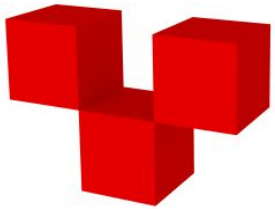
Creating 3D Games in Godot

alvaro.delcastillo@gmail.com



A live workshop

https://github.com/Voxelers/3d/projects/4

Join it!

About me

Open Source developer for around 30 years.

First steps in gaming world to teach my children how to program.

McThings: Project to create procedural worlds in Minecraft using Python. Born during the 3 months covid confinement from “Juntos desde Casa” project.

Co-founder of Voxelers with Javier Cuervo pushing the conversion from 3D shells worlds to solid ones based on voxels.

In love with Godot, I gave a 2D games workshop (slides) two years ago in esLibre 2020. But 3D is cooler! And Godot is great for it! Let’s go!!!

Why should you be here?

Games are fun so this **workshop** is about **fun**

The video game industry is huge, and it is now in disrupting moment: ray tracing, voxels and procedural generation (infinite generation) based on AI are emerging

Game engines lower the learning curve and the cost to create games

Godot is an Open Source engine that is competing with the market leaders (Unreal and Unity) and is embracing the computer graphics revolution (Vulkan)

It is not only about games but interactive digital experiences (G4 Movie Maker)

I have reached this field some months ago, so I know the initial steps to come in

Godot 3.1

8 Oct 2018

Shooter Game Demo

Workshop Steps

Introduction to Game Development

The story behind the game

Godot introduction

Installing Godot, the game and play it!

Layout of the game

Models with materials in the game

Animating the models

World details

Coding the game

Sounds and music

Interactivity design

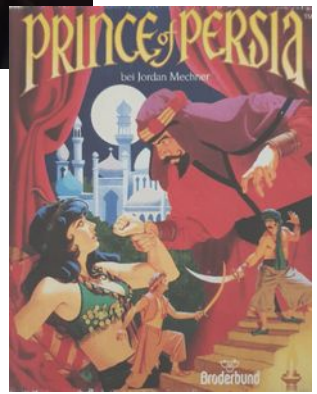
The Physics engine

Collisions

Experience playing the game

Introduction to Game Development

1. Prototype and test key game elements as early as possible.
2. Build the game in incremental steps. Don't make big design documents.
3. As you go, continue to strengthen what's strong and cut what's weak.
4. Be open to the unexpected. Make the most of serendipity.
5. Make sure the player always has a goal (and knows what it is).
6. Give players clear, continuous feedback as to whether they're getting closer or further from a goal.
7. Sometimes a cheap trick is better than an expensive one.
8. The moment when the game first becomes playable is the moment of truth. Don't be surprised if it isn't as much fun as you expected.
9. Listen to the voice of criticism — it's always right (but you have to figure out in what way).
10. Your original vision is not sacred. It's just a rough draft.
11. However much you cut, you'll wish you had cut it sooner.
12. Don't be afraid to consider BIG changes.
13. Draw inspiration from life and primary sources, not just other games and fiction.
14. Develop personal practices that remind you to step back and see the big picture (engage the right side of your brain).
15. Work on paper, not just on screens.
16. Constraints are your friend. They force you to seek creative and elegant solutions.
17. When you discover what the heart of the game is, realign everything to support it.
18. Put your ego aside.
19. Keep a journal.
20. Nobody knows what will succeed.



http://jordanmechner.com/downloads/20_tips.pdf (Review)

The story behind the game

Official Godot tutorials based on this game

Initiated with the 2D version

Evolutionated to a 3D version

Several changes to show new things in the workshop (GitHub code):

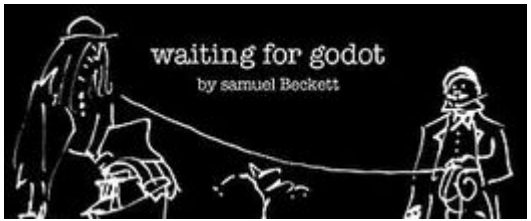
- Animations
- Materials and Textures
- Firing
- World layout

Godot: The Open Source Game Engine

Official synopsis:

«Godot Engine is a **feature**-packed, **cross-platform game engine** to **create 2D and 3D games** from a **unified interface**. It provides a comprehensive set of common tools, so users can focus on making games without having to reinvent the wheel. **Games can be exported** in one click **to a number of platforms**, including the major desktop platforms (Linux, macOS, Windows) as well as mobile (Android, iOS) and web-based (HTML5) platforms.»

All under the Open Source **MIT license**



Installing Godot, the game and play it!

https://godotengine.org/download -> Standard Edition for your System

Unzip the downloaded file “unzip Godot_v3.4.4-stable_x11.64.zip” and execute the only file inside it “Godot_v3.4.4-stable_x11.64”

Download the game from “https://github.com/Voxelers/godot-3d-dodge-the-creeps” with “Code->Download Zip” and import the zip file in Godot

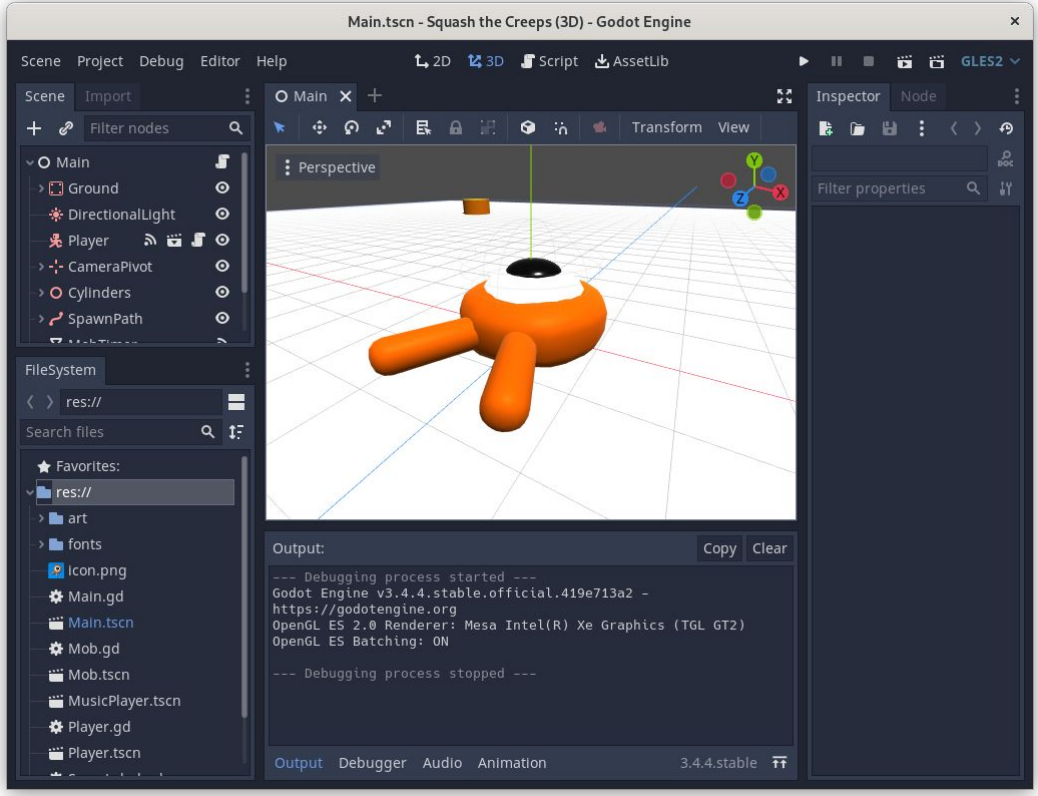
Execute the game (F5). You need to reach **5 points** to continue with the workshop

Quick overview: The game inside Godot

Three column default design:

- Active Scene and Resource Viewers
- Scene editor (2D/3D/Script/Lib)
- Node properties editor

Default Scene already selected



Godot Engine Concepts

All creations in Godot are done inside the Godot Editor (IDE)

All in Godot are Nodes

Nodes are organized in hierarchies

Nodes are added to Scenes

Complex scenes are built composing simple reusable and configurable scenes

The logic is implemented in GDScript, similar to Python but with better threading support. Logic is inside the scenes.

Godot can be extended using C++ for full performance

The Layout of the Game

Field in which the game is played: 3D plane, solid, over which the player and the mobs will move

Player starting position

Mobs paths (they are easy ones, just following a random selected path)

Camera and lights! In this game the lights are internals ones

Scenes: Main which includes Player, Mob and Bullet scenes

Support scenes: Music Player, Dissolve shader testing

Models of the game

3D models normally done outside Godot: **Blender**!

Introduction to modelling techniques: sculpting …

Textures and materials

Low resolution (polys) final models

Exporting from Blender and importing in Godot

Animation of the models

The animation could be done in Blender or Godot. In this game all done in Godot.

It is how models seems to be alive

The animation changes according to the movement

The animation could be done also with Material Shader: Dissolve effect

Player animation when it is destroyed (a kind of dissolve)

Mountain dissolve (using shaders) when the game ends

World Details

Ground is a plane Mesh

Added a new texture: pink marble

Four cylinders to mark the limits of the world

A new mountain to climb it and escape from the mobs: all the climb and descend motion is done by the physics engine

Coding the game

Each Node can have it own logic, coded as a GDScript

Scripts can be parametrized with exported variables (Mob and Bullet scenes in Main node)

Nodes can interact using signals (MobTimer to fire mobs generation)

All nodes from the playing scene are started and their logic executed (concurrently)

Godot engine calls automatically methods in the nodes to execute the logic

For each frame the engine calls **_process** than must finish before the next frame call!

Sounds and music

In this game we have just the background song

It is easy using the media player to add sounds:

- when the player jumps
- when the player hits a mob or jump over it
- when the mob destroys the player

Nice exercise to improve the game. Are you ready?

Interactivity design

Interactivity is key for the player experience with the game

Player with mobs (squash, hit, fire)

Mobs vs mobs (none)

Mobs and player with the ground and mountains

How to implement all this logic?

The Physics engine

If you use nodes with physics properties, the animation logic of the nodes (movement, collision detection, collision response) are handled mostly by the physics engine. 2D and 3D are pretty similar conceptually.

The physics nodes in Godot are:

- Static: Area, StaticBody
- Movement: RigidBody and KinematicBody

Player, Mobs and Bullets are KinematicBody: the movement is controlled by our logic but they have collision detection

The ground and the mountain are StaticBody: collision detection and no movement

Collisions

Collisions detection is one of the trickier parts of Godot (done by physics)

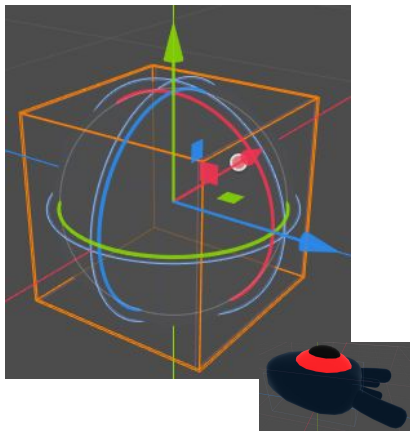
The key logic of the game depends on collisions: a mob (creeper) is squashed by the player, a bullet hits a mob, a mob hits a player or the player scale the mountain

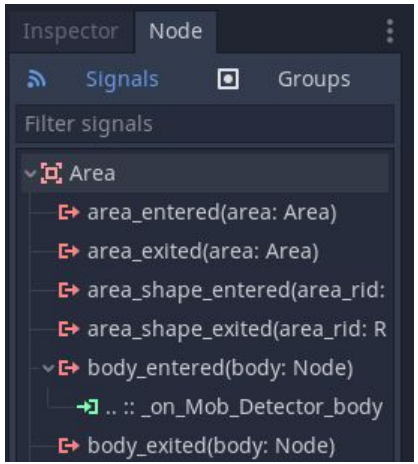
You need to model how the collision will be detected (shapes), in which zones (areas) and the logic to apply once the collision is detected

Collision: Debugging the game logic

Godot IDE has a debugger integrated that can be used during game execution to trace the logic of the nodes, the state, the signal triggering and so on

Let’s debug the collision between a Bullet and a Mob






```
30 func _physics_process(_delta):
31     move_and_slide(velocity)
32
33 func _on_VisibilityNotifier_screen_exited():
34     queue_free()
35
36 func _on_Mob_Detector_body_entered(mob):
37     print("The bullet hit a mob")
38     mob.hit_by_bullet()
39     queue_free()
40
```

Experience playing the game

8. The moment when the game first becomes playable is the moment of truth. Don't be surprised if it isn't as much fun as you expected.

Once you have a playable version of the game, it is time to experience it and to understand if it is fun

It is time to iterate over the game and to polish it. Details matter!

Remember the Pareto principle: 20% of time to develop 80% of the game, and 80% of the time to finish the 20% pending (post)

Games are all about user experience. Without it, you won't have players.

That’s all Folks!

Questions?

https://github.com/Voxelers/3d/projects/4