WIGGLE ESCAPE - GAME DOCUMENTATION

1. PROJECT OVERVIEW

Project Name:	Wiggle Escape
Engine:	Unity (Universal Render Pipeline 2D)
Platform:	Mobile (Android/iOS) + Desktop
Genre:	Puzzle Game
Language:	C#
Test Phone:	Samsung A36

Wiggle Escape is a sliding puzzle game where players must guide colored snakes off a grid by tapping them to move in their designated exit direction. The challenge lies in determining the correct order to move snakes, as they can block each other's paths.

2. GAME CONCEPT & RULES

OBJECTIVE

Clear all snakes from the grid by tapping them in the correct order.

CORE MECHANICS
----
1. SNAKE MOVEMENT:
   - Each snake has a fixed exit direction (shown by arrow on head)
   - Tapping a snake makes it move continuously in its exit direction
   - Snakes stop if they hit another snake or grid boundary
   - Snakes exit when they reach the edge in their exit direction

2. COLLISION RULES:
   - Snakes cannot pass through other snakes
   - If a snake hits an obstacle, it returns to its starting position
   - Each blocked move counts as a MISTAKE

3. LIVES SYSTEM:
   - Players start with 3 lives per level
   - Each mistake costs 1 life
   - Losing all lives = Game Over
   - Lives reset when restarting or moving to next level

4. VICTORY CONDITION:
   - All snakes successfully exit the grid
   - Level is marked as completed
   - Player can proceed to next level

STRATEGIC ELEMENTS

- Players must analyze snake positions and exit directions
- Some snakes block others - order matters
- Use the HINT feature to highlight an escapeable snake
- Use EXIT PATHS to visualize snake trajectories

3. PROJECT STRUCTURE

```
Assets/
├── Editor/
│   └── LevelGeneratorEditor.cs    # Custom inspector for level generation
│
├── Scripts/
│   ├── Core/
│   │   ├── GameManager.cs         # Main game logic controller
│   │   ├── GridManager.cs         # Grid system and cell management
│   │   ├── Snake.cs               # Snake behavior and rendering
│   │   ├── CameraController.cs    # Camera zoom and pan controls
│   │   └── ProgressManager.cs     # Save/load game progress
│   │
│   ├── Input/
│   │   └── InputController.cs     # Touch/mouse input handling
│   │
│   ├── Level/
│   │   ├── LevelData.cs           # ScriptableObject for level data
│   │   └── LevelGenerator.cs      # Procedural level generation
│   │
│   ├── UI/
│   │   ├── GameUI.cs              # In-game UI management
│   │   └── LevelSelectionUI.cs    # Level selection screen
│   │
│   └── Effects/
│       └── ConfettiManager.cs     # Victory celebration effects
│
├── ScriptableObjects/
│   └── Levels/
│       ├── lvl1.asset             # Pre-made level 1
│       ├── lvl2.asset             # Pre-made level 2
│       └── ... (lvl3-10)          # Additional levels
│
├── Scenes/
│   └── GameScene.unity            # Main game scene
│
├── Settings/
│   ├── UniversalRP.asset          # URP settings
│   └── Renderer2D.asset        # 2D renderer configuration
│
└── Plugins/
    └── Demigiant/DOTween/       # Animation library
```

4. ARCHITECTURE OVERVIEW

DESIGN PATTERNS USED
----
1. SINGLETON PATTERN:
   - GameManager, GridManager, GameUI, LevelSelectionUI
   - ProgressManager, ConfettiManager
   - Provides global access to core systems

2. COMPONENT-BASED:
   - Snake components attached dynamically to GameObjects
   - Modular UI elements

3. SCRIPTABLEOBJECT DATA:
   - LevelData stores level configurations
   - Separates data from logic

DATA FLOW

```
┌─────────────────┐
│ LevelSelectionUI│ ──Select Level──> ┌─────────────┐
└─────────────────┘                   │ GameManager │
                      └──────┬──────┘
                        │

┌──────────────────────┼──────────────────────┐
        ▼              ▼              ▼
     ┌──────────────┐   ┌──────────────┐
┌─────────────┐
     │ GridManager  │   │  Snake(s)    │   │  GameUI    │
     │ (Grid State) │   │ (Game Objects) │   │ (Display)  │
     └──────────────┘   └──────────────┘
└─────────────┘
        ▲              ▲
        │              │
     ┌──────────────┐   ┌──────────────┐
     │InputController│───────│ Player Input  │
     └──────────────┘   └──────────────┘
```

5. CORE SYSTEMS

5.1 GAMEMANAGER

Location: Assets/Scripts/Core/GameManager.cs

PURPOSE:
Central controller managing game state, level loading, and win/lose conditions.

KEY PROPERTIES:
```
- levels: List<LevelData>     # All available levels
- maxMistakes: int            # Lives per level (default: 3)
- snakePrefab: GameObject     # Template for spawning snakes
- snakeContainer: Transform   # Parent object for snake instances
```

- RestartLevel()
  Resets current level to initial state.

- LoadNextLevel()
  Advances to the next level in sequence.

- ReturnToLevelSelection()
  Saves progress and returns to menu.

- OnSnakeExited()
  Called when a snake successfully exits. Checks for level completion.

- OnMistake()
  Called when player makes a wrong move. Decrements lives.

- ToggleHint()
  Highlights a snake that can currently escape.

- ToggleExitPaths()
  Shows/hides exit direction lines for all snakes.

- GetActiveSnakes()
  Returns list of current snakes in play.

GAME STATES:
- Level Selection → Level Playing → Level Complete/Game Over → Level Selection

----

5.2 GRIDMANAGER

Location: Assets/Scripts/Core/GridManager.cs

PURPOSE:
Manages the game grid, cell occupancy, and coordinate conversion.

KEY PROPERTIES:
```
- gridWidth: int              # Horizontal cell count
- gridHeight: int             # Vertical cell count
- cellSize: float             # World units per cell (default: 1)
- gridColor: Color            # Background cell color
```

- GridToWorldPosition(Vector2Int gridPos) → Vector3
  Converts grid coordinates to world space.

- WorldToGridPosition(Vector3 worldPos) → Vector2Int
  Converts world coordinates to grid space.

- IsWithinBounds(Vector2Int gridPos) → bool
  Checks if position is inside grid.

- IsCellOccupied(Vector2Int gridPos) → bool
  Checks if a snake occupies the cell.

- SetCellOccupancy(Vector2Int gridPos, Snake snake)
  Marks cell as occupied/empty.

- GetSnakeAtCell(Vector2Int gridPos) → Snake
  Returns snake at specified cell.

- IsExitCell(Vector2Int gridPos) → bool
  Checks if position is outside grid (exit point).

- ClearOccupancy()
  Clears all occupancy data.

COORDINATE SYSTEM:
- Grid origin (0,0) is bottom-left
- X increases to the right
- Y increases upward
- World position is centered on grid

5.3 SNAKE SYSTEM

Location: Assets/Scripts/Core/Snake.cs

PURPOSE:
Controls individual snake behavior, movement, and visual representation.

KEY PROPERTIES:
```
- snakeColor: Color           # Snake's display color
- exitDirection: Vector2Int   # Direction snake moves (up/down/left/right)
- moveSpeed: float            # Movement speed (default: 12)
```

- CanMove() → bool
  Checks if snake can move (not blocked immediately).

- MoveSnake() → IEnumerator
  Coroutine that animates snake movement until blocked or exited.

- GetSegments() → List<Vector2Int>
  Returns current segment grid positions.

- SetHighlight(bool highlight)
  Shows/hides hint glow effect.

- SetExitPathVisible(bool visible)
  Shows/hides exit direction line.

- ForceStopAnimations()
  Immediately stops all animations.

- ForceCleanup()
  Clears snake from grid occupancy.

STATE PROPERTIES:
```
- IsMoving: bool              # Currently animating
- HasExited: bool             # Successfully left grid
- SegmentCount: int           # Number of body segments
- LastMoveWasInvalid: bool    # Hit obstacle during last move
```

VISUAL COMPONENTS:
- Head: Larger, brighter segment with direction arrow
- Body: Series of pill-shaped segments
- Connectors: Fill gaps between segments for smooth appearance
- Highlight Glow: Pulsing yellow effect for hints
- Exit Path Line: Shows trajectory to grid edge

MOVEMENT ALGORITHM:
1. Check if path is clear (CanMove)
2. If clear, start moving in exit direction
3. Record path history as head moves
4. Body segments follow head's path
5. Continue until hitting obstacle or exiting
6. If blocked mid-move, animate return to start position

5.4 INPUT SYSTEM

Location: Assets/Scripts/Input/InputController.cs

PURPOSE:
Handles player input (touch/mouse) and triggers snake movement.

BEHAVIOR:
1. Detects tap/click on screen
2. Raycasts to find snake under tap point
3. Validates snake can move
4. Initiates movement or counts mistake

INPUT FLOW:

```
┌─────────────────┐    ┌─────────────────┐    ┌───────────────┐
│ Touch/Click  │ ──> │ Raycast Check │ ──> │ Snake Found │
└─────────────────┘    └─────────────────┘    └───────┬───────┘
                            │
┌───────────────────────────────┴─────────────────────────────
┐
          ▼                         ▼
     ┌──────────────────┐                  ┌──────────────────┐
     │ Can Move = TRUE │                  │ Can Move = FALSE│
     │ → Start Move    │                │ → Count Mistake │
     │ → Wait Complete │                 │ → Shake Snake   │
     │ → Check Result  │                 └──────────────────┘
     └──────────────────┘
```

6. LEVEL SYSTEM

6.1 LEVELDATA (ScriptableObject)

Location: Assets/Scripts/Level/LevelData.cs

PURPOSE:
Stores level configuration as a reusable asset.

DATA STRUCTURE:

```
┌─────────────────────────────────────┐
│ LevelData                           │
├─────────────────────────────────────┤
│ gridWidth: int        # Grid columns      │
│ gridHeight: int       # Grid rows         │
│ snakes: List<SnakeData> # All snakes      │
└─────────────────────────────────────┘
          │
          ▼
┌─────────────────────────────────────┐
│ SnakeData                           │
├─────────────────────────────────────┤
│ color: Color          # Display color  │
│ segments: List<Vector2Int> # Body positions │
│ exitDirection: Vector2Int  # Move direction │
└─────────────────────────────────────┘
```

CREATING MANUALLY:
1. Right-click in Project window
2. Create → Wiggle Escape → Level Data
3. Configure in Inspector

6.2 LEVELGENERATOR

Location: Assets/Scripts/Level/LevelGenerator.cs
Editor: Assets/Editor/LevelGeneratorEditor.cs

PURPOSE:
Procedurally generates solvable levels with guaranteed solutions.

CONFIGURATION:
- gridWidth/gridHeight: Grid dimensions
- snakeCount: Target number of snakes

- minLength/maxLength: Snake body size range
- curveChance: Probability of snake turning (0-1)
- snakeColors: Color palette for snakes

ALGORITHM OVERVIEW:
1. INITIALIZATION:
   - Create empty grid
   - Initialize random number generator with seed

2. SNAKE PLACEMENT LOOP:
   For each snake to place:
   a) Find random empty cell for head
   b) Choose random exit direction
   c) Build body using random walk
   d) Check for facing conflicts (early elimination)
   e) Temporarily place snake
   f) SIMULATE entire level to verify solvability
   g) If simulation passes, keep snake; otherwise, reject

3. SOLVABILITY SIMULATION:
   - Clone grid and snake states
   - Find snake with clear path to exit
   - Remove that snake from simulation
   - Repeat until all snakes removed OR deadlock
   - If all removed = SOLVABLE

4. FINALIZATION:
   - Log statistics
   - Run final verification

KEY METHODS:
- GenerateLevel(int seed) → List<SnakePlacement>
  Main entry point for generation.

- CreateCandidateSnake(Vector2Int headPos, int targetLength)
  Creates a potential snake using random walk.

- IsFacingAnotherSnake(Vector2Int headPos, Vector2Int exitDir)
  Early elimination check for facing conflicts.

- TryCommitSnake(SnakePlacement newSnake)
  Attempts to permanently place snake with simulation verification.

- SimulateSolvability(List<SnakePlacement> snakes)
  Verifies level is solvable by simulating all snakes escaping.

USING THE GENERATOR:
1. Add LevelGenerator component to a GameObject

2. Configure parameters in Inspector
3. Click "GENERATE LEVEL" button
4. Preview result in Inspector
5. Click "SAVE AS ASSET" to export

See: levelGeneratorInstructions.txt for detailed usage guide.

7. UI SYSTEM

7.1 GAMEUI

Location: Assets/Scripts/UI/GameUI.cs

PURPOSE:
Manages all in-game UI elements during gameplay.

COMPONENTS:

```
┌─────────────────────────────────────────────────────────────
──────────────┐
│ GAMEPLAY PANEL                                   │
├─────────────────────────────────────────────────────────────
──────────────┤
│ ┌──────────────────┐
┌────────────────────────────────────────────┐  │
│ │ Level Text      │  │ Life Indicators (●●●)           │   │
│ │ "Level 1"       │  │ Red = Active, Faded = Lost        │   │
│ └──────────────────┘
└────────────────────────────────────────────┘  │
│                                          │
│ ┌─────────┐ ┌─────────┐ ┌─────────┐ ┌─────────┐          │
│ │ Back   │ │Refresh │ │ Hint   │ │ Paths  │          │
│ │ Button │ │ Button │ │ Button │ │ Button │          │
│ └─────────┘ └─────────┘ └─────────┘ └─────────┘          │
└─────────────────────────────────────────────────────────────
──────────────┘
```

OVERLAY PANELS:

```
┌───────────────────────┐  ┌───────────────────────┐
│ LEVEL COMPLETE     │   │ GAME OVER        │
├───────────────────────┤  ├───────────────────────┤
│ "Level Complete!"  │   │ "Game Over!"      │
│                 │   │                 │
│ [Next Level]       │   │ [Restart]         │
│ [Restart]          │    │ [Back to Menu]     │
└───────────────────────────┘  └───────────────────────────┘
```

KEY METHODS:
- UpdateLivesUI(int remainingLives)
  Updates life indicators with animation for lost lives.

- UpdateLevelUI(int level)
  Sets level number display.

- ShowLevelComplete()
  Displays victory panel with confetti.

- ShowGameOver()
  Displays game over panel.

- ShowGameplayUI() / HideGameplayUI()
  Controls visibility of gameplay elements.

BUTTON HANDLERS:
- OnRestartClicked: Restarts current level
- OnNextLevelClicked: Loads next level
- OnRefreshClicked: Restarts level (in-game)
- OnHintClicked: Toggles hint highlight
- OnBackClicked: Returns to level selection
- OnShowPathsClicked: Toggles exit path visualization

7.2 LEVELSELECTIONUI

Location: Assets/Scripts/UI/LevelSelectionUI.cs

PURPOSE:
Displays level selection menu with progress indicators.

VISUAL DESIGN:

```
┌─────────────────────────────────────────────────────────────────────
────────────────┐
│ LEVEL SELECTION PANEL                              │
├─────────────────────────────────────────────────────────────────────
────────────────┤
│                                               │
│   ┌─────────┐  ┌─────────┐  ┌─────────┐  ┌─────────┐  ┌─────────┐        │
│   │Level │  │Level │  │Level │  │Level │  │Level │        │
│   │  1   │  │  2   │  │  3   │  │  4   │  │  5   │        │
│   │ (●)  │  │ (●)  │  │ (○)  │  │ (○)  │  │ (○)  │        │
│  └──────────┘  └──────────┘  └──────────┘  └──────────┘  └──────────┘        │
│  [GREEN]   [YELLOW]  [WHITE]   [WHITE]   [WHITE]         │
│  Completed  In Progress Not Started                 │
│                                    │
└──────────────────────────────────────────────────────────────────────────
──────────────────┘
```

BUTTON COLORS:
- GREEN: Level completed
- YELLOW: Level started but not finished (has saved progress)
- WHITE: Level not yet attempted

KEY METHODS:
- ShowLevelSelection()
  Displays the level menu and refreshes button states.

- HideLevelSelection()
  Hides the level menu.

- RefreshLevelButtons()
  Updates all button appearances based on progress.

8. CAMERA SYSTEM

Location: Assets/Scripts/Core/CameraController.cs

PURPOSE:
Controls camera zoom, pan, and automatic fitting to grid.

FEATURES:
1. AUTO-FIT: Camera automatically sizes to fit entire grid
2. PINCH ZOOM: Two-finger gesture for mobile zoom
3. PAN: Drag to move camera view
4. BOUNDS CLAMPING: Prevents camera from leaving grid area

CONTROLS:

```
┌──────────────────────────────────────────────────────────────────────────
───────────┐
│ MOBILE                  │ DESKTOP (Editor)          │
├──────────────────────────────────────────────────────────────────────────
───────────┤
│ Pinch: Zoom in/out          │ Scroll Wheel: Zoom        │
│ Drag: Pan camera            │ Middle Mouse Drag: Pan      │
│                   │ Right Mouse Drag: Pan       │
```

```
└──────────────────────────────────────────────────────────────────────
──────────┘
```

CONFIGURATION:
- minZoom: Minimum orthographic size (closest zoom)
- baseMaxZoom: Default maximum zoom
- zoomSpeed: Zoom sensitivity
- boundsPadding: Extra space around grid edges
- extraZoomOutMargin: How much beyond grid fit to allow

KEY METHODS:
- OnGridChanged()
  Call when loading new level to refit camera.

- ResetCamera()
  Recenters and refits camera to grid.

9. PROGRESS & SAVE SYSTEM

Location: Assets/Scripts/Core/ProgressManager.cs

PURPOSE:
Persists game progress using PlayerPrefs.

DATA STRUCTURES:

LevelProgress:

```
┌─────────────────────────────────────────────────┐
│ levelIndex: int                    │
│ hasStarted: bool                   │
│ isCompleted: bool                  │
│ mistakeCount: int                  │
│ snakeStates: List<SnakeState>      │
└─────────────────────────────────────────────────┘
```

SnakeState:

```
┌─────────────────────────────────────────────────┐
│ segments: List<Vector2Int>         │
│ hasExited: bool                    │
└─────────────────────────────────────────────────┘
```

STORAGE:
- Data serialized as JSON
- Stored in PlayerPrefs with key "LevelProgress_[index]"
- Persists across sessions

KEY METHODS:
- SaveLevelProgress(int levelIndex, int mistakes, List<Snake> snakes)
  Saves current level state.

- GetLevelProgress(int levelIndex) → LevelProgress
  Retrieves saved progress for a level.

- MarkLevelCompleted(int levelIndex)
  Flags level as completed.

- HasSavedProgress(int levelIndex) → bool
  Checks if level has resumable progress.

- ClearLevelProgress(int levelIndex)
  Deletes saved data for one level.

- ClearAllProgress()
  Wipes all saved data.

10. VISUAL EFFECTS

Location: Assets/Scripts/Effects/ConfettiManager.cs

PURPOSE:
Creates celebratory confetti burst on level completion.

CONFIGURATION:
- confettiCount: Number of particles (default: 50)
- burstDuration: Animation length (default: 2 seconds)
- spreadRadius: How far particles spread
- confettiSize: Base particle size
- confettiColors: Color palette array

ANIMATION:
1. Particles spawn at screen center
2. Burst upward and outward
3. Fall with gravity simulation
4. Rotate continuously
5. Shrink and fade at end
6. Auto-destroy after animation

USAGE:
Called automatically by GameUI.ShowLevelComplete()
Can also call: ConfettiManager.Instance.PlayConfetti()

11. THIRD-PARTY DEPENDENCIES

DOTWEEN (Demigiant)

Location: Assets/Plugins/Demigiant/DOTween

PURPOSE:
High-performance animation library used for:
- UI animations (panels, buttons)
- Snake movement and effects
- Life indicator animations
- Confetti particle movement

COMMON USAGE:
- transform.DOMove() - Position animation
- transform.DOScale() - Scale animation
- transform.DOShakePosition() - Shake effect
- image.DOColor() - Color transitions
- DOTween.Sequence() - Chained animations

DOCUMENTATION: http://dotween.demigiant.com/

TEXTMESH PRO (Unity)

Location: Assets/TextMesh Pro

PURPOSE:
Advanced text rendering for UI elements.

12. SCENE SETUP GUIDE

REQUIRED GAMEOBJECTS

```
1. MANAGERS (Empty GameObjects)
   ├── GameManager
   │   └── Component: GameManager.cs
   │       - Assign levels list
   │       - Set maxMistakes
   │       - Assign snakeContainer
   │
   ├── GridManager
   │   └── Component: GridManager.cs
```

```
│
├── InputController
│    └── Component: InputController.cs
│
├── ProgressManager
│    └── Component: ProgressManager.cs
│
└── ConfettiManager
   └── Component: ConfettiManager.cs

2. CAMERA
└── Main Camera
   └── Components:
      - Camera (Orthographic)
      - CameraController.cs

3. UI CANVAS
└── Canvas (Screen Space - Overlay)
   ├── GameplayPanel
   │    ├── LevelText (TextMeshProUGUI)
   │    ├── LifeIndicatorContainer (Horizontal Layout)
   │    ├── BackButton
   │    ├── RefreshButton
   │    ├── HintButton
   │    └── ShowPathsButton
   │
   ├── LevelCompletePanel (Initially inactive)
   │    ├── Title Text
   │    ├── NextLevelButton
   │    └── RestartButton
   │
   ├── GameOverPanel (Initially inactive)
   │    ├── Title Text
   │    └── RestartButton
   │
   └── LevelSelectionPanel
      └── LevelButtonContainer (Grid Layout)

   UI Controllers:
   - GameUI.cs (on Canvas or child)
   - LevelSelectionUI.cs (on Canvas or child)

4. CONTAINERS
└── SnakeContainer (Empty GameObject)
   - Assigned to GameManager.snakeContainer
```

INSPECTOR SETUP CHECKLIST

□ GameManager.levels populated with LevelData assets
□ GameManager.snakeContainer assigned
□ GameUI references assigned (panels, buttons, containers)
□ LevelSelectionUI.levelButtonContainer assigned
□ Camera has CameraController component
□ All singleton managers present in scene

13. CREATING NEW LEVELS

METHOD 1: Using Level Generator (Recommended)
----
1. Create empty GameObject, add LevelGenerator component
2. Configure grid size and snake parameters
3. Click "GENERATE LEVEL"
4. Click "SAVE AS ASSET"
5. Add saved asset to GameManager.levels list

METHOD 2: Manual Creation
----
1. Right-click in Project → Create → Wiggle Escape → Level Data
2. Set gridWidth and gridHeight
3. Add snakes to the snakes list:
   - Set color
   - Add segment positions (head first, then body)
   - Set exitDirection
4. Add to GameManager.levels list

SNAKE SEGMENT ORDERING
----
segments[0] = Head position
segments[1] = First body segment (adjacent to head)
segments[n] = Last body segment (tail)

EXIT DIRECTIONS

```
Vector2Int.up    = (0, 1)   = Snake exits upward
Vector2Int.down  = (0, -1)  = Snake exits downward
Vector2Int.left  = (-1, 0)  = Snake exits left
Vector2Int.right = (1, 0)   = Snake exits right
```

DESIGN TIPS

- Ensure level is solvable (use generator for guarantee)
- Avoid snakes facing each other on same axis
- Vary snake lengths for interesting puzzles
- Consider "key" snakes that must exit first

14. KEYBOARD SHORTCUTS

```
IN-GAME SHORTCUTS (Desktop/Editor)
----------------------------------
R          - Restart current level
N          - Skip to next level
Escape     - Return to level selection

CAMERA CONTROLS (Desktop/Editor)
--------------------------------
Scroll     - Zoom in/out
Middle Mouse Drag - Pan camera
Right Mouse Drag  - Pan camera
```