Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 22.Б07-мм

Реализация поведения неигровых персонажей в игре Edge of Madness

Плоскарев Вадим Владимирович

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
старший преподаватель кафедры системного программирования, к.т.н., Ю. В. Литвинов

Санкт-Петербург
2024

Оглавление

Введение

В современном мире видеоигры становятся неотъемлемой частью нашей культуры. Жанры компьютерных игр весьма разнообразны, но наиболее популярными в настоящее время являются: шутеры (от первого (FPS, *First-person shooter*) или третьего (TPS, *Third-person shooter*) лица), ролевые игры (RPG, *role-playing game*), стратегии в режиме реального времени, хорроры. Данный факт подтверждается рейтингом самых продаваемых игр [9] в Steam [7] — онлайн-платформе, предназначенной для распространения компьютерных игр. На момент написания 7 из 10 игр, находящихся на вершине рейтинга по продажам, являются представителями жанров, перечисленных выше. Одну из ключевых ролей в успешности игр в данных категориях (а также во множестве других) играет хороший искусственный интеллект (далее ИИ). Стоит отметить, что, в отличие от более широкой трактовки, принятой в информатике, на протяжении всей работы под ИИ подразумевается система управления действиями игрока-компьютера. ИИ улучшает множество аспектов игрового процесса.

Хороший ИИ усложняет прохождение игры для пользователя, делая его более интересным и уникальным. Создается ощущение противостояния с реальным человеком, который может адаптироваться к действиям противника, менять свою тактику, заставлять игрока изменять стратегию своего прохождения. Именно игры с умным ИИ зачастую запоминаются геймерам.

Интеграция готовой системы ИИ в игру является довольно трудоемким процессом, во время которого можно столкнуться с проблемами совместимости системы и видеоигры. Создание собственной системы ИИ предоставляет возможность легкой настройки и адаптации под конкретный геймплей. Поэтому было решено создать систему базового ИИ и интегрировать ее в игру Edge Of Madness.

Edge of Madness — прототип хоррор-детектив игры от первого лица, действия которой разворачиваются в городе N в начале прошлого века. Главный герой, детектив, получает задание: раскрыть исчезнове-

ние человека. Начиная с расследования, игра постепенно интегрирует элементы шутера и стелса. С некоторого момента игрок будет иметь возможность взаимодействовать с неигровыми персонажами, благодаря чему в игре создаются уникальные возможности развития сюжета в зависимости от действий игрока. Стоит отметить, что игровые механики Edge of Madness предполагают довольно базовое поведение.

Главными целями бота является патрулирование территории и убийство врага. Соответственно, когда в пределах видимости нет пользовательского персонажа, приоритетом является патруль, при обнаружении главного героя ИИ делает выбор: перейти или не перейти в состояние атаки. Основными возможностями данной системы ИИ является: патрулирование местности по заданным точкам (waypoints) и по произвольному маршруту в пределах определенного радиуса; обнаружение игрока с помощью различных сенсоров (звук, зрение); атака пользователя, если это приоритетно; сканирование местности и запоминание мест частого пребывания главного героя, с целью корректировки своего маршрута.

1 Постановка задачи

1.1 Цель

Целью данной учебной практики является изучение движка Unity путем создания системы ИИ, обладающего перечисленными выше возможностями, и последующей интеграции данного ИИ в игру Edge Of Madness.

1.2 Задачи

Для решения данной цели были поставлены следующие задачи:

1. Выполнить обзор существующих решений и используемых технологий.
2. Разработать логику поведений ИИ.
3. Реализовать описываемые выше возможности ИИ.
4. Провести апробацию.

2 Обзор

2.1 Существующие подходы

Существуют множество подходов к реализации ИИ в видеоиграх, одними из основных являются следующие [1]:

1. Конечные автоматы (State Machine):

   Конечный автомат — это совокупность четырех множеств: набора состояний объекта, набора событий, таблицы переходов между состояними, стартового состояния. Пусть у нашего ИИ есть два состояния: патрулирование и атака. Когда ИИ замечает пользовательского персонажа, происходит событие «увидеть врага», и осуществляется переход между состояними «патрулирование» и «атака».

   Разработчики используют данный подход в реализации ИИ по следующим причинам: четкость различения состояний, которая позволяет управлять сложными системами поведений; предоставляется удобный способ управления переходами между состояними.

2. Дерево поведений (Behavior Tree):

   Если рассматривать более сложную систему ИИ, содержащую большее количество возможностей, конечные автоматы будут являться «жестким» подходом, менее восприимчивым к изменениям системы. В данном случае используется более гибкий подход — дерево поведений. В процессе разработки выделяется дерево, в котором узлы — это действия либо условия переходов. Переходы осуществляются в зависимости от результата выполнения действия или условия. Если действие выполнено успешно, система переходит к соответствующему дочернему узлу; в случае неудачи она может выбирать альтернативные ветви или условия.

   Одним из основных преимуществ деревьев поведений является возможность легко вносить изменения в поведение персонажа.

Можно дополнять новые узлы или модифицировать существующие, при этом не затрагивая всю систему.

3. Goal-Oriented Action Planning (GOAP):

   Данный метод проектировки ИИ отличается тем, что система в режиме реального времени выбирает последовательность поведений для достижения своей цели. Данный метод применяется в большинстве случаев для игр с похожими типами врагов. Тогда использование GOAP позволяет добавить вариативности в поведение ИИ, ведь у каждого персонажа будет своя цель и свой план для ее достижения.

   Например, пусть у ИИ среди его пула целей (target pool) есть убийство врага. Тогда, если цель будет находиться в зоне видимости, шанс атаки будет возрастать. Но система должна также учитывать уровень здоровья (если у него мало здоровья, то очевидно он должен существовать в более оборонительном режиме), наличие оружия, количество патронов и другие возможные факторы. Из совокупности всех условий формируется шанс на атаку врага. Если вероятность атаки слишком мала, то надо сменить цель на более приоритетную: укрыться в безопасном месте и восстановиться, или любая другая возможность из пула целей.

4. Правила и условия (Rule-based system):

   Следующий подход представляет собой набор правил, которые определяют переходы между состояниями персонажа. Данные правила в большинстве случаев имеют вид «if-then». Например, пусть ИИ находится в состоянии патрулирования. *Если* пользовательский персонаж подойдет к ИИ на слишком короткое расстояние (заранее прописанное в коде), *то* система переходит в состояние атаки. В отличии от конечных автоматов, rule-based system предоставляет возможность создания более гибкой и легко изменяемой системы. В случае проектирования такой системы, разработчикам достаточно просто определить правила, состоящие

из условий. Например, вместо того чтобы явно создавать событие «увидеть персонажа» и связывать его с определенным состоянием, можно просто написать правило «если обнаружен персонаж, то перейти в такое состояние». Этот подход делает процесс проектирования и поддержки системы более простым, поскольку изменения в логике принятия решений могут быть внесены путем добавления, изменения или удаления правил. Не требуется переписывание всей структуры, как в случае с конечными автоматами, где изменение состояний и переходов может быть более трудоемким.

2.2 Существующие решения

Для сравнения отбирались игры, про реализацию ИИ в которых имеются положительные отзывы. Целью обзора является анализ систем ИИ, используемых в различных играх.

F.E.A.R. [6] — одна из первых видеоигр, разработчики которой сделали акцент на создании умного ИИ. Они применили в своей игре метод GOAP. ИИ использовал систему построения путей не только для перемещения, но и для создания плана действий. Система могла выбрать одну из 70 задач (у каждого типа NPC свои задачи и цели), а затем использовать какую-либо комбинацию действий для ее выполнения. Одним из показательных моментов является, например, поведение ИИ в битве с игроком. Во время боя ИИ сам выбирает тактику поведения: с какой стороны подойти к главному персонажу, в какой момент бросить гранату и т.д. Если игрок будет вести себя слишком агрессивно, то система перейдет в оборонительный режим, если же у игрока мало здоровья, то ИИ попытается перейти в атаку.

Отсюда была взята идея добавления цели: патрулирование и убийство противника, а также возможность перехода в состояние атаки, основываясь на данных, полученных в режиме реального времени.

Doom [5] — основная логика ИИ в данной игре реализована с помощью конечных автоматов. То есть это набор определенных состояний,

в которых может находиться ИИ, и которые определяет поведение ИИ, также это список правил, условий, позволяющих изменять состояния персонажа.

Изначально все NPC находятся в состоянии покоя. Получив информацию, что игрок поблизости, они переходят в состояние See, в котором уже есть возможность перемещения. При обнаружении главного героя, ИИ попытается атаковать его, переходя в состояния рукопашного боя (melee) или атаки с расстояния (range). После завершения атаки, он снова перейдет в состояние see. Помимо этого, ИИ имеет возможность получить урон и умереть. Использование простых принципов конечных автоматов и наборов поведений позволяет создать эффективную систему ИИ, которая запомнилась пользователям по агрессивному поведению персонажей, по возможности обнаружения игрока с помощью разных методов (реакция на звуки, выстрелы, движения).

Из этой игры была взята основная логика системы ИИ: использование правил и условий для создания базовой системы ИИ.

Alien: Isolation [4] — данная игра была разработана на основе культового фильма «Чужой». Основной задачей было передать атмосферу фильмов, безвыходность положения и почти полную неуязвимость главного врага (ксеноморфа). Для избавления от преднаписанности сюжета было решено использовать систему дерева поведений. У ИИ есть более 100 действий в системе дерева поведений, в верхнем уровне находятся около 30 основных поведений, выбирая которые система переключается на выполнение более мелких действий в поддереве. С течением времени у ИИ открываются новые поведения, это создает ощущение, что система обучается по ходу игры и становится умнее. Очень хорошо здесь развита система обнаружения игрока: ИИ наделен сенсорами, на показаниях которых основывается система поиска пути до игрока. ИИ может «чувствовать» шаги, слышать звуки, обнаруживать игрока, если тот идет близко к нему сзади.

Использование способов обучения ИИ в робототехнике

Одним из способов обучения робота является наблюдение за окружающей средой и реакция на происходящие в ней изменения с постепен-

ным накоплением дополнительной информации, помогающей ИИ принимать те или иные решения.

Исходя из данной концепции, было решено добавить возможность сохранения позиций, где был замечен пользовательский персонаж, с целью корректировки области патрулирования.

В результате обзора были рассмотрены несколько игр, содержащих исключительные системы ИИ, а также один из способов обучения ИИ в робототехнике. Было решено реализовать систему ИИ, основанную на принципе rule-based, так как данный подход к реализации является наиболее простым и больше подходит для начального этапа изучения данной темы.

2.3 Использованные технологии

Для разработки системы ИИ был выбран движок Unity Engine (далее Unity) [8], так как он является простым в изучении и хорошо подходит для начального знакомства с разработкой игр.

Для описания логики поведения объектов в игре в Unity используются скрипты — код, написанный на языке программирования C#. Для взаимодействия с движком следует использовать пространство имен UnityEngine. Пространство имен UnityEngine.AI содержит классы и методы, необходимые для работы с ИИ и навигацией агента по сцене.

- Класс NavMeshAgent предоставляет возможность навигации объектов по игровой сцене:
  - SetDestination — метод, с помощью которого устанавливается точка назначения пути агента. Данный метод находит оптимальный маршрут по навигационной сетке к конечной цели объекта;
  - remainingDistance — свойство, необходимое для получения оставшегося расстояния до итоговой цели на сцене;
- Класс NavMesh представляет поверхность, по которой возможна навигация агента на сцене. Она используется для определения

потенциально достижимых мест, препятствий (NavMeshObstacle), которые агент должен обходить.

- Класс Animator предоставляет возможности управления анимациями объектов в игре. С помощью него осуществляется управление проигрыванием анимаций и переходами между ними.

- Ассеты — уже созданные графические, анимационные или иные компоненты, а также анимации, были скачаны с сайта Mixamo. Стоит отметить, что Mixamo [2] предоставляет бесплатные ресурсы с разрешением на использование в соответствии с их лицензией.

3 Реализация

В этом разделе будут описаны основные игровые механики, а также особенности реализации некоторых из них.

3.1 Основные игровые механики

- Для осуществления навигации по сцене на ИИ был добавлен компонент NavMeshAgent.

- Для осуществления движения ИИ на него был добавлен компонент Animator.

  В компоненте Animator были добавлены различные анимации:

  - Idle — анимация состояния покоя: *Idle*;
  - Walking — анимация ходьбы, используется в состоянии *Patrol* (см рис. 2);
  - Medium Run — анимация, используемая для состояния *Chase* и *Hiding*;
  - Shooting — анимация, нужная для состояния *Attack* (см рис. 3);
  - Death — анимация, смерти персонажа, необходима для состояния *Death*.

- Более подробно о том, как устроены переходы между состояниями (см рис. 1):

  - Изначально ИИ находится в состоянии покоя. ИИ начинает патрулирование через две секунды после старта. Также ИИ может перейти сразу в состояние *Chase*, если выполнятся определенные условия;
  - После 50 секунд патрулирования ИИ возвращается в состояние *Idle*. Если система при патрулировании обнаружила вра-

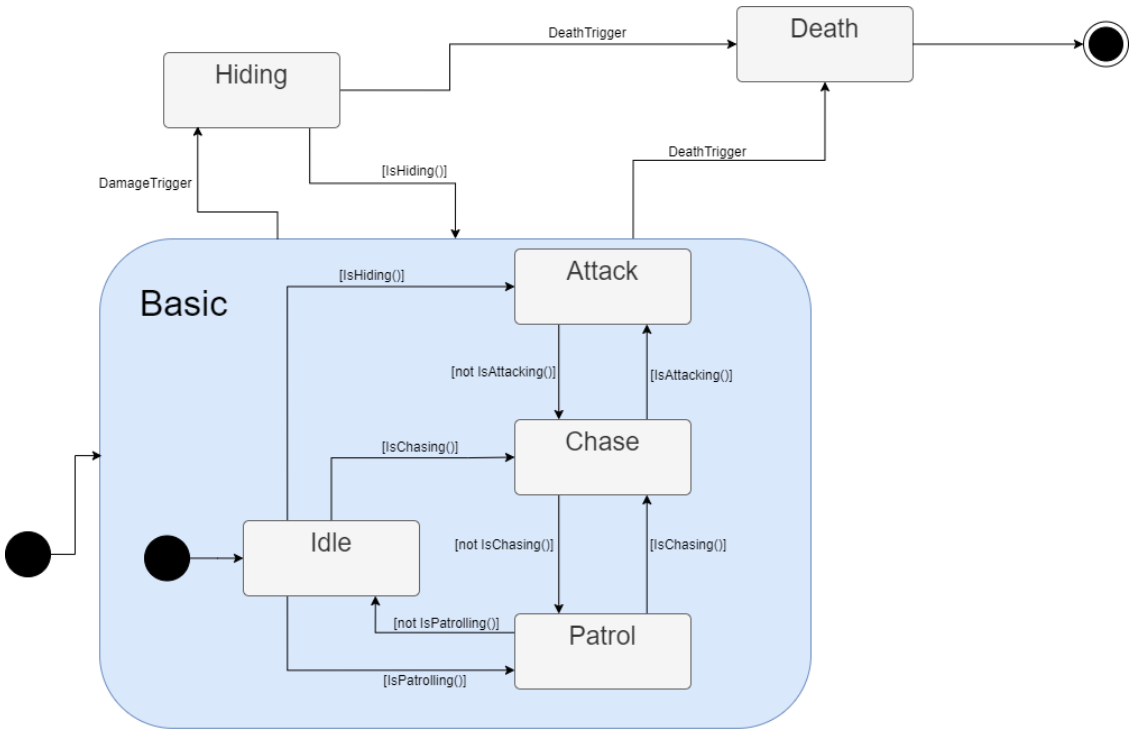

Рис. 1: Диаграмма состояний и их переходов

жеского персонажа (с помощью зрения или слуха), то она переходит в состояние *Chase*;

– Если расстояние между ИИ и пользователем становится менее 10, то система переходит в состояние атаки;

– Из любого состояния при срабатывании триггера *DeathTrigger* ИИ переходит в состояние смерти;

– Из любого состояния если срабатывает триггер *DamageTrigger* на получение урона (недостаточного для смерти), то ИИ переходит в состояние Hiding и будет стараться найти объект на сцене, за которым можно спрятаться и уже оттуда попытаться атаковать соперника.

Подробнее о реализации некоторых методов.

3.2 Патрулирование

Unity предоставляет встроенные инструменты для работы с навигацией, которые позволяют создавать оптимальные маршруты на сцене.

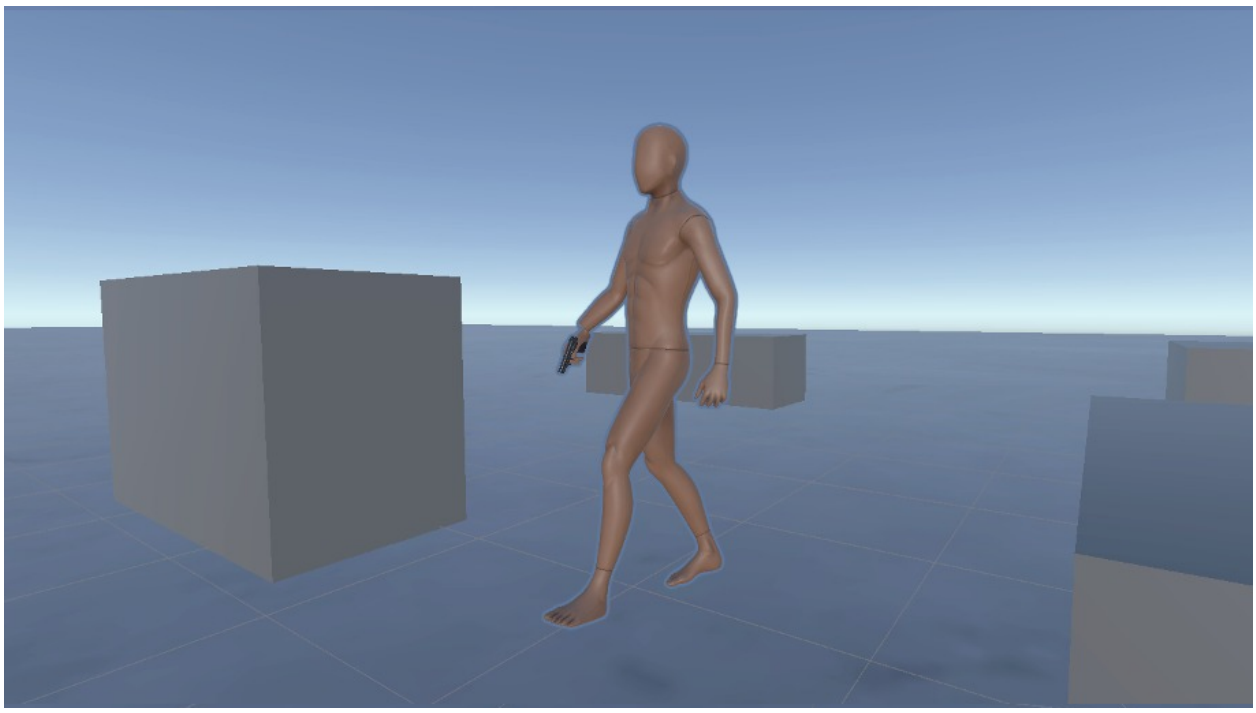

Рис. 2: ИИ в состоянии патрулирования (patrol)

3.2.1 Waypoints

Для реализации данного метода на сцену вручную были добавлены пустые объекты, которые и являются *путевыми точками*. То есть, маршрут ИИ будет проходить только через данные точки. С помощью NavMeshAgent.SetDestination() автоматически строится маршрут к заданной цели. При достижении путевой точки, ИИ выбирает новую цель, из списка, и прокладывает маршрут к ней. Преимуществом данного метода является простота реализации (т.к. вся навигация обеспечивается с помощью встроенных методов), полный контроль поведения ИИ.

3.2.2 RandomPoints

Для разнообразия поведения ИИ была добавлена возможность «свободного патрулирования». При таком подходе создается радиус патрулирования — радиус, в пределах которого будет сгенерирована случайная точка на NavMesh. Далее ИИ выбирает случайную точку и направляется к ней. При этом *центр сферы патрулирования* остается неизменным.

Встроенные инструменты Unity (методы класса NavMeshAgent) обеспечивают не только удобство в работе с навигацией, но и гарантируют, что ИИ автоматически адаптирует свой маршрут, учитывая окружающую среду и обеспечивая оптимальное движение даже при наличии препятствий.

3.3 Обнаружение пользовательского персонажа

ИИ оснащен двумя сенсорами: оптическими и звуковыми, обеспечивающими ему способность осуществлять обнаружение противника.

3.3.1 С помощью BoxCast

Данный метод предназначен для определения наличия игрока в определенном объеме пространства перед собой.

С помощью функции Physics.BoxCast проверяется, есть ли столкновения лучей с объектами в объеме перед ботом. Если столкновение произошло и объект столкновения — это игрок, то начинается преследование.

3.3.2 SoundDetection

Данный метод отвечает за обнаружение пользовательского персонажа по звуку. Функция *SoundDetection* проверяет, находится ли игрок в пределах для распознования по бегу и актирован ли у пользовательского игрока режим бега. Если все условия выполняются, то ИИ начинает преследование.

3.4 Укрытие за объектом на сцене

При получении урона ИИ попытается найти укрытие от игрока с подходящей высотой (т.е. препятствие не должно быть слишком маленьким).

Сначала с помощью Physics.OverlapSphereNonAlloc выбираются все объекты в определенном радиусе от ИИ. Затем исключаются непод-

ходящие: не рассматриваются больше те, которые находятся слишком близко к игроку или слишком низкие. Потом происходит поиск для каждого объекта ближайшей точки на NavMesh, так чтобы объект оказался между пользовательским персонажем и ИИ. Эта точка устанавливается как цель для ИИ. Если ИИ не нашел подходящего укрытия, то устанавливается флаг «IsAttacking», предполагая, что система продолжит атаковать игрока.

3.5 Реализация выстрелов

При выполнении выстрела в игре, в первую очередь активируется визуальный эффект вспышки из ствола оружия — muzzleFlash. Этот эффект служит визуализацией момента выстрела.

Затем, с использованием Physics.Raycast, создается луч, направленный вперед от позиции ИИ. Далее сохраняется информация о пересечении луча с каким-либо объектом.

Если у объекта есть компонент Health, вызывается метод DetectionDamage(), который определяет величину урона для пользовательского персонажа, в зависимости от места попадания.

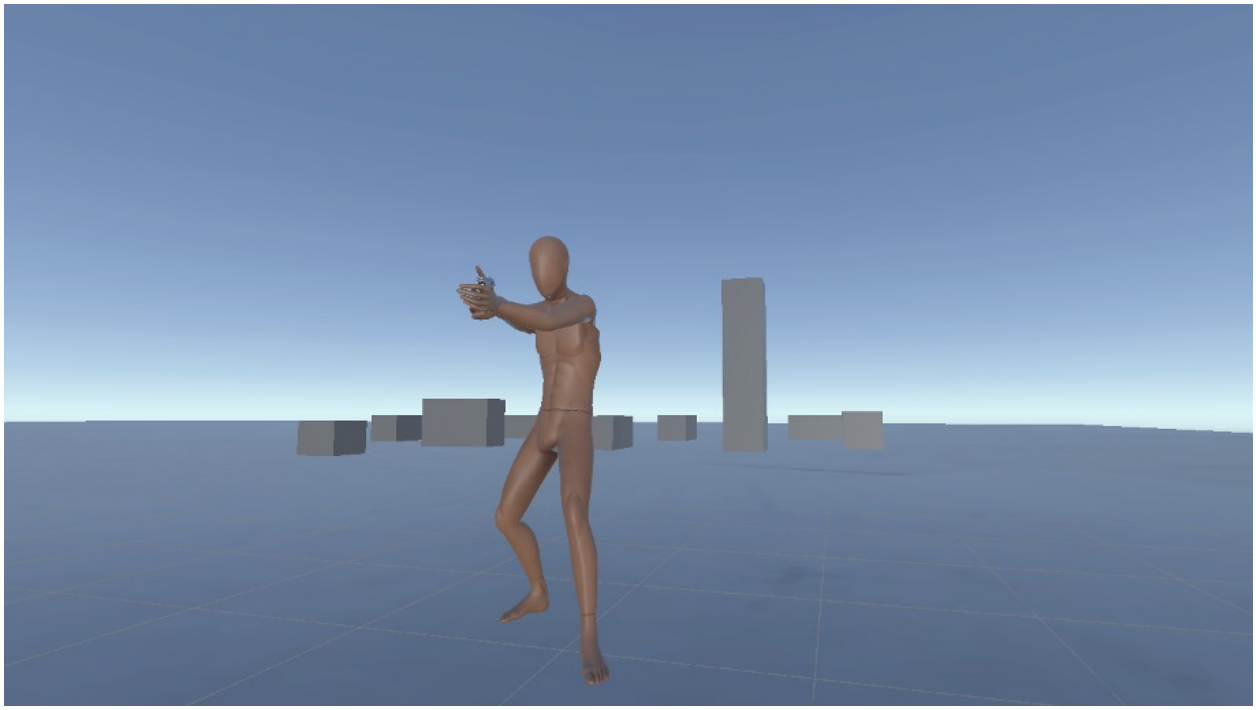

Рис. 3: ИИ в состоянии атаки (attack)

3.6 Запоминание мест

Реализованные методы SavePlayerPosition() и LoadPlayerPosition() отвечают за сохранение и загрузку позиций игрока, в которых ранее ИИ обнаруживал пользовательского персонажа.

В самом начале десериализуются данные о предыдущих местах обнаружения. Затем, суммируя координаты X и Z из загруженных позиций, вычисляется новый центр патрулирования. Этот процесс позволяет учесть предыдущие местоположения игрока и адаптировать маршрут ИИ.

3.7 Архитектура

Диаграмма классов представлена на рис. 4.

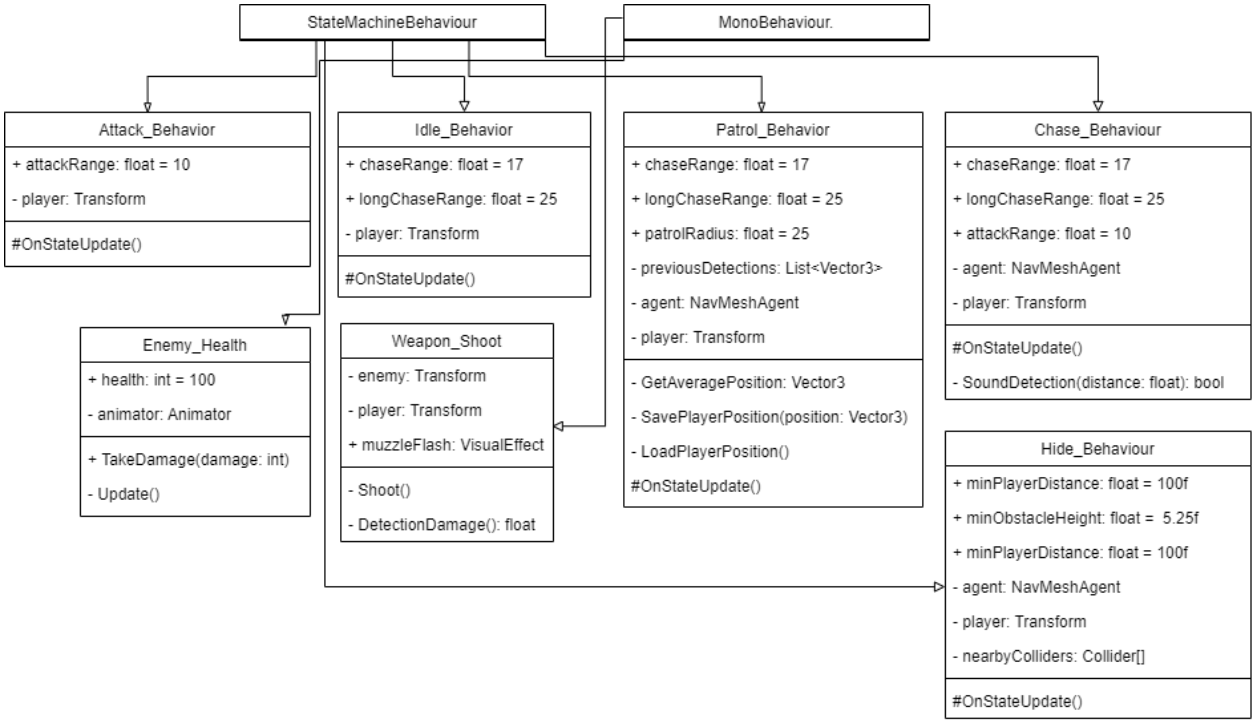

Рис. 4: Диаграмма классов

- Все классы, отвечающие за поведение ИИ в различных состояниях, наследуются от StateMachineBehaviour — класса, предназначенного для создания пользовательских поведений для различных состояний в Unity. Например, метод OnStateUpdate() в классе

Idle_Behavior переопределяет соответствующий метод из базового класса, что позволяет задать новое поведение во время обновления состояния.

- Все нестатические классы в Unity наследуются от класса MonoBehaviour. Например, Weapon_Shoot, отвечающий за стрельбу (метод Shoot()) или Enemy_Health, отвечающий за нанесение урона уже самому ИИ.

4 Апробация

Используя методику System Usability Scale [3], была проведена апробация поведения неигровых персонажей.

Оценка SUS находится в диапазоне от 0 до 100 баллов, причём показатель в 68 баллов является средним для данной методики. В опросе приняли участие пять человек, им было предложено ответить на вопросы, касающиеся реализации системы управления вражеским персонажем.

По итогам опроса средний балл среди участников составил 71. По результатам можно отметить, что большинство респондентов считают, что данная система ИИ подходит для выполнения базовых задач. Однако система поведений нуждается в усложнении в будущем.

Кроме оценок, участники также дали важные комментарии, касающиеся поведения ИИ.

- При получении урона ИИ сразу пытается спрятаться, что открывает игроку возможность легко убить его.
- При обнаружении по звуку анимации слегка быстро «сменяются», что создает неприятный эффект.

Заключение

В итоге были достигнуты следующие результаты.

- Был выполнен обзор существующих решений и используемых технологий.
- Была разработана логика поведений ИИ.
- Были реализованы планируемые возможности ИИ.
- Была проведена апробация.

Исходный код (ветка) доступен по ссылке: https://github.com/LeonidLodygin/EdgeOfMadness/tree/dev (дата обращения: 5 января 2024 г.).

Исходный код (пул-реквест) доступен по ссылке: https://github.com/LeonidLodygin/EdgeOfMadness/pull/6 (дата обращения: 5 января 2024 г.).

Список литературы

[1] Decision-making AI in digital games. — URL: https://www.diva-portal.org/smash/get/diva2:1673140/FULLTEXT01.pdf (дата обращения: 4 января 2024 г.).

[2] Mixamo. — URL: https://www.mixamo.com/#/ (дата обращения: 4 января 2024 г.).

[3] System usability scale. — URL: https://en.wikipedia.org/wiki/System_usability_scale (дата обращения: 4 января 2024 г.).

[4] Wikipedia. Alien: Isolation. — URL: https://en.wikipedia.org/wiki/Alien:_Isolation (дата обращения: 4 января 2024 г.).

[5] Wikipedia. Doom (franchise). — URL: https://en.wikipedia.org/wiki/Doom_(franchise) (дата обращения: 4 января 2024 г.).

[6] Wikipedia. F.E.A.R. — URL: https://en.wikipedia.org/wiki/F.E.A.R._(video_game) (дата обращения: 4 января 2024 г.).

[7] Wikipedia. Steam. — URL: https://ru.wikipedia.org/wiki/Steam (дата обращения: 4 января 2024 г.).

[8] Wikipedia. Unity (game engine). — URL: https://en.wikipedia.org/wiki/Unity_(game_engine) (дата обращения: 4 января 2024 г.).

[9] Лидеры продаж Steam. — URL: https://store.steampowered.com/charts/topselling/RU (дата обращения: 4 января 2024 г.).