УДК 004

Создание контролер управление персонажа на игровом движке Godot (часть 2)

Черкашин Александр Михайлович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье описан процесс создание персонажа для управления от первого лица на игровом движке Godot. В работе использовался язык программирование GDScript и конструктор игр для создания персонажа от первого лица и контроллер управление. В результате работы был создан персонаж и контроллер управление и написано программа на языке GDScript и использован игровой движок Godot.
Ключевые слова: Godot, Character, GDScript.

Creating a character controller using the Godot game engine (part 2)

Cherkashin Alexander Mihailovich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article describes the process of creating a character for first-person control on the Godot game engine. The work used the GDScript programming language and a game designer to create a first-person character and controller control. As a result of the work, a character and a controller were created, a program was written in GDScript and the Godot game engine was used.
Keywords: Godot, Character, GDScript.

1 Введение

1.1. Актуальность исследования

Актуальность исследование заключается в том что поскольку обеспечивает разработчиков мощным инструментарием для реализации увлекательного геймплея. Godot предоставляет гибкие средства для анимации, управления персонажем, а также возможность легкой интеграции с другими элементами игры. Создание персонажа в Godot позволяет разработчикам легко воплощать свои идеи в жизнь, создавая уникальный игровой опыт для игроков.

1.2. Цель исследования

Целью работы создания контроллер управление персонажа игровой движок Godot.

1.3. Обзор исследований

В работе Й. Холфелд описывает как Godot Engine — относительно новый игровой движок, появившийся в 2014 году и конкурирующий с ведущими игроками рынка. Чтобы понять ее актуальность, исследуются две основные платформы онлайн-торговли и широко используемые игровые движки: Steam и itch.io. В основном эти выводы сравниваются со справочными данными за 2018 год. Оказывается, движок Godot приобрел огромную актуальность в 2020 году и теперь, похоже, является одним из ведущих игроков в инди-индустрии. Точные причины определить сложно. Однако эта статья дает много подсказок для дальнейших исследований в этой области [1].

Исследование, проведенное Г. Ц. Уллманн сравнивает графы вызовов двух движков с открытым исходным кодом: Godot 3.4.4 и Urho3D 1.8. Хотя инструменты статического анализа могут предоставить исследователя общую картину без точных путей графа вызовов, использование профилировщика, такого как Callgrind, позволяет исследователя также просматривать порядок и частоту вызовов. Эти графики дают исследователь представление о конструкции двигателей. Г. Ц. Уллманн показали, что с помощью Callgrind авторы могут получить высокоуровневое представление об архитектуре движка, которое можно использовать для ее понимания. В будущей работе Г. Ц. Уллманн намерены применить как динамический, так и статический анализ к другим движкам с открытым исходным кодом, чтобы понять архитектурные шаблоны и их влияние на такие аспекты, как производительность и обслуживание [2].

Р. Розен. исследует, как можно использовать современные технологии игровых движков для создания механизмов визуального ввода и обратной связи, которые стимулируют исследовательское и живое программирование. Р. Розен сообщает об опыте создания среды визуального программирования для Machinations, предметно-ориентированного языка для игрового дизайна. Автор исследователь делится первоначальными выводами о том, как автоматизировать разработку графических и древовидных редакторов в Godot, игровом движке с открытым исходным кодом. В результаты исследование показывают, что современные технологии игровых движков обеспечивают прочную основу для будущих исследований языков программирования [3].

2. Рабочий процесс

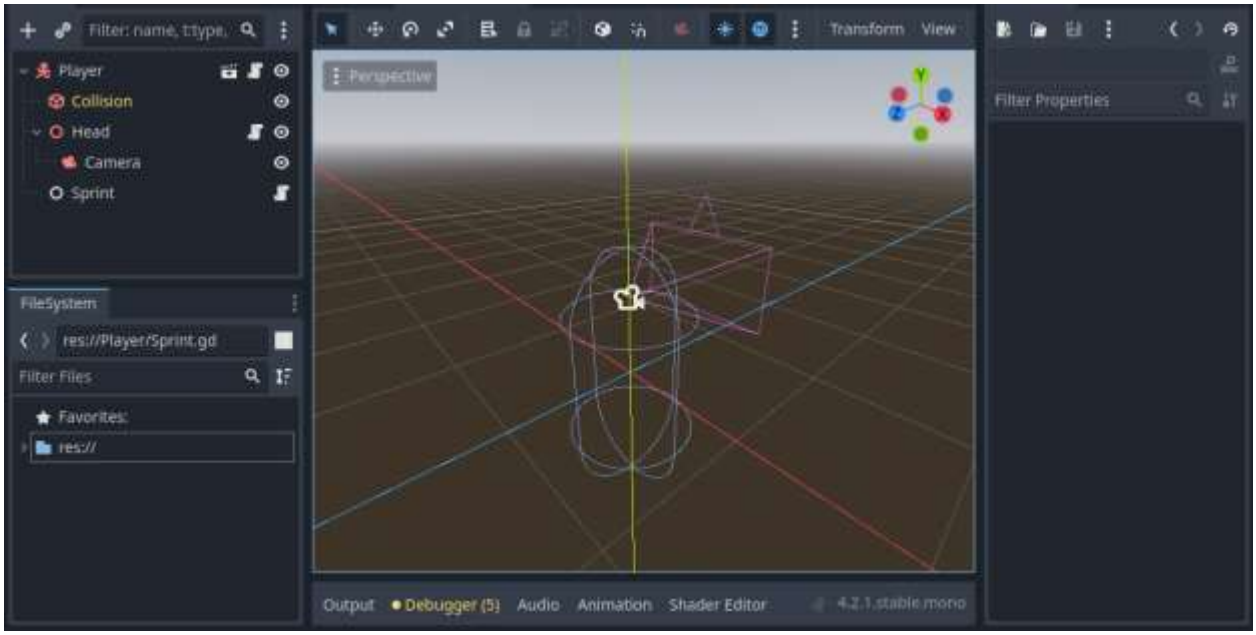

Рисунок 1. Объект игрок «Player.tscn»

Мы создали скрипт и назвали «MovementController» и сохранили файл «res://Player/MovementController.gd».

Листинг 2.2. GDScript «MovementController.gd».

```
1  extends MovementController
2
3  func _input(event: InputEvent) -> void:
4          # Mouse look (only if the mouse is captured).
5          if event is InputEventMouseMotion and Input.get_mouse_mode() == Input.MOUSE_MODE_CAPTURED:
6                  $Head.input_action(event.relative,
7                          Input.get_vector(&"ui_left", &"ui_right", &"ui_down", &"ui_up")
8                  )
9
10 func _physics_process(delta :float) -> void:
11         self.input_action(delta,
12                 Input.get_vector(&"move_back", &"move_forward", &"move_left", &"move_right"),
13                 Input.get_axis(&"sit", &"jump")
14         )
15         $Sprint.input_action(Input.is_action_pressed(&"sprint"))
```

Строка 1. Наследование класса «MovementController».

Строка 3. Функция вызывается когда есть входное событие [4].

Строка 4 — 8 и 11 — 15. Вызывается когда клавиши нажаты w, s, a, d, и space, для перемещение игрока.

Строка 16. Вызывается ускорение игрока нажата клавиша shift.

В Player прикрепили скрипт и сохранили в файл «res://Player/Player.gd».

Скрипт выполняет перемещение игрока управляемый клавиатурой (листинг 2.3).

Листинг 2.3. GDScript «Player.gd».

```
1  extends CharacterBody3D
2  class_name MovementController
3
4  @export var gravity_multiplier := 3.0
5  @export var speed := 10
6  @export var acceleration := 8
```

```
7   @export var deceleration := 10
8   @export_range(0.0, 1.0, 0.05) var air_control := 0.3
9   @export var jump_height := 10
10  var direction := Vector3()
11  var input_axis := Vector2()
12  var input_jump := float(0.0)
13  var fall_down = true
14  # Get the gravity from the project settings to be synced with RigidDynamicBody nodes.
15  @onready var gravity: float = (ProjectSettings.get_setting("physics/3d/default_gravity")
16                  * gravity_multiplier)
17
18  func input_action(delta: float, input_axis:Vector2, input_jump:float) -> void:
19      self.input_axis = input_axis
20      self.input_jump = input_jump
21      direction_input()
22      if self.fall_down:
23          if is_on_floor():
24              if self.input_jump: #Input.is_action_just_pressed(&"jump"):
25                  velocity.y = jump_height
26          else:
27              self.velocity.y -= gravity * delta
28      accelerate(delta)
29      move_and_slide()
30
31  func direction_input() -> void:
32      direction = Vector3()
33      var aim: Basis = get_global_transform().basis
34      direction = aim.z * -self.input_axis.x + aim.x * self.input_axis.y
35
36  func accelerate(delta: float) -> void:
37      # Using only the horizontal velocity, interpolate towards the input.
38      var temp_vel := velocity
39      if self.fall_down:
40          temp_vel.y = 0
41      var temp_accel: float
42      var target: Vector3 = direction * speed
43
44      if not self.fall_down:
45          target.y = self.input_jump * speed
46          #print("target: ", target)
47      if direction.dot(temp_vel) > 0:
48          temp_accel = acceleration
49      else:
50          temp_accel = deceleration
51
52      if not is_on_floor():
53          temp_accel *= air_control
54      temp_vel = temp_vel.lerp(target, temp_accel * delta)
55      velocity.x = temp_vel.x
56      if not self.fall_down:
57          velocity.y = temp_vel.y
58      velocity.z = temp_vel.z
```

Таблица 1. Список переменный в листинг 2.3.

Строка	Название	Значение	Описание
4	gravity_multiplier	3.0	Умножение на ускорение свободного падения
5	speed	10	Скорость
6	acceleration	8	Ускорение
7	deceleration	10	Замедление
8	air_control	0.3	Сопротивление воздуха
9	jump_height	10	Высота прыжка
10	direction	Vector3()	Вектор направление перемещение

11	input_axis	Vector2()	Вектор перемещение при нажатый клавише
12	input_jump	0.0	Значение нажатый клавише прыжке
13	fall_down	true	Если значение истина то игрок падает.
15	gravity		Взято из встроенный переменный свойство «physics/3d/default_gravity» и умноженный на gravity_multiplier. Ускорение свободного падения

Строка 18. Функция обработчик клавиатурный ввода для перемещение и прыжок игрока.

Строка 22 — 27. Обработка прыжок и падение игрока.

Строка 28. accelerate - Обработка ускорение и замедление игрока.

Строка 29. move_and_slide - Перемещает тело на основе скорости (переменная velocity) [5].

Строка 31 — 34. Глобальная расположение игрока умноженный на нажатый клавиатурой w, s, a, d.

Строка 38. temp_vel — текущая скорость игрока.

Строка 39 — 40. Если игрок падает то устанавливаем temp_vel.y = 0.

Строка 41. temp_accel — ускорение.

Строка 42. target — Направление (нажатый клавишам для перемещение) умноженное на скорость (speed).

Строка 44 — 46. Если игрок весит на воздухе то прыжок (нажата прыжок) умноженной на скорость (speed).

Строка 47 — 50. Если есть действия силы (нажатый клавишам для перемещение) то ускорение, иначе — замедление.

Строка 52. Если игрок не касается на пол.

Строка 53. Сопротивление воздуха.

Строка 54. Линейная интерполяция от «temp_vel» (скорость) до «target» (направление), по весу temp_accel * delta (задержка по времени).

Строка 55 — 58. Изменяем скорость игрока.

Мы создали скрипт и назвали «Head» и сохранили в файл «res://Player/Head.gd» и прикрепили объекту «Head».

Скрипт выполняет вращение камеры управляемый курсором (листинг 2.4).

Листинг 2.4. GDScript «Head.gd».

```
1  extends Node3D
2
3  @export_node_path("Node3D") var cam_path := NodePath("Camera")
4  @onready var cam: Node3D = get_node(cam_path)
5  @export var mouse_sensitivity := 2.0
6  @export var y_limit := 90.0
7  var mouse_axis := Vector2()
8  var rot := Vector3()
9  var joystick_axis := Vector2()
10
```

```
11  # Called when the node enters the scene tree for the first time.
12  func _ready() -> void:
13          mouse_sensitivity = mouse_sensitivity / 1000
14          y_limit = deg_to_rad(y_limit)
15
16  func input_action(mouse_axis: Vector2, joystick_axis: Vector2 = Vector2.ZERO) -> void:
17          self.mouse_axis = mouse_axis
18          self.joystick_axis = joystick_axis
19          camera_rotation()
20
21  # Called every physics tick. 'delta' is constant
22  func _physics_process(delta: float) -> void:
23          if self.joystick_axis != Vector2.ZERO:
24                  self.mouse_axis = self.joystick_axis * 1000.0 * delta
25                  camera_rotation()
26
27  func camera_rotation() -> void:
28          # Horizontal mouse look.
29          self.rot.y -= mouse_axis.x * mouse_sensitivity
30          # Vertical mouse look.
31          self.rot.x = clamp(self.rot.x - mouse_axis.y * mouse_sensitivity, -y_limit, y_limit)
32          get_owner().rotation.y = self.rot.y
33          rotation.x = self.rot.x
```

Строка 3. cam_path — Путь объект камеры.

Строка 4. cam — Получение объекта камеры.

Строка 5. mouse_sensitivity — Интенсивность перемещение курсора для вращение камеры.

Строка 6. y_limit — Максимальный угол вращение камеры.

Строка 7. mouse_axis — Направление курсора.

Строка 8. rot — Текущие вращение камеры.

Строка 9. joystick_axis — Направление джойстика.

Строка 13. mouse_sensitivity деленный на 1000.

Строка 14. y_limit. Преобразование градусы в радианы.

Строка 16 — 19. Функция принимающий ввод мыши и джойстик.

Строка 21 — 25. Только для joystick_axis используется для вращение камеры в зависимости нажатой направление джойстика.

Строка 29. Вращение камеры по осью y полученный курсора по осью x умноженный на mouse_sensitivity.

Строка 31 Вращение камеры по осью x полученный курсора по осью y умноженный на mouse_sensitivity, с ограничение угол вращение y_limit.

Строка 32. Вращение по осью y объекта «Player».

Строка 33. Вращение по осью x объекта «Head».

Мы создали скрипт и назвали «Sprint» и сохранили в файл «res://Player/Sprint.gd» и прикрепили объекту «Sprint».

Скрипт выполняет ускорение игрока нажатой клавиатурой (листинг 2.5).

Листинг 2.5. GDScript «Sprint.gd».

```
1   extends Node
2
3   @export_node_path("MovementController") var controller_path := NodePath("../")
4   @onready var controller: MovementController = get_node(controller_path)
5
6   @export_node_path("Node3D") var head_path := NodePath("../Head")
7   @onready var cam: Node3D = get_node(head_path).cam
8
9   @export var sprint_speed := 16
10  @export var fov_multiplier := 1.05
```

```
11  @onready var normal_speed: int = controller.speed
12  @onready var normal_fov: float
13  var input_sprint := float(0.0)
14
15  func _ready() -> void:
16          if self.cam is Camera3D:
17                  normal_fov = cam.fov
18
19  func input_action(input_sprint:float):
20          self.input_sprint = input_sprint
21
22  # Called every physics tick. 'delta' is constant
23  func _physics_process(delta: float) -> void:
24          if can_sprint():
25                  controller.speed = self.sprint_speed
26                  if self.cam is Camera3D:
27                          cam.set_fov(lerp(cam.fov, normal_fov * fov_multiplier, delta * 8))
28          else:
29                  controller.speed = normal_speed
30                  if self.cam is Camera3D:
31                          cam.set_fov(lerp(cam.fov, normal_fov, delta * 8))
32
33  func can_sprint() -> bool:
34          return (controller.is_on_floor() and self.input_sprint #Input.is_action_pressed(&"sprint")
35                          and controller.input_axis.x >= 0.5)
```

Строка 3. controller_path — Путь объекта «Player».

Строка 4. Получение объекта «Player».

Строка 6. Путь объекта «Head».

Строка 7. Получение объекта «Head».

Строка 9. sprint_speed = 16 — Скорость при нажатом клавишей shift.

Строка 10. fov_multiplier := 1.05 — Искажение поле угол обзора камеры при ускоренном движение.

Строка 11. normal_speed — Нормальная скорость без ускоренном движение.

Строка 12. normal_fov — Нормальная поле угол обзора камеры без ускоренном движение.

Строка 13. input_sprint — Значение нажатой клавишей shift, для ускоренном движение.

Строка 15 — 17. Установка переменной normal_fov для получение угол обзора камеры без ускоренном движение.

Строка 19 — 20. Обработчик ввода для ускоренном движение.

Строка 24. Проверка способность игрока ускорятся.

Строка 25. Увеличиваем скорость до sprint_speed.

Строка 26 — 27. Изменяем угол обзора камеры при ускоренном движений если это возможно.

Строка 29. Замедляем скорость до нормальный скорость.

Строка 30 — 31. Изменяем угол обзора камеры при нормальной движений если это возможно.

Строка 33 — 35. Проверяется касается ли игрок на пол и нажато клавиша shift.

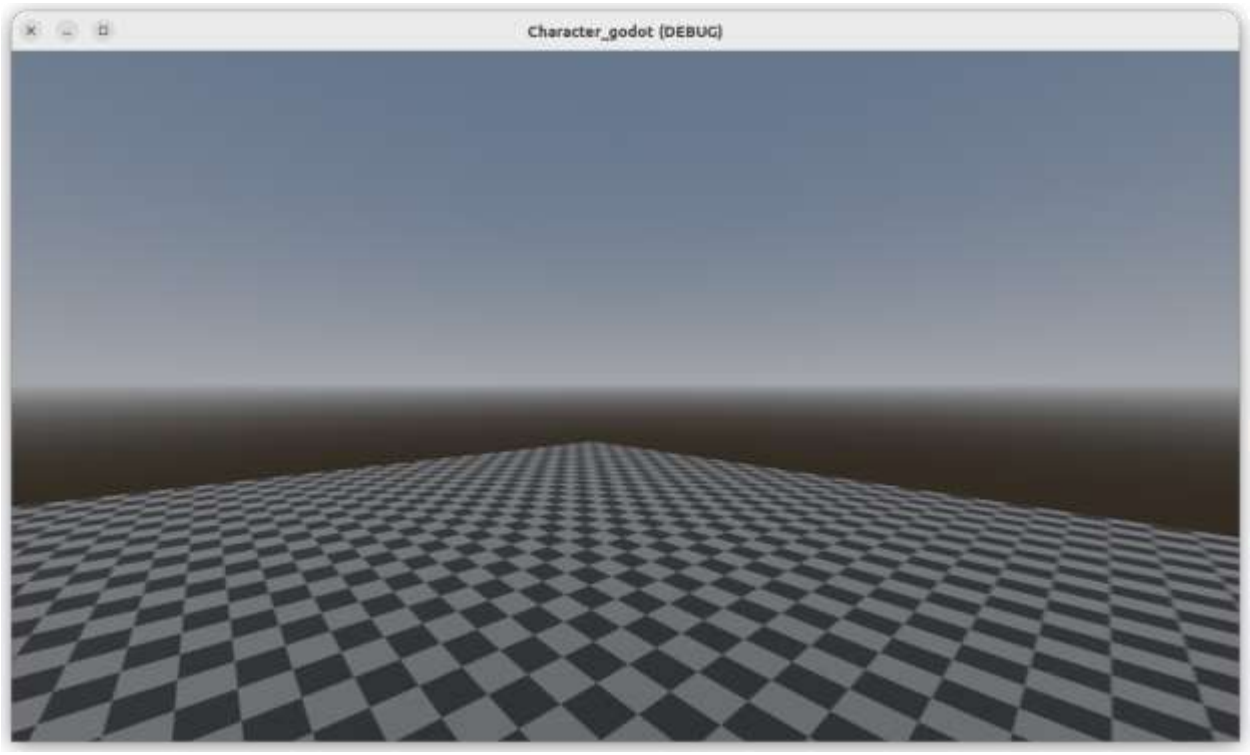

Рисунок 2. Игровой процесс

3 Выводы

В данное статьи была создана игра который игрок может управлять контроллером, перемещаться и прыгать. В результате работы был написан скрипт на языке GDScript, с помощью которого игрок может управлять, перемещается, прыгать и вращать камеру.

Библиографический список

1. Holfeld J. On the relevance of the Godot Engine in the indie game development industry //arXiv preprint arXiv:2401.01909. 2023.
2. Ullmann G. C. et al. Game engine comparative anatomy //International Conference on Entertainment Computing. Cham: Springer International Publishing, 2022. – С. 103-111.
3. van Rozen R. Game Engine Wizardry for Programming Mischief //Proceedings of the 2nd ACM SIGPLAN International Workshop on Programming Abstractions and Interactive Notations, Tools, and Environments. 2023. С. 36-43.
4. Node — Godot Engine (4.2) documentation in English URL: https://docs.godotengine.org/en/4.2/classes/class_node.html#class-node-private-method-input (дата обращения: 2024-01-27).
5. CharacterBody3D — Godot Engine (4.2) documentation in English URL: https://docs.godotengine.org/en/4.2/classes/class_characterbody3d.html#class-characterbody3d-method-move-and-slide (дата обращения: 2024-01-28).