# Assignment 4 – 2D Game Project in the Godot C#

## Learning objectives

In this assignment students will apply their knowledge of C# within the Godot Game Engine to build a simple 2D game. By the end of the project you should be comfortable with:

- Constructing levels using **TileMap** and **TileSet** resources. A tilemap is simply a grid of tiles used to create a game's layout, and you must create a TileSet before you can paint tiles into a TileMap [1].
- Implementing a **player character** using `CharacterBody2D` and moving it using Godot's physics methods (`move_and_slide` and `velocity`). The **top-down movement** recipe shows how to multiply a direction vector (obtained from `Input.get_vector`) by a speed and then call `move_and_slide()` [2].
- Adding **sprite animations** with `AnimatedSprite2D`. Godot's docs explain that `AnimatedSprite2D` is similar to `Sprite2D` except that it stores multiple textures as frames; animations are defined in a `SpriteFrames` resource [3].
- Creating **particle effects**. Godot provides a simple but flexible particle system; particle systems are used to simulate complex physical effects such as sparks, fire, magic particles, smoke or mist [4]. You will learn how to configure `Particles2D` with a `ParticlesMaterial` and optionally randomise parameters [5].
- Implementing **2D navigation** to allow enemies to chase the player. For pathfinding to work you must add navigation support to the TileSet by expanding its **Navigation Layers** section and selecting a layer [6]. An enemy can then use a `NavigationAgent2D` to compute and follow a path [7].
- Applying **simple physics** – gravity, collision detection and acceleration. In the platform-character recipe, a `CharacterBody2D` node with a sprite and collision shape uses `velocity.y += gravity * delta` to apply gravity, sets horizontal velocity from player input, calls `move_and_slide()` and allows jumping when the character is on the floor [8].

## Assignment overview

Develop a small 2D game in Godot 4.4 using C# (GDScript is **not** allowed). The game may be top-down or side-scrolling (your choice), but it must include the following elements:

1. **Tile-based world** – Build your level using a `TileMap` node and one or more `TileSet` resources. Each tile will represent part of the world (floor, walls, obstacles, etc.). Remember: a tilemap is a grid of tiles used to lay out the game, and you need to create a TileSet before you can paint tiles [1].
2. **Player character** – Use a `CharacterBody2D` node as the root of your player. Add a `Sprite2D` (or `AnimatedSprite2D`) and a `CollisionShape2D` as children. Move the character in `_PhysicsProcess()` using a **velocity** vector and call `move_and_slide()` or

`move_and_collide()` to handle collisions. For top-down movement you can take the input vector with `Input.get_vector()` and multiply by a speed, then pass it to `move_and_slide()` [2]. For a platformer, apply gravity each frame (`velocity.y += gravity * delta`) and only allow jumping when `is_on_floor()` returns `true` [8].

3. **Sprite animation** – Create animations for your player and at least one enemy. Use the `AnimatedSprite2D` node or a `Sprite2D` combined with an `AnimationPlayer`. According to the Godot class reference, `AnimatedSprite2D` is like `Sprite2D` but carries multiple textures as animation frames; the frames are stored in a `SpriteFrames` resource and configured via the editor [3]. At minimum, implement idle and movement animations. You should be able to change animations in code (e.g., play a "walk" animation when moving).
4. **Enemies with pathfinding** – Your level must include at least one enemy that can chase or patrol. Implement enemy movement using **2D navigation**:
5. Add navigation layers to your TileSet by expanding its **Navigation Layers** section and adding a layer [6]. Paint navigation onto walkable tiles via the TileSet editor.
6. Attach a `NavigationAgent2D` node to your enemy. In C#, call `set_movement_target()` on the agent to follow the player's position [7]. Use the agent's `velocity_computed` signal (or poll its `next_path_position`) to move the enemy toward the next waypoint.
7. Enemies should avoid walls and obstacles defined in the TileMap.
8. **Particle effects** – Create at least two particle systems using `Particles2D`. For example, use a particle effect when the player picks up an item, when an enemy is defeated, or for environmental effects such as torches or smoke. Remember that particle systems simulate complex effects like sparks, fire or mist [4]. Add a `ParticlesMaterial` to each `Particles2D` node; adjust parameters such as lifetime, speed and randomness [5].
9. **Interactions and simple physics** – Implement interactions between the player and the environment:
10. **Collisions** – The player cannot walk through walls; collisions should be handled by the physics engine. Walls should be defined in the TileSet via the collision layer (as shown in the platformer recipe) or by adding `CollisionShape2D` nodes to static obstacles.
11. **Collectibles or projectiles** – Add an item (e.g., coin, key) that the player can pick up. When collected, play a particle effect and increment a score or change a game state.
12. **Optional: gravity and jumping** – If you choose a side-scrolling game, use gravity and jumping as in the platform-character example [8]. Include friction/acceleration for smoother movement [9].
13. **UI and feedback** – Display minimal UI (score, health or timers). Use Godot's UI nodes (such as `Label` and `TextureProgressBar`) to show game state.
14. **Polish and creativity** – Feel free to add sound effects, backgrounds, power-ups, etc. Creativity will be rewarded, but all core requirements must be met.

## Implementation requirements

- **Programming language** – All scripts must be written in C#. Use partial classes where necessary to organise code. Scripts must be well-commented and use descriptive variable and method names. Avoid duplicating code; factor common behaviour into base classes or helper methods.
- **Project structure** – Organise your project directory logically (e.g., `scenes/`, `scripts/`, `assets/`). Provide a `Main` scene that loads your level and attaches the player and enemy instances.
- **Version control** – Use Git to track your progress. Make regular commits with meaningful messages.

- **README** – Include a `README.md` explaining how to run your game, the controls, and any assets you used.

## Deliverables

1. **Godot project** – Submit the url for your projects Github repository.

2. **Gameplay video** – Record a short clip demonstrating your game: the player moving, enemies using navigation to pursue, sprite animations, particle effects, and collision interactions. Add the link to the video as a comment in canvas.

## Assessment criteria

Your assignment will be graded according to the following rubric:

| Criterion | Description |
|---|---|
| **Functionality - 2 pts** | Does the game run without errors? Are all required features implemented (tilemap, player movement, navigation, animation, particles, physics)? |
| **C# code quality - 2pts** | Are classes and methods well-structured and documented? Does the code make good use of object-oriented principles? |
| **Use of Godot features - 2 pts** | Appropriate use of TileMap/TileSet, NavigationAgent2D, AnimatedSprite2D, Particles2D, CollisionShape2D and CharacterBody2D (with `move_and_slide()`, gravity and input) [2] [8]. |
| **Game design & polish - 2 pts** | Level design, enemy behaviour, UI and overall game feel. Creative touches (sound, story, original art) are a plus. |
| **Documentation & submission - 2 pts** | Presence of README, clear instructions for running the game, and proper packaging of the project. |

## Getting started (See Sample Code)

. Start by creating a new Godot 4.4 project. Add a `Main` scene containing a `TileMap` and load your TileSet. You can draw a simple room or maze.
• To configure collisions for tiles, assign a `Collision` layer and draw the shape in the TileSet panel. Similarly, define **Navigation Layers** on walkable tiles to allow pathfinding [6].
For sprite animations, import your sprite sheet into the project and create an `AnimatedSprite2D`. Use the SpriteFrames editor to define “idle” and “walk” animations [3].
Add a player scene using `CharacterBody2D`. In the attached C# script, declare exported variables for speed, gravity and other parameters. In `_PhysicsProcess()`, update the velocity and call `move_and_slide()` [2].
Create an enemy scene with `CharacterBody2D` and `NavigationAgent2D`. In the script, assign the agent’s `target_position` to the player’s global position each physics frame and apply the

computed velocity. Make sure the enemy cannot cross walls by properly painting navigation and collision layers [6] [7].

- Design at least two particle effects using `Particles2D`. Use small textures or sprite sheets; adjust lifetime, gravity, speed and randomness to get the desired effect [4].
- Test frequently. Make sure the enemy follows the player without getting stuck; ensure collisions work and that animations and particles trigger at the right times.

Good luck, and have fun crafting your 2D game!

---


[1] Learn Godot 4 by Making a 2D Platformer — Part 4: Level Creation #1 - DEV Community
https://dev.to/christinec_dev/learn-godot-4-by-making-a-2d-platformer-part-4-level-creation-1-4obb

[2] Top-down movement :: Godot 4 Recipes
https://kidscancode.org/godot_recipes/4.x/2d/topdown_movement/index.html

[3] CoCalc -- AnimatedSprite2D.xml
https://cocalc.com/github/godotengine/godot/blob/master/doc/classes/AnimatedSprite2D.xml

[4] [5] Particle systems (2D) — Godot Engine (3.6) documentation in English
https://docs.godotengine.org/en/3.6/tutorials/2d/particle_systems_2d.html

[6] [7] Pathfinding Guide for 2D Top-View Tiles in Godot 4.3 - casraf.dev
https://casraf.dev/2024/09/pathfinding-guide-for-2d-top-view-tiles-in-godot-4-3/

[8] [9] Platform character :: Godot 4 Recipes
https://kidscancode.org/godot_recipes/4.x/2d/platform_character/