"Shadow Swap"

A platformer where you switch between a character and their shadow to solve puzzles.

Core Mechanics:

1. Dual Characters: You control a normal character (e.g., a knight, robot, or adventurer) and their shadow.
2. The "real" character moves normally on solid platforms.
3. The "shadow" can only move on dark/shadowy areas (inverted platforms).

Switching Mechanic:

1. Press a key (e.g., Space) to switch control between the character and their shadow.
2. When you're controlling one, the other stays idle.
3. Some obstacles require coordinating both (e.g., stepping on switches, blocking enemies).

Environment Interaction:

1. Light sources (torches, lamps) can create new shadows or remove them.
2. Shadows dynamically change based on light positions (simple line-based shadows).

Goal: Reach the end of each level by using both characters wisely.

Example Level Design:

* A gap exists where the real character can’t pass, but their shadow can walk on a shadowy wall.
* A moving light source makes shadows shift, requiring timing to progress.

**Shadow Swap - 2-Week Agile Development Plan**

**Sprint Overview (7 Sprints × 2 Days Each)**

**Sprint 1: Foundation & Main Character (Days 1-2)**

**Goal**: Establish core Phaser setup and main character movement

**Deliverables:**

* ✅ Basic Phaser 3 project structure
* ✅ Main character sprite with animations (idle, walk, jump)
* ✅ Keyboard input handling (WASD/Arrow keys)
* ✅ Basic gravity and jump mechanics
* ✅ Simple test level with colored rectangles as platforms

**Technical Tasks:**

* Initialize Phaser scene with physics system
* Create Player class with sprite management
* Implement movement state machine (idle, moving, jumping, falling)
* Add basic collision detection with static platforms
* Set up asset loading system

**Acceptance Criteria:**

* Character moves smoothly left/right
* Character jumps and falls realistically
* Character stops on platforms
* Basic level layout visible

**Sprint 2: Shadow Character & Switching (Days 3-4)**

**Goal**: Add shadow character and core switching mechanic

**Deliverables:**

* ✅ Shadow character sprite (darker/translucent version)
* ✅ Switch control system (Space key toggle)
* ✅ Visual indicators for active/inactive characters
* ✅ Shadow follows main character position initially

**Technical Tasks:**

* Create Shadow class inheriting from Player
* Implement character switching logic
* Add visual feedback (outline, transparency, particle effects)
* Create CharacterManager to handle dual character state
* Add UI indicators for current active character

**Acceptance Criteria:**

* Press Space to switch between characters
* Only active character responds to input
* Clear visual distinction between active/inactive
* Shadow mirrors main character's position when not controlled

**Sprint 3: Platform Collision System (Days 5-6)**

**Goal**: Robust collision detection and platform interaction

**Deliverables:**

* ✅ Solid platforms for main character
* ✅ Improved physics with proper collision bounds
* ✅ Multiple platform types (ground, floating platforms)
* ✅ Basic level geometry expanded

**Technical Tasks:**

* Refine collision detection with proper hitboxes
* Create Platform class with different collision properties
* Implement one-way platforms (can jump through from below)
* Add collision layers for different character types
* Optimize collision performance

**Acceptance Criteria:**

* Character stands solidly on all platforms
* Can jump through one-way platforms from below
* No glitching through platform edges
* Smooth character movement on slopes/edges

**Sprint 4: Shadow Collision Areas (Days 7-8)**

**Goal**: Shadow-specific movement zones and restrictions

**Deliverables:**

* ✅ Shadow-only movement areas (dark zones)
* ✅ Shadow cannot move in "light" areas
* ✅ Visual distinction between shadow and light zones
* ✅ First puzzle requiring both characters

**Technical Tasks:**

* Create ShadowZone class for shadow-walkable areas
* Implement zone-based collision for shadow character
* Add visual styling for shadow zones (darker backgrounds, patterns)
* Create simple puzzle requiring character coordination
* Prevent shadow movement outside designated areas

**Acceptance Criteria:**

* Shadow can only move in designated dark areas
* Main character cannot access shadow-only zones
* Visual feedback when attempting invalid moves
* Complete first puzzle using both characters

**Sprint 5: Light Sources & Dynamic Shadows (Days 9-10)**

**Goal**: Interactive light sources that modify shadow zones

**Deliverables:**

* ✅ Torch/lamp objects that emit light
* ✅ Dynamic shadow zone generation based on light
* ✅ Moving light sources (pendulum torches)
* ✅ Timing-based shadow puzzles

**Technical Tasks:**

* Create LightSource class with radius and intensity
* Implement basic shadow casting algorithm (raycasting/geometric)
* Update shadow zones in real-time based on light positions
* Add animated light sources with movement patterns
* Create puzzle requiring timing with moving shadows

**Acceptance Criteria:**

* Light sources visibly affect shadow areas
* Shadow zones update as lights move
* Puzzles require timing shadow movement with light cycles
* Performance remains smooth with multiple light sources

**Sprint 6: Interactive Elements (Days 11-12)**

**Goal**: Switches, pressure plates, and coordination puzzles

**Deliverables:**

* ✅ Pressure switches activated by characters
* ✅ Doors/barriers controlled by switches
* ✅ Puzzles requiring both characters simultaneously
* ✅ Character-specific interactive elements

**Technical Tasks:**

* Create Switch and Door classes with trigger systems
* Implement character detection on pressure plates
* Add puzzle logic requiring coordination
* Create some switches only shadow can activate
* Add visual feedback for switch states

**Acceptance Criteria:**

* Switches activate when character steps on them
* Doors open/close based on switch states
* Some puzzles require both characters on switches simultaneously
* Clear visual feedback for all interactive states

**Sprint 7: Level Design & Polish (Days 13-14)**

**Goal**: Complete level, UI, and gameplay polish

**Deliverables:**

* ✅ Complete tutorial level showcasing all mechanics
* ✅ Game UI (health, level progress, instructions)
* ✅ Sound effects and background music
* ✅ Level completion/restart system
* ✅ Basic menu system

**Technical Tasks:**

* Design comprehensive level using all implemented mechanics
* Add audio system with sound effects
* Create UI overlays for game information
* Implement level completion detection
* Add restart/menu functionality
* Performance optimization and bug fixes

**Acceptance Criteria:**

* Complete playable level from start to finish
* All mechanics work together cohesively
* Audio feedback enhances gameplay
* Smooth performance on target platforms
* Intuitive UI and controls

**Daily Development Process:**

**Day Structure:**

* **Morning**: Planning & architecture decisions
* **Afternoon**: Implementation & testing
* **End of Day**: Commit working build, demo functionality

**Testing Protocol:**

* Each sprint must produce a playable build
* Test all previous functionality still works
* Document any discovered issues for next sprint

**Risk Mitigation:**

* Keep scope flexible - features can be simplified if needed
* Prioritize core mechanics over visual polish early
* Maintain working build at all times

**Development Rules**

**You are a Game Architect / Senior Game Developer**

1. If you think you need more context or if I have missed something out from the documentation while giving you instructions or if the documentation and already existing system conflicts with what I am asking you to implement, ask me clarifying questions to produce a better output
2. Follow SOLID Principles
3. Data/Configuration should always be in a separate file, there should be no hardcoded values inside scripts. That way I can change the configuration at any given moment. Always give me complete rewrite of data file
4. Each script should have a single responsibility
5. Use Design Patterns to create systems
6. Create Managers to handle similar types of things
7. If there is CSS to be created each css should contain selectors related to that functionality. For example if we are creating css for player then the file should be named player.css and every selector should be chained using .player .[next selector]
8. If a script needs to add new functions, give me script name along with new functions, if the new functions need to be called from an existing function from the same script, give me the full updated existing function
9. If a script needs to modify either 1 or two functions at max, then give me the script name along with new updated complete functions, if there are more than 2 function udpates, give me complete rewrite of script
10. Always give me complete rewrite of css