epsi

l'école d'ingénierie
informatique

EPSI BORDEAUX - I1 EISI

Module Atelier - Architecture applicative (7/9)

Code Module	Durée	Titre Diplôme	Bloc de Compétences	Promotion	Auteur
ARCE842 - DEVE702	20h	EISI / RNCP 35584	Concevoir & Développer des solutions applicatives métiers	2023/2024	Julien COURAUD

13. Grille de jeu interactive

Cette section sera également présentée sous la forme d'un exercice, comme pour la partie sur les dés et les choix. Vous n'aurez à votre disposition que les templates Client et la liste des méthodes du GameService côté serveur (pas leur implémentation). Vous aurez donc à produire la logique de jeu au sein du 'index.js' et l'implémentation des méthodes du service côté WebSocket Server.

Composant graphique 'Grid'

```
// app/components/board/grid/grid.component.js

import React, { useEffect, useContext, useState } from "react";
import { View, Text, TouchableOpacity, StyleSheet } from "react-native";
import { SocketContext } from "../../../contexts/socket.context";

const Grid = () => {

    const socket = useContext(SocketContext);

    const [displayGrid, setDisplayGrid] = useState(true);
    const [canSelectCells, setCanSelectCells] = useState([]);
    const [grid, setGrid] = useState(
        Array(5).fill().map(() => Array(5).fill().map(() => (
            { viewContent: '', id: '', owner: null, canBeChecked: false }
            ))
        ));

    const handleSelectCell = (cellId, rowIndex, cellIndex) => {
        if (canSelectCells) {
            socket.emit("game.grid.selected", { cellId, rowIndex, cellIndex });
        }
    };

    useEffect(() => {
        socket.on("game.grid.view-state", (data) => {
            setDisplayGrid(data['displayGrid']);
            setCanSelectCells(data['canSelectCells'])
            setGrid(data['grid']);
        });
    }, []);
```

```
    return (
        <View style={styles.gridContainer}>
            {displayGrid &&
                grid.map((row, rowIndex) => (
                    <View key={rowIndex} style={styles.row}>
                        {row.map((cell, cellIndex) => (
                            <TouchableOpacity
                                key={cell.id}
                                style={[
                                    styles.cell,
                                    cell.owner === "player:1" && styles.playerOwnedCell,
                                    cell.owner === "player:2" && styles.opponentOwnedCell,
                                    (cell.canBeChecked && !(cell.owner === "player:1") && !(cell.owner === "p
                                    rowIndex !== 0 && styles.topBorder,
                                    cellIndex !== 0 && styles.leftBorder,
                                ]}
                                onPress={() => handleSelectCell(cell.id, rowIndex, cellIndex)}
                                disabled={!cell.canBeChecked}
                            >
                                <Text style={styles.cellText}>{cell.viewContent}</Text>
                            </TouchableOpacity>
                        ))}
                    </View>
                ))}
        </View>
    );
};

const styles = StyleSheet.create({
    gridContainer: {
        flex: 7,
        justifyContent: "center",
        alignItems: "center",
        flexDirection: "column",
    },
    row: {
        flexDirection: "row",
        flex: 1,
        width: "100%",
        justifyContent: "center",
        alignItems: "center",
    },
    cell: {
        flexDirection: "row",
        flex: 2,
        width: "100%",
        height: "100%",
        justifyContent: "center",
        alignItems: "center",
        borderWidth: 1,
        borderColor: "black",
    },
    cellText: {
        fontSize: 11,
    },
    playerOwnedCell: {
        backgroundColor: "lightgreen",
        opacity: 0.9,
    },
    opponentOwnedCell: {
        backgroundColor: "lightcoral",
        opacity: 0.9,
    },
    canBeCheckedCell: {
        backgroundColor: "lightyellow",
    },
    topBorder: {
        borderTopWidth: 1,
    },
    leftBorder: {
        borderLeftWidth: 1,
    },
});
```

```
export default Grid;
```

Méthodes du 'GameService'

```
// websocket-game-server/services/game.service.js

// ...

const GRID_INIT = [
    [
        { viewContent: '1', id: 'brelan1', owner: null, canBeChecked: false },
        { viewContent: '3', id: 'brelan3', owner: null, canBeChecked: false },
        { viewContent: 'Défi', id: 'defi', owner: null, canBeChecked: false },
        { viewContent: '4', id: 'brelan4', owner: null, canBeChecked: false },
        { viewContent: '6', id: 'brelan6', owner: null, canBeChecked: false },
    ],
    [
        { viewContent: '2', id: 'brelan2', owner: null, canBeChecked: false },
        { viewContent: 'Carré', id: 'carre', owner: null, canBeChecked: false },
        { viewContent: 'Sec', id: 'sec', owner: null, canBeChecked: false },
        { viewContent: 'Full', id: 'full', owner: null, canBeChecked: false },
        { viewContent: '5', id: 'brelan5', owner: null, canBeChecked: false },
    ],
    [
        { viewContent: '≤8', id: 'moinshuit', owner: null, canBeChecked: false },
        { viewContent: 'Full', id: 'full', owner: null, canBeChecked: false },
        { viewContent: 'Yam', id: 'yam', owner: null, canBeChecked: false },
        { viewContent: 'Défi', id: 'defi', owner: null, canBeChecked: false },
        { viewContent: 'Suite', id: 'suite', owner: null, canBeChecked: false },
    ],
    [
        { viewContent: '6', id: 'brelan6', owner: null, canBeChecked: false },
        { viewContent: 'Sec', id: 'sec', owner: null, canBeChecked: false },
        { viewContent: 'Suite', id: 'suite', owner: null, canBeChecked: false },
        { viewContent: '≤8', id: 'moinshuit', owner: null, canBeChecked: false },
        { viewContent: '1', id: 'brelan1', owner: null, canBeChecked: false },
    ],
    [
        { viewContent: '3', id: 'brelan3', owner: null, canBeChecked: false },
        { viewContent: '2', id: 'brelan2', owner: null, canBeChecked: false },
        { viewContent: 'Carré', id: 'carre', owner: null, canBeChecked: false },
        { viewContent: '5', id: 'brelan5', owner: null, canBeChecked: false },
        { viewContent: '4', id: 'brelan4', owner: null, canBeChecked: false },
    ]
];

const GameService = {

    init: {
        gameState: () => {
            const game = { ...GAME_INIT };
            game['gameState']['timer'] = TURN_DURATION;
            game['gameState']['deck'] = { ...DECK_INIT };
            game['gameState']['choices'] = { ...CHOICES_INIT };
            game['gameState']['grid'] = [ ...GRID_INIT];
            return game;
        },

        // ...

        grid: () => {
            return { ...GRID_INIT };
        }
    },

    send: {
        forPlayer: {

            // ...
```

```
            gridViewState: (playerKey, gameState) => {

                return {
                    displayGrid: true,
                    canSelectCells: (playerKey === gameState.currentTurn) && (gameState.choices.availableChoi
                    grid: gameState.grid
                };

            }
        }
    },

    choices: {
        findCombinations: (dices, isDefi, isSec) => {
            const availableCombinations = [];
            const allCombinations = ALL_COMBINATIONS;

            const counts = Array(7).fill(0); // Tableau pour compter le nombre de dés de chaque valeur (de 1
            let hasPair = false; // Pour vérifier si une paire est présente
            let threeOfAKindValue = null; // Stocker la valeur du brelan
            let hasThreeOfAKind = false; // Pour vérifier si un brelan est présent
            let hasFourOfAKind = false; // Pour vérifier si un carré est présent
            let hasFiveOfAKind = false; // Pour vérifier si un Yam est présent
            let hasStraight = false; // Pour vérifier si une suite est présente
            let sum = 0; // Somme des valeurs des dés

            // Compter le nombre de dés de chaque valeur et calculer la somme
            for (let i = 0; i < dices.length; i++) {
                const diceValue = parseInt(dices[i].value);
                counts[diceValue]++;
                sum += diceValue;
            }

            // Vérifier les combinaisons possibles
            for (let i = 1; i <= 6; i++) {
                if (counts[i] === 2) {
                    hasPair = true;
                } else if (counts[i] === 3) {
                    threeOfAKindValue = i;
                    hasThreeOfAKind = true;
                } else if (counts[i] === 4) {
                    threeOfAKindValue = i;
                    hasThreeOfAKind = true;
                    hasFourOfAKind = true;
                } else if (counts[i] === 5) {
                    threeOfAKindValue = i;
                    hasThreeOfAKind = true;
                    hasFourOfAKind = true;
                    hasFiveOfAKind = true;
                }
            }

            const sortedValues = dices.map(dice => parseInt(dice.value)).sort((a, b) => a - b); // Trie les v

            // Vérifie si les valeurs triées forment une suite
            hasStraight = sortedValues.every((value, index) => index === 0 || value === sortedValues[index -

            // Vérifier si la somme ne dépasse pas 8
            const isLessThanEqual8 = sum <= 8;

            // Retourner les combinaisons possibles via leur ID
            allCombinations.forEach(combination => {
                if (
                    (combination.id.includes('brelan') && hasThreeOfAKind && parseInt(combination.id.slice(-1
                    (combination.id === 'full' && hasPair && hasThreeOfAKind) ||
                    (combination.id === 'carre' && hasFourOfAKind) ||
                    (combination.id === 'yam' && hasFiveOfAKind) ||
                    (combination.id === 'suite' && hasStraight) ||
                    (combination.id === 'moinshuit' && isLessThanEqual8) ||
                    (combination.id === 'defi' && isDefi)
                ) {
                    availableCombinations.push(combination);
                }
            });
```

```
            const notOnlyBrelan = availableCombinations.some(combination => !combination.id.includes('brelan'

            if (isSec && availableCombinations.length > 0 && notOnlyBrelan) {
                availableCombinations.push(allCombinations.find(combination => combination.id === 'sec'));
            }

            return availableCombinations;
        }
    },

    grid: {

        resetcanBeCheckedCells: (grid) => {
            const updatedGrid = // TODO

            // La grille retournée doit avoir le flag 'canBeChecked' de toutes les cases de la 'grid' à 'fals

            return updatedGrid;
        },

        updateGridAfterSelectingChoice: (idSelectedChoice, grid) => {

            const updatedGrid = // TODO

            // La grille retournée doit avoir toutes les 'cells' qui ont le même 'id' que le 'idSelectedChoic

            return updatedGrid;
        },

        selectCell: (idCell, rowIndex, cellIndex, currentTurn, grid) => {
            const updatedGrid = // TODO

            // La grille retournée doit avoir avoir la case selectionnée par le joueur du tour en cours à 'ow
            // Nous avons besoin de rowIndex et cellIndex pour différencier les deux combinaisons similaires

            return updatedGrid;
        }

    },

    // ...
}

module.exports = GameService;
```

Logique de jeu côté WebSocket Server

- `createGame`

  --> Au deux clients concernés: on envoie la grille de jeu initialisée au tout début de la partie.

  --> Dans le `setInterval()`, toutes les secondes, lorsque c'est la fin du tour:

  - Nous prévoyons de réinitialiser la surbrillance des combinaisons possibles qu'avait le joueur précédent. Pour cela nous utiliseront la méthode: `GameService.grid.resetcanBeCheckedCells()`,
  - Puis nous mettrons à jour la vue `updateClientsViewGrid(game)` des deux joueurs.

- Lorsque que le `socket.on('game.choices.selected')` est appelé, donc à la selection d'un choix par le joueur qui à la main. Après la mise à jour du flag 'idSelectedChoice' du sous objet 'gameState.choices':

  - Nous mettons à jour les cases précédemment selectionables avec la méthode: `GameService.grid.resetcanBeCheckedCells()`
  - Puis nous mettons également à jour la grille avec notre méthode: `GameService.grid.updateGridAfterSelectingChoice()`. Celle-ci ira modifier le flag 'canBeChecked' des cases liées à l'id du choix passé par l'utilisateur.

    - Puis nous mettons à jour la vue `updateClientsViewGrid(game)` des deux joueurs.

- Lorsque que le `socket.on('game.grid.selected')` est appelé:

```
socket.on('game.grid.selected', (data) => {

  const gameIndex = GameService.utils.findGameIndexBySocketId(games, socket.id);



  // La sélection d'une cellule signifie la fin du tour (ou plus tard le check des conditions de victoires)
  // On reset l'état des cases qui étaient précédemment clicables.
  games[gameIndex].gameState.grid = GameService.grid.resetcanBeCheckedCells(games[gameIndex].gameState.grid);
  games[gameIndex].gameState.grid = GameService.grid.selectCell(data.cellId, data.rowIndex, data.cellIndex, g

  // TODO: Ici calculer le score
  // TODO: Puis check si la partie s'arrête (lines / diagolales / no-more-gametokens)

  // Sinon on finit le tour
  games[gameIndex].gameState.currentTurn = games[gameIndex].gameState.currentTurn === 'player:1' ? 'player:2'
  games[gameIndex].gameState.timer = GameService.timer.getTurnDuration();

  // On remet le deck et les choix à zéro (la grille, elle, ne change pas)
  games[gameIndex].gameState.deck = GameService.init.deck();
  games[gameIndex].gameState.choices = GameService.init.choices();

  // On reset le timer
  games[gameIndex].player1Socket.emit('game.timer', GameService.send.forPlayer.gameTimer('player:1', games[ga
  games[gameIndex].player2Socket.emit('game.timer', GameService.send.forPlayer.gameTimer('player:2', games[ga

  // et on remet à jour la vue
  updateClientsViewDecks(games[gameIndex]);
  updateClientsViewChoices(games[gameIndex]);
  updateClientsViewGrid(games[gameIndex]);
});
```