Part 1: The Core Reel Mechanic

Goal: Build the "Base", create a functional slot machine that spins, stops, and handles basic symbol logic. A machine with 3 or 5 reels. You press a button, they accelerate (with blur), spin, and stop one by one with a mechanical bounce, landing on ***pre-specific*** target symbols.

- Setup & The Hierarchy Strategy
  - Concept: Slot games have deep hierarchies (Machine -> ReelGroup -> Reel -> SymbolContainer -> Symbol).
  - Task: Set up the project. Create the visual layout using Mask components (essential for hiding symbols as they scroll off-screen).
  - Hint: "Clipping" is your best friend here.
- The "Infinite Scroll" Logic
  - Concept: How to make 5 symbols look like an infinite spinning reel.
  - Task: Implement the "Loop" logic. When a symbol moves below the bottom threshold ($y < -100$), instantly move it to the top ($y = +500$) and swap its texture.
  - Hint: node.position.y -= speed * dt.
- State Machines (The Brain)
  - Concept: Slots are strict State Machines: IDLE -> SPINNING_ACCEL -> SPINNING_CONST -> STOPPING -> RESULT.
  - Task: Implement a basic State Machine that controls the reel speed.
- Easing & The "Bounce"
  - Concept: "Mechanical Feel." A reel shouldn't just stop instantly; it needs a "Recall" (bounce back) effect.
  - Task: Use tween(node) with easing: 'backOut' for the stop animation. This gives the reel weight.
  - Key Learning: Mastering the Tween system is 50% of slot programming.
- Symbol Configuration (ScriptableObjects)
  - Concept: Data-driven design.
  - Task: Create a configuration file (JSON or TypeScript Class) defining Symbol IDs (0=Cherry, 1=7, 2=Wild) and map them to SpriteFrames.
- The Result Matrix
  - Concept: Separation of View and Data.
  - Task: Generate a random array [1, 3, 4] (the target result). The reels must spin indefinitely *until* they receive this array, then stop at those specific symbols.
- Blurred Textures (Motion Blur)
  - Concept: Speed illusion.
  - Task: Detect when the reel speed is high. Swap the crisp "Symbol" sprite with a "Blurred Symbol" sprite. Swap it back when slowing down.
  - Effect: This makes the spin look smooth and professional (60fps feel).
- Part 1 Review - The "Base Game"
  - Deliverable: Test build on web, and (APK for android device or installed build on iphone)

Part 2: Menus, Popups & Scene Flow

Goal: Create a seamless "User Experience" wrapper around the slot machine. We will build a scalable `PopupManager`, handle scene transitions, and implement "Modal" logic (pausing the world while the UI is open).

- The UI Architecture (Stack System)
  - Concept: Don't just put all popups in the scene and toggle `active`. Use a `PopupManager`.
  - Task: Create a generic `PopupBase` class. Create a `PopupManager` that handles a Stack of popups.
  - Logic: When `PopupManager.show(Settings)` is called:
    1. Load the prefab.
    2. Play an "Open" animation.
    3. Push to stack.
    4. Darken the background.
- Blocking Input (The Modal Problem)
  - Problem: If a generic "Settings" popup is open, you shouldn't be able to click the "Spin" button behind it.
  - Solution: The `BlockInputEvents` Component.
  - Task: Create a "Scrim" (black semi-transparent background). Add `BlockInputEvents` to it. Ensure this Scrim is always behind the *topmost* popup but in front of the game.
- The Main Menu (Lobby)
  - Concept: Scene Management.
  - Task: Create a `LobbyScene`. It needs:
    1. A "Play" button (transitions to GameScene).
    2. A "Quit" button.
    3. PersistRootNode: Create a node that survives scene transitions (perfect for background music that shouldn't cut off when changing scenes).
- Scene Transitions (Loading Screens)
  - Concept: Async Loading.
  - Task: Create a fake "Loading..." scene.
  - Code: Use `director.preloadScene()` while showing a progress bar (0% to 100%).
  - App Dev Note: Unlike apps, games often freeze the main thread while initializing. A transition scene masks this.
- The Pause Logic (TimeScale)
  - Concept: Freezing the Game vs. Freezing the App.
  - Task: Implement the Pause Button.
  - Crucial Logic:
    1. *Bad Way:* `director.pause()` (This stops *everything*, including the Pause Menu animations!).

        2. *Good Way:* Set a global flag `GameManager.isPaused = true`. In your slot reel update loop, add `if (isPaused) return;`.
    - Task: Make the Pause Menu animate in (bounce) even though the reels are frozen.
- The "Paytable" (ScrollViews)
    - Concept: Complex Layouts. Slot games have massive "How to Play" screens.
    - Task: Master the `ScrollView`, `Mask`, and `Layout` components.
    - Challenge: Create a dynamic Paytable where the text updates based on the current Bet amount (e.g., "5x Cherries pays $50" becomes "$100" if bet is doubled).
- Dynamic "Toast" Messages
    - Concept: Non-blocking feedback (like Android Toasts).
    - Task: Create a system for small messages ("Not enough coins!", "Connected to Server").
    - Implementation: A prefab pool of labels that fade in, float up, and fade out.
- Tweening UI (Juice)
    - Concept: UI in games must feel "alive."
    - Task:
        1. Button Press: Scale button to 0.9 on `TOUCH_START`, restore to 1.0 on `TOUCH_END`.
        2. Popup Open: Scale from 0 to 1 with `easing: 'backOut'`.
        3. Popup Close: Scale from 1 to 0 with `easing: 'backIn'`.
- Settings & Local Storage
    - Task: Connect the UI to the Data.
    - Features:
        1. Music Volume Slider (controls the `AudioManager`).
        2. SFX Toggle.
        3. Save these preferences to `sys.localStorage` so they persist when the app restarts.
- Part 2 Review - The "Full Flow"
    - Deliverable:
        1. Open App -> Show Splash Screen.
        2. Load Main Menu (Music starts).
        3. Click Play -> Loading Bar -> Game Scene.
        4. Play Game -> Pause -> Change Volume -> Resume.
        5. Game Over -> Popup Result -> Click "Home" -> Back to Main Menu.

Part 3: Audio, Particles & "The Juice"

Goal: Make it addictive. This phase focuses on **Audio**, **Particles**, and **Spine Animations** (industry standard for Slots). Coins fly, numbers tick up, and the reels stop with a satisfying "Clack" sound.

- The Audio Manager (Layers)
    - Concept: Slot audio is complex. You need parallel tracks.
    - Task: Build an AudioManager.

        - Channels: BGM (Loop), SFX (Spin button, Reel Stop), Voiceover ("Big Win!").
        - Logic: The "Reel Stop" sound must trigger exactly when the reel bounce tween finishes.
- Audio Dynamics (Pitch & Urgency)
    - Concept: Anticipation.
    - Task: If the player lands 2 "Scatter" symbols (waiting for the 3rd to trigger a bonus), loop a "tension" sound and speed up the visual spin of the remaining reels.
- Particle Systems (Cocos Built-in)
    - Concept: Visual feedback for wins.
    - Task: Create a "Coin Explosion" effect.
    - Settings: Learn to manipulate Gravity, Emission Rate, and LifeTime in the Cocos Particle Editor. Trigger this only on a "Win" state.
- Spine/DragonBones Animations
    - Concept: Modern slots don't use static sprites for high-value symbols; they use skeletal animation.
    - Task: Import a Spine animation (e.g., a character that waves when you win).
    - Code: sp.Skeleton.setAnimation(0, 'win_loop', true).
    - Performance Note: Spine is expensive. Learn to pause the animation when the reel is spinning to save CPU.
- Line Logic & Visual Connectors
    - Concept: Showing the user *why* they won.
    - Task: Draw lines over the winning symbols using Graphics (Cocos drawing API) or instantiate prefab borders around winning items.
- The "Win Rollup" (Number Ticking)
    - Concept: Psychological reward.
    - Task: Create a text label that counts up from 0 to the Won Amount.
    - Math: currentScore = lerp(currentScore, targetScore, dt * speed). Hook a "ticking" sound to this update.
- User Interaction & Auto-Spin
    - Concept: UI State Management.
    - Task: Implement "Hold to Auto-Spin". This requires a mini-state machine in the UI (Idle, Pressed, Auto-Mode).
- Part 3 Review - The "Standard Game"
    - Deliverable: The game now sounds and looks like a casino game. Coins fly, numbers tick up, and the reels stop with a satisfying "Clack" sound.

Part 4: Architecture, Bundles & Optimization

Goal: Prepare for a real store release. Slots are heavy on assets (hundreds of textures), so efficient loading is critical.

- Asset Bundles (Critical for Slots)
    - Scenario: A casino app has 50 different slot games. You cannot load all of them at launch.

    - Task: Move all assets (Sounds, Textures, Prefabs) for this specific slot into a Bundle named "Slot_Pharaoh".
    - Code: assetManager.loadBundle('Slot_Pharaoh', ...) only when the user selects this game from the lobby.
- Texture Atlases & Batching
    - Problem: 5 reels x 3 rows = 15 symbols. If they are separate images, that's 15 draw calls.
    - Solution: Use Auto Atlas.
    - Task: Pack all symbols into one sheet. Ensure the DrawCall count in the profiler drops to near 1 (for the reels).
- Object Pooling (Advanced)
    - Context: Particle effects (coins) and symbols.
    - Task: Never destroy a symbol. When it scrolls off-screen, recycle it. Create a CoinPool for the big win animations.
- Server Integration (Simulation)
    - Concept: Determinism. The Client is a "Puppet."
    - Task: Mock a server response: { result: [1,1,1], winAmount: 500 }.
    - Flow: Button Click -> Send Request -> Wait -> Receive Result -> Start Visual Spin -> Stop at Result. (Don't start spinning until you know the result, or use a "dummy spin" phase while waiting).
- Handling Network Latency
    - Task: What if the server takes 5 seconds to reply?
    - Implementation: Create an "Infinite Spin" state that loops the spinning animation until the data arrives, then transitions to the "Stop Sequence."
- Mobile Optimization (Battery & Heat)
    - Task: Cap the frame rate (game.frameRate = 30 or 60).
    - Logic: If the user is idle for 10 seconds, drop FPS to 30 to save their battery. Wake up to 60 FPS on touch.
- Shaders for "Big Win"
    - Concept: Make the winning symbols glow or flash.
    - Task: Apply a custom "Shine" shader (UV sliding effect) to the sprite only when a win occurs.
- Android/iOS Deployment
    - Task: Handle Device Orientation (Force Landscape).
    - Task: Handle "Notch" safe areas specifically for the Spin Button (don't let the home bar overlap the spin button).
- Final Architecture Review
    - Checklist:
        - Are assets unloaded when returning to the lobby?
        - Is the Logic (Math) separated from the View (Nodes)?
        - Is the memory usage stable after 100 auto-spins?
        - How much is the amount of transferred data for the first load for a web platform?