Signal Decoder — All Code (Single-Line Plain Format)

===== .gitignore =====
node_modules
dist
.env
.vscode
.DS_Store
===== README.md =====
Signal Decoder — The Invisible Pattern Game

This is a React + TypeScript (Vite) implementation of the frontend assignment.

What you get
- 5x5 grid puzzle where squares flash following hidden rules.
- Levels implemented (1..5) matching the assignment.
- No UI libraries used; Tailwind is configured.
- Build: Vite

Run locally
1. Install dependencies:
```bash
npm install
```
2. Run dev server:
```bash
npm run dev
```
3. Build:
```bash
npm run build
npm run preview
```

Deploy to Vercel
1. Create a **private** GitHub repository and push this project.
2. Sign in to Vercel, import the repo, and use default settings.
- Framework: Vite
- Build command: `npm run build`
- Output directory: `dist`
3. Deploy. Vercel will provide a live URL.

Notes
- The flashing uses the level's answer set as the visible flashing indices for demo clarity.
- Customize timing or make flashing more complex as desired.
===== index.html =====
<!doctype html>
<html>
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>
<title>Signal Decoder</title>
</head>
<body>

<script type="module" src="/src/main.tsx"></script>
</body>
</html>
===== package.json =====
{
"name": "signal-decoder",
"version": "1.0.0",
"private": true,
"scripts": {
"dev": "vite",
"build": "vite build",
"preview": "vite preview"
},
"dependencies": {
"react": "^18.2.0",
"react-dom": "^18.2.0"
},
"devDependencies": {
"typescript": "^5.0.0",
"vite": "^5.0.0",
"@types/react": "^18.0.0",
"@types/react-dom": "^18.0.0",

```
"tailwindcss": "^3.0.0",
"postcss": "^8.0.0",
"autoprefixer": "^10.0.0"
}
}
===== postcss.config.cjs =====
module.exports = {
plugins: {
tailwindcss: {},
autoprefixer: {},
},
}
===== tailwind.config.cjs =====
module.exports = {
content: ['./index.html', './src/**/*.{ts,tsx}'],
theme: {
extend: {},
},
plugins: [],
}
===== tsconfig.json =====
{
"compilerOptions": {
"target": "ESNext",
"useDefineForClassFields": true,
"lib": ["DOM","ESNext"],
"jsx": "react-jsx",
"module": "ESNext",
"moduleResolution": "bundler",
"strict": true,
"esModuleInterop": true,
"skipLibCheck": true,
"forceConsistentCasingInFileNames": true,
"resolveJsonModule": true,
"isolatedModules": true,
"noEmit": true
},
"include": ["src"]
}
===== vercel.json =====
{
"builds": [
{ "src": "package.json", "use": "@vercel/static-build", "config": { "distDir": "dist" } }
]
}
===== vite.config.ts =====
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
plugins: [react()],
})
===== src/App.tsx =====
import React, { useState } from 'react'
import Game from './components/Game'

export default function App(){
return (
<div className="game-container">
<h1 className="text-3xl font-bold mb-4">Signal Decoder — The Invisible Pattern Game</h1>
<Game />
</div>
)
}
===== src/main.tsx =====
import React from 'react'
import { createRoot } from 'react-dom/client'
import App from './App'
import './styles.css'

createRoot(document.getElementById('root')!).render(
<React.StrictMode>
<App />
</React.StrictMode>
)
===== src/styles.css =====
@tailwind base;
```

```
@tailwind components;
@tailwind utilities;

/* Basic styles for the game */
body {
@apply bg-gray-100 dark:bg-gray-900 text-gray-900 dark:text-gray-100;
font-family: Inter, ui-sans-serif, system-ui, -apple-system, "Segoe UI", Roboto;
}
.game-container {
@apply max-w-3xl mx-auto p-6;
}
.grid {
@apply grid grid-cols-5 gap-2;
}
.cell {
@apply w-16 h-16 sm:w-20 sm:h-20 flex items-center justify-center border rounded cursor-pointer select-none;
transition: background-color 0.25s, transform 0.12s;
}
.cell.flashing {
@apply bg-yellow-300;
transform: scale(1.05);
}
.cell.selected {
@apply ring-4 ring-offset-2 ring-indigo-400;
}
.cell.correct {
@apply bg-green-400 text-white;
}
.cell.incorrect {
@apply bg-red-400 text-white;
}
.controls { @apply flex items-center gap-3 mt-4; }
===== src/components/Cell.tsx =====
import React from 'react'

type Props = {
index:number
flashing:boolean
selected:boolean
onClick?: ()=>void
reveal:boolean
isCorrect:boolean
isIncorrect:boolean
isMissed:boolean
}

export default function Cell({index, flashing, selected, onClick, reveal, isCorrect, isIncorrect, isMissed}:Pr
const classes = ['cell']
if(flashing) classes.push('flashing')
if(selected) classes.push('selected')
if(reveal && isCorrect) classes.push('correct')
if(reveal && isIncorrect) classes.push('incorrect')
if(reveal && isMissed) classes.push('incorrect')
return (
<div className={classes.join(' ')} onClick={onClick}>
<span className="text-sm">{index}</span>
</div>
)
}
===== src/components/Game.tsx =====
import React, { useEffect, useMemo, useState } from 'react'
import { getAnswerForLevel } from '../logic/LevelLogic'
import Grid from './Grid'
import ResultModal from './ResultModal'

const FLASH_DURATION = 1000 // ms per flash
const FLASH_CYCLES = 10 // total seconds ~10

export default function Game(){
const [level, setLevel] = useState(1)
const [isFlashing, setIsFlashing] = useState(true)
const [flashIndices, setFlashIndices] = useState<number[]>([])
const [selected, setSelected] = useState<Set<number>>(new Set())
const [showResult, setShowResult] = useState(false)
const [resultDetail, setResultDetail] = useState<{correct:number[], incorrect:number[], missed:number[]}>({cor

const answer = useMemo(()=> getAnswerForLevel(level), [level])
```

```
useEffect(()=>{
// generate flashing cycle sequence (for visual demo we'll repeat a flashing pattern)
setIsFlashing(true)
setSelected(new Set())
setShowResult(false)
const seq = answer.slice() // use the answer as the flashing set for clarity
setFlashIndices(seq)
const timer = setTimeout(()=> {
setIsFlashing(false)
}, FLASH_CYCLES * FLASH_DURATION)
return ()=> clearTimeout(timer)
}, [level, answer])

function toggleSelect(index:number){
if(isFlashing) return
setSelected(prev=>{
const copy = new Set(prev)
if(copy.has(index)) copy.delete(index)
else copy.add(index)
return copy
})
}

function submit(){
const selectedArr = Array.from(selected).sort((a,b)=>a-b)
const correct = selectedArr.filter(i=> answer.includes(i))
const incorrect = selectedArr.filter(i=> !answer.includes(i))
const missed = answer.filter(i=> !selectedArr.includes(i))
setResultDetail({correct, incorrect, missed})
setShowResult(true)
}

function nextLevel(){
setLevel(l => l+1)
}

function reset(){
setLevel(1)
}

return (
<div>
<div className="mb-3">
<strong>Level:</strong> {level}
</div>
<Grid
flashing={isFlashing}
flashIndices={flashIndices}
selected={selected}
onCellClick={toggleSelect}
reveal={showResult}
resultDetail={resultDetail}
/>
<div className="controls">
{!isFlashing && !showResult && (
<button className="px-4 py-2 bg-indigo-600 text-white rounded" onClick={submit}>Submit Answer</button>
)}
{!isFlashing && showResult && (
<>
<button className="px-4 py-2 bg-green-600 text-white rounded" onClick={nextLevel}>Next Level</button>
<button className="px-4 py-2 border rounded" onClick={reset}>Restart</button>
</>
)}
{isFlashing && <div>Observe the flashing sequence... (~10s)</div>}
</div>

<ResultModal open={showResult} detail={resultDetail} onClose={()=> setShowResult(false)} />
</div>
)
}
===== src/components/Grid.tsx =====
import React from 'react'
import Cell from './Cell'

type Props = {
flashing: boolean
```

```
flashIndices: number[]
selected: Set<number>
onCellClick: (index:number)=>void
reveal: boolean
resultDetail: {correct:number[], incorrect:number[], missed:number[]}
}

export default function Grid({flashing, flashIndices, selected, onCellClick, reveal, resultDetail}:Props){
// render 5x5 grid
const cells = Array.from({length:25}, (_,i)=>i)
return (
<div className="grid">
{cells.map(i=>(
<Cell
key={i}
index={i}
flashing={flashing && flashIndices.includes(i)}
selected={selected.has(i)}
onClick={()=> onCellClick(i)}
reveal={reveal}
isCorrect={resultDetail.correct.includes(i)}
isIncorrect={resultDetail.incorrect.includes(i)}
isMissed={resultDetail.missed.includes(i)}
/>
))}
</div>
)
}
===== src/components/ResultModal.tsx =====
import React from 'react'

export default function ResultModal({open, detail, onClose}:{open:boolean, detail:{correct:number[], incorrect
if(!open) return null
return (
<div className="fixed inset-0 flex items-center justify-center">
<div className="bg-white dark:bg-gray-800 p-6 rounded shadow-lg w-96">
<h3 className="text-xl font-semibold mb-3">Results</h3>
<p>Correct picks: {detail.correct.length}</p>
<p>Incorrect picks: {detail.incorrect.length}</p>
<p>Missed: {detail.missed.length}</p>
<div className="mt-4 flex justify-end gap-2">
<button onClick={onClose} className="px-3 py-1 border rounded">Close</button>
</div>
</div>
</div>
)
}
===== src/logic/LevelLogic.ts =====
export function getRowCol(index:number){
return {row: Math.floor(index/5), col: index % 5}
}

export function isPrime(n:number){
if(n < 2) return false
for(let i=2;i*i<=n;i++) if(n % i === 0) return false
return true
}

export function getAnswerForLevel(level:number){
const indices = Array.from({length:25}, (_,i)=>i)
switch(level){
case 1:
return indices.filter(i=> i % 2 === 0)
case 2:
return indices.filter(i=>{
const {row,col} = getRowCol(i)
return row === col || (row + col) === 4
})
case 3:
return indices.filter(i=> isPrime(i))
case 4:
// center 12 and neighbors: indices 7,11,12,13,17
return [7,11,12,13,17]
case 5:
return indices.filter(i=>{
const {row,col} = getRowCol(i)
return (row + col) % 3 === 0
```

```
})
default:
// for extra levels, fall back to even indices
return indices.filter(i=> i % 2 === 0)
}
}
```