l'école d'ingénierie informatique

EPSI BORDEAUX - I1 EISI

Module Atelier - Architecture applicative (6/9)

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

12. Les choix et les combinaisons disponibles

Cette section sera également présentée sous la forme d'un exercice, comme pour la partie sur les dés. Cependant cette fois-ci 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 de la méthode de recherche de combinaisons disponibles dans le 'GameService' (TODO en commentaires dans le code à compléter).

Composant 'Choices'

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

const Choices = () => {

    const socket = useContext(SocketContext);

    const [displayChoices, setDisplayChoices] = useState(false);
    const [canMakeChoice, setCanMakeChoice] = useState(false);
    const [idSelectedChoice, setIdSelectedChoice] = useState(null);
    const [availableChoices, setAvailableChoices] = useState([]);

    useEffect(() => {

        socket.on("game.choices.view-state", (data) => {
            setDisplayChoices(data['displayChoices']);
            setCanMakeChoice(data['canMakeChoice']);
            setIdSelectedChoice(data['idSelectedChoice']);
            setAvailableChoices(data['availableChoices']);
        });

    }, []);

    const handleSelectChoice = (choiceId) => {

        if (canMakeChoice) {
            setIdSelectedChoice(choiceId);
            socket.emit("game.choices.selected", { choiceId });
        }

    };
```

```
    return (
        <View style={styles.choicesContainer}>
            {displayChoices &&
                availableChoices.map((choice) => (
                    <TouchableOpacity
                        key={choice.id}
                        style={[
                            styles.choiceButton,
                            idSelectedChoice === choice.id && styles.selectedChoice,
                            !canMakeChoice && styles.disabledChoice
                        ]}
                        onPress={() => handleSelectChoice(choice.id)}
                        disabled={!canMakeChoice}
                    >
                        <Text style={styles.choiceText}>{choice.value}</Text>
                    </TouchableOpacity>
                ))}
        </View>
    );
};

const styles = StyleSheet.create({
    choicesContainer: {
        flex: 1,
        flexDirection: "row",
        flexWrap: "wrap",
        justifyContent: "space-between",
        paddingHorizontal: 10,
        borderBottomWidth: 1,
        borderColor: "black",
        backgroundColor: "lightgrey"
    },
    choiceButton: {
        backgroundColor: "white",
        borderRadius: 5,
        marginVertical: 5,
        alignItems: "center",
        justifyContent: "center",
        width: "100%",
        height: "10%"
    },
    selectedChoice: {
        backgroundColor: "lightgreen",
    },
    choiceText: {
        fontSize: 13,
        fontWeight: "bold",
    },
    disabledChoice: {
        opacity: 0.5,
    },
});

export default Choices;
```

Méthodes du GameService

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

const CHOICES_INIT = {
    isDefi: false,
    isSec: false,
    idSelectedChoice: null,
    availableChoices: [],
};

const ALL_COMBINATIONS = [
    { value: 'Brelan1', id: 'brelan1' },
    { value: 'Brelan2', id: 'brelan2' },
    { value: 'Brelan3', id: 'brelan3' },
    { value: 'Brelan4', id: 'brelan4' },
```

```
    { value: 'Brelan5', id: 'brelan5' },
    { value: 'Brelan6', id: 'brelan6' },
    { value: 'Full', id: 'full' },
    { value: 'Carré', id: 'carre' },
    { value: 'Yam', id: 'yam' },
    { value: 'Suite', id: 'suite' },
    { value: '≤8', id: 'moinshuit' },
    { value: 'Sec', id: 'sec' },
    { value: 'Défi', id: 'defi' }
];

// ...

const GameService = {

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

        // ...

        choices: () => {
            return { ...CHOICES_INIT };
        }
    },

    send: {
        forPlayer: {

            // ...

            choicesViewState: (playerKey, gameState) => {
                const choicesViewState = {
                    displayChoices: true,
                    canMakeChoice: playerKey === gameState.currentTurn,
                    idSelectedChoice: gameState.choices.idSelectedChoice,
                    availableChoices: gameState.choices.availableChoices
                }
                return choicesViewState;
            }
        }
    },

    // ...

    choices: {
        findCombinations: (dices, isDefi, isSec) => {

            const allCombinations = ALL_COMBINATIONS;

            // Tableau des objets 'combinations' disponibles parmi 'ALL_COMBINATIONS'
            const availableCombinations = [];

            // Tableau pour compter le nombre de dés de chaque valeur (de 1 à 6)
            const counts = Array(7).fill(0);

            let hasPair = false; // check: paire
            let threeOfAKindValue = null; // check: valeur brelan
            let hasThreeOfAKind = false; // check: brelan
            let hasFourOfAKind = false; // check: carré
            let hasFiveOfAKind = false; // check: yam
            let hasStraight = false; // check: suite
            let sum = 0; // sum of dices

            // ------------------------------------
            // TODO: Vérifier les combinaisons possibles
            // ------------------------------------

            // return available combinations
```

```
            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);
                }
            });

            return availableCombinations;
        }
    }
}
```

Logique de jeu

- `createGame`

  --> Aucune modification à l'init (aucun choix ne peut être fait).
  --> Dans le `setInterval()`, toutes les secondes, lorsque c'est la fin du tour. Nous prévoyons de réinitialiser les choix (`GameService.init.choices();`), puis nous mettrons à jour la vue `updateClientsViewChoices(game)`)

- Lorsque que le `socket.on('game.dices.roll')` est appelé (pour tous les lancers):

```
// combinations management
const dices = { ...games[gameIndex].gameState.deck.dices};
const isDefi = false;
const isSec = games[gameIndex].gameState.deck.rollsCounter === 2;
const combinations = GameService.choices.findCombinations(dices, isDefi, isSec);

// we affect changes to gameState
games[gameIndex].gameState.choices.availableChoices = combinations;

// emitters
updateClientsViewChoices(games[gameIndex]);
```

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

```
socket.on('game.choices.selected', (data) => {
    // gestion des choix
    const gameIndex = GameService.utils.findGameIndexBySocketId(games, socket.id);
    games[gameIndex].gameState.choices.idSelectedChoice = data.choiceId;

    updateClientsViewChoices(games[gameIndex]);
});
```