Учебная практика – 1 игра platformer

Содержание

[Введение 2](#_Toc191977099)

[Разработка моделей и механик игры 2](#_Toc191977100)

[Ресурсы 9](#_Toc191977101)

[Заключение 9](#_Toc191977102)

Введение

В этом отчете я расскажу о разработке 2D платформера с использованием игрового движка Godot. Проект представляет собой игру в жанре платформер с процедурной генерацией уровней, системой очков и врагами. Игрок управляет персонажем, который может бегать, прыгать и уворачиваться от врагов, собирая очки за выживание. По мере набора очков игра становится сложнее - враги появляются чаще и становятся сильнее.

Разработка моделей и механик игры

1. Настройка проекта

Начал я с настройки проекта в файле project.godot. В нём я задал основные параметры игры. Например, указал test\_level.tscn как главную сцену, которая будет загружаться при запуске. Также я настроил параметры отображения в секции display, определив ширину и высоту окна, а также режим растягивания (stretch\_mode="viewport" и stretch\_aspect="keep"), чтобы игра корректно отображалась на разных экранах. В секции physics я установил параметры физики 2D, такие как гравитация.

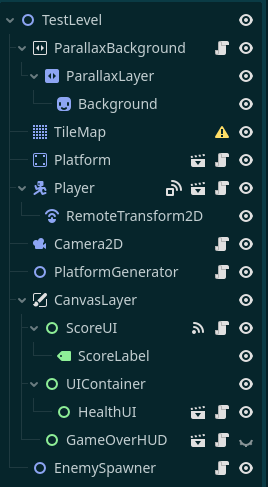


Рисунок 1 - сцена test\_level

1. Управление персонажем

Затем я перешёл к созданию персонажа. Скрипт player.gd, прикреплённый к узлу Player, отвечает за всё его поведение. Я расширил класс CharacterBody2D, чтобы получить доступ к функциям физического тела. В скрипте я определил переменные для скорости (speed), силы прыжка (jump\_velocity), максимальной скорости падения (max\_fall\_speed) и гравитации (gravity).

В функции \_physics\_process(delta) я обрабатываю ввод игрока. Если нажата клавиша “ui\_left” (стрелка влево), я устанавливаю отрицательное значение скорости по оси X, и персонаж движется влево. Аналогично для “ui\_right”. Если персонаж находится на полу (is\_on\_floor()) и нажата клавиша “ui\_accept” (обычно пробел), я вызываю функцию jump(), которая устанавливает скорость по оси Y равной jump\_velocity, заставляя персонажа подпрыгнуть. Я также применяю гравитацию, постоянно увеличивая скорость по оси Y, пока персонаж не коснётся земли.

В функции \_process(delta) я управляю таймером неуязвимости (invincibility\_timer). Если персонаж получает урон, этот таймер запускается, и в течение короткого времени персонаж не получает урона.

Функция die() обрабатывает смерть персонажа. Я отключаю столкновения, проигрываю анимацию смерти и перезапускаю уровень через некоторое время.



Рисунок 2 - главный персонаж

1. Анимация и ресурсы

Для анимации персонажа я использовал узел AnimatedSprite2D в сцене player.tscn. В инспекторе Godot я настроил SpriteFrames, содержащие кадры для каждой анимации: бег, прыжок, атака, смерть. В скрипте player.gd я переключаю анимации в зависимости от состояния персонажа. Например, если персонаж движется по земле, я включаю анимацию “run”. Если он прыгает, я включаю анимацию “jump”.

1. Взаимодействие с объектами

Персонаж взаимодействует с платформами и врагами. В сцене test\_level.tscn я добавил TileMap с настроенным TileSet, который содержит тайлы для платформ. Я настроил слои столкновений, чтобы персонаж мог стоять на платформах. Для платформ, от которых можно отталкиваться, я использовал сцену Platform.tscn, которая содержит Area2D и CollisionShape2D для определения области взаимодействия. Когда персонаж входит в эту область, он получает импульс вверх.



Рисунок 3 - платформа обычная

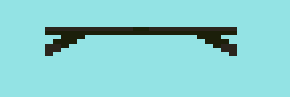


Рисунок 4 - платформа пружинистая

1. Пользовательский интерфейс

Я создал пользовательский интерфейс для отображения здоровья и очков. Скрипт health\_ui.gd, прикреплённый к узлу HealthUI, отображает текущее здоровье персонажа. В функции \_process(\_delta) я постоянно обновляю текст метки HealthLabel, отображая текущее здоровье. Я также изменяю цвет метки в зависимости от уровня здоровья: зелёный, если здоровье больше 60%, жёлтый – если больше 30%, и красный – если меньше. Скрипт score\_ui.gd аналогичным образом отображает очки игрока, обновляя текст метки ScoreLabel.



Рисунок 5 - overlay окончания игры

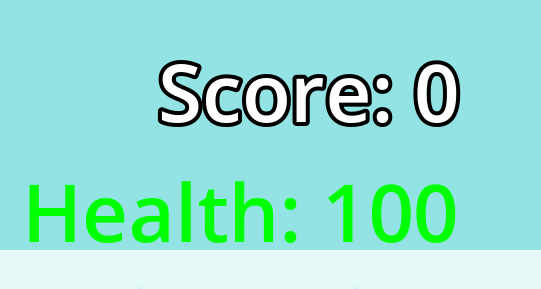


Рисунок 6 - overlay Здоровья / Очков

1. Генерация платформ

Скрипт platform\_generator.gd отвечает за процедурную генерацию платформ. В функции \_ready() я инициализирую генератор случайных чисел. В функции \_process(\_delta) я проверяю, нужно ли генерировать новые платформы. Если самая высокая платформа (highest\_platform\_y) находится выше, чем позиция камеры минус некоторый буфер (VIEWPORT\_BUFFER), я вызываю функцию generate\_platforms().

В generate\_platforms() я создаю несколько платформ. Я использую rng.randf() для случайного выбора между обычной платформой (BasePlatform) и пружинящей платформой (BouncePlatform) с вероятностью BOUNCE\_PLATFORM\_CHANCE. Я добавляю платформу в сцену и устанавливаю её позицию. Координата Y определяется константой PLATFORM\_Y\_SPACING, а координата X – случайным образом в пределах ширины экрана.

Функция \_on\_score\_changed реагирует на изменения счета игрока. Когда счет достигает определенного порога, спавнится платформа из сцены ground.tscn.

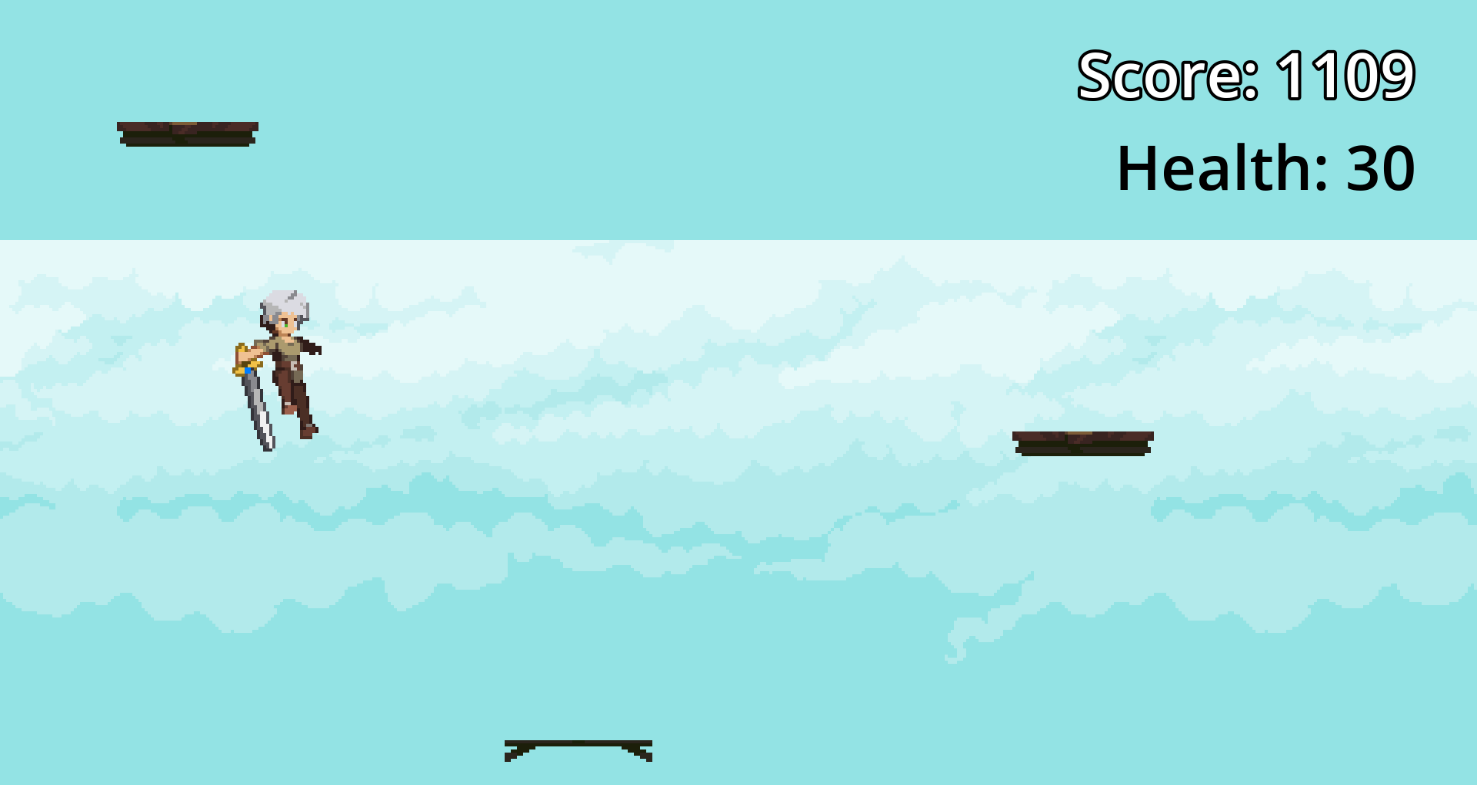


Рисунок 7 - Момент генерации платформ

1. Управление уровнем

Сцена test\_level.tscn объединяет все элементы игры: игрока, платформы, врагов, фон и пользовательский интерфейс. Я добавил TileMap для создания статических платформ, ParallaxBackground для создания эффекта параллакса, Camera2D для следования за игроком, PlatformGenerator для создания динамических платформ, EnemySpawner для создания врагов, а также узлы CanvasLayer для размещения элементов пользовательского интерфейса.

1. Взаимодействие с врагами

Скрипт enemy\_spawner.gd управляет появлением врагов. В функции \_ready() я устанавливаю начальные значения интервала появления (current\_spawn\_interval) и максимального количества врагов (current\_max\_enemies). В функции \_process(delta) я использую таймер (spawn\_timer), чтобы определять, когда нужно создать нового врага.

В функции spawn\_enemy() я создаю экземпляр сцены врага (enemy\_scene). Я определяю случайную позицию для врага, учитывая размеры экрана и положение камеры. Враги могут появляться слева, справа или сверху. Я также масштабирую характеристики врага (здоровье, урон, скорость) в зависимости от текущего счёта игрока, делая их сильнее по мере прохождения игры. Я подключаю сигнал enemy\_died врага к функции \_on\_enemy\_destroyed, чтобы уменьшать счётчик текущих врагов при смерти врага.

Функция update\_difficulty обновляет параметры спавна врагов в зависимости от текущего счета игрока.



Рисунок 8 - взаимодейсвие врага к игроку

1. Камера игры

Скрипт game\_camera.gd управляет камерой. Я расширил класс Camera2D. В функции \_process(\_delta) я проверяю, поднялся ли игрок на новую максимальную высоту (highest\_reached\_y). Если да, то я обновляю позицию камеры по оси Y. Это обеспечивает плавное вертикальное следование за игроком.

1. Контроллер фона

Скрипт background\_controller.gd создаёт эффект параллакса для фона. Я расширил класс ParallaxBackground. В функции \_process(\_delta) я вычисляю смещение камеры и использую его для прокрутки фона. Это создаёт иллюзию глубины сцены.

1. HUD окончания игры

Скрипт GameOverHUD.gd управляет экраном( Рисунок 5 ) “Game Over”. Я расширил класс Control. В функции show\_game\_over() я показываю экран, останавливаю игру (get\_tree().paused = true) и создаю эффект плавного появления с помощью Tween. Функция \_on\_RestartButton\_pressed() обрабатывает нажатие кнопки “Restart”. Она создаёт эффект плавного исчезновения, перезагружает сцену и возобновляет игру.

1. Интерфейс очков

Скрипт score\_ui.gd отображает очки игрока. Я расширил класс Control. В функции \_process(\_delta) я обновляю текст метки ScoreLabel, отображая текущий счёт. Я также использую сигнал score\_changed для оповещения других частей игры об изменении счёта.

1. Интерфейс здоровья

Скрипт health\_ui.gd отображает здоровье игрока. Я расширил класс Control. В функции update\_health\_display(health) я обновляю текст метки HealthLabel и изменяю её цвет в зависимости от уровня здоровья (зелёный, жёлтый, красный). Я также обрабатываю сигналы player\_died и player\_respawned для обновления отображения здоровья при смерти и возрождении игрока.

Ресурсы

Сцена ссылается на различные внешние ресурсы, включая скрипты, упакованные сцены и темы, которые определяют поведение и внешний вид узлов. Это позволяет легко управлять ресурсами и изменять их при необходимости. Использование внешних ресурсов также способствует модульности и повторному использованию кода, что упрощает разработку и поддержку проекта.

Заключение

В этом отчете я подробно описал процесс создания 2D платформера в Godot, начиная с настройки проекта и заканчивая реализацией основных игровых механик. Я рассказал о управлении персонажем, анимации, взаимодействии с объектами, создании пользовательского интерфейса, генерации платформ, управлении уровнем и работе с врагами. Проект позволил мне получить практический опыт разработки игр и глубже изучить возможности движка Godot.