Definitely Normal Physics - Technical Specification

🏗️ System Architecture

Game Engine

Recommended: Godot 4.2+ or Unity 2022.3 LTS

Why Godot:

- Free and open-source
- Excellent 2D physics engine
- Small export size (<30MB)
- Built-in scripting (GDScript) easy to learn
- Cross-platform export (mobile, web, desktop)

Why Unity:

- More AI IDE support/documentation
- Larger asset store
- Better mobile performance optimization
- More third-party tools

Project Structure

```
definitely-normal-physics/
├── assets/
│   ├── sprites/
│   │   ├── player/
│   │   ├── platforms/
│   │   ├── hazards/
│   │   └── ui/
│   ├── audio/
│   │   ├── music/
│   │   └── sfx/
│   └── fonts/
├── scripts/
│   ├── core/
│   │   ├── PhysicsManager.gd
│   │   ├── GameManager.gd
│   │   └── InputManager.gd
│   ├── player/
│   │   ├── PlayerController.gd
│   │   └── PlayerAnimator.gd
```

```
│   ├── physics_states/
│   │   ├── PhysicsState.gd (base class)
│   │   ├── NormalPhysics.gd
│   │   ├── ReverseGravity.gd
│   │   ├── LowGravity.gd
│   │   └── [... 20 total states]
│   ├── level/
│   │   ├── LevelManager.gd
│   │   ├── PhysicsTrigger.gd
│   │   └── Checkpoint.gd
│   └── ui/
│       ├── MenuManager.gd
│       ├── HUD.gd
│       └── DeathScreen.gd
├── levels/
│   ├── world_01/
│   │   ├── level_01.tscn
│   │   ├── level_02.tscn
│   │   └── [... 10 levels]
│   └── [... 10 worlds]
└── README.md
```

🎮 Core Systems

1. Physics State System

Base Physics State Class

gdscript

```gdscript
# PhysicsState.gd
class_name PhysicsState
extends Node

# Properties
var gravity_scale: float = 1.0
var gravity_direction: Vector2 = Vector2.DOWN
var friction: float = 0.5
var bounce: float = 0.0
var speed_multiplier: float = 1.0
var jump_multiplier: float = 1.0
var controls_reversed: bool = false
var input_delay: float = 0.0

# Visual feedback
var tint_color: Color = Color.WHITE
var particle_effect: String = ""
var sound_effect: String = ""

# Called when state becomes active
func on_enter() -> void:
    pass

# Called when state becomes inactive
func on_exit() -> void:
    pass

# Called every frame while active
func update_physics(delta: float, player: CharacterBody2D) -> void:
    pass

# Get state name for debugging
func get_state_name() -> String:
    return "Normal"
```

Example: Reverse Gravity State

gdscript

```gdscript
# ReverseGravity.gd
extends PhysicsState

func _init():
    gravity_scale = 1.0
    gravity_direction = Vector2.UP
    tint_color = Color(0.8, 1.0, 1.0, 1.0)  # Light cyan
    sound_effect = "physics_reverse"

func on_enter() -> void:
    # Play sound, spawn particles, etc.
    AudioManager.play_sfx(sound_effect)

func get_state_name() -> String:
    return "Reverse Gravity"
```

Example: Zero Friction State

gdscript

```gdscript
# ZeroFriction.gd
extends PhysicsState

func _init():
    friction = 0.0
    tint_color = Color(0.7, 0.9, 1.0, 1.0)  # Icy blue
    particle_effect = "ice_sparkle"
    sound_effect = "physics_ice"

func update_physics(delta: float, player: CharacterBody2D) -> void:
    # Ice physics - player keeps sliding
    if player.is_on_floor():
        player.velocity.x = lerp(player.velocity.x, 0.0, friction * delta)

func get_state_name() -> String:
    return "Zero Friction"
```

---

2. Physics Manager

gdscript

```gdscript
# PhysicsManager.gd
extends Node

# Singleton
static var instance: PhysicsManager

# Current active physics state
var current_state: PhysicsState
var state_stack: Array[PhysicsState] = []

# All available physics states
var physics_states: Dictionary = {}

func _ready():
    instance = self
    load_all_states()
    set_state("Normal")

func load_all_states():
    # Register all physics states
    physics_states["Normal"] = NormalPhysics.new()
    physics_states["ReverseGravity"] = ReverseGravity.new()
    physics_states["LowGravity"] = LowGravity.new()
    physics_states["HighGravity"] = HighGravity.new()
    physics_states["ZeroFriction"] = ZeroFriction.new()
    # ... register all 20 states

func set_state(state_name: String, instant: bool = false):
    # Exit current state
    if current_state:
        current_state.on_exit()

    # Enter new state
    current_state = physics_states[state_name]
    current_state.on_enter()

    # Update visual feedback
    update_visual_feedback(instant)

    # Notify player
    EventBus.physics_changed.emit(current_state)

func update_visual_feedback(instant: bool):
    # Tint screen, update particles, etc.
    var tween = create_tween()
    if instant:
```

```
        VisualEffects.set_screen_tint(current_state.tint_color)
    else:
        tween.tween_property(VisualEffects, "screen_tint",
                             current_state.tint_color, 0.3)

func get_current_gravity() -> Vector2:
    return current_state.gravity_direction * current_state.gravity_scale * 980.0

func get_current_friction() -> float:
    return current_state.friction

# For level 51+ - stack multiple physics
func push_state(state_name: String):
    state_stack.push_back(current_state)
    set_state(state_name)

func pop_state():
    if state_stack.size() > 0:
        var previous = state_stack.pop_back()
        set_state(previous.get_state_name())
```

---

3. Player Controller

gdscript

```gdscript
# PlayerController.gd
extends CharacterBody2D

# Movement constants
const BASE_SPEED = 300.0
const BASE_JUMP_VELOCITY = -500.0
const ACCELERATION = 2000.0

# Current physics modifiers (updated by PhysicsManager)
var speed_multiplier: float = 1.0
var jump_multiplier: float = 1.0
var gravity_scale: float = 1.0
var controls_reversed: bool = false

# Input buffer
var input_queue: Array = []
var input_delay: float = 0.0

func _ready():
    EventBus.physics_changed.connect(_on_physics_changed)

func _physics_process(delta):
    # Get current physics
    var physics = PhysicsManager.instance.current_state

    # Apply gravity
    if not is_on_floor():
        var gravity = PhysicsManager.instance.get_current_gravity()
        velocity += gravity * delta * physics.gravity_scale

    # Handle input (with delay if needed)
    var input = get_processed_input()
    handle_movement(input, delta, physics)
    handle_jump(input, physics)

    # Apply friction
    if is_on_floor():
        velocity.x = lerp(velocity.x, 0.0,
                          physics.friction * delta * 10.0)

    # Apply bounce (if bouncy physics active)
    if physics.bounce > 0 and is_on_floor_only():
        velocity.y = -abs(velocity.y) * physics.bounce

    move_and_slide()
```

```
func get_processed_input() -> Dictionary:
    var current_input = {
        "move_left": Input.is_action_pressed("move_left"),
        "move_right": Input.is_action_pressed("move_right"),
        "jump": Input.is_action_just_pressed("jump")
    }

    # Handle input delay
    if input_delay > 0:
        input_queue.append({
            "input": current_input,
            "time": Time.get_ticks_msec()
        })

        # Process delayed input
        if input_queue.size() > 0:
            var oldest = input_queue[0]
            if Time.get_ticks_msec() - oldest.time >= input_delay * 1000:
                current_input = oldest.input
                input_queue.pop_front()

    # Handle reversed controls
    if controls_reversed:
        var temp = current_input.move_left
        current_input.move_left = current_input.move_right
        current_input.move_right = temp

    return current_input

func handle_movement(input: Dictionary, delta: float, physics: PhysicsState):
    var direction = 0.0
    if input.move_right:
        direction += 1.0
    if input.move_left:
        direction -= 1.0

    if direction != 0:
        var target_speed = direction * BASE_SPEED * physics.speed_multiplier
        velocity.x = move_toward(velocity.x, target_speed,
                                 ACCELERATION * delta)

func handle_jump(input: Dictionary, physics: PhysicsState):
    if input.jump and is_on_floor():
        velocity.y = BASE_JUMP_VELOCITY * physics.jump_multiplier

func _on_physics_changed(new_state: PhysicsState):
    # Update cached physics values
```

```
    speed_multiplier = new_state.speed_multiplier
    jump_multiplier = new_state.jump_multiplier
    gravity_scale = new_state.gravity_scale
    controls_reversed = new_state.controls_reversed
    input_delay = new_state.input_delay

func die():
    EventBus.player_died.emit()
    # Reset to checkpoint
    get_tree().reload_current_scene()
```

---

4. Physics Trigger System

gdscript

```gdscript
# PhysicsTrigger.gd
extends Area2D

@export var target_physics_state: String = "ReverseGravity"
@export var trigger_type: String = "enter"  # "enter", "timer", "distance"
@export var delay: float = 0.0
@export var one_time: bool = true

var has_triggered: bool = false

func _ready():
    body_entered.connect(_on_body_entered)

func _on_body_entered(body):
    if body.is_in_group("player") and not has_triggered:
        if trigger_type == "enter":
            trigger_physics_change()

func trigger_physics_change():
    if one_time:
        has_triggered = true

    if delay > 0:
        await get_tree().create_timer(delay).timeout

    PhysicsManager.instance.set_state(target_physics_state)

    # Visual indicator (optional, for debugging)
    if OS.is_debug_build():
        print("Physics changed to: ", target_physics_state)
```

---

5. Level Manager

gdscript

```gdscript
# LevelManager.gd
extends Node

var current_level: int = 1
var current_world: int = 1
var death_count: int = 0
var level_start_time: float = 0.0

var checkpoint_position: Vector2 = Vector2.ZERO

func _ready():
    EventBus.player_died.connect(_on_player_died)
    EventBus.level_complete.connect(_on_level_complete)
    level_start_time = Time.get_ticks_msec()

func _on_player_died():
    death_count += 1
    # Update UI
    EventBus.death_count_updated.emit(death_count)

func _on_level_complete():
    var completion_time = (Time.get_ticks_msec() - level_start_time) / 1000.0
    var stars = calculate_stars(death_count, completion_time)

    # Save progress
    SaveManager.save_level_completion(current_world, current_level,
                                      stars, death_count, completion_time)

    # Show completion screen
    EventBus.show_level_complete.emit(stars, death_count, completion_time)

func calculate_stars(deaths: int, time: float) -> int:
    if deaths <= 5:
        return 3
    elif deaths <= 10:
        return 2
    else:
        return 1

func set_checkpoint(position: Vector2):
    checkpoint_position = position

func respawn_at_checkpoint():
    var player = get_tree().get_first_node_in_group("player")
```

```
    if player:
        player.global_position = checkpoint_position
```

Data Structures

Level Data Format (JSON)

```json
```

```
{
  "level_id": "world_01_level_03",
  "world": 1,
  "level": 3,
  "name": "The Ice Surprise",
  "physics_sequence": [
    {
      "trigger_type": "distance",
      "trigger_value": 5.0,
      "physics_state": "ZeroFriction",
      "duration": -1
    },
    {
      "trigger_type": "position",
      "trigger_position": [15.0, 10.0],
      "physics_state": "LowGravity",
      "duration": -1
    }
  ],
  "platforms": [
    {"position": [0, 0], "size": [2, 1], "type": "normal"},
    {"position": [5, 2], "size": [3, 1], "type": "moving"}
  ],
  "hazards": [
    {"position": [8, 5], "type": "spike"},
    {"position": [12, 3], "type": "saw", "movement": "horizontal"}
  ],
  "checkpoints": [
    {"position": [10, 8]}
  ],
  "exit": {"position": [20, 12]},
  "star_thresholds": {
    "3_star_deaths": 5,
    "2_star_deaths": 10,
    "3_star_time": 30.0
  }
}
```


Save Data Format

json

```
{
  "player_id": "uuid-here",
  "settings": {
    "music_volume": 0.8,
    "sfx_volume": 1.0,
    "show_death_counter": true
  },
  "progress": {
    "world_01": {
      "level_01": {
        "completed": true,
        "stars": 3,
        "best_deaths": 3,
        "best_time": 25.5,
        "total_attempts": 8
      }
    }
  },
  "unlocked_cosmetics": ["skin_red", "trail_fire"],
  "total_playtime": 3600,
  "total_deaths": 247
}
```

🎨 Asset Specifications

Sprite Sizes

- Player: 32x32 pixels
- Platform tiles: 32x32 pixels (tileable)
- Hazards: 32x32 or 64x64 pixels
- UI elements: Vector or high-res (will scale)

Color Palette

```
Normal:       #FFFFFF (white)
Low Gravity:  #A0D8F0 (light blue)
High Gravity: #E06060 (red)
Zero Friction: #B0E8FF (icy blue)
Bouncy:       #FFB0FF (pink)
Background:   #2C2C3C (dark gray)
Platforms:    #5A5A6E (medium gray)
```

Hazards: #FF4040 (danger red)
Exit: #40FF40 (safe green)

Animation Requirements

- Player idle: 2 frames
- Player run: 4 frames
- Player jump: 1 frame
- Player fall: 1 frame
- Player death: 3 frames
- Platform moving: 2 frames (optional)
- Hazard spinning: 4 frames

🔊 Audio Specifications

Music Tracks Needed

1. Menu Theme (looping, 90 BPM)
2. World 1-3 Theme (upbeat, 120 BPM)
3. World 4-6 Theme (intense, 140 BPM)
4. World 7-10 Theme (chaotic, 160 BPM)
5. Victory Theme (short, 30 seconds)

Sound Effects List

1. Player jump
2. Player land
3. Player death (quick, not annoying)
4. Physics change (subtle whoosh)
5. Checkpoint reached
6. Star collect
7. Level complete
8. UI button click
9. UI button hover
10. Spike hit
11. Platform moving
12. Exit door open

📱 Mobile Optimization

Performance Targets

- Maintain 60 FPS on mid-range devices (2018+)
- Maximum 50 draw calls per frame
- Texture atlas: Single 2048x2048 texture
- Object pooling for particles

Battery Optimization

- Pause game when app backgrounded
- Reduce particle effects on low battery mode
- Screen dimming disabled during gameplay

Touch Controls

- Virtual joystick size: 120x120 dp
- Jump button size: 100x100 dp
- Dead zone: 15% of joystick radius
- Visual feedback on touch (button press animation)

🧪 Testing Checklist

Physics Testing

- [ ] All 20 physics states work correctly
- [ ] Physics transitions are smooth
- [ ] Combined physics (2-3 states) don't break game
- [ ] Player can't clip through platforms
- [ ] Death triggers are fair

Platform Testing

- [ ] iOS 13+ (iPhone 8, XR, 12, 14)
- [ ] Android 8+ (Samsung S8, Pixel 3, OnePlus 9)
- [ ] WebGL (Chrome, Firefox, Safari)
- [ ] Windows 10/11
- [ ] macOS 10.14+

Balance Testing

- [ ] Levels 1-10 completable by 90% of testers
- [ ] Levels 11-50 completable by 60% of testers
- [ ] Levels 51-100 completable by 30% of testers
- [ ] Average player reaches level 30

🚀 Build & Deployment

Build Configurations

Mobile (iOS/Android)

```
Resolution: 1920x1080 (scales to device)
Target FPS: 60
Physics FPS: 60
Compression: ETC2 (Android), ASTC (iOS)
Build size target: <50MB
```

Web (WebGL)

```
Resolution: Scalable (960x540 minimum)
Target FPS: 60
Compression: WebP
Max memory: 256MB
```

Desktop (Windows/Mac)

```
Resolution: 1920x1080 default (fullscreen/windowed)
Target FPS: 60
Compression: PNG
Executable size: <100MB
```

Version Control

- Git repository structure
- Branch strategy: main, develop, feature/*
- Semantic versioning: v1.0.0

📊 Analytics Events to Track

1. `level_start` - {world, level, attempt_number}
2. `level_complete` - {world, level, deaths, time, stars}
3. `player_death` - {world, level, death_position, physics_state}
4. `physics_change` - {from_state, to_state, player_position}
5. `session_start` - {platform, version}
6. `session_end` - {duration, levels_played}
7. `premium_purchase` - {price, timestamp}
8. `ad_watched` - {placement, completed}
9. `share_clicked` - {content_type}
10. `settings_changed` - {setting_name, new_value}

🔒 Security Considerations

Anti-Cheat

- Server-side validation for leaderboards
- Checksum validation for save files
- Rate limiting on API calls
- Encrypted player preferences

Data Privacy

- GDPR compliant (EU)
- COPPA compliant (US, age 13+)
- Minimal data collection
- Clear privacy policy
- User can delete all data

🛠️ Development Tools

Recommended Tools

- **IDE:** Visual Studio Code with GDScript extension
- **Version Control:** Git + GitHub
- **Pixel Art:** Aseprite or Pixelorama

- **Sound Effects:** BFXR or sfxr
- **Music:** Beepbox or FL Studio
- **Level Design:** Godot's built-in tilemap editor
- **Analytics:** GameAnalytics or Unity Analytics

AI IDE Integration

- Use AI for code generation (boilerplate, physics states)
- AI-assisted level design (generate platform layouts)
- AI bug detection and optimization suggestions

This technical spec provides the foundation. Next, I'll create specific AI IDE prompts!