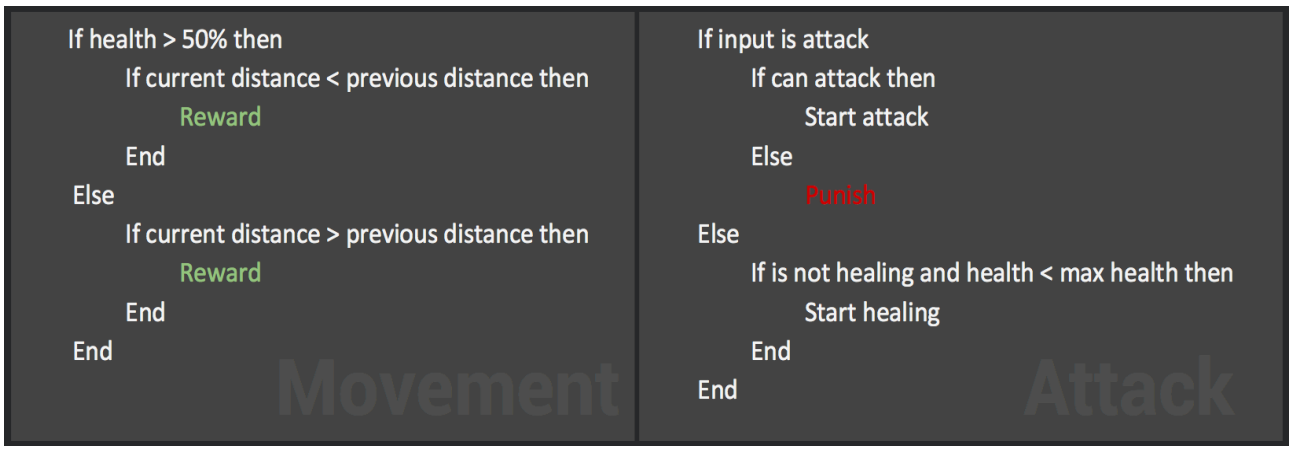

Рисунок 3. Функция награждения.

Далее с помощью консольных команд запускается процесс обучения, где в игровом режиме агент выполняет хаотические действия, за которые либо получает очки, либо теряет. Чем больше времени обучается модель, тем лучше, однако все зависит от конкретного случая. После обучения формируется модель поведения для агента, которая по сути является графом данных Tensor Flow (используется в модуле Python API), содержащем математические операции и оптимизированные веса.

Всё, что необходимо сделать в конце, – это определить эту модель как модель поведения агента в Unity, и агент будет следовать поведению, которому его обучили.

ВЫБОР ЦЕЛЕВЫХ ПАРАМЕТРОВ

После знакомства с технологией *ml-agents* была выдвинута идея, предлагающая отойти от жёсткого назначения награды или наказания разработчиком и делегировать эту задачу некому плагину. В таком случае из игровой среды необходимо получать только текущее состояние и действие агента. Процесс обучения сводится непосредственно к игровому процессу, в ходе которого нужно собирать историю состояний агента и формировать «датасет» (англ. dataset).

После обучения и сбора данных специальный алгоритм внутри плагина проанализирует датасет и проведет автоматически разбиение на «плохое» / «хорошее» с учётом выбранного целевого параметра, который и необходимо максимизировать в процессе разбиения.

Выдвинутая идея призвана автоматизировать процесс обучения агентов, тем самым ускорить внедрение ИИ в игровую среду и облегчить разработку логики NPC. Перейдём к детальному описанию реализации алгоритма.

Первым делом необходимо решить, как среда Unity и Python будут взаимодействовать. Для решения этой задачи был выбран плагин *Unity-Python connector* на основе *IronPython* [5] – реализации языка *Python*, предназначенной для платформы Microsoft .NET или Mono, который является транслятором компилирующего типа.

Плагин *Unity-Python connector* был использован для запуска скриптов *Python* в среде *Unity*. Схема взаимодействия модулей представлена на рис. 4.

В модуле *Python ML* заключена главная логика плагина. В нем работают *python*-скрипты для обработки данных, обучение моделей машинного обучения и назначение тех или иных действий агентам.

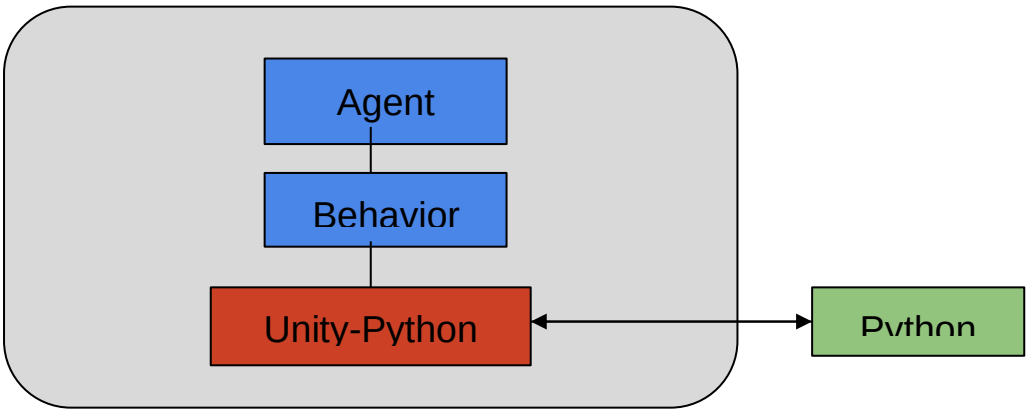

Рис. 4. Взаимодействие модулей.

Идея подхода заключается в следующем: в классе поведения агента необходимо инициализировать состояние, которое в процессе обучения собирается в набор необработанных данных, являющийся по сути историей поведения агента. В состоянии может быть собрана такая информация, как количество жизней персонажа, расстояние до врага, расстояние до укрытия и т. д., также в состояние должно войти текущее действие объекта. Это очень гибкий подход, так как все зависит от конкретного случая. Может быть сколько угодно параметров в состоянии и сколько угодно большой набор данных. После сбора истории необходимо проанализировать набор данных и выполнить его чистку: исключить из набора записи о состояниях, приводящих к уменьшению целевого для агента параметра. От игровой среды требуется получить этот целевой параметр, от которого будет отталкиваться python скрипт-анализатор. Например, если решено держать максимальным количество жизней персонажа, то можно предположить, что действия и параметры состояния, приводящие к наиболее высокому целевому параметру, являются «правильными» и их стоит придерживаться. Для реализации этой мысли *python*-скрипт вычисляет медиану целевого параметра на всём наборе данных и отсекает «плохие» состояния.

Далее формируется модель классификатора, которая обучается предсказывать действия агента на подготовленном наборе данных. В качестве реализации алгоритмов машинного обучения была использована библиотека *scikit-learn* [6, 7], а в качестве модели классификатора – дерево решений.

Класс *DecisionTreeClassifier* библиотеки *scikit-learn* способен выполнять мультиклассовую классификацию для набора данных. Внутри класса реализован

алгоритм, который прогнозирует значения целевой переменной, исходя из правил принятия решения.

Визуализация алгоритма дерева решений представлена на рис. 5, на котором можно видеть, что дерево решений состоит из «ветвей» (ребра графа) и «листьев». На «ветвях» хранится информация о значениях атрибутов, от которых зависит целевая функция, а на «листьях» – значения самой функции.

Для распознавания действия в режиме реального времени в модели поведения агента необходимо отправить запрос через *IronPython* на скрипт-предсказатель, который на вход принимает новое текущее состояние агента, классифицирует по его параметрам состояния через обученную модель и на выходе отдает действие, которое необходимо совершить агенту.

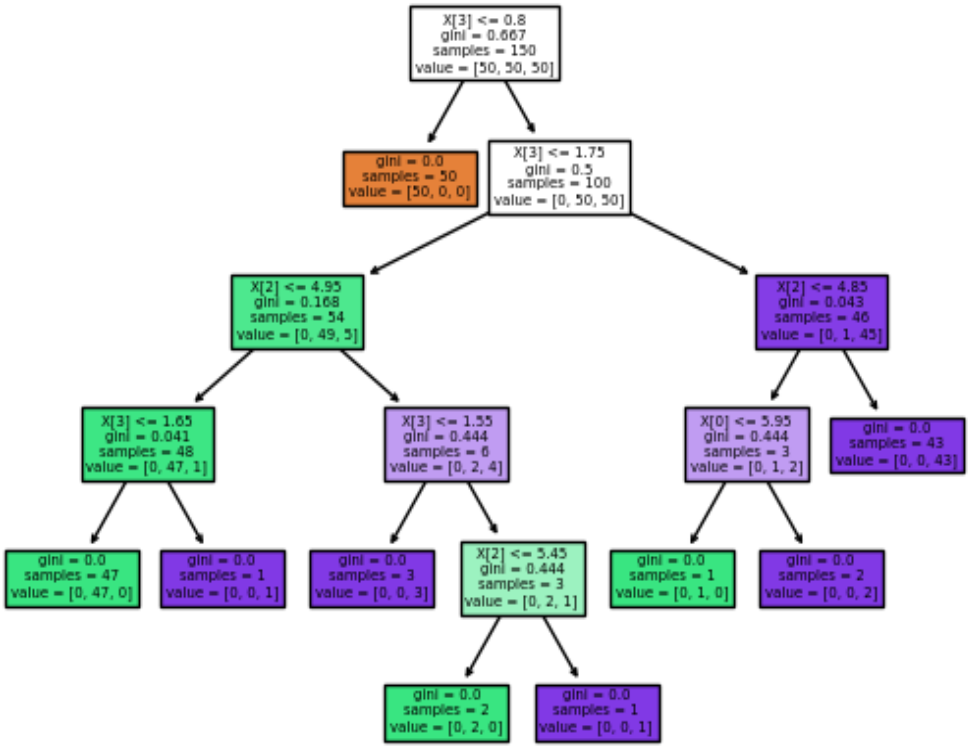

Рисунок 5. Визуализация алгоритма «Дерево решений».

Для классификации был выбран алгоритм ID3 [8]. Существенные отличия представленного здесь подхода от изложенного в [8] в том, что после сбора данных производится дополнительный фильтр выборки оптимизирующим алгоритмом по целевому параметру.

Таким образом, используется реальная игровая среда для динамической настройки поведения NPC с оптимизацией по целевому параметру.

ОБСУЖДЕНИЕ РЕЗУЛЬТАТОВ

В итоге был получен плагин для конкретного игрового движка, который непрерывно обеспечивает обмен информацией между игровой средой и адаптированными алгоритмами машинного обучения.

Тестирование подхода проводилось на игре Roguelike 2D [13] (см. рис. 6), где агентам предложено было классифицировать своё поведение, исходя из параметров количества жизней, расстояния до врага, расстояния до укрытия и возможности атаковать.

Рисунок 6. Игровая среда в Roguelike2D.

В качестве целевого параметра был выбран параметр *количества жизней*. Также были определены действия для агентов: *атаковать, прятаться, бродить, лечиться*.

Реализация процесса принятия решения может выглядеть следующим образом (см. листинг 1):

```
1 ArrayList playerState = new ArrayList();
2 playerState.Add(food);
3 playerState.Add(distanceToEnemy);
```

```
4 playerState.Add(destanceToClosestShelter);
5 playerState.Add(canNowAttack);
6 dynamic py = engine.ExecuteFile(@"Assets/Python/pyton-unity-re-
search/python-ml/recognize.py");
7 dynamic obj = py.RecognizeModel(playerState);
8 Debug.Log(obj.recognize());
```

Листинг 1. Пример процесса принятия решения.

В данном примере информация о действии получается каждый тик игры. В качестве метрик оценки качества обучения модели были использованы точность и полнота.

Точность обучения модели классификатора (precision) — 93.75%, а полнота (recall) — 94%. Для визуализации оценки (см. рис. 7) были использованы утилиты *scikitplot* и *precision_recall_curve*.

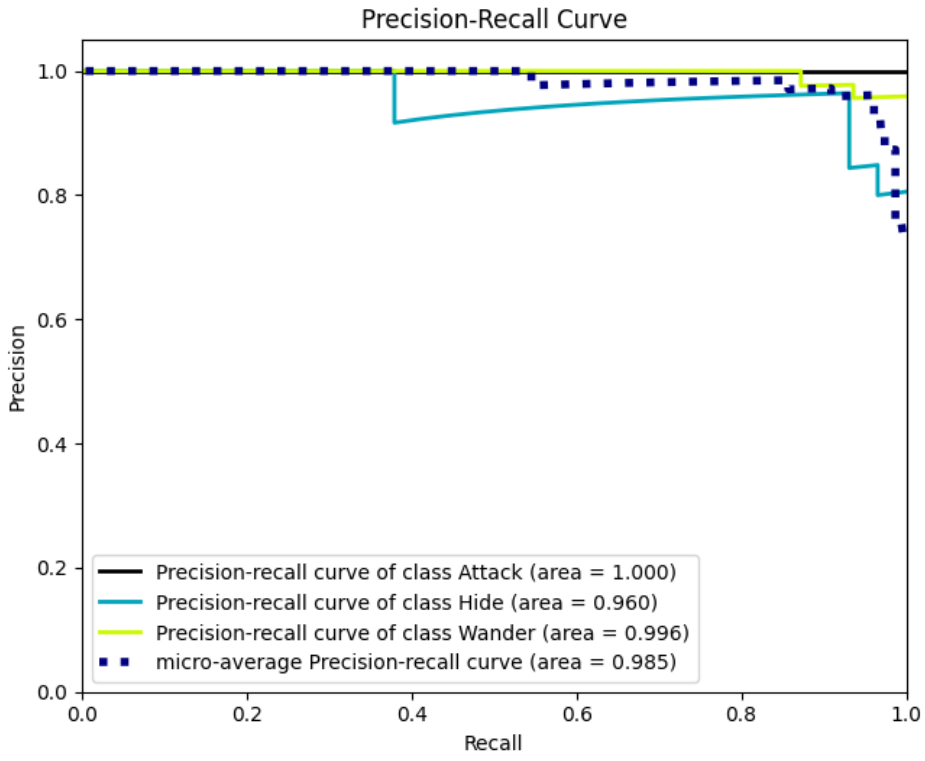

Рисунок 7. Визуализация оценки качества обучения модели.

Работу плагина иллюстрирует демонстрационное видео [10]. Исходный код размещён в репозитории git [11] и доступен как для исследования, так и может быть использован для развития предложенного подхода.

ЗАКЛЮЧЕНИЕ

Предложен подход для реализации поведения NPC в играх на движке *Unity*. В основу его реализации легли алгоритмы машинного обучения на *Python*, а также алгоритм-анализатор, который делил состояния по целевому параметру на «плохие» и «хорошие». Обучение модели дерева решений на собранном наборе данных показало неплохие результаты, в качестве метрик оценки качества обучения были использованы точность и полнота обучения.

СПИСОК ЛИТЕРАТУРЫ

1. *Бакиров А.Р., Костюк Д.И., Лазарев Е.Н., Хафизова А.Р.* Опыт создания неигровых персонажей в виртуальных мирах // Электронные библиотеки. 2016. Т. 19. № 6. С. 502–520.

2. *Juliani A., Berges V., Vckay E., Gao Y., Henry H., Mattar M., Lange D.* Unity: A General Platform for Intelligent Agents / Open-source library // 2018. URL: https://github.com/Unity-Technologies/ml-agents

3. Juliani A. Introducing: Unity Machine Learning Agents Toolkit // 2017. URL: https://blogs.unity3d.com/2017/09/19/introducing-unity-machine- learning-agents/

4. *Tang W., Lai J.* Enhancing Agent Intelligence with Behavior Duplication // Advanced Materials Research. 2012. Vols. 403–408. P. 1266–1269.

5. *Hugunin J.* IronPython / Open-source library // 1st ver. 2006: last ver. 2018. URL: https://ironpython.net

6. Scikit-learn / Open source library // 1st ver. 2007: last ver. 2020. URL: https://scikit-learn.org

7. *Pedregosa F. et al.* Scikit-learn: Machine Learning in Python // Journal of Machine Learning Research (JMLR 12). 2011. P. 2825–2830.

8. *Li Y., Xu D.-W.* A Game AI based on ID3 Algorithm // Conference: 2016 2nd International Conference on Contemporary Computing and Informatics. 2016. P. 681–687.

9. *De Freitas J.M., De Souza F.R., Bernardino H.S.* Evolving Controllers for Mario AI Using Grammar-based Genetic Programming // IEEE Congress on Evolutionary Computation (CEC). 2018. P. 1–8.

10. ITIS-DML-M2020-ParenyukLN. (2020) Python Unity Research [демонстрационная видеозапись] // YouTube. 9 марта 2020. – (https://www.youtube.com/watch?v=4AqrpTxCXYk).

11. Плагин поведения NPC для игрового движка UNITY / Project repository // 2020. – URL: https://github.com/parenyukln/pyton-unity-research

12. *C.-U. Lim, R. Baumgarten, S. Colton, C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A.I. Esparcia-Alcazar, C.-K. Goh, J.J. Merelo, F. Neri, M. Preuß, J. Togelius, G.N. Yannakakis.* Evolving behaviour trees for the commercial game defcon // Applications

of Evolutionary Computation, Berlin, Heidelberg: Springer Berlin Heidelberg. 2010. P. 100–110.

13. 2D Roguelike tutorial // Unity Technologies. 2015. URL: https://learn.unity.com/project/2d-roguelike-tutorial

---

NPC BEHAVIOR PLUGIN DEVELOPMENT FOR GAME ENGINE UNITY

L. N. Parenyuk[1], V. V. Kugurakova[2]

[1,2] Higher School ITIS. Kazan Federal University

[1]parenyukln@gmail.com, [2]vlada.kugurakova@gmail.com

Abstract

There are various approaches for creating artificial intelligence in games, and each has both advantages and disadvantages. This study describes an authoring implementation of the NPC behavior task using machine learning algorithms that will be associated with the Unity environment in real time. This approach can be used in game development.

Keywords*: Unity, python, machine learning, AI, NPC, NPC behavior, game development, scikit-learn.*

REFERENCES

1. *Bakirov A.R., Kostyuk D.I., Lazarev E.N., Hafizova A.R.* The experience of creating non-player characters in virtual worlds // Russian Digital Libraries. 2016. Vol. 19. No 6. P. 502–520.

2. *Juliani A., Berges V., Vckay E., Gao Y., Henry H., Mattar M., Lange D.* Unity: A General Platform for Intelligent Agents / Open-source library // 2018. URL: https://github.com/Unity-Technologies/ml-agents

3. Juliani A. Introducing: Unity Machine Learning Agents Toolkit // 2017. URL: https://blogs.unity3d.com/2017/09/19/introducing-unity-machine- learning-agents/

4. *Tang W., Lai J.* Enhancing Agent Intelligence with Behavior Duplication // Advanced Materials Research. 2012. Vols. 403–408. P. 1266–1269.

5. *Hugunin J.* IronPython / Open-source library // 1st ver. 2006: last ver. 2018. URL: https://ironpython.net

6. Scikit-learn / Open source library // 1st ver. 2007: last ver. 2020. URL: https://scikit-learn.org

7. *Pedregosa F. et al.* Scikit-learn: Machine Learning in Python // Journal of Machine Learning Research (JMLR 12). 2011. P. 2825–2830.

8. *Li Y., Xu D.-W.* A Game AI based on ID3 Algorithm // Conference: 2016 2nd International Conference on Contemporary Computing and Informatics. 2016. P. 681–687.

9. *De Freitas J.M., De Souza F.R., Bernardino H.S.* Evolving Controllers for Mario AI Using Grammar-based Genetic Programming // IEEE Congress on Evolutionary Computation (CEC). 2018. P. 1–8.

10. ITIS-DML-M2020-ParenyukLN. (2020) Python Unity Research [демонстрационная видеозапись] // YouTube. 9 марта 2020. – (https://www.youtube.com/watch?v=4AqrpTxCXYk).

11. NPC behavior plugin for the UNITY game engine / Project repository // 2020. URL: https://github.com/parenyukln/pyton-unity-research

12. *C.-U. Lim, R. Baumgarten, S. Colton, C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A.I. Esparcia-Alcazar, C.-K. Goh, J.J. Merelo, F. Neri, M. Preuß, J. Togelius, G.N. Yannakakis.* Evolving behaviour trees for the commercial game defcon // Applications of Evolutionary Computation, Berlin, Heidelberg:Springer Berlin Heidelberg. 2010. P. 100–110.

13. 2D Roguelike tutorial // Unity Technologies. 2015. URL: https://learn.unity.com/project/2d-roguelike-tutorial

СВЕДЕНИЯ ОБ АВТОРАХ

ПАРЕНЮК Леонид Николаевич – магистрант Высшей школы информационных технологий и интеллектуальных систем Казанского федерального университета. Сфера научных интересов – разработка игр.

Leonid Nikolaevich PARENYUK – Higher School of Information Technologies and Intelligent Systems of Kazan Federal University. His research interests include game development.

email: parenyukln@gmail.com

КУГУРАКОВА Влада Владимировна – к.т.н., доцент кафедры программной инженерии Высшей школы информационных технологий и интеллектуальных систем Казанского федерального университета, руководитель НИЛ SIM. Сфера научных интересов – реалистичность визуализации и симуляций, иммерсивность VR.

Vlada Vladimirovna KUGURAKOVA, docent of Higher School of Information Technology and Intelligent Systems, Head of Laboratory «Virtual and simulation technologies in biomedicine». Research interests include realism of simulation, immersion VR.

email: vlada.kugurakova@gmail.com.

Материал поступил в редакцию 3 апреля 2020 года