Reflected Path

1. Game Concept and Genre

- **Concept:** "Reflected Path" is a puzzle game designed to test and challenge the player's visual memory and spatial reasoning. Players are presented with a geometric path illuminated briefly on one side of a symmetrical grid. Their objective is to accurately recreate the mirror image (reflection) of this path on the opposite side of the grid, adhering to specific constraints like time, available "ink" (tile placements), and a limit on mistakes.
- **Genre:** Puzzle / Logic / Memory Game
- **Theme:** Abstract / Minimalist / Geometric

2. Target Audience

- Players who enjoy puzzle games focusing on memory, pattern recognition, and spatial logic.
- Individuals seeking a mentally stimulating yet accessible gameplay experience suitable for short sessions.
- Players who appreciate minimalist aesthetics and clear, understandable mechanics.
- Casual to mid-core puzzle enthusiasts.

3. Core Gameplay Mechanics and Rules

- **Path Presentation:** At the start of each level, the "original path" on the left grid half is revealed tile by tile sequentially, accompanied by a sound cue.
- **Memorization:** The player must memorize the shape and position of this revealed path.
- **Reflection & Drawing:** After the original path fades (with a color transition animation), the player uses the left mouse button to click on empty tiles on the *right* half of the grid, attempting to draw the exact geometric reflection of the memorized path across the central vertical symmetry line.
- **Immediate Feedback:** Clicking a tile provides instant visual and audio feedback:
    - **Correct Tile:** Turns green (with color transition animation) and plays a "correct" sound.
    - **Incorrect Tile:** Turns red (with color transition animation), plays an "incorrect" sound, and increments the mistake counter.

- **Constraints:**
    - **Time Limit:** Each level must be completed within a specific time limit (default or level-specific). A warning sound plays repeatedly during the last 10 seconds. Running out of time results in Game Over.
    - **Ink Limit:** The player can only place a limited number of tiles per level (default or level-specific). Running out of ink before completing the path results in Game Over.
    - **Mistake Limit:** Placing an incorrect tile counts as a mistake. Exceeding the mistake limit (default or level-specific) results in Game Over.
- **Level Completion:** A level is successfully completed when all tiles corresponding to the correct reflection have been placed (turned green) without exceeding any limits.
- **Game Progression:** Upon level completion, there is a brief transition period before the next level starts automatically. The game currently features 25 levels of increasing complexity.
- **Game Completion/Over:** Completing all levels successfully ends the game with a congratulatory message. Failing due to time, ink, or mistakes ends the game with a corresponding message. In either end state, the player can click or press a key to restart from Level 1.

4. Graphical Style and Key Visual Elements

- **Style:** The game employs a minimalist, clean, and functional 2D graphical style. The focus is on clarity and minimizing visual clutter to aid player focus on the puzzle itself.
- **Color Palette:** A dark, low-saturation background (dark bluish-grey) provides contrast for the brighter, more saturated colors used for interactive elements. Empty tiles are a slightly lighter shade than the background. Key colors include:
    - Light Blue (`COLOR_TILE_ORIGINAL_PATH`): For the path to be memorized.
    - Green (`COLOR_TILE_CORRECT`): For correctly placed reflection tiles.
    - Red (`COLOR_TILE_INCORRECT`): For incorrectly placed tiles.
    - White (`COLOR_SYMMETRY_LINE`, `COLOR_UI_TEXT`): For the central line and UI text, ensuring high visibility.
- **Key Visual Elements:**
    - **Grid:** Clearly defined grid lines partition the play area.
    - **Tiles:** Square cells forming the grid. They change color based on their state, featuring a subtle border effect for slight depth and a smooth color transition animation when changing state.
    - **Symmetry Line:** A prominent, thick white vertical line dividing the grid, clearly indicating the axis of reflection.

    - **UI Display:** Simple text elements at the top of the screen displaying remaining Time, Ink, and Mistake Allowance.
    - **Messages:** Centered text overlays with a semi-transparent background are used for game state notifications (Level Complete, Game Over).
    - **Background Image:** A static background image provides subtle visual interest without distracting from the gameplay elements.
    - **Animations:**
        - Tile color transitions (fade between states).
        - Sequential reveal of the original path tiles.

5. Sound Effects

- **Sound Effects:** Used to provide immediate auditory feedback for player actions and game events:
    - `SOUND_CLICK`: Played when a tile is clicked (and ink is consumed).
    - `SOUND_CORRECT`: Played when a correct tile is placed.
    - `SOUND_INCORRECT`: Played when an incorrect tile is placed.
    - `SOUND_LEVEL_COMPLETE`: Played upon successful level completion.
    - `SOUND_GAME_OVER`: Played when the player fails due to time, ink, or mistakes.
    - `SOUND_PATH_SHOW`: Played for each tile revealed in the original path sequence.
    - `SOUND_TIMER_LOW`: A repeating warning sound played during the final seconds of the timer.

6. Control Scheme

- **Primary Control:** Mouse
    - **Left Mouse Button Click:** Used exclusively to select and place tiles on the right (player) side of the grid during the drawing phase. Also used to restart the game from Game Over / Game Complete screens.
- **Secondary Control:** Keyboard
    - **Any Key Press:** Can also be used to restart the game from Game Over / Game Complete screens.

7. Code Structure and Technical Challenges

- **Code Structure:** The project utilizes a modular structure within a `src/` directory to promote organization and maintainability:
    - `config.py`: Centralizes all constants, settings (colors, timings, limits), and initial Pygame subsystem configurations (like font loading).

    - `tile.py`: Defines the `Tile` class, encapsulating the state, drawing logic, and animation behavior of individual grid cells.
    - `level_data.py`: Isolates level path definitions and level-specific setting overrides (`LEVELS`, `LEVEL_SETTINGS`).
    - `game.py`: Contains the main `Game` class, which manages the game loop, state transitions, event handling, updates, rendering orchestration, and holds the primary game state variables (grid, timers, counters, etc.).
    - `main.py`: Serves as the simple entry point, responsible only for importing the `Game` class and initiating its `run()` method. It also handles `sys.path` modification to ensure modules within `src/` can be imported correctly.
    - `assets/`: Contains subdirectories for images and sounds, keeping media files separate from code.
- **Technical Challenges & Solutions:**
    - **Pygame Initialization Order:** Encountered errors ("No video mode set", "cannot convert") related to asset loading (images, fonts) before the display mode was initialized. Resolved by strictly enforcing the initialization order within `Game.__init__`: `pygame.init()` -> `pygame.display.set_mode()` -> initialize other subsystems (mixer, font) -> load assets (`.convert()`/`.convert_alpha()` called *after* display init).
    - **Tile Animation (** The `pygame.Color.lerp()` function occasionally threw a `ValueError` because floating-point inaccuracies could cause the interpolation factor to slightly exceed the expected [0.0, 1.0] range. Solved by explicitly clamping the progress value using `max(0.0, min(1.0, ...))` before passing it to `lerp()`.
    - **State Management Complexity:** Handling transitions between multiple game states (showing path, drawing, level transition, various game over states) required careful conditional logic within the `Game._update` method to ensure actions (like playing sounds or starting timers) occur only once at the correct moment. Using distinct state constants (strings) and transition flags/timers helped manage this.
    - **Modular Imports:** Ensuring modules within the `src/` directory could import each other correctly required modifying `sys.path` in `main.py` or using relative imports carefully. The `sys.path` modification approach was chosen for simplicity in this structure.