The Grand Adventure of the Jumping Star

Title Page & Dedication
THE GRAND ADVENTURE OF THE JUMPING STAR

A Beginner's Guide to Building Games with the Godot Engine
Written for Future Game Wizards (Ages 5 and Up!)

[ILLUSTRATION: A large, brightly colored cover illustration showing a cheerful, simple blue character standing on a green platform, waving hello. Yellow coins float in the background.]

Page 1: Introduction
Welcome to the Magic Box
Hello, future Game Wizard!
Have you ever wanted to build your own world? A world where a brave hero can jump high, collect shiny coins, and reach a secret finish line? That's what a Platformer is all about—jumping and exploring!
We are going to open a Magic Box called Godot Engine (say: Goh-doh). Godot is the greatest toy workshop for making games, and it uses a simple language called GDScript—which is really just a list of magic spells our hero can cast!
Ready to be the master builder of your very own 2D world? Let's begin the adventure!

[ILLUSTRATION: A small, friendly wizard character (representing the user) opening a large, glowing treasure chest labeled 'Godot Engine'.]

Page 2: Chapter 1 - Unlocking the Workshop
1.1 Finding Our Tools

First, we need the Magic Box!
Ask a grown-up to find the Godot Engine on the internet.
Download the file. It's so light, it doesn't even need to be installed!
Double-click the file to open the Project Manager.
1.2 Naming the Adventure
Click the big "New Project" button.
Give your game a brave name, like "The Jump Start".
Crucially, make sure the environment is set to 2D. This tells the workshop that we are building a flat world, like a drawing in a notebook!
Click "Create & Edit" to step inside the workshop.

Page 3: The Workshop Windows

Inside the workshop, you will see many windows. They all have important jobs!
The Scene Tab (Top Left): This is where we arrange our Nodes. Think of Nodes as our building blocks and toys.
The Inspector (Top Right): This is our magic wand! When you touch a toy (a Node), the Inspector lets you change its color, size, and powers.
The Script Editor: This is where we write the Magic Spells (the GDScript) to make things move and think!

[ILLUSTRATION: A simple, colorful diagram of the Godot editor, pointing out the Scene Tree (as a list of nested toys), the Inspector (as a color palette), and the 2D Viewport (as a blank stage).]

Page 4: Chapter 2 - Meet the Hero
2.1 Building Our Jumping Star

Every great adventure needs a brave hero! Our hero is a special node called CharacterBody2D that knows how to jump and run.
In the Scene Tab, click "Other Node" and pick the CharacterBody2D node.
Rename this node simply Player.
Give the hero eyes and armor by adding two children:
Sprite2D: This is their costume. Give it a bright color (like blue) in the Inspector.
CollisionShape2D: This is their invisible armor that stops them from falling through the world. Choose a CapsuleShape2D and stretch it to fit the hero's costume.

[ILLUSTRATION: A simple blue character capsule standing proudly, with a separate green outline of the CollisionShape floating next to it, labeled "Invisible Armor."]

Page 5: The Hero's Pocket and Spells
Our hero needs a pocket to hold all the shiny things they find! And they need a spell for running and jumping!

Select the root Player node and click the "Attach Script" scroll icon.
Type this magic spell (GDScript) into the editor. This is a very important spell!
THE JUMPING STAR'S MAGIC SPELL (Player.gd)

```
extends CharacterBody2D

# This is their starting speed!
const SPEED = 300.0
const JUMP_VELOCITY = -450.0

# Hero's Pocket! This starts empty.
var coins_collected = 0
var gravity = ProjectSettings.get_setting("physics/2d/default_gravity")

# --- The Coin Grabber Spell ---
func collect_coin():
    coins_collected += 1
    print("Coins in pocket: ", coins_collected)

func _physics_process(delta):
    # Gravity pulls the hero down!
    if not is_on_floor():
        velocity.y += gravity * delta

    # If you press SPACE and are on the floor, JUMP!
    if Input.is_action_just_pressed("ui_accept") and is_on_floor():
        velocity.y = JUMP_VELOCITY

    # Check Left or Right arrows
    var direction = Input.get_axis("ui_left", "ui_right")

    if direction:
        velocity.x = direction * SPEED
    else:
        velocity.x = move_toward(velocity.x, 0, SPEED * delta * 10)

    # This magic line makes the hero move and check for walls!
    move_and_slide()
```

Remember to save the hero's scene as Player.tscn!

Page 6: Chapter 3 - Building the Solid Ground
Our hero needs a solid ground to stand on. For ground that is solid and does not move, we use the StaticBody2D node.
Start a New Scene (Ctrl+N).
Make a basic Node2D and rename it Level.
Add a StaticBody2D child and rename it Platform.
Give the platform its costume and armor:
Sprite2D: Give it a green or brown color.
CollisionShape2D: Use a RectangleShape2D and stretch it long and flat to be the floor.
Bringing the Hero Home
Go back to your Player.tscn tab.
Click the "Link" icon on the Player node and link the Level.tscn scene into the world.
Move the ground so it is right under the hero's feet.
[ILLUSTRATION: A vertical strip showing the Scene Tree hierarchy: Level (root) -> Platform (Static Body) -> Sprite & CollisionShape.]
Page 7: Testing the Ground
Press the Play Button (F5)!

Your hero should fall gently and stop on the solid platform! Try running and jumping! If they fall through the floor, check that your Platform has the CollisionShape2D armor!
[ILLUSTRATION: The blue hero standing proudly on a green rectangle platform with a big checkmark icon above its head.]

Page 8: Chapter 4 - Collecting Shiny Sparkles

Our coins are sparkly ghosts! They aren't solid; they just need to know when the player touches them. We use the Area2D node for this.
4.1 The Coin Toy and Signal Wire
Start a New Scene (Ctrl+N).
Choose the Area2D node and rename it Coin.
Give the coin a bright yellow costume and a small, round armor (CircleShape2D).
Save the scene as Coin.tscn.
Connect the Wire: Select the Coin node, go to the Node tab, and double-click the body_entered signal. This tells Godot to run a spell when the coin is touched!

Page 9: The Coin's Disappearing Spell
The coin needs a spell that makes it vanish and fill the hero's pocket!

Select the root Coin node and attach this new script:
THE SPARKLE SPELL (Coin.gd)

```
extends Area2D

func _on_body_entered(body):
    # Is the thing that touched us the hero?
    if body.name == "Player":
        # 1. Yes! Call the hero's coin-grabbing spell!
        body.collect_coin()

        # 2. VANISH! POOF!
        queue_free()
```

Putting Sparkles in the World
Open your Level.tscn.
Link the Coin.tscn scene into your level.
Copy and paste lots of them!

[ILLUSTRATION: The blue hero jumping towards a floating yellow coin, with a little sparkle effect and a thought bubble showing the coin count going up.]

Page 10: Chapter 5 - The Grand Finale Lock
Our adventure needs an end! We'll make a secret exit door that only opens when the hero's pocket is full.
5.1 The Goal Door Toy
The exit door is also an Area2D.
Start a New Scene (Ctrl+N). Choose Area2D and rename it Goal.
Give it a big green costume and matching armor.
Save the scene as Goal.tscn.
5.2 The Door's Secret Key Check Spell
The door needs a spell to count the coins in the hero's pocket.
Select the root Goal node and attach this final script. Connect the body_entered signal just like you did with the coin!

THE FINAL DOOR SPELL (Goal.gd)

```
extends Area2D

# IMPORTANT: This number MUST match the total coins in your level!
@export var required_coins = 3

func _on_body_entered(body):
    if body.name == "Player":
        # Does the hero have enough coins?
        if body.coins_collected >= required_coins:
            print("You Win! The Grand Finale is unlocked!")
            get_tree().quit()
        else:
            print("Locked! You need ", required_coins - body.coins_collected, " more coins!")
```

Page 11: The Victory!

The Final Steps

Open your Level.tscn.

Link the Goal.tscn scene and place the door at the end of your level.

VERY IMPORTANT: Select the Goal in the Scene Tab. In the Inspector, find Required Coins and type the total number of coins you placed in your world!

You did it! You are now a Game Wizard!

Run the game and test your creation:

Try to reach the door without all the coins (it should stay locked!).

Collect every sparkle and run to the door for the ultimate victory!

[ILLUSTRATION: The hero standing at the Goal door, which is glowing green, celebrating a victory with a shower of gold sparkles!]

Back Cover: What's Next?

You mastered the Jump Start! Now, where should our hero go next?

In our next adventure, we could learn how to:

Make the Background Magical (using Parallax Scrolling)

Give the Hero a Voice! (adding Music and Sound Effects)

Find out more in the next chapter of "The Grand Adventure of the Jumping Star!”

The Grand Adventure of the Jumping Star

Page 12: Chapter 6 - The Moving Landscape
Making the Background Magical
Did you know your level looks flat because the clouds, the mountains, and the hero all move at the same speed? To make the world look real and deep, we use a magic trick called Parallax Scrolling (say: Parra-lax).
Parallax Scrolling is simple: Things that are far away (like mountains and stars) move very slowly, and things that are close (like the hero and the platform) move quickly. This makes your level look incredibly vast!

6.1 The Parallax Layer
First, we need a special container for our faraway scenery.
Open your Level.tscn scene again.
Add a new node called ParallaxBackground and rename it Skybox. Make sure it is the very first child node under the root Level node!
Add a child to the Skybox called ParallaxLayer and rename it FarClouds. This layer will hold our slowest, furthest background.
Add a Sprite2D child to FarClouds. Give it a light blue color and stretch it to cover the entire game screen. This is our sky!

[ILLUSTRATION: A simple diagram of three layers: 1. Close (Hero/Platform moving fast), 2. Middle (Trees moving slower), 3. Far (Clouds moving slowest).]

Page 13: The Secret Power of Motion

Now, we tell the FarClouds layer how slowly to move.
Select the FarClouds (ParallaxLayer) node.
In the Inspector dock, find the Motion section.
We need to change the Scale property, specifically the X value.
If the hero moves 100 pixels, this is how far the background will move:

Set the Scale: X value to 0.1.
6.2 Adding Closer Hills
Let's add a second layer that moves slightly faster than the clouds but slower than the hero.
Add a second ParallaxLayer child to the Skybox. Rename it NearHills.
Add a Sprite2D child to NearHills. Give it a darker green color and place it just above the floor platform.
Select the NearHills (ParallaxLayer) node.
In the Inspector, set the Scale: X value to 0.5.

Page 14: Final Adjustments
The Repetition Spell (Mirroring)
When the hero runs far enough, the background will run out! To fix this, we need to tell the layer to mirror itself so it repeats forever.
Select the NearHills (ParallaxLayer) node.
In the Inspector dock, find the Motion section again.
Set the Mirroring: X value to 1024. This number is a size guess; it tells the layer to repeat itself every 1024 pixels. If your level is bigger, you can try 2048!
Repeat this step for the FarClouds layer as well!
The Big Test!

Press the Play Button (F5) and make your hero run!
You will see:
The Platforms move quickly (they are close).
The NearHills move slower (they are further away).
The FarClouds barely move at all (they are miles away!).

[ILLUSTRATION: A visual split screen showing the hero running. On the left, the background moves slowly. On the right, a cartoon character says, "Wow, the world looks so deep!"]
That was a big dose of magic! Now the world truly feels enormous.

Page 15: Chapter 7 - The Super Stamping Tool

Building Blocks for Giant Worlds
In our last chapter, we placed one Platform at a time. That's fine for a small start, but what if we want a giant, winding level that goes on forever? We need a faster, better tool!
We will use the TileMap node. Think of a TileMap as a magical sheet of paper, and we will create a TileSet—a giant box filled with tiny stamps we can use to paint the level very quickly!
7.1 Making a Box of Stamps (The TileSet)

The TileSet is our collection of blocks.
Open your Level.tscn scene.
Add a new child node called TileMap. Rename it GroundPainter.
Select the GroundPainter node. In the Inspector dock, find the Tile Set property.
Click [empty] and choose "New TileSet".
Click the new TileSet resource to open the TileSet Editor at the bottom of the screen.
Now we need to draw our first stamp!
In the TileSet Editor, click the + icon and choose "New Atlas".
In the Atlas panel, click "Texture" and choose a simple image or a solid color from Godot's built-in resources to be your block.

Click "Add Single Tile." You now have a block in your stamp collection!
[ILLUSTRATION: A simple drawing of a bucket overflowing with small, identical square blocks (the TileSet). The TileMap node is a paintbrush sweeping across the scene.]

Page 16: Giving the Stamps Armor
Remember the hero needs armor (CollisionShape2D) to stand on the ground? Well, our stamps need armor too, or the hero will fall right through the drawn blocks!
We call this armor the Physics Layer.
Stay in the TileSet Editor. Click on your new stamp.
In the right-hand panel, click the "Physics" tab.
Click the + icon to add a new Physics Layer.
Now, use the drawing tool that appears (it looks like a square or a rectangle) to draw a blue rectangle over the entire stamp block. This blue shape is the invisible armor.
This tells the hero's physics engine: "Stop here! This block is solid!"
7.2 Deleting the Old Floor
If you still have your old, single Platform (StaticBody2D) from Chapter 3 in your level, you should delete it now. The TileMap will be the only ground!
[ILLUSTRATION: A magnifying glass showing a single stamp block. A blue outline (the collision shape) is carefully drawn around the edge of the block.]

Page 17: Painting the Giant Level
Now for the best part—drawing the level instantly!
Go back to the main 2D view by clicking the "2D" tab at the top.
Select the GroundPainter (TileMap) node.
The TileMap Editor sidebar should open on the right. If it doesn't, click on the Node to open it.
Click on the stamp you created in the TileSet sidebar.
Now, move your mouse over the main scene and start clicking and dragging! You are now painting solid ground onto your level, block by block!
You can quickly draw tall walls, long winding paths, or small floating islands. The hero will recognize every single one of these painted blocks as solid ground because you gave the stamp its armor!
The Big Test!
Press the Play Button (F5) and try out your painted world!
You will see:
Your hero can run and jump on every block you drew.
You can now make a much, much bigger level than before!

[ILLUSTRATION: A vibrant, large spread showing a complex, painted level with winding paths and floating islands. The blue hero is jumping between two islands.]
Great work, Game Wizard! You've mastered the art of drawing gigantic worlds instantly.

Page 18: Chapter 8 - Bumps and Bandages

The Great Challenge: Enemy Goblins!

Our hero is getting too good at jumping! It's time to introduce some challenge with an Enemy Goblin. Our Goblin will be a patrol guard, marching back and forth, and if the hero touches it, the hero loses some Health!

8.1 Building the Goblin Patrol

The Goblin is just like the hero—it's a moving character, so it also uses a CharacterBody2D node.

Start a New Scene (Ctrl+N).

Choose the CharacterBody2D node and rename it Goblin.

Give the Goblin a costume (Sprite2D) and armor (CollisionShape2D). Give it a scary color, like red or purple!

Save the scene as Goblin.tscn.

[ILLUSTRATION: A simple red or purple capsule shape with small feet, marching back and forth between two invisible points.]

Page 19: The Goblin's Marching Spell
The Goblin only needs to march left and right, turning around when it hits a wall.
Select the Goblin node and attach a new GDScript.

THE GOBLIN'S MARCHING SPELL (Goblin.gd)

```
extends CharacterBody2D

# How fast the goblin marches
const SPEED = 100.0

# Which direction is the goblin facing? (1.0 is Right, -1.0 is Left)
var direction = 1.0

func _physics_process(delta):
    # This is the marching speed
    velocity.x = direction * SPEED

    # If the goblin hits a wall, it needs to turn around!
    # is_on_wall() is a magic check built into CharacterBody2D.
    if is_on_wall():
        direction *= -1.0 # This spell flips the direction (1.0 becomes -1.0, and vice versa)

        # Optional: Flip the sprite so it looks like it's facing the right way!
        $Sprite2D.flip_h = direction < 0.0

    # This magic line makes the goblin move and check for walls!
    move_and_slide()
```

Placing the Goblin
Open your Level.tscn and link the Goblin.tscn scene.
Place the Goblin on a long platform. When you play, it should march back and forth perfectly!

Page 20: The Hero's Health Spell
Now our hero needs to know how much damage they can take! We update the hero's script to give them a health pocket.
Open the Player.gd script again.
Add a new variable at the top, right below coins_collected.
Update the Hero's Spell (Player.gd)

```
# ... (inside the Player.gd script) ...

# Hero's Pocket! This starts empty.
var coins_collected = 0

# --- Hero's Bandage Pocket! ---
var health = 3 # The hero starts with 3 points of health!

# ... (rest of the script) …
```

The Pain Spell
Now we teach the hero what to do when they get hurt. Add this brand new function to the bottom of the Player.gd script:
THE PAIN SPELL (Player.gd)

```
# ... (inside the Player.gd script, at the bottom) ...

# --- The Pain Spell ---
func take_damage():
    health -= 1 # Lose 1 health point
    print("Ouch! Health remaining: ", health)

    # Check if the hero is completely out of health!
    if health <= 0:
        print("Game Over!")
        # This quits the game when the hero is defeated!
        queue_free() # Makes the hero disappear!
```

Page 21: The Goblin's Attack Spell
Now we need the Goblin to call the hero's take_damage() spell. Since the Goblin is moving, we can't use its main body for the attack; we give it an extra tiny invisible Area2D around its feet for hitting the hero!
Open the Goblin.tscn scene.
Add a child node to the Goblin called Area2D and rename it HurtBox.
Add a CollisionShape2D child to the HurtBox. Make this shape slightly bigger than the Goblin's armor.

8.2 Connecting the Attack Wire
Select the HurtBox (Area2D) node.
Go to the Node tab (next to the Inspector).
Double-click the body_entered signal.

Connect this signal to the Goblin.gd script (where all the Goblin's spells live).
Now, open Goblin.gd and add the final attack spell:
THE GOBLIN'S ATTACK SPELL (Goblin.gd)

```
# ... (inside the Goblin.gd script, at the bottom) ...

# --- The Attack Spell ---
func _on_hurt_box_body_entered(body):
    # Check if the thing that touched the HurtBox is the hero!
    if body.name == "Player":
        # Call the hero's pain spell!
        body.take_damage()
```

The Big Test!
Press the Play Button (F5) and try to bump into your Goblin!
You will see:
The Goblin will march back and forth.
Every time the hero touches the Goblin's HurtBox, the health count in your console will go down!
When the health hits 0, the hero vanishes, and the game ends!

[ILLUSTRATION: The blue hero recoils with a small 'thwack' effect after touching the red Goblin. A small heart icon is shown, with one heart turning gray.]
Fantastic! You've successfully added enemies and a health system!
We have a beautiful, challenging, and functional platformer!