УДК 004.431.5

РАЗРАБОТКА СИСТЕМЫ ВИЗУАЛЬНОГО ВОСПРИЯТИЯ ИГРОВЫХ АГЕНТОВ В ВИДЕОИГРАХ

А. М. Примаченко[1] [0009-0008-6396-2035], М. Р. Хафизов[2] [0000-0001-7275-9102]

[1, 2]Казанский (Приволжский) федеральный университет, Казань, 420008, Россия

[1]primachenko.artem@mail.ru, [2]murkorp@gmail.com

Аннотация

Представлен алгоритм функционирования системы визуального восприятия для игровых агентов, реализованный в игровом движке Unity. Предложенный метод основан на сравнении изображений с двух камер, учитывающих сложные визуальные эффекты (освещение, тени, маскировку), и дополнен проверкой прямой видимости, учетом скорости движения объекта, и механикой постепенного обнаружения. Тестирование системы показало значительное повышение реалистичности обнаружения по сравнению с традиционными методами при сохранении производительности в пределах небольшой дополнительной нагрузки на процессор. Проведена оптимизация алгоритма с использованием Unity Job System и динамической активации камер. Проведен также анализ научной литературы по схожим решениям, выявлены их сильные и слабые стороны. Результаты могут быть применены в разработке видеоигр для создания реалистичного поведения неигровых персонажей, особенно в играх с элементами скрытности.

Ключевые слова: *видеоигры, искусственный интеллект, система восприятия, NPC, неигровые персонажи, игровые агенты, стелс-механики, Unity, рендеринг, компьютерное зрение, оптимизация, гейм-дизайн.*

ВВЕДЕНИЕ

Системы визуального восприятия неигровых персонажей (non-player character, NPC) в видеоиграх являются важным элементом, обеспечивающим реалистичность и интерактивность игрового процесса. Как отмечено в ряде работ (см., например, [1–7]), информация о видимости (или линии обзора) является ключевым элементом, позволяющим NPC строить пространственное представление о мире игры и принимать решения в реальном времени. Традиционные подходы, такие как проверка углов обзора и использование отражения лучей (или рейкастов, от англ. raycast) [8], обладают ограничениями, связанными с недостаточным учетом визуальных факторов, таких как освещение, тени и маскировка. Это приводит к неестественному поведению NPC, особенно в играх, использующих игровые механики невидимости («стелс-играх», от англ. stealth), где требуется высокая степень реализма [9].

Наиболее распространённым способом реализации системы зрения для неигровых персонажей является алгоритм, состоящий из двух шагов: проверка нахождения целевого объекта в определённой зоне в игровом мире, часто динамически изменяющейся в зависимости от положения персонажа и называемой полем зрения, и проверка на наличие препятствий между персонажем и целевым объектом. Хотя такой подход прост в реализации, он не учитывает множество факторов реального визуального восприятия, что снижает убедительность игрового опыта.

Целью проведенного исследования была разработка системы визуального восприятия, которая учитывает сложные визуальные эффекты, обеспечивает гибкость настройки и оптимизирована для реального времени. Объектом исследования выступают методы реализации визуального восприятия NPC, а предметом — алгоритм, основанный на сравнении изображений и дополнительных механиках.

1. СВЯЗАННЫЕ РАБОТЫ

Современные исследования в области систем визуального восприятия NPC предлагают разнообразные подходы.

Понятие базового рейкастинга (raycasting) для определения первого столкновения луча с объектом сцены было введено ещё в 1989 году [8].

В [10] представлен разработанный алгоритм, основанный на проверке нахождения объекта в поле зрения, представленного усеченной пирамидой, и использовании рейкаста для исключения препятствий. Этот метод включает проверку трех основных условий.

1. Ограничительная рамка (Bounding box) целевого объекта должна находиться внутри усечённой пирамиды поля зрения персонажа либо пересекать хотя бы одну из шести её граней (рис. 1).

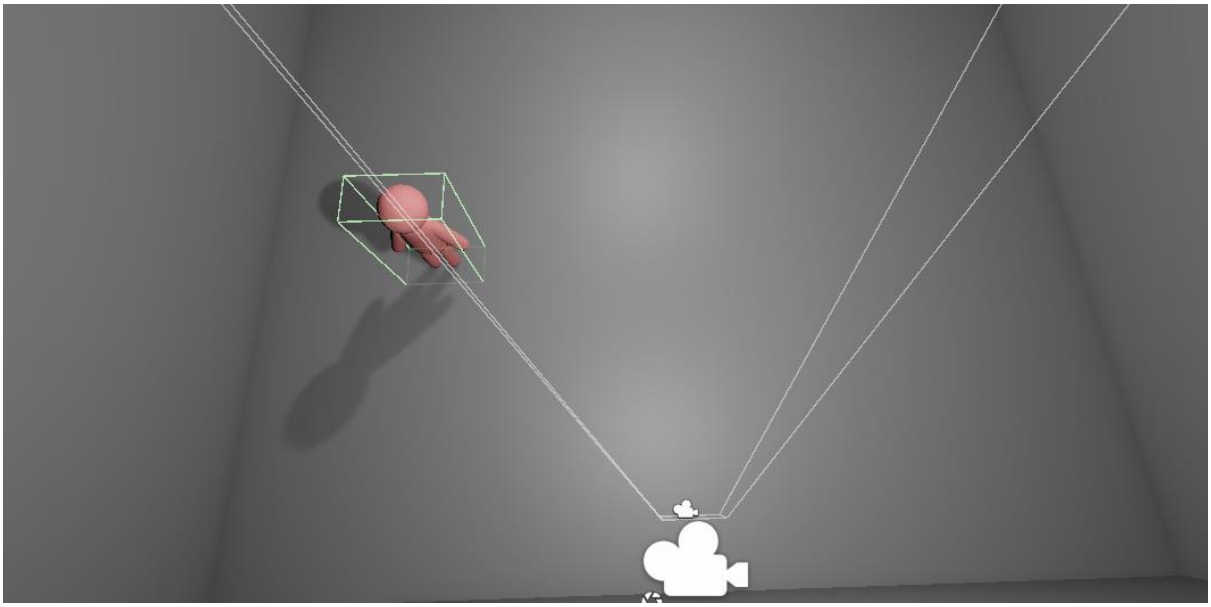

Рис. 1. Пересечение усечённой пирамиды поля зрения и ограничительной рамки цели

2. Угол между вектором, обозначающим направление взгляда персонажа, и вектором между персонажем и целевым объектом должен быть меньше, чем половина заранее заданного угла обзора персонажа.
3. Raycast, выпущенный из координат персонажа в сторону целевого объекта, не должен пересечь по пути какой-либо посторонний объект (рис. 2).

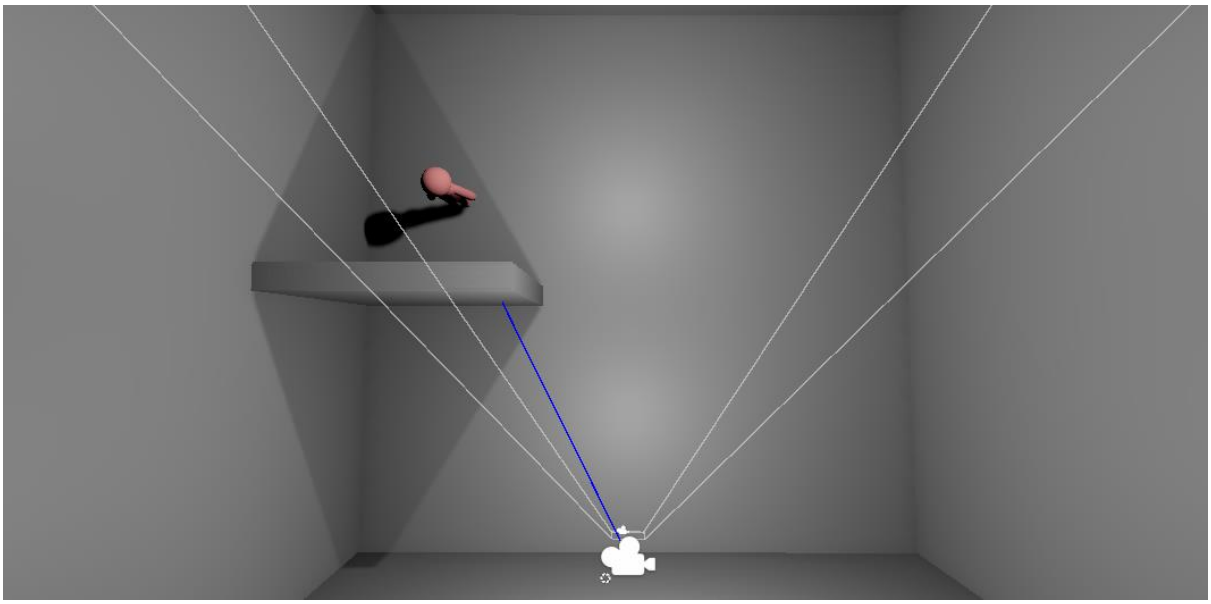

Рис. 2. При трассировке лучей в сторону цели луч сталкивается с препятствием

Метод прост в реализации, однако не учитывает визуальные эффекты, такие как тени или освещение, что снижает его применимость в сложных сценах.

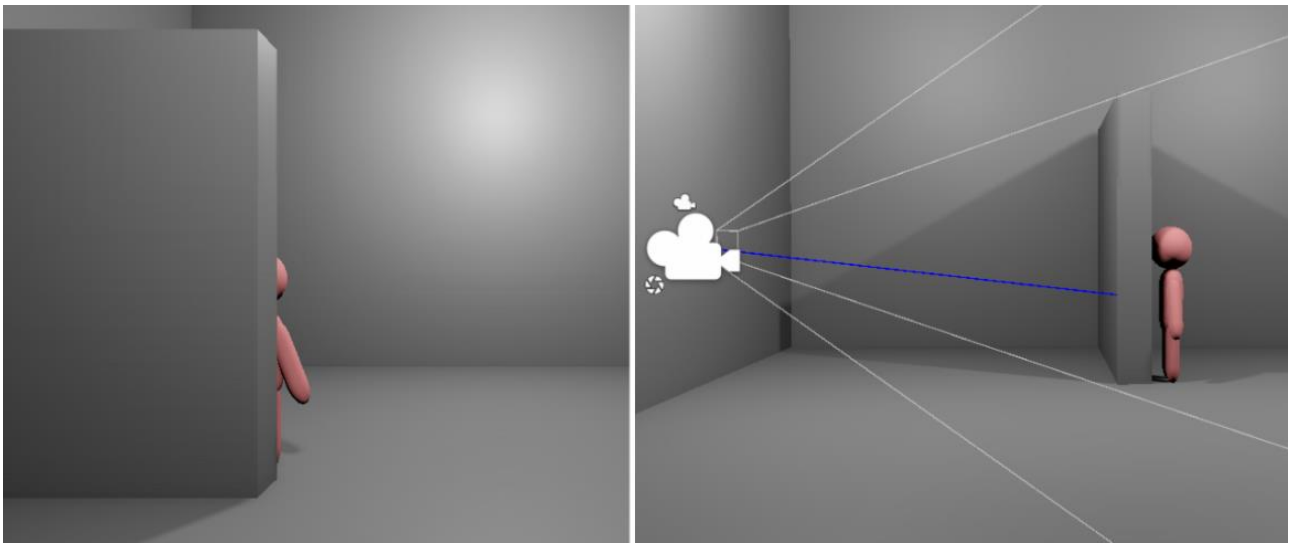

Рис. 3. Raycast заблокирован препятствием несмотря на то, что значительная часть цели видна наблюдателю

Использование одного Raycast может привести к ситуации, когда относительно небольшой объект, например тонкая труба, может заблокировать выпущенный луч несмотря на то, что с точки зрения реального человека персонаж должен видеть небольшую часть целевого объекта (рис. 3).

Другой подход сосредоточен на стелс-механиках, предлагая модель постепенного обнаружения с учетом расстояния и угла обзора [11]. Этот подход повышает реализм в играх с элементами скрытности, но не решает проблему ложных срабатываний, вызванных тенями или отражениями. Авторы отмечают, что мгновенное обнаружение персонажа игрока может показаться нечестным и привести к неудовлетворению игрой, поэтому неигровым персонажам обычно требуется некоторое время, чтобы «заметить» игрока, уже находящегося внутри их поля зрения. По аналогии с реальной жизнью, позиции вблизи неигровых персонажей и прямо перед направлением их взгляда часто связывают с большим риском обнаружения, чем позиции вдали от них и/или на границе их «периферийного зрения».

Плагин NPC Eyes Sight System для Unreal Engine использует точки на скелетных сетках и их проекции на тени, что улучшает обнаружение частично видимых объектов [12]. В такой реализации неигровые персонажи используют для обнаружения других персонажей точки на их скелетных сетках (англ. skeletal mesh). Если искусственный интеллект внутри игры обнаружит в своём поле обзора хотя бы одну такую точку, не прикрытую препятствием, он тут же узнает о местоположении персонажа. Такая реализация значительно уменьшает количество ситуаций, когда небольшое препятствие, лишь частично закрывающее целевой объект, полностью предотвращает его обнаружение персонажами. Однако проецирование множества точек относительно множества возможных источников света – непростая операция, которая может оказать значительное влияние на производительность игры.

Для анализа изображений с камеры NPC в [13] исследовано применение методов машинного зрения, таких как сверточные нейронные сети. Для определения и классификации объектов в поле зрения игрового агента предложено использовать алгоритмы компьютерного зрения. Этот подход обеспечивает высокую точность распознавания, приближая восприятие NPC к человеческому, и позволяет имитировать такие особенности восприятия, как распознавание образов и категоризация объектов. Однако вычислительная сложность этого подхода делает его непригодным в реальном времени без специализированного оборудования, что ограничивает его применимость в обычных игровых проектах.

Для разработки поведения NPC предложен альтернативный подход [14, 15], основанный на применении алгоритмов машинного обучения. Этот метод отходит от жесткого назначения наград и наказаний разработчиком, делегируя эту задачу плагину, который анализирует данные о состоянии агента и автоматически классифицирует поведение на «хорошее» и «плохое» с учетом выбранного целевого параметра. Хотя данный подход направлен на общую систему принятия решений NPC, а неспецифически на визуальное восприятие, его методология динамической настройки поведения в реальной игровой среде может быть адаптирована и для задач визуального восприятия.

Более сложные и реалистичные системы, обеспечивающие правдоподобное поведение NPC, опираются на расширенные подходы: в [16–20] рассмотрены архитектурные решения, позволяющие выбирать тактические позиции и моделировать восприятие в условиях динамической игровой среды. Указанные работы подчёркивают, что ограничение количества тестов видимости в реальном времени может негативно сказаться на многообразии реакций NPC, что оправдывает необходимость разработки более эффективных методов оценки видимости.

Перспективное применение нейронных методов с использованием всенаправленных расстояний (Neural Omnidirectional Distance Fields), представленное в [21], демонстрирует значительное ускорение проверки видимости, что позволяет снизить вычислительную нагрузку по сравнению с традиционными подходами. Это подтверждает актуальность поиска альтернатив классическим raycasting-методам.

Алгоритм, предложенный в настоящей работе, сочетает сравнение текстур, рейкасты и оптимизацию, что отличает его от существующих решений и характеризуется простотой, универсальностью и производительностью.

2. РАЗРАБОТКА АЛГОРИТМА

Для реализации системы зрения игрового агента необходимо определить, видит ли наблюдатель цель в своём поле зрения. Простые методы: проверка углов обзора и расстояния – не учитывают сложные визуальные эффекты, такие как освещение, тени, отражения и маскировка. Поэтому необходим более точный и гибкий подход, который мог бы учитывать визуальные особенности сцены.

2.1. Основной алгоритм

Проблема: традиционные методы определения видимости не учитывают визуальные эффекты, такие как освещение, тени и маскировка, что снижает реализм поведения NPC.

Решение: основной алгоритм системы зрения основан на сравнении изображений, полученных с двух камер, прикреплённых к наблюдателю. Камера 1 рендерит сцену с целью, а Камера 2 — без цели. Обе камеры рендерят ту же сцену, что и основная камера игрока, включая уровень и освещение. Это позволяет учитывать все указанные выше визуальные эффекты (рис. 4).

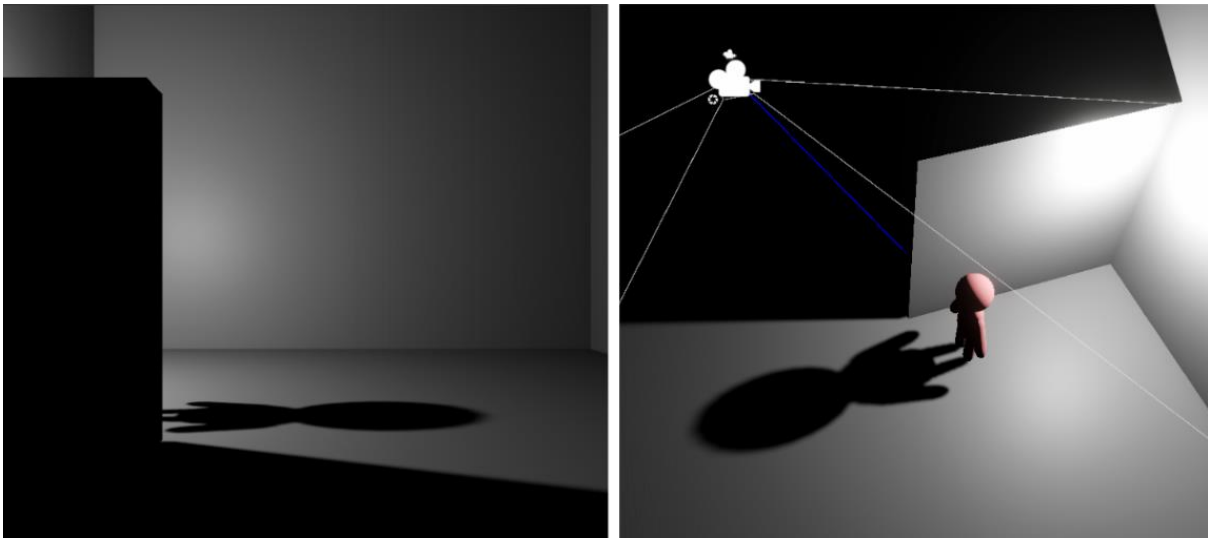

Рис. 4. Наблюдатель видит тень цели, но не саму цель

Шаги алгоритма:

1. Рендеринг сцен в текстуры. Камера 1 включает цель в рендер, а Камера 2 — исключает её.

2. Конвертация текстур с камер в формат Texture2D для дальнейшей обработки.

3. Сравнение пикселей. Вычисление разности пикселей V по каналам RGB и их суммирование для получения степень заметности цели. Сравнение V с пороговым значением $V_{\min}$. Для каждой пары соответствующих пикселей текстур вычисляется их разность по каналам RGB. Эти разности суммируются, чтобы получить степень заметности цели V:

$$V = \frac{\sum_{i=1}^{N}(|R_1 - R_2| + |G_1 - G_2| + |B_1 - B_2|)}{3\ N},$$

где R_1, G_1, B_1 – значения каналов RGB для i-го пикселя на 1-ой текстуре, R_2, G_2, B_2 – значения каналов RGB для i-го пикселя на 2-ой текстуре, N – общее количество пикселей.

4. Пороговое значение. Если V превышает пороговое значение $V_{\min}$, считается, что наблюдатель видит цель, т. е. если $V > V_{\min}$, то цель видима.

Преимущества:

- Алгоритм учитывает визуальные особенности сцены (освещение и тени).
- Гибкость: можно настраивать пороговое значение $V_{\min}$ для разных наблюдателей.

Недостатки:

- Высокая вычислительная сложность из-за необходимости рендерить сцену несколько раз и вычислять разность текстур.
- Чувствительность к мелким деталям, что может привести к ложным срабатываниям.

2.2. Маскировка объектов

Проблема: изначальный алгоритм сравнивает отдельные пиксели, что делает его чрезмерно чувствительным к мелким деталям. Например, если цель – это персонаж, одетый в лесной камуфляж на фоне лесной сцены, даже небольшие различия в текстуре могут привести к обнаружению. Это не позволяет объекту «сливаться» с окружающей обстановкой.

Решение: для уменьшения чувствительности к мелким деталям применяется размытие по Гауссу. Размытие снижает резкость изображения, что позволяет объекту маскироваться. Такой подход также является хорошим приближением того, как работает зрительное восприятие человека в реальности [22, 23]. Сложность алгоритма размытия по Гауссу составляет $O(nr^2)$, где n – количество пикселей, а r – радиус круга, по которому усредняется каждый пиксель (другими словами, сила размытия).

Вместо традиционного гауссова фильтра можно применять фильтр Box Blur, который выполняет простое усреднение значений пикселей в квадратном окне.

При нескольких последовательных пропусках Box Blur позволяет достичь результата, практически неотличимого от размытия по Гауссу, однако имеет линейную вычислительную сложность $O(n)$, что значительно снижает затраты по сравнению с гауссовым фильтром [24].

Шаги решения:

1. Размытие по Гауссу: перед сравнением текстур применяется размытие по Гауссу. Это уменьшает резкость изображения и снижает чувствительность алгоритма к мелким деталям. Радиус определяет степень размытия и обычно устанавливается в диапазоне от 2 до 5 пикселей.
2. Среднеквадратичная ошибка (MSE): вместо простой суммы разностей используется среднеквадратичная ошибка для расчёта V:

$$V = \frac{\sum_{i=1}^{N}((R_1-R_2)^2+(G_1-G_2)^2+(B_1-B_2)^2)}{3\ N},$$

что позволяет уменьшить влияние незначительных изменений в поле зрения и увеличить вес серьёзных различий.

Преимущества:

- Уменьшение чувствительности к мелким деталям.
- Возможность маскировки объектов в сложных условиях (например, камуфляж).
- Увеличение устойчивости к шумам и артефактам.

Недостатки:

- Увеличение вычислительной сложности из-за размытия.
- Необходимость настройки параметров размытия для разных сцен.

2.3. Проверка прямой видимости

Проблема: наблюдатель может реагировать на тени, отражения или малую часть объекта так же, как и на сам объект. Это приводит к ложным срабатываниям, когда наблюдатель «видит» цель, хотя на самом деле видит только её тень или отражение.

Решение: для устранения этой проблемы введена проверка рейкастами. На модели цели равномерно распределяются X точек, которые проверяются на видимость с помощью рейкастов (рис. 5).

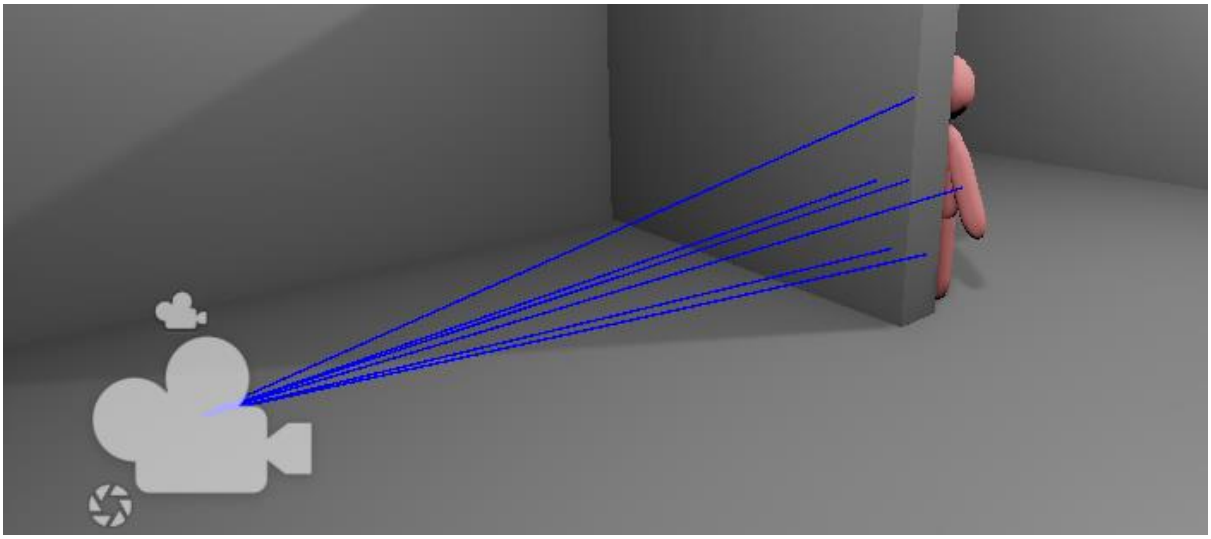

Рис. 5. Пучок из 6 рейкастов, направленных в разные части цели

Если количество X_{pass} точек, прошедших проверку, превышает пороговое значение $X_{\min}$, считается, что наблюдатель видит сам объект. Если $X_{\text{pass}} < X_{\min}$, то наблюдатель видит тень, отражение или малую часть объекта. В этом случае наблюдатель получает информацию о координатах цели и при помощи отдельной логики пытается получить визуальное подтверждение объекта.

Шаги решения:

1. Распределение точек на цели: на модели цели равномерно распределяются X точек, где X обычно принимает значения от 5 до 20 в зависимости от размера и сложности модели.
2. Проверка рейкастами: для каждой точки выполняется рейкаст от наблюдателя до точки. Если луч не пересекает препятствий, точка считается видимой.
3. Определение видимости цели: если количество X_{pass} видимых точек превышает пороговое значение $X_{\min}$, цель считается видимой. В противном случае наблюдатель видит только тень или отражение.

4. Попытка визуального подтверждения: если наблюдатель обнаружил объект, но не имеет прямой видимости, то он попытается получить визуальное подтверждение объекта: например, повернуться в его сторону или же при помощи алгоритма поиска пути дойти до позиции объекта.

Преимущества:

- Уменьшение ложных срабатываний.
- Более реалистичное поведение наблюдателя.
- Возможность настройки порогового значения $X_{\min}$ для разных ситуаций.

Недостатки:

- Увеличение вычислительной сложности из-за необходимости выполнения рейкастов.
- Необходимость настройки количества точек и порогового значения.
- Необходимость равномерной расстановки точек на модели объекта.

2.4. Влияние скорости движения объекта на его заметность

Проблема: в реальности быстро движущиеся объекты заметить легче, чем медленно движущиеся или статичные. Текущий алгоритм не учитывает скорость движения цели, что может привести к нереалистичному поведению наблюдателя.

Решение: для учёта скорости движения цели введён параметр eV – эффективное значение V, который рассчитывается как произведение V на множитель, зависящий от скорости движения цели. Чем выше скорость цели, тем больше множитель, что увеличивает заметность объекта.

Шаги решения:

1. Расчёт скорости цели: определяется скорость движения цели на основе её позиции в предыдущих кадрах.
2. Расчёт множителя: множитель α для V рассчитывается на основе скорости цели. Например, множитель может быть линейно зависимым от скорости.
3. Расчёт V: эффективное значение V рассчитывается как $eV = V\alpha$.
4. Если $eV > V_{\min}$, значит, наблюдатель видит цель.

Преимущества:

- Более реалистичное поведение наблюдателя.
- Динамическая система, учитывающая движение объекта.
- Создает дополнительный геймплейный элемент для стелс-игр, где игрок должен двигаться медленно, чтобы оставаться незамеченным.

Недостатки:

- Необходимость настройки зависимости множителя от скорости.
- Может потребоваться дополнительная балансировка для предотвращения слишком легкого обнаружения быстро движущихся объектов.

2.5. Механика постепенного обнаружения

Проблема: в некоторых случаях обнаружение не должно быть мгновенным. Например, наблюдатель может постепенно «замечать» цель, что добавляет реализма и в случае стелс-игр делает геймплей более честным с точки зрения игрока в ситуации, когда наблюдателями являются противники, а объектом – сам игрок.

Решение: для реализации постепенного обнаружения введён параметр Det – степень обнаружения, который изменяется в зависимости от значения V и скорости обнаружения $\mathrm{Det}_{\mathrm{speed}}$.

Если $V > V_{min}$, то Det увеличивается на $\Delta\mathrm{Det}\,\mathrm{Det}_{\mathrm{speed}}$ каждый кадр. Если $V < V_{min}$, то Det уменьшается на $\Delta\mathrm{Det}\,\mathrm{Det}_{\mathrm{speed}}$. Когда Det превысит пороговое значение $\mathrm{Det}_{\min}$, цель считается обнаруженной.

Шаги решения:

1. Расчёт $\Delta\mathrm{Det}$: определяется изменение степени обнаружения на основе значения V.

2. Изменение Det: каждый кадр Det изменяется на $\Delta\mathrm{Det}\ \mathrm{Det}_{\mathrm{speed}}$ в зависимости от того, превышает ли V пороговое значение $\mathrm{V}_{\mathrm{min}}$.

3. Обнаружение цели: если $\mathrm{Det} > \mathrm{Det}_{\mathrm{min}}$, цель считается обнаруженной.

Преимущества:

- Более реалистичное поведение наблюдателя.
- Возможность настройки скорости обнаружения для разных наблюдателей.
- Улучшенный игровой опыт для стелс-игр, дающий игроку время на реакцию.

Недостатки:

- Необходимость настройки параметров $\mathrm{Det}_{\mathrm{speed}}$ и $\mathrm{Det}_{\mathrm{min}}$.
- Может потребоваться дополнительная логика для реакции наблюдателя на «подозрительный» объект, когда $0 < \mathrm{Det}_{\mathrm{min}} < \mathrm{Det}_{\mathrm{min}}$.

2.6. Оптимизация

Проблема: алгоритм имеет высокую вычислительную сложность, что может негативно сказаться на производительности игры, особенно при большом количестве наблюдателей и целей.

Решение: для повышения производительности предложены следующие оптимизации.

1. Использование Unity Job System: алгоритм выполняется в многопоточном режиме с использованием Unity Job System, что позволяет распределить нагрузку на несколько ядер процессора.

2. Уменьшение частоты расчёта V: расчёт V выполняется не в каждом кадре, а с определённым интервалом, что снижает нагрузку на систему.

3. Активация рендера камер: рендер 1 и 2 камер активируется только в тех кадрах, когда необходимо вычислить V, и отключается в остальное время.

4. Исключение заведомо невидимых объектов: алгоритм не применяется для объектов, которые заведомо находятся вне поля зрения наблюдателя (например, слишком далеко или в другой части сцены).

5. Снижение разрешения текстур: для расчёта V используются текстуры с более низким разрешением, чем основная камера игры, что значительно снижает количество пикселей для обработки.

6. Кэширование результатов: результаты рейкастов кэшируются на несколько кадров, если позиции наблюдателя и цели значительно не изменились.

Преимущества:

- Увеличение производительности до 300% по сравнению с наивной реализацией.
- Снижение нагрузки на CPU с 12–15% до 3–5% в среднем.
- Возможность масштабирования для большого количества наблюдателей.

Недостатки:

- Необходимость настройки интервалов расчёта V.
- Усложнение кода из-за многопоточности.
- Потенциальное снижение точности при использовании низкого разрешения текстур.

Помимо описанных оптимизаций, перспективным направлением является применение нейронных методов для аппроксимации проверки видимости. Как показано в [21], использование нейронных Omnidirectional Distance Fields позволяет заменить традиционные рейкасты и ускорить проверку видимости до 9 раз по сравнению с классическим подходом. Этот подход демонстрирует потенциал для дальнейшей оптимизации систем искусственного интеллекта в играх, особенно в сценариях с большим числом наблюдателей.

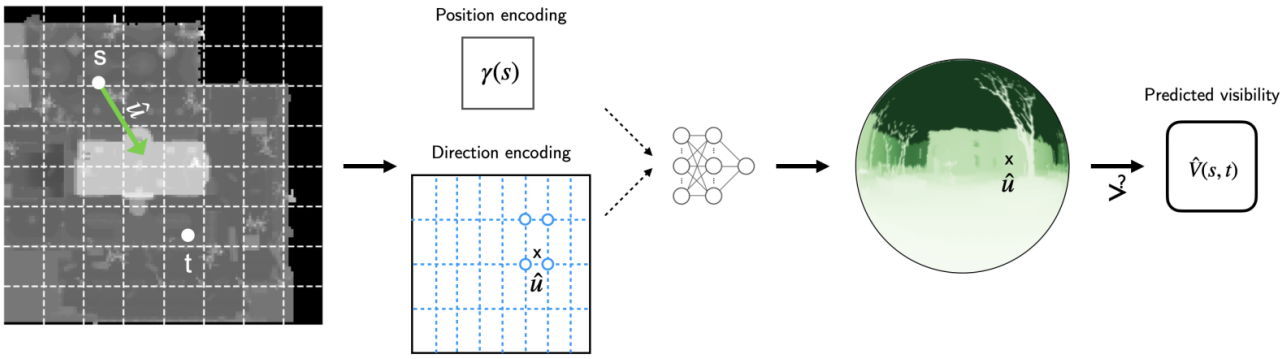

Рис. 6. Подход к аппроксимации видимости между 3D-позициями в статичных игровых сценах с помощью нейронного представления поля всенаправленных расстояний (ODF) [21]

3. ТЕСТИРОВАНИЕ

3.1. Методология тестирования

Для оценки эффективности разработанного алгоритма было проведено комплексное тестирование по нескольким аспектам:

1. Тестирование производительности.
   - Измерение FPS[1] при различном количестве наблюдателей (от 1 до 20).
   - Сравнение с традиционными методами на одинаковых сценах.
   - Профилирование CPU и GPU использования.
2. Тестирование точности обнаружения.
   - Определение процента успешных обнаружений в различных условиях видимости.
   - Оценка частоты ложных срабатываний.
   - Сравнение с ожидаемыми результатами, основанными на восприятии человека.
3. Тестирование в различных игровых условиях.
   - Дневная и ночная сцены с разным освещением.
   - Тестирование маскировки объектов (камуфляж, тени).
   - Сцены с отражающими поверхностями (зеркала, вода).

[1] FPS сокр. от англ. Frame per seconds – количество кадров в секунду.

3.2. Инструменты тестирования

Для проведения тестов были разработаны следующие вспомогательные компоненты.

1. Тепловая карта отличий текстур: визуализирует различие между текстурами с камер 1 и 2, помогая настраивать пороговые значения и улучшать алгоритм.

2. Панель отладки: позволяет во время игры включать/выключать различные компоненты алгоритма (размытие, проверка рейкастами и т. д.), что упрощает тестирование.

3. Система логирования: записывает все случаи обнаружения и параметры наблюдателя и цели для дальнейшего анализа.

4. Сцены для тестирования: созданы специальные сцены, имитирующие различные игровые ситуации, с контролируемыми параметрами освещения и геометрии.

5. Функциональное тестирование: обход технического ограничения стандартного класса Input в Unity путём создания системы автоматизации тестирования, которая позволяет записывать и воспроизводить сценарии взаимодействия пользователя с виртуальной средой [25]. Для тестирования алгоритма визуального восприятия был разработан набор сценариев, моделирующих различные ситуации обнаружения: движение объекта с разной скоростью, перемещение в условиях разной освещенности, попытки маскировки и другие. Применение такого подхода позволило значительно сократить время тестирования за счёт автоматического многократного прогона идентичных тестовых сценариев и сравнения результатов с ожидаемыми, что особенно эффективно для проверки функциональности обнаружения объектов в сложных условиях, когда требуется точная имитация различных входных данных.

3.3. Результаты тестирования

Производительность

Измерения производительности показали, что оптимизированная версия алгоритма добавляет в среднем 3–5% нагрузки на CPU по сравнению с традици-

онным подходом. При этом использование Unity Job System позволило эффективно распределить вычисления между ядрами процессора, что особенно заметно на многоядерных системах. Тестирование на различных устройствах показало, что алгоритм хорошо масштабируется и может быть настроен для разных уровней производительности.

Тестирование точности обнаружения показало значительное улучшение по сравнению с традиционными методами. Разработанный алгоритм значительно лучше работает в сложных условиях видимости и имеет гораздо меньшую частоту ложных срабатываний, когда видны только тени или отражения объекта.

Существенное влияние на эффективность алгоритма оказывает настройка параметров.

1. Влияние радиуса размытия: увеличение радиуса размытия от 2 до 5 пикселей повышало эффективность маскировки на 32%, но снижало точность определения границ объекта.
2. Влияние порогового значения $V_{\min}$: оптимальное значение $V_{\min}$ составило 0.03 для большинства сцен, обеспечив баланс между чувствительностью и устойчивостью к шумам.
3. Влияние количества точек для рейкастов: увеличение числа точек с 5 до 15 повышало точность определения прямой видимости на 27%, но увеличивало нагрузку на CPU на 8%.
4. Влияние $\mathrm{Det}_{\mathrm{speed}}$: значения $\mathrm{Det}_{\mathrm{speed}}$ в диапазоне 0.5–1.0 обеспечивали наиболее реалистичное постепенное обнаружение для стелс-игр.

3.4. Рекомендации по настройке

На основе проведенных тестов были сформулированы следующие рекомендации по настройке параметров алгоритма.

1. Для игр с высоким темпом действия:
   - $V_{\min}$ = 0.04–0.05 (для быстрого обнаружения);
   - $\mathrm{Det}_{\mathrm{speed}}$ = 1.5–2.0 (для быстрой реакции);
   - интервал расчета *V* = 0.1–0.2 с.;
   - количество точек для рейкастов равно 5–8.

2. Для стелс-игр:

- $V_{\min}$ = 0.02–0.03 (для более тонкого обнаружения);
- $\mathrm{Det}_{\mathrm{speed}}$ = 0.3–0.7 (для постепенного обнаружения);
- интервал расчета V = 0.2–0.3 с.;
- количество точек для рейкастов равно 10–15.

3. Для игр с ограниченными ресурсами (мобильные платформы):

- коэффициент уменьшения разрешения равен 3–4;
- пропуск проверки прямой видимости для отдаленных объектов;
- увеличение интервала расчета V до 0.4–0.5 с.;
- использование более простого алгоритма размытия.

4. РЕЗУЛЬТАТЫ И ИХ ОБСУЖДЕНИЕ

Разработанный алгоритм успешно учитывает визуальные эффекты, такие как освещение и маскировка, и обеспечивает гибкость настройки для различных типов NPC. Оптимизация позволяет применять его в реальном времени, однако тестирование выявило необходимость калибровки параметров в зависимости от сцены.

4.1. Сравнение с существующими решениями

Сравнение (табл. 1) с традиционными методами [10] показало, что предложенный подход превосходит их по реализму, но уступает в простоте реализации. В отличие от решений с машинным зрением [13], алгоритм не требует мощного оборудования, что делает его более доступным.

Таблица 1. Сравнение с существующими решениями

Характеристика	Традиционный подход [10]	NPC Eyes Sight System [12]	CV [13]	Разработанный алгоритм
Учет освещения и теней	низкий	средний	высокий	высокий
Реакция на маскировку и камуфляж	низкий	низкий	высокий	средний
Различение объекта и его тени/отражения	низкий	средний	высокий	высокий
Вычислительная сложность	низкая	средняя	очень высокая	средняя
Настраиваемость для разных типов NPC	средняя	средняя	высокая	высокая

4.2. Преимущества разработанного алгоритма

1. Универсальность: алгоритм может быть настроен для различных игровых жанров и условий видимости.
2. Учет визуальных эффектов: система учитывает освещение, тени, отражения и маскировку, что значительно повышает реализм.
3. Оптимизация: благодаря предложенным методам оптимизации, алгоритм может эффективно работать даже на устройствах с ограниченными ресурсами.
4. Постепенное обнаружение: механика постепенного обнаружения делает поведение NPC более реалистичным и улучшает игровой опыт в стелс-играх.
5. Гибкость настройки: широкий набор параметров позволяет тонко настроить систему для каждого типа NPC и игровой ситуации.

4.3. Ограничения и направления дальнейшего развития

Несмотря на значительные преимущества, у разработанного алгоритма есть ряд ограничений:

1. Зависимость от качества рендеринга: Качество работы алгоритма зависит от графического движка и качества рендеринга сцены.

2. Необходимость настройки параметров: Для оптимальной работы в различных сценах требуется тщательная настройка параметров.

3. Ограниченная применимость для динамически изменяющихся условий: при резком изменении освещения или окружения может потребоваться перенастройка параметров.

Направления дальнейшего развития алгоритма включают:

1. Адаптивная настройка параметров: разработка механизма автоматической подстройки параметров в зависимости от условий сцены.

2. Интеграция с другими системами восприятия: комбинирование визуального восприятия со слуховым и другими видами сенсоров для более комплексного поведения NPC.

3. Применение методов машинного обучения: использование упрощенных нейронных сетей для классификации видимых объектов без значительного увеличения вычислительной нагрузки.

ЗАКЛЮЧЕНИЕ

Представлен новый подход к реализации системы визуального восприятия для неигровых персонажей в видеоиграх. Разработанный алгоритм основан на сравнении изображений с двух камер и учитывает сложные визуальные эффекты (освещение, тени и маскировка), что значительно повышает реализм поведения NPC.

Ключевые достижения работы включают:

1. Разработку основного алгоритма сравнения изображений для определения видимости объектов.

2. Реализацию дополнительных механик, таких как учет маскировки, проверка прямой видимости, влияние скорости движения и постепенное обнаружение.

3. Оптимизацию алгоритма для работы в реальном времени с использованием Unity Job System, динамической активации камер и других методов.

4. Всестороннее тестирование алгоритма в различных условиях и формулирование рекомендаций по его настройке.

Результаты тестирования показали, что разработанный алгоритм обеспечивает более реалистичное поведение NPC по сравнению с традиционными методами, особенно в сложных условиях видимости, при этом сохраняя приемлемую производительность.

Предложенный подход может найти применение в различных жанрах видеоигр, особенно в играх с элементами скрытности, где реалистичное визуальное восприятие NPC является критическим фактором для создания увлекательного игрового процесса.

Дальнейшие исследования могут быть направлены на адаптивную настройку параметров алгоритма, интеграцию с другими системами восприятия и применение методов машинного обучения для улучшения классификации видимых объектов.

СПИСОК ЛИТЕРАТУРЫ

1. *Миллингтон Я.* Искусственный интеллект для игр / Я. Миллингтон, Дж. Фанж. СПб.: Питер, 2021. 816 с.

2. *Рабинович З.Л.* Методы и алгоритмы искусственного интеллекта в компьютерных играх: учеб. пособие. М.: Физматлит, 2018. 320 с.

3. *Петров А.В.* Искусственный интеллект в трехмерных играх. Программирование и моделирование поведения персонажей. М.: ДМК Пресс, 2019. 452 с.

4. *Buckland M.* Programming Game AI by Example. Sudbury: Jones & Bartlett Learning, 2022. 522 c.

5. *Ostuni D., Galante E.T.* Towards an AI playing Touhou from pixels: a dataset for real-time semantic segmentation // 2021 IEEE Conference on Games (CoG). 2021. P. 1–5. https://doi.org/10.1109/CoG52621.2021.9619112.

6. *Tutum C., AbdulQuddos S., Miikkulainen R.* Generalization of Agent Behavior through Explicit Representation of Context // 2021 IEEE Conference on Games (CoG). 2021. P. 1–7. https://doi.org/10.1109/CoG52621.2021.9619141.

7. *Guerrero-Romero C., Perez-Liebana D.* MAP-Elites to Generate a Team of Agents that Elicits Diverse Automated Gameplay // 2021 IEEE Conference on Games (CoG). 2021. P. 1–8. https://doi.org/10.1109/CoG52621.2021.9619142.

8. *Glassner A.S. (Ed*.). An introduction to ray tracing. Morgan Kaufmann, 1989.

9. *Panwar H.* The NPC AI of the last of us: a case study // arXiv preprint arXiv:2207.00682. 2022.

10. *Mahmoud I., Jaffal Y., Wloka D*. A Vision Simulation Algorithm for Non-Player Character in Static Scene // University of Kassel, Germany. 2014. P. 6.

11. *Tremblay J., Torres P.A., Verbrugge C.* Measuring Risk in Stealth Games // Foundations of Digital Games. Liberty of the Seas, 2014. P. 8.

12. NPC Eyes Sight System – PRO.
URL: https://www.fab.com/listings/6b54716a-dd21-414d-b78f-384068de14b7

13. *Erdelyi C.* Using Computer Vision Techniques to Play an Existing Video Game // California State University San Marcos. 2019. P. 49.

14. *Паренюк Л.Н., Кугуракова В.В.* Разработка плагина поведения NPC для игрового движка Unity // Электронные библиотеки. 2020. Т. 23(5). С. 1044–1057. http://doi.org/10.26907/1562-5419-2020-23-5-1044-1057.

15. *Estgren M.* Modelling NPC perception using supervised learning // Uppsala University, Sweden. 2021. 8 p. URL: https://sciion.se/assets/papers/npc-perception.pdf

16. *Bourg D.M., Seemann G.* AI for Game Developers. O'Reilly Media, Inc. 2004.

17. *Jack M.* Tactical Position Selection: An Architecture and Query Language. In Game AI Pro 360. CRC Press. 2019. P. 1–24.

18. *McIntosh T.* Human Enemy AI in The Last of Us. In Game AI Pro 360. CRC Press, 2019. P. 13–24.

19. *Welsh R.* Crytek's Target Tracks Perception System. In Game AI Pro: Collected Wisdom of Game AI Professionals. 2013. Vol. 403. 411 p.

20. *Walsh M.* Modeling Perception and Awareness in Tom Clancy's Splinter Cell Blacklist. In Game AI Pro 360. CRC Press. 2019. P. 73–86.

21. *Ying Z., Edwards N., Kutuzov M.* Efficient Visibility Approximation for Game AI using Neural Omnidirectional Distance Fields // Proceedings of the ACM on Computer Graphics and Interactive Techniques. 2024. Vol. 7, No. 1. P. 1–15.

22. Image Comparison Tuned to Human Perception // Computer Science Stack Exchange. 2015.
URL: https://cs.stackexchange.com/questions/48862/image-comparison-tuned-to-human-perception

23. *Pramod R.T., Katti H., Arun S.P.* Human peripheral blur is optimal for object

recognition // Vision Research. 2022. Vol. 200. P. 108083.

24. Fastest Gaussian Blur (in Linear Time) // Algorithms and Stuff. 2014. URL: https://blog.ivank.net/fastest-gaussian-blur.html

25. *Кугуракова В.В., Бедрин О.А.* Система автоматизации функционального тестирования для платформы Unity // Вестник компьютерных и информационных технологий. 2020. Т. 17, № 12. С. 47–52. https://doi.org/10.14489/vkit.2020.12.pp.047-052

DEVELOPMENT OF A VISUAL PERCEPTION SYSTEM FOR GAME AGENTS IN VIDEO GAMES

A. M. Primachenko[1] [0009-0008-6396-2035], M. R. Khafizov[2] [0000-0001-7275-9102]

[1, 2]Kazan Federal University, Kazan, 420008, Russia

[1]primachenko.artem@mail.ru, [2]murkorp@gmail.com

Abstract

The developed algorithm of the visual perception system for game agents, implemented in the Unity game engine, is presented. The proposed method is based on the comparison of images from two cameras, taking into account complex visual effects (lighting, shadows, camouflage), and supplemented with line-of-sight verification, taking into account the speed of the object, and the mechanics of gradual detection. Testing of the system has shown a significant increase in realistic detection compared to traditional methods, while maintaining performance within a small additional load on the processor. The algorithm was optimized using Unity Job System and dynamic camera activation. The scientific literature on similar solutions has also been analyzed and their strengths and weaknesses have been identified. The results can be applied in video game development to create realistic behavior of non-player characters, especially in games with stealth elements.

Keywords: *video games, artificial intelligence, perception system, NPC, non-player characters, game agents, stealth mechanics, Unity, rendering, computer vision, optimization, game design.*

REFERENCES

1. *Millington I., Funge J.* Artificial Intelligence for Games. Saint Petersburg: Piter. 2021. 816 p. (in Russian).

2. *Rabinovich Z.L.* Methods and Algorithms of Artificial Intelligence in Computer Games: A Tutorial. Moscow: Fizmatlit. 2018. 320 p. (in Russian).

3. Petrov A.V. Artificial Intelligence in 3D Games: Programming and Modelling of Character Behavior. Moscow: DMK Press. 2019. 452 p. (in Russian).

4. *Buckland M.* Programming Game AI by Example. Sudbury: Jones & Bartlett Learning, 2022. 522 p.

5. *Ostuni D., Galante E.T.* Towards an AI playing Touhou from pixels: a dataset for real-time semantic segmentation // 2021 IEEE Conference on Games (CoG). 2021. P. 1–5. https://doi.org/10.1109/CoG52621.2021.9619112.

6. *Tutum C., AbdulQuddos S., Miikkulainen R.* Generalization of Agent Behavior through Explicit Representation of Context // 2021 IEEE Conference on Games (CoG). 2021. P. 1–7. https://doi.org/10.1109/CoG52621.2021.9619141.

7. *Guerrero-Romero C., Perez-Liebana D.* MAP-Elites to Generate a Team of Agents that Elicits Diverse Automated Gameplay // 2021 IEEE Conference on Games (CoG). 2021. P. 1–8. https://doi.org/10.1109/CoG52621.2021.9619142.

8. *Glassner A.S. (Ed*.). An introduction to ray tracing. Morgan Kaufmann, 1989.

9. *Panwar H.* The NPC AI of the last of us: a case study.
arXiv preprint arXiv:2207.00682. 2022.

10. *Mahmoud I., Jaffal Y., Wloka D*. A Vision Simulation Algorithm for Non-Player Character in Static Scene // University of Kassel, Germany. Kassel, 2014. P. 6.

11. *Tremblay J., Torres P.A., Verbrugge C.* Measuring Risk in Stealth Games // Foundations of Digital Games. Liberty of the Seas, 2014. P. 8.

12. NPC Eyes Sight System PRO.
URL: https://www.fab.com/listings/6b54716a-dd21-414d-b78f-384068de14b7

13. *Erdelyi C.* Using Computer Vision Techniques to Play an Existing Video Game // California State University San Marcos. 2019. P. 49.

14. *Parenyuk L.N., Kugurakova V.V. NPC behavior plugin development for game engine Unity* // Russian Digital Libraries. 2020. Vol. 23 (5). P. 1044–1057.
https://doi.org/10.26907/1562-5419-2020-23-5-1044-1057 (In Russian).

15. *Estgren M.* Modelling NPC perception using supervised learning // Uppsala University, Sweden. 2021. 8 p.
URL: https://sciion.se/assets/papers/npc-perception.pdf

16. *Bourg D.M., Seemann G.* AI for Game Developers. O'Reilly Media, Inc. 2004.

17. *Jack M.* Tactical Position Selection: An Architecture and Query Language. In Game AI Pro 360. CRC Press. 2019. P. 1–24.

18. *McIntosh T.* Human Enemy AI in The Last of Us. In Game AI Pro 360. CRC Press, 2019. P. 13–24.

19. *Welsh R.* Crytek's Target Tracks Perception System. In Game AI Pro: Collected Wisdom of Game AI Professionals. 2013. Vol. 403. 411 p.

20. *Walsh M.* Modeling Perception and Awareness in Tom Clancy's Splinter Cell Blacklist. In Game AI Pro 360. CRC Press. 2019. P. 73–86.

21. *Ying Z., Edwards N., Kutuzov M.* Efficient Visibility Approximation for Game AI using Neural Omnidirectional Distance Fields // Proceedings of the ACM on Computer Graphics and Interactive Techniques. 2024. Vol. 7, No. 1. P. 1–15.

22. Image Comparison Tuned to Human Perception // Computer Science Stack Exchange. 2015.
URL: https://cs.stackexchange.com/questions/48862/image-comparison-tuned-to-human-perception

23. *Pramod R.T., Katti H., Arun S.P.* Human peripheral blur is optimal for object recognition // Vision Research. 2022. Vol. 200. P. 108083.

24. Fastest Gaussian Blur (in Linear Time) // Algorithms and Stuff. 2014.
URL: https://blog.ivank.net/fastest-gaussian-blur.html

25. *Kugurakova V.V., Bedrin O.A.* Functional test automation system for unity applications // Vestn. Komp'yut. Inform. Tekhnol. 2020. Vol. 17, No. 12. P. 47–52. https://doi.org/10.14489/vkit.2020.12.pp.047-052 (In Russian).

СВЕДЕНИЯ ОБ АВТОРАХ

ПРИМАЧЕНКО Артём Михайлович – магистрант Института информационных технологий и интеллектуальных систем Казанского федерального университета. Направление исследований: визуальное восприятие игровых агентов в видеоиграх.

Artyom Mikhailovich PRIMACHENKO – Master's student at the Institute of Information Technologies and Intelligent Systems of Kazan Federal University. Research area: visual perception of game agents in video games.

email: primachenko.artem@mail.ru

ORCID: 0009-0008-6396-2035

ХАФИЗОВ Мурад Рустэмович – старший преподаватель Института ИТИС КФУ. Сфера научных интересов: разработка видеоигр, виртуальная реальность, синтетические данные.

Murad Rustemovich KHAFIZOV – senior lecturer at the Institute of Information Technologies and Intelligent Systems, Kazan Federal University. Research interests: game development, virtual reality, synth data.

e-mail: murkorp@gmail.com

ORCID: 0000-0001-7275-9102

Материал поступил в редакцию 2 февраля 2025 года