УДК 004

Создание неигрового персонажа на игровом движке Godot (часть 2)

Черкашин Александр Михайлович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье описан процесс написания программы поведения неигрового персонажа (NPC) на игровом движке Godot. В работе использовался язык программирование GDScript и объекты в конструкторе игр на движке Godot для создания неигрового персонажа (NPC), а также определение системы кланов для поведения неигрового персонажа. В результате работы создан неигровой персонаж, написанный на языке GDScript для выполнения поведения неигрового персонажа и определена система кланов на игровом движке Godot.
Ключевые слова: Godot, GDScript, NPC.

Creation of a non-player character on the Godot game engine (part 2)

Cherkashin Alexander Mihailovich
Sholom-Aleichem Priamursky State University
student

Abstract
This article describes the process of writing a program behavior of a non-player character (NPC) on the Godot game engine. The work used the GDScript programming language and objects in the designer of the Godot game engine to create a non-player character (NPC) as well as the definition of the clans system, and the behavior of a non-player character was described. As a result of the work, a non-player character was created written in the GDScript language to perform the behavior of a non-player character and a system of clans on the Godot game engine was determined.
Keywords: Godot, GDScript, NPC.

1 Введение

1.1. Актуальность исследования

Актуальность исследование заключается в том, что сценарий поведения NPC позволяет создавать разнообразные и уникальные ситуации в игре, которые повышают интерес и вовлеченность игрока. Например, NPC может быть дружелюбным, враждебным, нейтральным, помогать или мешать игроку, реагировать на его действия и выборы, обладать своими целями и мотивами и т. д.

Сценарий поведения NPC также позволяет создавать NPC, которые ведут себя естественно и логично в соответствии с их характером, ролью, ситуацией и окружением. Например, NPC может бегать, прятаться, атаковать, защищаться, разговаривать, торговать, выполнять задания и многое другое.

Сценарий поведения NPC также предоставляет разработчикам технические возможности для создания различных визуальных эффектов, таких как динамические и интерактивные анимации, эффекты освещения, деформации объектов и многое другое, что делает его важным инструментом для разработки VFX.

Таким образом, сценарий поведения NPC на игровом движке Godot остается актуальным и важным для создания качественных игр. Если у вас есть дополнительные вопросы или вам требуется более подробная информация, не стесняйтесь задавать.

1.2. Цель исследования

Целью работы создания неигрового персонажа на игровом движок Godot.

1.3. Обзор исследований

Исследование авторов статьи Г. Гомес и др. представляет собой работу, посвященную разработке инструмента для проведения экспериментов с обучением с подкреплением в играх. Авторы исследования разрабатывают инструмент Ai4u, который позволяет создавать и запускать различные сценарии обучения с подкреплением в играх, используя платформу Unity. Инструмент Ai4u предоставляет удобный интерфейс для взаимодействия между агентами и средой, а также для сбора и анализа данных. Авторы обсуждают преимущества и возможности применения инструмента Ai4u в игровой индустрии. Они также рассматривают потенциальные вызовы и ограничения, связанные с его внедрением. В статье представлены результаты исследования, включая примеры использования инструмента Ai4u и анализ его эффективности. Авторы также обсуждают возможности дальнейшего развития и улучшения инструмента Ai4u. Таким образом исследование авторов статьи Г. Гомес и др. представляет интерес для разработчиков игр и специалистов в области обучения с подкреплением. Оно предлагает новый инструмент для проведения экспериментов с обучением с подкреплением в играх, что может привести к улучшению качества и интеллектуальности игровых проектов [1].

Исследование автора статьи И. Зеиметз представляет собой работу, посвященную изучению парапсихологических отношений между игроками и вымышленными персонажами в игре Trouble Brewing. Автор исследования определяет парапсихологические отношения как односторонние психологические связи между индивидами и вымышленными персонажами, которые создаются в процессе взаимодействия с медиа-продуктами. Автор анализирует факторы, которые влияют на формирование и развитие таких отношений, такие как эмпатия, идентификация, вовлеченность и привязанность. Так же автор выбирает игру Trouble Brewing как объект исследования, так как она представляет собой интерактивную историю, в

которой игроки могут взаимодействовать с различными вымышленными персонажами, которые имеют свои характеры, мотивации и цели. Автор описывает основные элементы игры, такие как сюжет, геймплей, механики и дизайн. Автор исследования проводит качественное исследование, используя методы наблюдения, интервью и анализа контента. Автор собирает данные от 12 участников, которые играли в Trouble Brewing, и анализирует их с помощью тематического анализа. Автор исследования представляет результаты исследования, включая основные темы, которые выявлены в данных, такие как типы парапсихологических отношений, факторы, которые влияют на их формирование и развитие, и эффекты, которые они оказывают на игровой опыт. Автор также обсуждает возможности дальнейшего развития и улучшения исследования. Таким образом исследование автора статьи И. Зеиметз представляет интерес для разработчиков игр и специалистов в области психологии и коммуникации. Оно предлагает новый взгляд на парапсихологические отношения между игроками и вымышленными персонажами в играх, что может привести к улучшению качества и эмоциональности игровых проектов [2].

Исследование авторов статьи М. А. Норюшан и др. представляет собой работу, посвященную разработке неигровой персонажа (Non-Character Player, NCP) с использованием алгоритма самообучения для искусственно-интеллектуальных игр. Авторы исследования определяют Неигровой персонажа как сущность в игре, которая не является персонажем, но имеет влияние на игровой процесс. Например, Неигровой персонажем может быть животное, растение, предмет или событие. Авторы анализируют различные типы и функции Неигровой персонажей в играх. Авторы исследования применяют алгоритм самообучения для разработки Неигровой персонажа, который может адаптироваться к поведению игрока и окружающей среде. Авторы используют метод обучения с подкреплением, который позволяет Неигровой персонажу учиться на основе своих действий и их последствий. Авторы исследования выбирают искусственно-интеллектуальные игры как объект исследования, так как они представляют собой игры, в которых искусственный интеллект играет важную роль в создании интересного и сложного игрового опыта. Авторы описывают основные характеристики и примеры таких игр. Авторы исследования представляют результаты исследования, включая примеры разработанных Неигровой персонажей и анализ их эффективности. Авторы также обсуждают возможности дальнейшего развития и улучшения разработанного алгоритма самообучения. Таки образом исследование авторов статьи М. А. Норюшан и др. представляет интерес для разработчиков игр и специалистов в области искусственного интеллекта. Оно предлагает новый подход к разработке Неигровой персонажей с использованием алгоритма самообучения для искусственно-интеллектуальных игр, что может привести к улучшению интерактивности и сложности игровых проектов [3].

2. Рабочий процесс

Чтобы определить неигрового персонажа Ally и Enemy, в экземпляр класса MainScene (главный сцена), содержится метод init_clans (Листинг 2.1) который мы задали и определили кланы (табл. 1).

Таблица 1. Кланы

Исходный группа персонажа	Отношение	Обратное отношение	Целевой группа персонажа
City	Enemy	Enemy	Balloon
Player	Enemy	Enemy	Balloon
Player	Ally	Ally	City
EvilCube	Enemy	Enemy	Balloon
EvilCube	Enemy	Enemy	Player
EvilCube	Neutral	Neutral	City

Листинг 2.1. Инициализация кланов.

```
1  func init_clans():
2          if not self.clans.is_empty():
3                  return
4          self.clans["Player"] = Clan.new()
5          self.clans["Player"].setAlly("Player")
6          self.clans["Player"].setAlly("City")
7          self.clans["Player"].setEnemy("Balloon")
8          self.clans["Player"].setEnemy("EvilCube")
9
10         self.clans["City"] = Clan.new()
11         self.clans["City"].setAlly("City")
12         self.clans["City"].setAlly("Player")
13         self.clans["City"].setEnemy("Balloon")
14         self.clans["City"].setNeutral("EvilCube")
15
16         self.clans["Balloon"] = Clan.new()
17         self.clans["Balloon"].setAlly("Balloon")
18         self.clans["Balloon"].setEnemy("Player")
19         self.clans["Balloon"].setEnemy("City")
20         self.clans["Balloon"].setEnemy("EvilCube")
21
22         self.clans["EvilCube"] = Clan.new()
23         self.clans["EvilCube"].setAlly("EvilCube")
24         self.clans["EvilCube"].setEnemy("Player")
25         self.clans["EvilCube"].setEnemy("Balloon")
26         self.clans["EvilCube"].setNeutral("City")
```

Персонаж в игре определяется базовой класс Mob_base.

В Mob_base содержится переменная id_clans который идентифицирует персонажа принадлежность кланам. При инициализации в MainScene содержится метод init_clan который устанавливает соответствующий клан каждого персонажа. По идее если, например, персонаж Balloon1 атаковал на Balloon2 (изначально Ally) то могут стать как Enemy, но в данном случае недостатки могут быть что установка клан Balloon как Enemy может распространятся как на всех персонажей, относящийся к Balloon а не отдельный конкретный персонажа.

А также в Mob_base содержится health для определения уровень здоровье персонажа. При инициализации Mob_base, прикрепляет сигнал sig_damage который будет вызывать метод action_damage если будет персонаж атакован, то теряет очко здоровье, если сигнал sig_damage не подключен, то персонаж неуязвим.

В Balloon и игрока очко здоровье указано 100.

Если неигровой персонаж Balloon атакован (получен сигнал sig_damage, например, Player атаковал Balloon (рис 3) и Balloon начинает в ответ атаковать Player) то вызывается метод reaction_damage (прикреплен к сигналу sig_damage). Который reaction_damage задает attacking (переменная указывающий источник атаки персонажа), и переходит в состояние state_solve в st_attacked. Если очко здоровье достигнет до 0, то объект удаляется (функция queue_free).

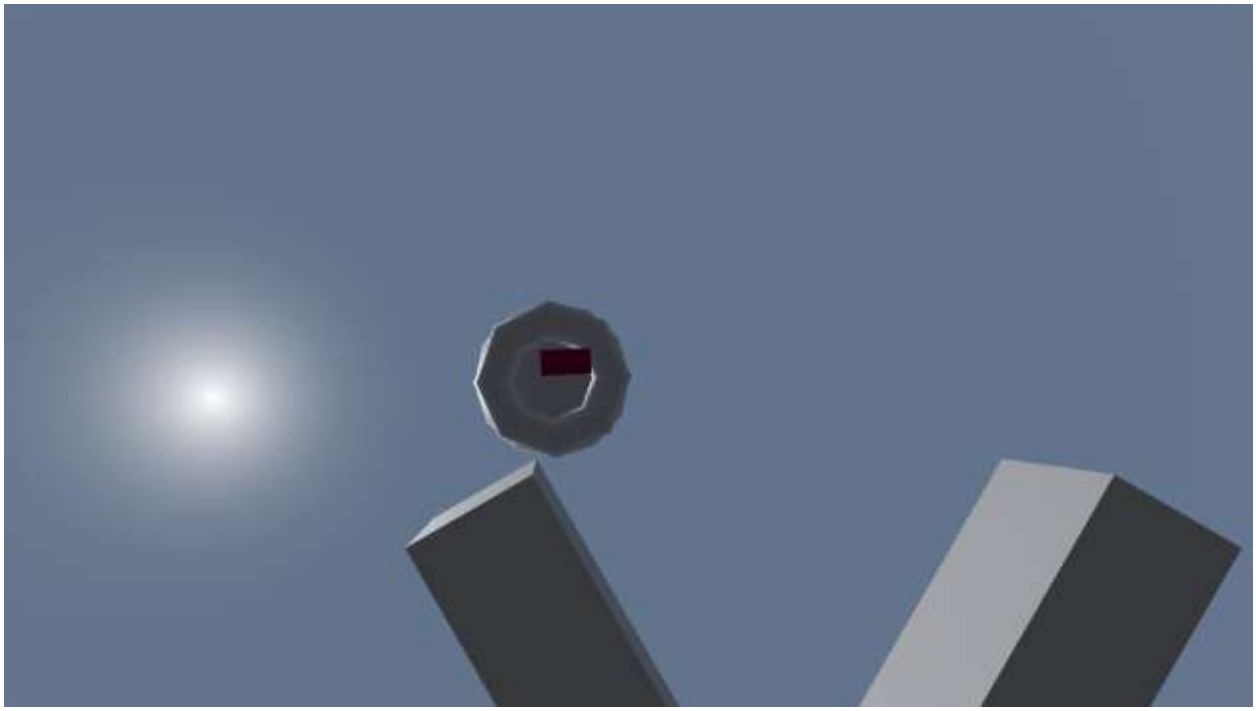

Рисунок 1. Balloon атакует Player

Состояние st_attacked с начало пытается получить список персонажи (get_list_target) а затем находит ближайший Ally который ближе к атакующий противника (find_ally_to_defence) и отправляет команду атаковать, а сам Balloon может продолжить в состояние st_select_target. В остальном случае состояние st_attacked имеет схожесть st_select_target.

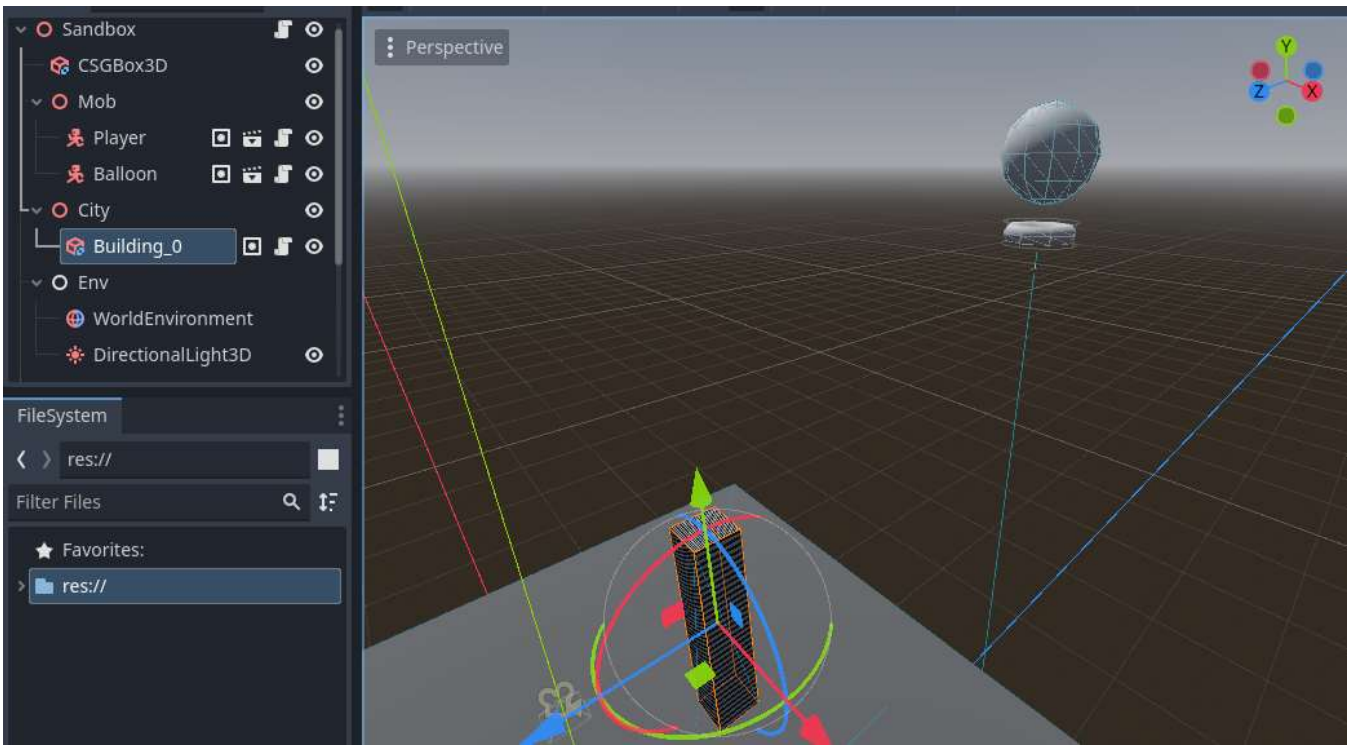

Рисунок 2. Balloon и Building_0

Объект Building_0 (рис 2) по типу CSGBox3D. Имеет базовый класс Building_base, мы задали очко здоровье 1000.

В отличие от Building_base и Mob_base имеет разные базовые классы, Mob_base имеет базовый класс CharacterBody3D (класс которые предназначены для контроля пользователя. На них вообще не влияет физика; Для других типов тел, таких как персонаж или жесткое тело, они такие же, как статическое тело. Однако у них есть два основных применения [4]) а Building_base имеет базовый класс Node3D.

Building_base и Mob_base не предусмотрено множественные наследование на языке GDScript [5].

Чтобы определить принадлежность классам Building_base и Mob_base мы предусмотрели функцию is_unity (листинг 2.2).

Листинг 2.2. Проверка принадлежность Building_base и Mob_base.

```
1  func is_unity(obj:Node):
2          if is_instance_of(obj, Mob_base) or is_instance_of(obj, Building_base):
3                  return true
4          return false
```

Класс Bullet_base определяет объектов который при попадании персонажи теряет очко здоровье.

В Bullet_base содержится переменная damage который вычитает очко здоровье, и subject которое позволяет идентифицировать атакующий, чтобы пропускать себя (не «повреждать» самой себя).

Метод is_vulnerable сравнивает тело и subject (атакующий) а затем проверяет принадлежность класса Building_base и Mob_base. Возвращает логический тип.

Метод is_attackable тоже самое как is_vulnerable но только отсеивает столкновение тело самой себя и атакующий.

Листинг 2.3. Методы is_attackable и is_vulnerable.

```
1  func is_attackable(body:Node3D) -> bool:
2          if self.subject != null:
3                  if self.subject == body:
4                          return false
5          if self == body:
6                  return false
7          return true
8  func is_vulnerable(body:Node3D) -> bool:
9          if self.subject != null:
10                 if self.subject == body:
11                         return false
12         if Root.is_unity(body):
13                 return true
14         return false
```

Метод attack отправляет сигнал sig_damage который был столкновение с персонажем.

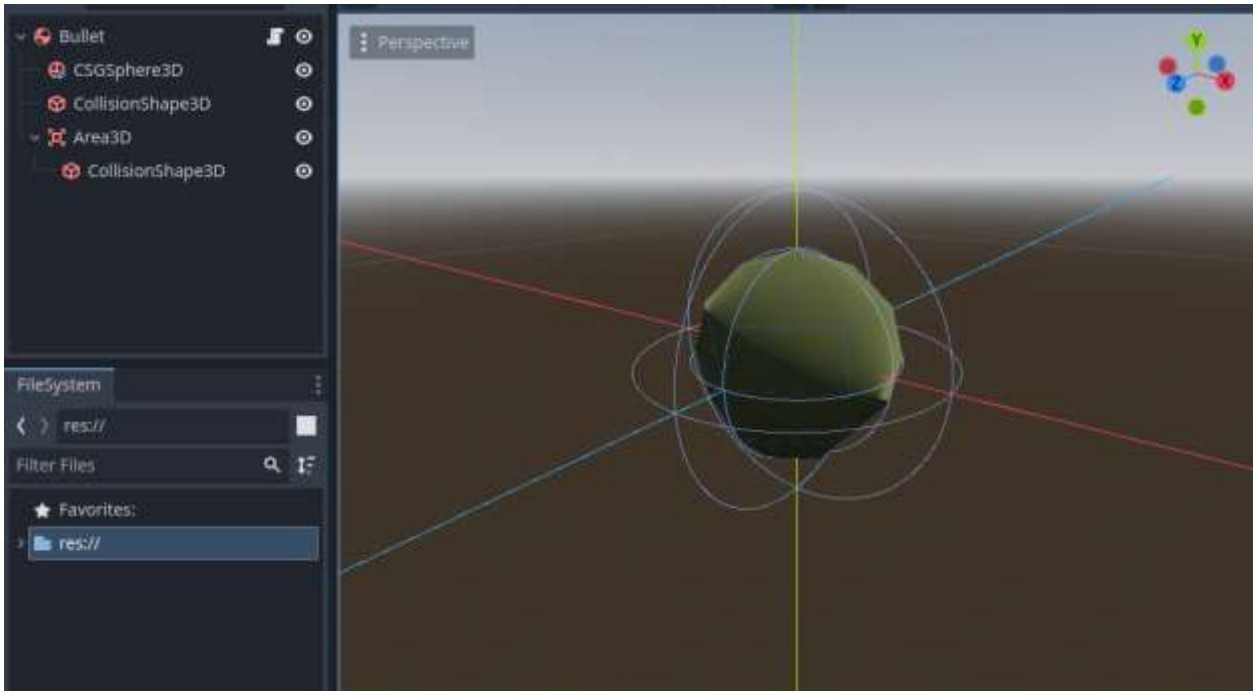

Рисунок 3. Объект Bullet

CSGSphere3D — Визуальная форма, CollisionShape3D — Область столкновение, Bullet — тип RigidBody3D, слой столкновение 4, а маска — 1, 2, 4.

Area3D — детекторы столкновение который будут обнаруживать столкновение персонажа и отправлять сигнал sig_damage. CollisionShape3D — область столкновение (рис 3).

3 Выводы

В данное статьи создан неигровой персонаж и описано отношение к другим персонажам. В результате работы был создан неигровой персонаж, кланы (отношение к другим персонажам), а так же было определен базовый

класс, который позволяет грамотно спроектировать структура классов для персонажа и определить набор поведения неигрового персонажа.

Библиографический список

1. Gomes G. et al. Ai4u: A tool for game reinforcement learning experiments //2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames). IEEE, 2020. С. 19-28.
2. Zeimetz I. Trouble Brewing: An exploration in parasocial relationships with fictional characters. 2021.
3. Noryushan M. A. et al. Development of Non-Character Player Using Self-Learning Algorithm for Artificial Intelligent Games //International Journal of Engineering & Technology. 2018. Т. 7. №. 2.28. С. 204-205.
4. CharacterBody3D — Godot Engine (4.2) documentation in English URL: https://docs.godotengine.org/en/4.2/classes/class_characterbody3d.html (дата обращения: 2024-01-30).
5. How to Extend to multiple scripts - #2 by system - Archive - Godot Forum // Godot Forum URL: https://forum.godotengine.org/t/how-to-extend-to-multiple-scripts/25994/2 (дата обращения: 2024-01-30).