УДК 004

# Создание неигрового персонажа на игровом движке Godot

*Черкашин Александр Михайлович*
*Приамурский государственный университет имени Шолом-Алейхема*
*Студент*

**Аннотация**
В данной статье описан процесс написание программы поведения неигрового персонажа (NPC) на игровом движке Godot. В работе использовался язык программирование GDScript и объекты в конструкторе игр на движке Godot для создание неигровой персонаж (NPC) а также было описано поведение неигрового персонажа. В результате работы был создан неигровой персонаж, написанный на языке GDScript на игровом движке Godot.
**Ключевые слова:** Godot, GDScript, NPC.

# Creation of a non-player character on the Godot game engine

*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 the objects in the designer of the Godot game engine to create a non-player character (NPC) as well as described the behavior of a non-player character. As a result of the work, a non-player character was created written in the GDScript language on the Godot game engine.
**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*

Я. Тернер представляет собой работу, посвященную разработке адаптивной системы объектов и анимации для разработки игр. Автор исследования предлагает разработку системы, которая позволяет объектам в игре адаптироваться к различным условиям и ситуациям. Это позволяет создавать более реалистичные и интерактивные игровые миры. В исследовании также рассматривается разработка адаптивной системы анимации, которая позволяет объектам в игре адаптироваться к различным действиям и событиям. Это позволяет создавать более плавные и реалистичные анимации персонажей и объектов в игре. Автор статьи обсуждает преимущества и возможности применения разработанной системы в игровой индустрии. Он также рассматривает потенциальные вызовы и ограничения, связанные с ее внедрением. В статье представлены результаты исследования, включая примеры использования системы и анализ ее эффективности. Автор также обсуждает возможности дальнейшего развития и улучшения системы.

Я. Тернер представляет интерес для разработчиков игр и специалистов в области компьютерной графики и анимации. Оно предлагает новые подходы к созданию адаптивных игровых объектов и анимации, что может привести к улучшению качества и реалистичности игровых проектов [1].

Исследование авторов статьи М. Шаттен, И. Томич, Б. О. Дурич. Представляет собой работу, посвященную разработке программных интерфейсов для оркестрации облачных сервисов в компьютерных играх. В исследовании предлагается разработка программных интерфейсов, которые позволят оркестрировать облачные сервисы в компьютерных играх. Это может включать в себя управление вычислительными ресурсами, хранением данных, сетевыми соединениями и другими облачными возможностями. Преимущества и возможности применения разработанных программных интерфейсов в компьютерных играх. Они также рассматривают потенциальные вызовы и ограничения, связанные с их внедрением. В статье представлены результаты исследования, включая примеры использования программных интерфейсов и анализ их эффективности. Таким образом авторы

также обсуждают возможности дальнейшего развития и улучшения разработанных интерфейсов.

Исследование авторов статьи М. Шаттен, И. Томич, Б. О. Дурич. Представляет интерес для разработчиков компьютерных игр и специалистов в области облачных вычислений. Оно предлагает новые подходы к оркестрации облачных сервисов в компьютерных играх, что может привести к улучшению производительности и возможностей игровых проектов [2].

Исследование авторов статьи И. И. Шагианто и др. представляет собой работу, посвященную разработке двухмерной игры на базе Android с использованием нечеткой логики для NPC (неперсонажей). Авторы исследования разрабатывают двухмерную игру на базе Android, которая имеет жанр приключения и экшена. Игра состоит из трех уровней, каждый из которых имеет различные препятствия и врагов. В исследовании также применяется нечеткая логика для NPC, которая позволяет им иметь различные уровни агрессии, интеллекта и скорости в зависимости от ситуации. Это позволяет создавать более интересные и разнообразные сценарии в игре. В преимущества и возможности применения разработанной игры и нечеткой логики для NPC. Они также рассматривают потенциальные вызовы и ограничения, связанные с их внедрением. В статье представлены результаты исследования, включая примеры игрового процесса и анализ эффективности нечеткой логики для NPC. Авторы также обсуждают возможности дальнейшего развития и улучшения разработанной игры. Таким образом исследование авторов статьи И. И. Шагианто и др. представляет интерес для разработчиков двухмерных игр на базе Android и специалистов в области нечеткой логики. Оно предлагает новые подходы к созданию двухмерных игр с использованием нечеткой логики для NPC, что может привести к улучшению интерактивности и разнообразности игровых проектов [3].

## 2. Рабочий процесс

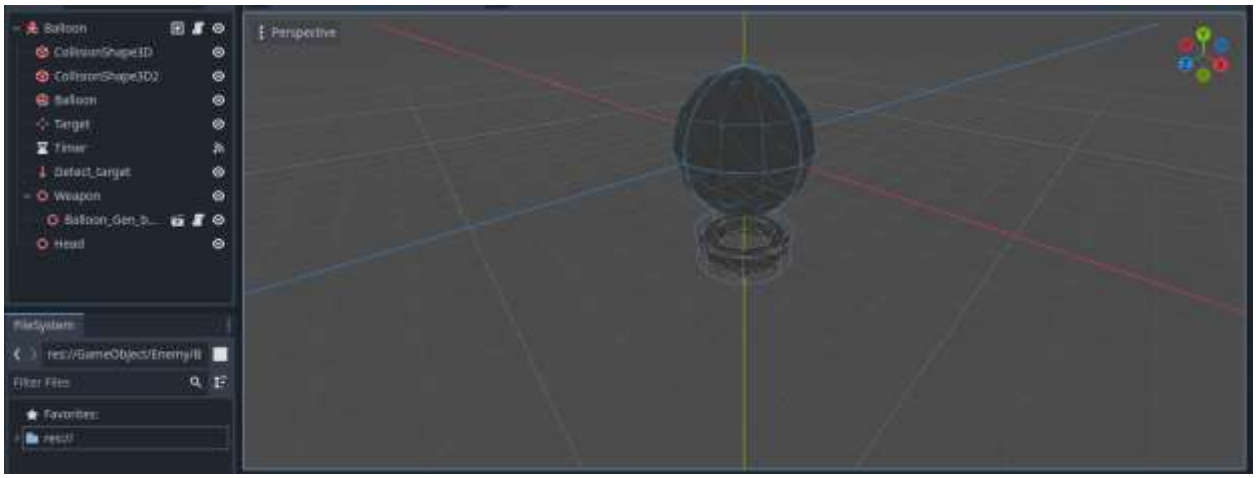


Рисунок 1. Неигровой персонаж Balloon

Неигровой персонаж не контролируется игрока (человек), а управляется определенной заданной программной [4].

Для начала работы создали проект и назвали его «Boy_with_Slingshot». Мы создали неигровой персонаж (рис 1) и назвали его «Balloon», цель неигрового персонажа нападать на игрока и на Building. А цель игрока защитить Building и победить против Balloon. Чтобы этого работало мы создали несколько внутри «Balloon» объектов:

CollisionShape3D — тип CollisionShape3D, форма сфера, радиус 2.5.

CollisionShape3D2 - тип CollisionShape3D, высота 1.2, радиус 2. Расположение y: -4.

Это используется обнаружение столкновение препятствие.

Balloon — тип CSGSphere3D, радиус 2.5, включили столкновение и указали слой столкновение 2, а слой маска 1, 3, 4. Это визуализирует форму персонажа.

Target — тип Marker3D, расположение y: -6. Это используется для появление объектов в начальной точки чтобы персонаж мог сбросить предмет чтобы нападать на игрока.

Timer — тип Timer. Это для задержки между появление новых объектов для атаки на игрока.

Detect_target — тип RayCast3D, Target Position y: -100 (расположение конечной точке), слой столкновение 1, 2, 4, Collide With → Areas на On (Это включает для столкновение объекта типа Area3D). Это для наведение на цель.

Weapon — тип Node3D, расположение y: -4. Это набор средств против игрока.

Balloon_Gen_b... - тип Node3D. В данном случае выбран один средства против игрока.

В неигрового персонажа Balloon содержится набор 3 действие, это перемещение (state_move), атаковать (state_attack), решать (state_solve).

Листинг 2.1. Обработчик состояние.

```
1  func _process(delta):
2          if self.state_move != "":
3                  call(self.state_move)
4          if self.state_attack != "":
5                  call(self.state_attack)
6          if self.state_solve != "":
7                  call(self.state_solve)
```

В state_move (перемещаться) содержит 2 состояние — без действие и st_move_to (перемещение).

В state_attack (атаковать) содержит 2 состояния — без действия и st_shoot (атаковать).

В state_solve (решатель) содержит 3 состояния — без действия и st_select_target (активный поиски противника) и st_attacked (активное преследование противника).

Изначальное состояние Balloon находится st_select_target (искать и находить противников).

Если Balloon не знает Ally и Enemy (Переменная ls_enemy и ls_friends) то вызывает метод get_list_target()

В методе get_list_target с начало в текущий сцене объекты Mob получает список игровых и неигровых персонажей, а затем проверяет отношение к другим персонажам, для этого используется система кланов, и добавляет в список ls_enemy и ls_friends.

Как только Balloon составил список ls_enemy и ls_friends. Balloon находит ближайший по расположение цель, для этого вызывается метод get_prey_with_threat. Максимальный радиус нахождение ближайший цель 1000.

Для прохождения по списку ls_enemy, программа проверяет is_instance_valid и сравнивает null, чтобы избежать ошибки если объект не существует (или освобожден из памяти). А затем находит ближайший (наименьшей) дистанций между Balloon и другим персонажам.

В результате Balloon выбирает наиболее ближайшие цели для нападение (рис 2).

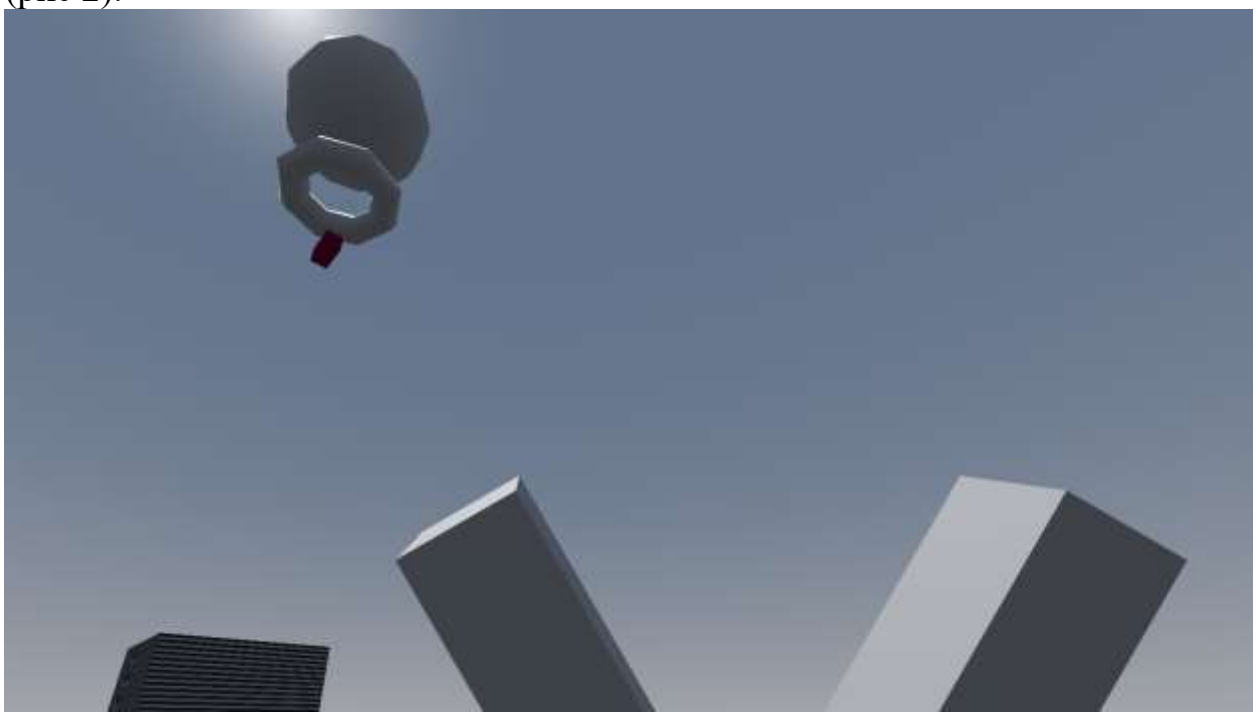

Рисунок 2. Balloon атакует Building

Переменная target_attack полученный возвратом вызова get_list_target. Определяет цель нападение.

Если значение target_attack не пусто, то переключает state_move в состояние st_move_to.

Состояние st_move_to выполняет простое действие перемещение Balloon, если target_attack нет (null), то не выполняет ничего.

Метод relay_race выполняет действия чтобы передать другого дружественный персонаж (например, другой Balloon) команду нападать противника. Если этот персонаж имеет состояние state_solve == "st_attacked"

то Ally будет slave.state_solve = "st_attacked" а этот self.state_solve = "st_select_target".

Листинг 2.2. Метод relay_race.

```
func relay_race() -> bool:
	if self.target_attack == null:
		return false
	var slave = null
	slave = self.find_ally_to_defence(self.target_attack)
	if is_instance_valid(slave) or slave == null:
		return false
	if slave != self and ((slave.target_attack == null and slave.state_solve == "st_select_target") or
			slave.state_solve == "st_attacked"):
		slave.target_attack = self.target_attack
		self.target_attack = null
		if self.state_solve == "st_attacked":
			self.state_solve = "st_select_target"
			slave.state_solve = "st_attacked"
		else:
			slave.state_solve = "st_select_target"
	return true
```

Метод find_ally_to_defence находит ближайший по списку ls_friends, Ally который ближе всего к цели противника. Аргумент threat — цель (листинг 2.3).

Листинг 2.3. Метод find_ally_to_defence.

```
func find_ally_to_defence(threat:Node3D):
	var dis:float = 0.0
	var dis_prev:float = 0.0
	var obj_prev:Node3D
	obj_prev = self._get_friend(0)
	dis_prev = obj_prev.global_position.distance_to(threat.global_position)
	for i in range(len(self.ls_friends)):
		if self.ls_friends[i] == null or not is_instance_valid(self.ls_friends[i]):
			continue
		dis = self.ls_friends[i].global_position.distance_to(threat.global_position)
		if dis <= dis_prev:
			dis_prev = dis
			obj_prev = self.ls_friends[i]
	return obj_prev
```

В состояние st_select_target, вызывает метод detect_target, если возрат был истинный, то переходит в состояние st_shoot из действия state_attack, а действия state_move в без действия, то есть Balloon не перемещается. Если detect_target возращает ложь то действия state_attack переходит в состояние без действия.

В detect_target выполняет проверяет столкновение луч Detect_target объекта на объект, который столкнулся с принадлежащий к классам Building_base и Mob_base, и проверяет является ли клан противник. Аргумент target используется для проверки конкретный объект, указанный в target. Возвращает истина если объект является противник и персонаж.

В состояние st_shoot выполняет посылает 2 последовательный сигнала start_shooting и end_shooting.

Мы назвали несколько слой столкновений для 3D Physics (табл. 1, рис 3).

Таблица 1. Слой столкновений 3D Physics

| Слой | Название |
|---|---|
| 1 | Static |
| 2 | Mob |
| 3 | Items |
| 4 | Object |

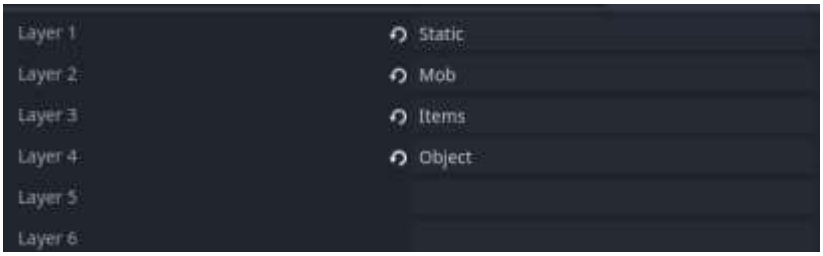


Рисунок 3. Слой столкновений 3D Physics

**3 Выводы**

В результате работы создан неигровой персонаж и написан на языке GDScript, определено поведение и действие неигровой персонаж.

**Библиографический список**

1. Turner I. Adaptable Object and Animation System for Game Development. 2023.
2. Schatten M., Tomicic I., Duric B. O. Towards application programming interfaces for cloud services orchestration platforms in computer games //Central European Conference on Information and Intelligent Systems. Faculty of Organization and Informatics Varazdin, 2020. С. 9-14.
3. Shagianto I. I. et al. Aplikasi Game 2D berbasis Andorid dengan Logika Fuzzy pada NPC (Non-Player Character) //JEITECH (JOURNAL OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY). 2023. Т. 1. №. 1. С. 41-55.
4. Non-player character - Wikipedia // Wikipedia URL: https://en.wikipedia.org/wiki/Non-player_character (дата обращения: 2024-01-29).