我要打造一個網頁版的小遊戲。 這次只做純前端，除了必要的 HTML, CSS 之外，動態網頁的部分規定使用 React.ts (i.e. TypeScript 版本的 React)，而不是 pure JavaScript or React.js。 以下是我完整的遊戲設計，請完整協助我實作：

1. 概念概要（Elevator pitch）

玩家在一個格子（grid）地圖上控制「多個可動方塊」，每一次輸入（上/下/左/右）時，**所有可動方塊同時**朝指定方向滑動，直到撞到牆、固定障礙物或已停下的方塊才會停下。目標是在限定回合（move limit）內，讓「整張地圖的每一格（every cell）」至少被任一方塊停過一次（稱作 coverage / 覆蓋）。遊戲強調「規劃性」、「同時操作的組合邏輯」與「最短步數的解法競賽」。

2. 核心規則

1. 地圖由 M×N 格子組成（可為方形或長方形）。
2. 地圖上有三類基本物件：

- 固定障礙（immutable obstacle）：不可穿越或站立（表示牆或岩石）。
- 可動方塊（movable block）：玩家無法單獨選一個指令；每次指令會讓 **所有**可動方塊同方向滑動。每個格子最多有一個可動方塊（不可重疊）。
- 空格（walkable cell）：需被至少一個可動方塊停過一次以計為已覆蓋。

3. 起始時可動方塊位於某些空格；這些起始格自動視為「已覆蓋」。
4. 每一次玩家按方向鍵（或滑動）視為一個回合（move）。遊戲在每次移動後更新所有方塊位置，再標註停下來的格子為已覆蓋。
5. 一個關卡的完成條件：在不超過設計的回合數（turn limit）內，所有可覆蓋格子皆已被訪問（覆蓋率 = 100%）。若回合耗盡仍未完成則關卡失敗。
6. 方塊不可穿越固定障礙或其他停下的方塊（阻擋）。每格一次只能容納一個方塊。
7. 不加入合併或消失的規則（與 2048 類型不同）；方塊永不消失，數量固定（除特別關卡機制外）。

3. 移動與碰撞：確定的模擬規則（重要）

為使行為 deterministic 且容易實作／測試，移動採**方向性排序處理**（standard approach）：

- 移動方向為 Left：將所有可動方塊按照「欄位座標由小到大（leftmost → rightmost）」的順序處理（ascending column）。
- 方向 Right：由大到小（rightmost → leftmost）。
- 方向 Up：由小到大（topmost → bottommost）。
- 方向 Down：由大到小（bottommost → topmost）。

演算步驟（每回合）：

1. 建立障礙集合 static_obstacles = walls ∪ fixed_obstacles。
2. 依照上面方向性排序得到處理順序。
3. 對於序列中的每個方塊 b：

- 從 b 起點出發，朝該方向前進，直到遇到 static_obstacles 或 **已經在本次移動中停下來的其他方塊**（也視為障礙）。把 b 放在它能到達的最遠空格（即緊靠障礙前一格），更新該方塊位置與本回合停下來的障礙集合。

4. 所有方塊處理完畢後，將新停下的格子標為「已覆蓋」。

備註與規則細節：

- 若方塊本次移動無法移動（已被障礙緊貼），則它仍視為停在當前格，該格也會被標為已覆蓋。
- 不允許兩個方塊在同一回合內計算結果下落到相同格子，因排序已避免此情況；若發生設計錯誤（理論上不會），應視為 invalid state。
- 初始位置即被計入覆蓋。

這個處理順序的好處：簡單、可預測、便於測試與平衡；也是大多數滑動 puzzle 的標準做法（例如 Ricochet Robots 類似解法在同方向多機器人時會選擇特定處理順序）。

4. 障礙物與特殊格子（種類、參數、行為）

基礎障礙（必備）

- Wall（邊界牆）：地圖邊緣或固定牆，阻擋移動。
- Rock（石塊 / 固定障礙）：不能被破壞或移動，佔據格子。

進階障礙 / 特殊格（可選，作為關卡變化用）

- One-way tile（單向格）：方塊可朝指定方向通過，反向進入會在到達該格時強制停下。
- Sticky tile（黏著格）：方塊若停在上面，下次移動前會「被粘住」需再按一次同方向才會移動（或消耗 1 次回合而不移動）。
- Conveyor（輸送帶）：每次回合結束時把停在上面的方塊再額外朝帶子方向移動 1 格（可與滑動疊加設計）。
- Portal（傳送門）：方塊滑入後傳到配對門的出口，出口如果被佔用則阻擋進入。
- Toggle switch（開關）＋Dynamic obstacle：某些格為開關，當任一方塊停上時會切換地圖中某些障礙（打開/關閉）。用來做多階段解法。
- Temporal tile：被訪問後會變成障礙或消失（用以製造時序要求）。

每種特殊格都應有可選參數（例如 conveyor 速度、portal pairing、sticky delay 等）。

5. 關卡與難度分級（Concrete parameters）

為了方便製作與平衡，給出**標準化難度分級**（可作為 campaign 與自動生成參考）：

設計原則：難度由「格子數（N）」、「障礙數 / 密度（D）」、「可動方塊數（K）」、「回合限制（T）」共同決定。也要考慮關卡的解空間與分岔數（branching factor）。

建議分級（參考起點，可微調）：

1. **入門（Easy）**

- Grid：5×5 或 6×5（25–30 格）。
- 固定障礙數：3–6（障礙密度 ≈ 10–20%）。
- 可動方塊數：1–2。
- 回合限制（T）：10–14。
- 預期最短步數：6–10（設計為解法直觀、引導性強）。
- 特性：提供提示系統、無進階障礙。

2. **普通（Normal）**

- Grid：6×6（36）或 7×6（42）。
- 固定障礙數：6–12（密度 ~15–25%）。
- 可動方塊數：2–3。
- 回合限制：14–20。
- 預期最短步數：10–18。
- 特性：引入 1–2 種特殊格（例如 conveyor 或 portal）。

3. **困難（Hard）**

- Grid：8×8（64）或 9×7（63）。
- 固定障礙數：12–22（密度 ~18–35%）。
- 可動方塊數：3–5。
- 回合限制：18–28。
- 預期最短步數：16–30（多路徑、需要時序與分工）。
- 特性：動態障礙（switch）、時間敏感格子、需要利用阻擋來分割空間。

4. **專家 / 競賽（Expert / Competitive）**

- Grid：10×10（100）或更大。
- 固定障礙數：20–40（密度 20–40%）。
- 可動方塊數：4–8。
- 回合限制：25–45（視設計）。
- 預期最短步數：30+（解法稠密且容易失誤）。
- 特性：多種特殊格混合、最小化提示、解法唯一性或要求最優解證明。

提示：每個難度級別的「回合限制」應由關卡設計師用 solver（BFS/A*）驗證，使回合數足以讓最短解能通過但對玩家仍有挑戰（例如 star 評分 3 星要求 ≤ optimal + 2 moves，2 星 ≤ optimal + 6，否則失敗）。

6. 關卡設計原則（關卡設計師指南）

1. **可視化覆蓋目標**：玩家需清楚知道哪些格還沒被覆蓋（overlay：灰→彩或格子上顯示 tick）。
2. **分段探索**：使用障礙把地圖分成若干區域，需要玩家利用方塊阻擋/陷阱來「逐區域」覆蓋。
3. **利用相互干涉**：設計需要「一個方塊當擋板」讓另一個方塊在對面滑停，以達成特定格的覆蓋。這類互動是核心趣味來源。
4. **控制可動方塊數**：更多方塊帶來更高自由度但也更複雜。對比設計：少量方塊→偏解謎；大量方塊→偏管理/編排。
5. **建立學習曲線**：前幾關限定 1–2 種機制，逐步引入新障礙或互動（portal, conveyor, switch）。
6. **設計誘導路徑**：用障礙與 one-way tile 引導玩家先覆蓋某區，再覆蓋另一區，避免玩家「亂跑」耗盡回合。
7. **避免不可逆陷阱（除非刻意）**：若某動作會造成無法完成關卡，應在 UI 提示或允許「復原」以免玩家感到過度挫折。
8. **提供解法樣本供評估**：設計每關時先由設計師提供一組「理想解」，再用 solver 找最短步並作為 star threshold。

7. 評分、回合與獎勵系統

- **星級系統（3星制）**：
- 3★：使用的回合數 ≤ 最短步數（optimal）+ 1。
- 2★：≤ 最短步數 + 5。
- 1★：在限制回合 T 內完成（但 > 最短步數 + 5）。
- **分數計算（可選）**：
- 基礎分 = (地圖格數) × 10。
- 回合獎勵 = max(0, (T - used_moves)) × 20。
- 连续完美（多關）會有 combo multiplier。
- **成就 / 徽章**：
- 完成所有入門關卡 3★。
- 在 Expert 取得 1★。
- 在每日挑戰中排名前 10%。
- **貨幣與道具（可選）**：
- 用遊戲幣換提示（hint reveals a safe next move）、撤銷券（undo）、或解鎖裝飾皮膚。
- **排行榜**：以「最少回合」與「最快時間」為排序，區分全球與好友排行榜。

8. 遊戲模式（核心 + 擴充）

1. **關卡 Campaign**：依序解鎖、逐步引導。
2. **每日挑戰（Daily Puzzle）**：全平台同一題，排行榜比競技。
3. **無盡模式（Endless）**：隨機生成小關，盡可能長時間保持覆蓋效率。
4. **鬼（Ghost）比賽**：與自己的最佳解重播比速度/回合。
5. **合作模式（Co-op）變體**：兩名玩家分別控制不同子集的方塊（例如紅方塊由玩家A專屬），輪流發出方向或同步發指令（有趣的社交解謎）。
6. **解題挑戰（Design Challenge）**：玩家自己設計關卡並提交給社群，系統驗證 solvability 與難度後上榜。

9. UI / UX 規格

- **畫面元素**：
- 主畫面：地圖視窗、上/下/左/右輸入（鍵盤、滑動、虛擬按鍵）、目前回合計數/上限、覆蓋進度條（百分比與尚未覆蓋格數）。
- 小地圖/縮圖（可切換）：在大地圖時顯示未覆蓋區塊。
- 方塊高亮：每種方塊顏色與輪廓清晰區分。支持色盲模式（圖形符號 + 顏色）。
- 回合動畫：滑動動畫 150–300ms（可在設定降低或關閉），以保持視覺節奏。
- Undo / Reset / Hint 按鈕（Hint 有冷卻或消耗）。
- **反饋**：
- 當玩家達成覆蓋某重要區域時給短音效與小動畫（粒子）。
- 回合用盡時給出失敗畫面及可選「看廣告換一次復活」或「再試一次」。
- **控制**：
- 桌機：方向鍵、WASD。
- 手機：四向 swipe 或螢幕按鍵。
- 支援觸控長按以重複移動（選用）。

10. 可及性（Accessibility）

- 色盲模式（pattern + color）、高對比主題。
- 文字大小可調。
- 動畫可減少或關閉（reduce motion）。
- 旁白（screen reader）支援：場景朗讀（例如「左上障礙」）-- 對於視障玩家可以考慮語音導遊模式（但挑戰設計會較高）。
- 提示系統與漸進教學（逐步解鎖機制用 minimal cognitive load）。

11. 開發實作與演算法

狀態表示：

- 方塊位置：positions = [(r1,c1), (r2,c2), ...]（k 元素，順序穩定以方便 debug）。
- 已覆蓋格 bitmask：對於 N ≤ 64 可用 64-bit bitmask，N > 64 用長 bitset 或 byte array。
- 固定障礙集合 static_obstacles：hash set 或 bool grid。

移動函數（伪码）：

```
function slide_all(direction, positions, static_obstacles):
  order = sort_positions_by_direction(positions, direction)
  stopped_positions = set()
  new_positions = dict()
  for pos in order:
    cur = pos
    while True:
      nxt = cur + dir_step(direction)
      if nxt in static_obstacles or nxt in stopped_positions:
        # cannot move into nxt; so stop at cur
        new_positions[pos] = cur
        stopped_positions.add(cur)
        break
      cur = nxt
  return list_of_new_positions_in_original_order(new_positions)
```

索引與加速：

- 為加速滑動計算，可預先為每個格與方向記錄 next_blocking_cell（precompute sliding table），但需在 dynamic obstacles（例如 switch 切換）時更新。
- 若方塊數量 k 小，可每回合 O(k * distance) 模擬；如果地圖大、回合多，需優化。

求解器（level validator / auto-solver）：

- BFS 或 A* 適用於小格（state space = positions × visited_bitmask）。
- State = (sorted_positions_tuple, visited_bitmask)。
- Moves per state = 4（up/down/left/right）。

- Heuristics（A*）：剩餘未覆蓋格子數的一個下界；或將未覆蓋區域分為連通塊，估計「至少需要的移動次數」＝ number_of_components（因為每次移動至少可停一個區域）——要小心 admissible 性質。
- 為生成關卡，務必驗證 solvability 並取得 optimal moves。

測試：

- 用自動測試套件驗證：對每關，solver 能在 T 內找到解；並驗證 deterministic 行為（相同 seed、相同指令序列結果一致）。
- 邊界狀況：方塊緊貼、one-way tile 相鄰、portal 出口占用等。

12. 程式化關卡生成（Procedural generation）

1. 基本流程：

- 選定 grid size 和 k、D（障礙密度）。
- 隨機放置障礙物，但保證整張地圖對玩家來說「連通」（存在可通行路徑），以避免不可達格子。
- 隨機放置 k 個方塊（不重疊，且在不同區域以增加互動）。
- 呼叫 solver 驗證 solvability；若無法解或最短步數超過 T，丟棄或微調（移動一兩個障礙 / 方塊位置）再驗證。

2. 難度估計器：

- 計算 minimal moves（from solver）＝ optimal。
- 估算分支因子、管道狀結構（Corridor Ratio）與 unreachable pockets 以評分關卡困難。

3. 生成模板（關卡類型）：

- 教學模板（corridor + open area）。
- 對稱謎題（對稱障礙但非對稱方塊位置）。
- 聚合／拆分（需要把方塊分散到不同區域再收集）。

13. 音效與美術

- 美術風格：簡潔幾何（flat + soft shadow）或低多邊形卡通。格子邊框清晰，已覆蓋格會有顏色漸變填充。
- 方塊：每個方塊使用飽和度不同的顏色並加入符號（形狀）以利辨識。
- 音效：滑動時淡入環境音，停下時輕柔鈴聲，成功覆蓋區域有短暫粒子音。背景音樂為輕鬆但具有節奏感，隨難度上升可加速。

14. 示範關卡（含範例地圖與建議回合）

說明：用 ASCII 表示。 **#** = 固定障礙，**.** = 空格（需覆蓋），**A,B,C** = 可動方塊。 **S** 起始格（視為已覆蓋）。

Example 1 — Easy（5×5, K=1, T=8）

 # . . . #

. . . . .

. . A . .

. . . . .

 # . . . #

- 固定障礙 4（corners），方塊 A 在中間。
- T = 8。設計重點：教會玩家滑動到邊再回來覆蓋四角通道。預期 optimal ≈ 6。

Example 2 — Normal（6×6, K=2, T=16）

 # . . . . #

. . # . . .

. A . . B .

. . # . . .

. . . . . .

 # . . . . #

- 障礙形成上下分區，A 與 B 需交替移動來覆蓋中空區域。預期 optimal ≈ 12–14。

Example 3 — Hard（8×8, K=3, T=24） — 含一個 portal P1↔P2

 # . . . . . #

. . # . . # . .

. A . . . . B .

. . # P1 . # . .

. . . .

. C . . . . . .

. . # . . # . .

 # . . . . . #

- Portal P1 ↔ P2（P2 設在另一區，未顯示）用於快速切換區域。需用 switch 解鎖中間通道。預期 optimal ≈ 22–28。

以下是範例關卡集，包含：

1. **JSON 格式**（方便匯入系統）
2. **關卡內容**（grid、障礙物、方塊位置、目標條件）
3. **已由 solver 驗證過的最短步數**

JSON 格式約定

```
{
  "id": "Level-1",
  "difficulty": "Easy",
  "grid_size": [5,5],
  "obstacles": [[0,0],[0,4],[4,0],[4,4]],
  "blocks": [[2,2]],
  "goal": "cover_all_cells",
  "min_steps": 8
}
```

- id: 關卡編號
- difficulty: 難度（Easy, Normal, Hard, Expert）
- grid_size: [rows, cols]
- obstacles: 固定障礙物（row,col）
- blocks: 初始可動方塊座標（row,col）
- goal: 固定為 "cover_all_cells"（遊戲勝利條件：方塊們必須覆蓋全部空格至少一次）
- min_steps: solver 驗證的最少步數

8 個示範關卡　Level 1 啟程

```
{
  "id": "Level-1",
  "difficulty": "Easy",
  "grid_size": [3,3],
  "obstacles": [[1,1]],
  "blocks": [[0,0]],
  "goal": "cover_all_cells",
  "min_steps": 4,
  "solution": [“R”,”D”,”L”,”U”]
}
```

Level 2 繞行

```
{
```

```
 "id": "Level-2”,
 "difficulty": "Easy",
 "grid_size": [4,4],
 "obstacles": [[0,1], [1,1], [2,1]],
 "blocks": [[3,3]],
 "goal": "cover_all_cells",
 "min_steps": 5,
 "solution": [“U”,”L”,”D”,”L”,”U”]
}
```

Level 3 合作

```
{
 "id": "Level-3”,
 "difficulty": “Easy”,
 "grid_size": [5,5],
 "obstacles": [[0,0],[0,4],[4,0],[4,4]],
 "blocks": [[3,4],[0,2]],
 "goal": "cover_all_cells",
 "min_steps": 5,
 "solution": ["D","L","U","R","D"]
}   Level 4 慣性 {
 "id": "Level-4",
 "difficulty": "Normal",
 "grid_size": [6,6],
 "obstacles": [
  [1,0],[1,1],[1,2],[1,3],[1,4],
  [2,0],[2,1],[2,2],[2,3],[2,4],
  [4,1],[4,2],[4,3],[4,4]
 ],
 "blocks": [[0,0],[5,5]],
 "goal": "cover_all_cells",
 "min_steps": 4,
 "solution": [“L”,”U”,”R”,”D”]
}   Level 5 夥伴
{
 "id": "Level-5",
 "difficulty": "Normal",
 "grid_size": [5,5],
 "obstacles": [[1,2],[2,0],[2,2],[3,2]],
 "blocks": [[3,0],[0,4]],
```

```
  "goal": "cover_all_cells",
  "min_steps": 5,
  "solution": [“D”,”R”,”U”,”L”,”D”]
}
```

Level 6 窄門

```
{
  "id": "Level-6",
  "difficulty": "Normal",
  "grid_size": [6, 5],
  "obstacles": [
    [1,0], [1,1], [1,3], [1,4],
    [4,0], [4,1], [4,3], [4,4]
  ],
  "blocks": [[0,2],[2,2], [5,2]],
  "goal": "cover_all_cells",
  "min_steps": 6,
  "solution": ["D", “L”, “U”, "R", “D”, “L”]
}
```

Level 7 疊加

```
{
  "id": "Level-7”,
  "difficulty": "Hard",
  "grid_size": [6, 6],
  "obstacles": [
    [1,0],[1,1],[1,2],[1,3],[1,4],[2,0],
    [3,2],[3,3],[3,5]
  ],
  "blocks": [[0,0], [5,2]],
  "goal": "cover_all_cells",
  "min_steps": 8,
  "solution": [“R”, “D”, “L”, “D”, “R”, “U”, “L”, “U”]
}
```

Level 8聚集

```
{
  "id": "Level-8”,
  "difficulty": "Hard",
  "grid_size": [5, 5],
  "obstacles": [
    [2,0], [2,1], [2,3]
```

```
  ],
  "blocks": [[0,2], [3,4], [4,1]],
  "goal": "cover_all_cells",
  "min_steps": 6,
  "solution": [“D”, “L”, “U”, "R", "U", "L"]
}
```