Marcel Bosch 2022

Projecte final CIFO

SUDO SPACE TRAVELER

Index

Descripció

Aquest és un projecte de creació d'un joc de sudoku partint d'una API que genera els trencaclosques i els resol de manera automàtica. Per donar personalitat a l'aplicació i un toc original els nombres es substitueixen per símbols.

La nova perspectiva del sudoku afegeix un concepte de joc diferent i igualment jugable. Tot i que a l'inici semblava que la idea de substituïr els números per símbols podria incrementar la dificultat del joc finalment, després de fer diverses proves, no ha estat així.

Referències API

- API utilitzada: Sudoku Board

Nou trencaclosques

```
GET /api/new-board
```

Paràmetre	Tipus	Descripció
`RapidAPI App`	`string`	**Required**. La ID de l'API a RapidAPI

`Request URL`	`string`	**Required**. rapidapi.com
`X-RapidAPI-Host`	`string`	**Required**. Header proporcionat per RapidAPI
`X-RapidAPI-Key`	`enum`	**Required**. Key segons l'usuari
`diff`	`string`	**Optional**. Dificultat del trencaclosques (per defecte 2)

Resoldre trencaclosques

```
GET /api/solve-board
```

Paràmetre	Tipus	Descripció
`RapidAPI App`	`string`	**Required**. La ID de l'API a RapidAPI
`Request URL`	`string`	**Required**. rapidapi.com
`X-RapidAPI-Host`	`string`	**Required**. Header proporcionat per RapidAPI
`X-RapidAPI-Key`	`enum`	**Required**. Key segons l'usuari
`sudo`	`string`	**Required**. Trencaclosques a resoldre

Idea descartada

La idea inicial era crear una pàgina tipus buscador utilitzant una de les API del llistat proporcionat: API personatges disney. En el buscador es podrien trobar personatges i donar "like" o posar comentaris.

Al analitzar l'API durant l'aprovació de la idea vam veure que només tenia dos endpoints:

```
{
  getAllCharacters: 'https://api.disneyapi.dev/characters',
  getOneCharacters: 'https://api.disneyapi.dev/characters/:id'
}
```

Això no permetia fer recerques directament a l'API i suposava un problema ja que l'únic que permetia era obtenir la informació complerta amb el primer GET. La funcionalitat desitjada pel projecte de fer diverses crides a l'API quedava massa reduïda i, per tant, descartada.

Primera API sudoku descartada

Un cop descartada la primera idea, fent recerca per rapidAPI va sorgir la idea de crear un joc de sudoku amb la peculiaritat de que els nombres fossin substituïts per símbols.

La primera API trobada va ser Solve-sudoku.

Aquesta API permetia descarregar el trencaclosques i la solució alhora però tenia una limitació de 50 cerques diàries i, per aconseguir la "key", calia fer un registre de la targeta de crèdit ja que, un cop superades les 50 cerques, es carregava un import.

Aquesta opció doncs, va quedar també descartada degut a que el nivell de peticions durant el desenvolupament podia superar les 50 cerques al dia i suposaria un cost.

Desenvolupament

Fase 1: estructura html i css

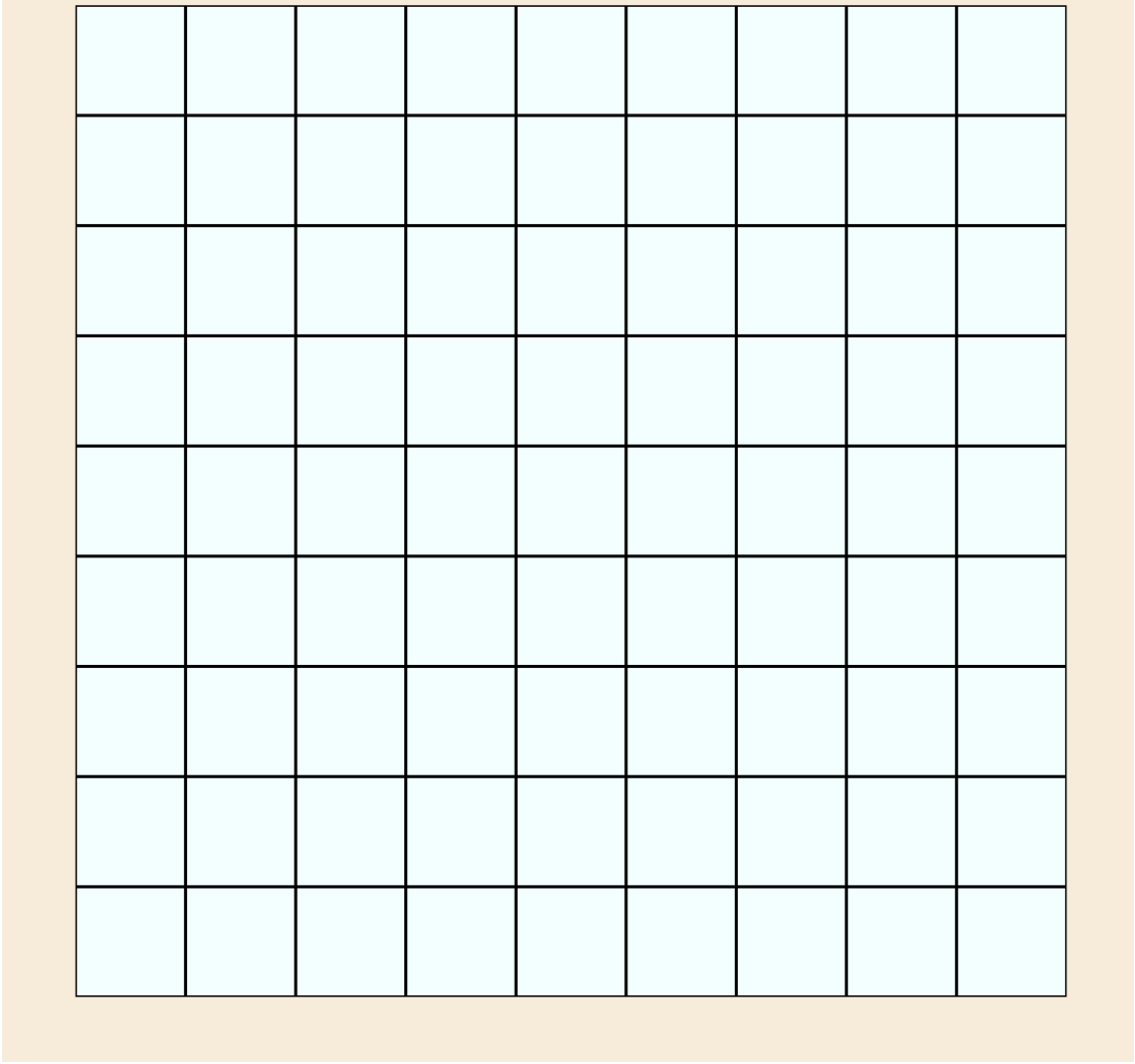
La primera fase consistia en veure possibles estructures per tal de presentar la pàgina i com poder fer la maquetació entenedora i senzilla.

La idea del joc és molt senzilla a nivell visual, un simple requadre central amb els 81 elements de tots els nombres al seu interior.

Vaig optar inicialment per crear una `<ul>` amb els 81 elements `<li>` interiors. Ja era obvi des d'un inici que aquests elements no haurien de ser creats de forma manual així però en fase de proves era molt més visual i entenedor.

Amb CSS la part més òbvia inicialment era convertir tots els elements de la llista en una grid. La mida de les cèl·les en vw ja crea una mica d'efecte "responsive" i, per tant, es poden fer tests en diverses mides.

La "responsivitat" en aquest projecte té un gran pès ja que es pretén que sigui un joc funcional a la majoria de dispositius (prioritzant telèfons).



No hi ha complicacions en aquest apartat.

Fase 2: Tests amb API i DOM

Aquesta fase consistia en testejar l'API i la seva relació amb el DOM per tal d'aconseguir renderitzar tots els elements del trencaclosques de manera correcte i veure possibles errors.

La pròpia documentació de l'API ja proporciona exemples de com fer les crides amb axios, fetch o XMLHttp.

La primera prova va ser amb fetch executant una crida i comprovant la correcta resposta guardada en una variable.

La data retornada consisteix en un array amb 9 elements (corresponents a les 9 files del sudoku) que contenen un altre array amb els 9 valors de cada cel·la.


```
{1 item
"response":{2 items
  "difficulty":"easy"
  "unsolved-sudoku":[9 items
```

```
      0:[9 items //Primera fila
        0:1 //Valor de la cel·la
        1:0
        2:0
        3:2
        4:5
        5:0
        6:0
        7:8
        8:9
      ]
      1:[...]9 items
      2:[...]9 items
      3:[...]9 items
      4:[...]9 items
      5:[...]9 items
      6:[...]9 items
      7:[...]9 items
      8:[...]9 items
  ]
}
```

Per tal de fer més fàcil la renderització dels elements dins de cada cèl·la el primer és concatenar cada fila per tal de tenir un array amb tots els valors junts. En altres API de sudoku de pagament la resposta sol ser així sense la divisió per files, per tant, en el cas de que en algun moment s'hagués de canviar l'API de referència seria també més fàcil fer el canvi.

Aquest array ens servirà prenent com a referència la grid per tal de que tingui cada valor el seu índex corresponent coincidint amb els elements child tal i com la tenim definida: `<ul class="grid">`.

Amb aquesta base ja podem imprimir els valors.

1	2	3					8	
4				9		2	3	
	8	9				5		
		1			4	6		
2					1	3		
5		6			9		2	
			6	7				5
			9			8		
		5				4		2

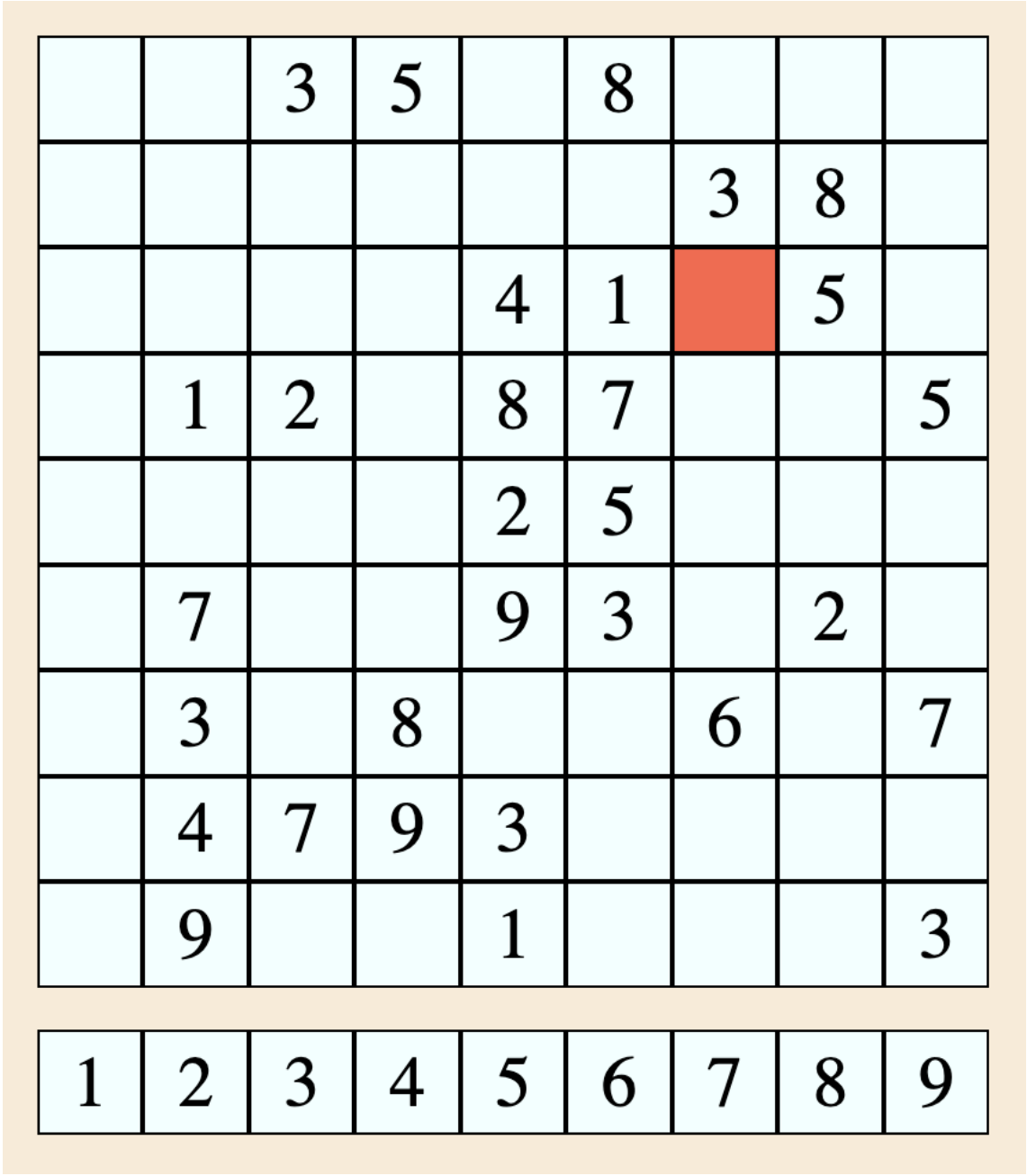
Complicacions:

- "Asincronicitat" i `fetch`:
  Les proves es van fer executant la funció de "getPuzzle" a l'inici, el comportament era correcte i tots els elements s'imprimien per pantalla al iniciar la pàgina carregant l'script afegint `async`.
  El problema va sorgir a l'hora d'afegir un botó per tal de fer la crida.
  Finalment resulta més fàcil fer la petició a l'API a través d'una funció `async` ja que quan fem servir un botó de "trigger" per imprimir els elements en pantalla podem fer servir `await` i no tindrem errors d'intentar renderitzar sense data. Fent servir `fetch` semblava que el codi es feia més extens amb masses línies de `.then` i era més difícil de llegir.

Fase 3: Testejant interacció de l'usuari

Aquesta fase consisteix en implementar el sistema per començar a modificar els valors de les cèl·les buides. Cal destacar visualment, en primer lloc, la cèl·la sel·leccionada. Simplement creant una variable la qual defineixi quina és aquella cel·la i afegir-li la classe "selected" que farà que el color de fons es modifiqui.

Al pensar en el sistema per introduir els valors de cada cel·la el més apropiat va semblar crear uns botons a la part inferior amb totes les opcions possibles. Visualment això concorda amb la grid principal i ens serveix perfectament per jugar en telèfon i d'altres dispositius sense afegir massa complicacions.



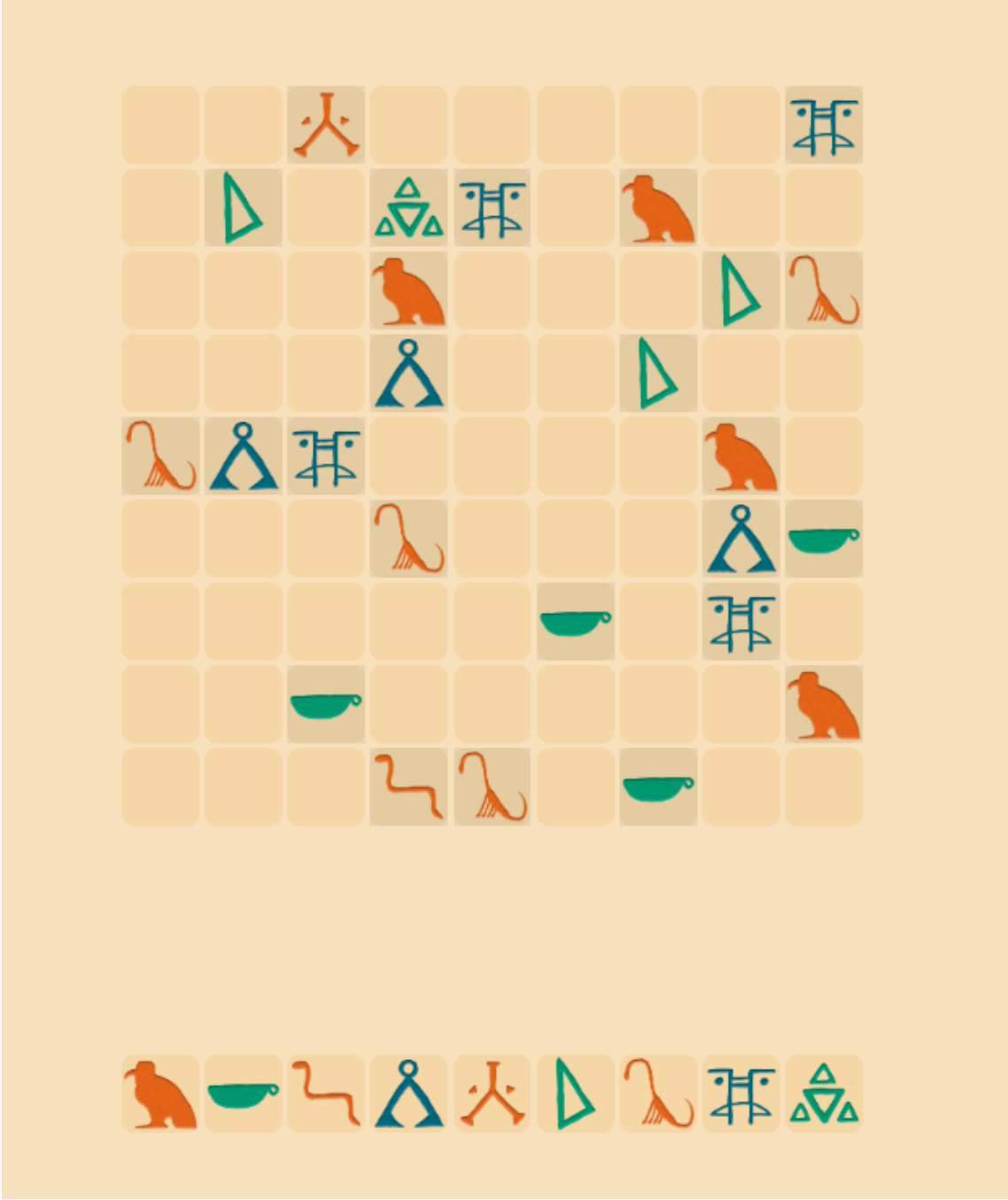
Complicacions:

- Descartem la idea d'utilitzar un input ja que en telèfons apareix el teclat que tapa part de la pantalla i es fa incòmode visualment. A més, la idea inicial de substituir els nombres per símbols ho faria extremadament incòmode.

Fase 4: Afegint símbols

Aquesta fase consisteix en substituïr tots els nombres per símbols. Cada símbol està creat amb photoshop amb la intenció de donar-li un toc egipci i relacionat amb el món de l'espai i els viatges espacials. Tenim 3 símbols que pertanyen a a jeroglifics egipcis, uns altres 3 que pertanyen als glifos utilitzats en els viatges de star gate i uns altres 3 que pertanyen a la simbologia de No Man's Sky.

Cada número és substituït per un dels símbols i, amb la barra de botons inferior, tenim la referència de cadescún.



Complicacions:

- Per tal de tenir una referència de quin número és cada símbol i d'altres propietas es fa testeig amb els atributs de data d'html. Finalment descartem l'opció ja que la única informació rellevant és el número i aquesta dada ja queda emmagatzemada en la propietat "alt" de la imatge. D'altres propietats tenen a veure amb la mateixa classe d'html i, per tant, finalment no és necessari utilitzar els data attributes.

Fase 5: Sudoku implementat

A aquestes alçades ja podem dir que la funcionalitat principal per jugar està implementada. Iniciem doncs proves per tal de saber si la solució del trencaclosques que hem aplicat és correcte o no. La mateixa API ja ens envia la solució del trencaclosques així que no caldrà fer una altra crida com passava amb altres APIs.

Apliquem la següent lògica per tal de veure els errors:

```
function checkUserSolution() {
    let wrongTiles = 0;

    const tiles = [].slice.call(grid.children);
    for (const [i, tile] of tiles.entries()) {
        if (tile.children[0].alt != solvedPuzzle[i]) {
            grid.children[i].classList.add("wrong");
            wrongTiles++;
        }
    }
    if (wrongTiles > 0) alert(`Wrong answer! you made ${wrongTiles} mistakes`);
    menuContainer.style.display = 'none';

}
```

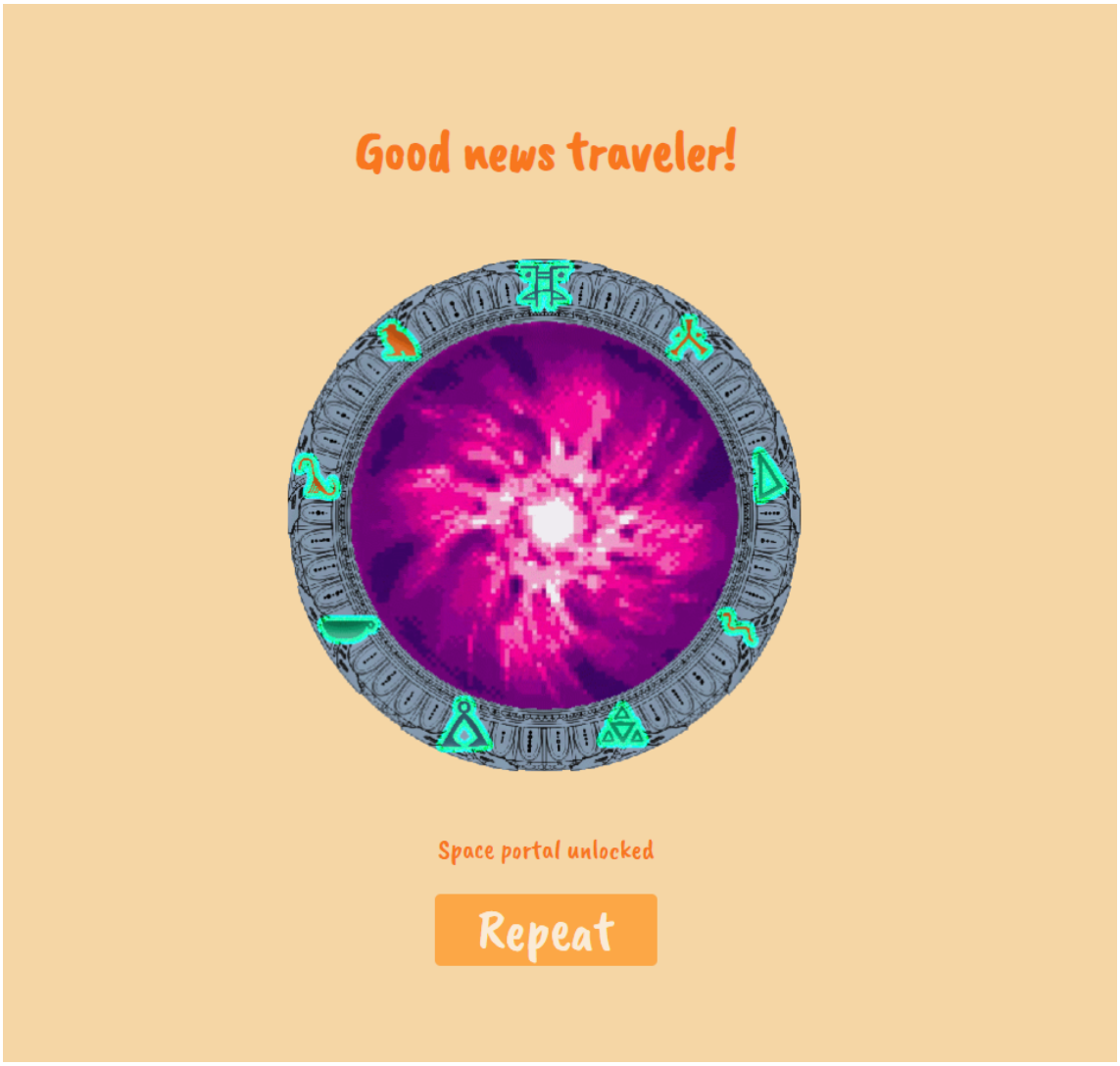
La const tiles ens crearà un array amb totes les cèl·les per tal de poder comprobar les respostes de l'usuari/a. El mateix array amb la solució ja el tenim plantejat perquè concordi amb els índex de la grid i d'aquesta manera ens corregeix les cèl·les incorrectes a les cuals afegim la classe "wrong" la qual canviarà el seu color de fons.

Fase 6: Menú d'inici i pantalla final

El menú d'inici consisteix en un contenidor que ocuparà tota la pantalla i portarà les opcions d'inici i d'enviar el puzzle per obtenir la seva resolució o informació dels errors que hem comès.

També tindrem l'opció de carregar un nou trencaclosques.

Un cop enviat el trencaclosques la pantalla final ens indicarà que hem desbloquejat la combinació correcte.



Fase 7: Últims detalls

Afegim un comptador amb el temps que es triga en resoldre el trencaclosques.

```
const timerDOM = document.getElementById('timer');

let timer;

let start = false;
let hours = 0;
let minutes = `0${0}`;
let seconds = `0${0}`;


const resetTimer = () => {
    minutes = `0${0}`;
    seconds = `0${0}`;
```

```
    hours = 0;
    timer = `${hours}:${minutes}:${seconds}`;
    timerDOM.innerText = timer;
};

const stopTimer = () => start = false;
const startTimer = () => start = true;

setInterval(() => {
    if (start) {
        seconds++
        seconds < 10 ? seconds = `0${seconds}` : seconds = seconds;
        if (seconds == 60) {
            minutes++;
            seconds = `0${0}`;
            seconds = seconds;
            minutes < 10 ? minutes = `0${minutes}` : minutes = minutes;
        }
        if (minutes == 60) {
            hours++
            minutes == 60 ? minutes = `0${0}` : minutes = minutes;
        }
        timer = `${hours}:${minutes}:${seconds}`;
        timerDOM.innerText = timer;
    }
}, 1000);


export { stopTimer, startTimer, resetTimer, timer };
```

L'app setejarà la variable start a true o false segons convingui. En els moments en els que el menú està obert estarà en false ja que el mateix menú actuarà com a pausa. Les condicions estàn implementades amb un == ja que el resultat final que mostrem per pantalla és una string degut a que hem de mostrar un 0 davant les unitats més petites de 10 per raons estètiques.

En aquesta fase també s'aplica refactorització movent el contingut de la crida a la API cap a un arxiu separat i aplicant diverses

Backlog

A mesura que s'ha anat generant el joc han sorgit diverses idees que no han pogut estar implementades:

- Afegir nivells de dificultat
- Afegir sons i música
- CRUD d'usuaris
- Ranking de millors puntuacions

Per altra banda, en una de les comunicacions de tutoria va sorgir una proposta de dividir les columnes o cèl·les per grups com passa en altres modalitats de jocs de sudoku i que ressalti una mica més de separació entre els grups de cada 3 cèl·les. Això no ha estat possible implementar-ho a temps degut a que el joc està basat en una grid i els gaps no poden ser diferents depenent de la columna o fila.

Les proves afegint marges, paddings, hombres, etc no donaven bon resultat estèticament i finalment s'ha optat per no implementar-ho ja que la solució més eficient possiblement sigui dividir el tauler en diverseses grids i replantejar tota la lògica.

Links del projecte

Repositori github

App a github pages