Space Operators

Présentation

Le vaisseau spatial dans lequel les joueurs naviguent tombe sans arrêt en panne, et des opérations de maintenance sont constamment nécessaires ! Mais il faut agir vite : se tromper dans ces opérations, prendre trop de temps, ou manquer de communication, et c'est tout le vaisseau qui risque d'exploser !

Space Operators est un jeu multijoueur en coopération, inspiré de Spaceteam, et dans lequel les joueurs auront tour à tour le rôle d'instructeur ou d'opérateur.

Un instructeur aura une opération à communiquer à un autre joueur qui sera alors opérateur, et qui devra la réaliser correctement. A chaque tour, les opérateurs et les instructeurs changent, et les joueurs ne savent pas qui est qui : il est impératif de se faire entendre pour donner la bonne instruction au bon opérateur !

Plus les tours passent, et plus la durée des tours (et donc du temps imparti pour la réalisation des tâches) est réduite.

Lorsqu'une tâche n'est pas complétée à temps, ou ne l'a pas été de la bonne manière, l'intégrité du vaisseau diminue, et le risque de désintégration augmente !

Fonctionnalités et écrans

Menu principal

Le menu principal affichera six éléments :

- Le nom du joueur avec la possibilité de le modifier
- L'ID unique du joueur généré de manière aléatoire (voir https://github.com/uuidjs/uuid)
- Un bouton "Créer une partie"
- Un bouton "Rejoindre une partie"
- Un bouton "Historique"
- Un bouton "Quitter"

Création d'une partie

Si un joueur décide de créer une partie, il sera redirigé vers un nouvel écran qui affichera :

- Un bouton retour
- L'ID de la partie à communiquer aux autres joueurs
- La liste des joueurs connectés (nom + statut prêt ou non)
- Un bouton "Démarrer la partie"

Quand tous les joueurs sont prêts, l'hôte peut cliquer sur le bouton "Démarrer la partie".

Rejoindre une partie

Si le joueur décide de rejoindre une partie, une popup s'affichera, lui demandant :

- L'ID de la partie
- Son nom de joueur (repris à partir du menu principal mais pouvant être modifié ici)
- De valider la connexion

Si la connexion est validée, il sera redirigé vers un nouvel écran affichant :

- Un bouton retour
- La liste des autres joueurs connectés (nom + statut prêt ou non)
- Un bouton permettant de passer son statut en "Prêt"

Partie démarrée

Lorsque la partie est démarrée, le nouvel écran devra afficher :

- Le numéro du tour actuel
- L'état de délabrement du vaisseau sous la forme d'une progress bar : elle tendra du vert au rouge en fonction de son remplissage

  Idée : lorsque la barre est quasiment pleine et le vaisseau sur le point d'exploser, la faire clignoter tel une alarme
- Le temps restant pour réaliser l'opération en cours

  Sous forme de compte à rebours, ou de progress bar … à vous de voir
- L'opération à réaliser ou l'instruction à donner
- Un bouton quitter

En attendant la prochaine opération ou instruction, afficher "En attente de la prochaine opération".

Historique

Lors du clic sur le bouton "Historique" dans le menu principal, le joueur sera redirigé vers une nouvelle page affichant l'historique de ses parties jouées :

- La liste des noms des joueurs
- Le nombre de tours réalisés

C'est au serveur de déterminer les paramètres de la partie :

- *La liste des opérations à réaliser*
- *La durée (dégressive) de chaque tour*
- *Qui sera opérateur ou instructeur à chaque tour*

A chaque tour, le serveur organisera les joueurs par pair, en déterminant aléatoirement qui sera instructeur, et qui sera opérateur.

Le serveur enverra à l'instructeur et à l'opérateur les mêmes données représentant l'opération à réaliser, dont la durée autorisée pour sa réalisation qui sera dégressive au fur et à mesure des tours.

Dès qu'une opération est réalisée ou que le temps imparti est dépassé, le serveur recevra la réponse de l'opérateur. En fonction de cette réponse (opération réussie, ou non), le serveur devra mettre à jour l'intégrité du vaisseau et transmettre cette nouvelle intégrité aux joueurs. L'intégrité du vaisseau est un pourcentage : à 100% l'intégrité du vaisseau est totale, à 0% le vaisseau est détruit.

Un nouveau tour commence une fois les réponses de tous les opérateurs reçues, ou à la fin d'un timeout.

Côté clients

Les joueurs recevront une opération à réaliser et auront un rôle aléatoire à chaque tour : soit opérateur, soit instructeur.

L'opérateur sera identifié par un code, aléatoire à chaque tour, sous la forme "XX-99" (par exemple BA-17, CG-96, ...). Cet identifiant sera également envoyé à l'instructeur afin qu'il sache à quel opérateur il doit donner ses instructions.

Si le joueur est instructeur, il aura affiché, en plus des éléments communs :

- L'identifiant de l'opérateur
- La description de la tâche à réaliser sous forme de texte

Si le joueur est opérateur, il aura affiché, en plus des éléments communs :

- Son identifiant
- La liste des éléments permettant de réaliser l'opération

Si le nombre de joueurs est impair, un joueur aléatoire sera systématiquement en attente.

S'il est en attente, ou que la tâche a été complétée avant la fin du tour, et en attendant le prochain tour, un message "En attente de la prochaine opération" sera affiché.

Si l'état de délabrement du vaisseau a atteint son maximum, la partie s'arrête, et un message de fin de partie s'affiche avec le nombre de tours complétés par l'équipe.
A vous d'être imaginatif sur la façon d'afficher la fin de la partie (animation, explosion, ...)

Spécifications des requêtes

La communication avec le serveur devra se faire de deux manière :

- Via une API REST mise à disposition par le serveur et permettant d'envoyer ou de récupérer des données à l'initiative des clients
- Via une connexion WebSocket afin que le serveur puisse envoyer des données aux clients en temps réel (par exemple les instructions en cours de partie). Voir : https://reactnative.dev/docs/network#websocket-support

Requêtes REST

Création d'une partie

Chemin	/create-game
Type	POST
Description	A envoyer lorsqu'un joueur souhaite créer une nouvelle partie

Données en corps de la réponse :

Nom	type	Description
id	String	ID de la partie créée

Exemple :

```
POST /create-game

Response body :
{
  "id": "56zer435G",
}
```

Changement de statut d'un joueur

Chemin	/ready/:id
Type	POST
Description	A envoyer lorsque le joueur clic sur le bouton "Prêt"

Données en paramètre de l'URL :

Nom	type	Description	Contraintes
id	String	ID du joueur	Requis

Exemple :

```
POST /ready/1b9d6bcd-bbfd-4b2d-9b5d-ab8dfbbd4bed
```

Requêtes WebSocket

Au lancement de l'application, une connexion WebSocket devra être établie avec le serveur sur l'URL https://space-operators.herokuapp.com/ et permettra au serveur d'envoyer des informations en temps réel au client. Les clients pourront également envoyer des informations au serveur par ce biais.

Pour comprendre comment créer une connexion WebSocket en React Native, voir : https://reactnative.dev/docs/network#websocket-support

Chaque message envoyé par le serveur aux clients sera au format JSON et respectera une structure donnée :

```
{
  "type": "TYPE_REQUETE",
  "data": {...}
}
```

Connexion à une partie

Sens	Client -> serveur
Type	connect
Description	A envoyer lorsqu'un joueur souhaite se connecter à une partie. Permet également au serveur d'identifier le joueur auquel appartient cette connexion.

Données :

Nom	type	Description	Contraintes
gameId	string	Identifiant de la partie	Requis
playerId	string	Identifiant unique du joueur	Requis
playerName	string	Nom du joueur	Requis

Exemple :

```
{
  "type": "connect",
  "data": {
    "gameId": "56zer435G",
    "playerId": "1b9d6bcd-bbfd-4b2d-9b5d-ab8dfbbd4bed",
    "playerName": "Toto"
  }
}
```

Mise à jour de la liste des joueurs

Sens	Serveur -> client
Type	players
Description	A envoyer à chaque joueur lorsqu'un nouveau joueur se connecte ou que le statut de l'un d'entre eux change avant le démarrage de la partie

Données :

Nom	type	Description	Contraintes
players	Player[]	Liste des joueurs	
players.name	String	Nom du joueur	
players.status	Boolean	Statut du joueur (prêt ou non)	

Exemple :

```
{
  "type": "players",
  "data": {
    "players": [
      {
        "name": "Player 1",
        "status": true
      },
      {
        "name": "Player 2",
        "status": false
      }
    ]
  }
}
```

Démarrage de la partie

Sens	Client -> serveur
Type	start
Description	A envoyer si tous les joueurs sont prêts, lors du clic sur le bouton "Démarrer"

Données : Aucune

Exemple :

```
{
  "type": "start"
}
```

Notification de démarrage de la partie

Sens	Serveur -> client
Type	start
Description	Envoyé à tous les joueurs par le serveur lorsque la partie est démarrée

Données : Aucune

Exemple :

```
{
  "type": "start"
}
```

Nouvelle opération

Sens	Serveur -> client
Type	operation
Description	A envoyer aux joueurs à chaque nouveau tour

Données :

Nom	type	Description	Contraintes
turn	Int	Tour actuel	
role	String	Indique si le joueur est opérateur ou instructeur	"operator" ou "instructor"
id	String	Identifiant de l'opérateur	Format XX-99 Ex : CI-86, LA-52
duration	Float	Durée en seconde pour réaliser l'opération	
description	String	Description de l'opération. A n'afficher qu'à l'instructeur qui devra décrire à l'opérateur quoi faire	
elements	Element[]	Liste des élément à afficher à l'opérateur pour permettre la réalisation de l'instruction	
elements.type	String	Type de l'élément	Voir chapitre suivant
elements.id	Int	Id de l'élément	
elements.valueType	String	Type de la valeur de l'élément	"string" ou "int" ou "float" ou "color"
elements.value	String	Valeur de l'élément	A parser en fonction du champ value-type
result	Result	Résultat attendu de l'opération	
result.buttons	ButtonResult	Résultat attendu pour les boutons	

result.buttons.order	String	Ordre de clic des boutons	"ordered" ou "random"
result.buttons.ids	Int[]	IDs des boutons devant être cliqués (dans l'ordre ou pas en fonction du champ "order")	
result.switchs	Int[]	IDs des switchs que l'opérateur doit activer	
result.links	Int[][]	Tableau des liens devant être reliés entre eux Exemple : [[1, 3], [2, 4]]	

Exemple :

```
{
  "type": "operation",
  "data": {
    "turn": 2,
    "role": "operator",
    "id": "CA-98",
    "duration": 4,
    "description": "Activer les switchs pairs et appuyer deux
fois sur les boutons rouges",
    "elements": [
      {
        "type": "switch",
        "id": 0,
        "valueType": "int",
        "value": 1
      },
      {
```

```
      "type": "switch",
      "id": 1,
      "valueType": "int",
      "value": 2
    },
    {
      "type": "switch",
      "id": 2,
      "valueType": "int",
      "value": 3
    },
    {
      "type": "switch",
      "id": 3,
      "valueType": "int",
      "value": 4
    },
    {
      "type": "button",
      "id": 4,
      "valueType": "color",
      "value": "#000"
    },
    {
      "type": "button",
      "id": 5,
      "valueType": "color",
      "value": "#FF0000"
```

```
      },
      {
        "type": "button",
        "id": 6,
        "valueType": "color",
        "value": "#00FF00"
      },
      {
        "type": "button",
        "id": 7,
        "valueType": "color",
        "value": "#FF0000"
      }
    ],
    "result": {
      "buttons": {
        "order": "random",
        "ids": [5, 5, 7, 7]
      },
      "switches": [1, 3]
    }
  }
}
```

Fin d'une opération

Sens	Client -> serveur
Type	finish
Description	A envoyer lorsqu'un opérateur a fini son opération, ou si le temps imparti est dépassé

Données :

Nom	type	Description	Contraintes
success	Boolean	true si l'opération a été correctement réalisée	
operator	String	Code de l'opérateur ayant réalisé (ou non) l'opération	XX-99

Exemple :

```
{
  "type": "finish",
  "data": {
    "id": "CI-93",
    "success": true
  }
}
```

Mise à jour de l'intégrité du vaisseau

Sens	Serveur -> client
Type	integrity
Description	A envoyer à tous les clients afin de mettre à jour l'intégrité du vaisseau si une opération n'a pas été réalisée correctement

Données :

Nom	type	Description	Contraintes
integrity	Float	Nouvelle valeur de l'intégrité du vaisseau. A 0%, le vaisseau est détruit.	

Exemple :

```
{
  "type": "integrity",
  "data": {
    "integrity": 50
  }
}
```

Vaisseau détruit : fin de la partie

Sens	Serveur -> client
Type	destroyed
Description	A appeler si l'intégrité du vaisseau atteint 0% afin de déclencher la fin de la partie

Données :

Nom	type	Description	Contraintes
turns	Int	Nombre de tours complétés par les joueurs avant la destruction du vaisseau	

Exemple :

```
{
  "type": "destroyed",
  "data": {
    "turns": 11
  }
}
```

Spécifications des éléments

Chaque opération à réaliser devra l'être à l'aide de différents éléments à afficher à l'opérateur.

Ces éléments peuvent être de différents types :

- Button
- Switch
- Link

Button

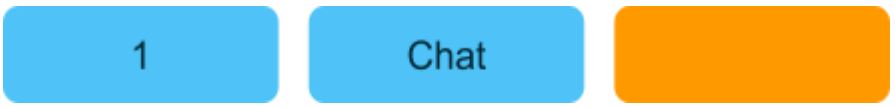
Un bouton peut être cliqué une ou plusieurs fois. Chaque clic doit être comptabilisé dans le résultat de l'opération.

Affichage :

La valeur d'un bouton sera directement affichée à l'intérieur de celui-ci.

Cependant, si le type de la valeur du bouton est "color" alors aucun texte ne sera affiché dans le bouton, mais sa couleur changera en fonction.

Exemples :



Switch

Un switch peut être activé ou non. A la fin de l'opération, les switch activés sont comptés dans le résultat de l'opération.

Affichage :

La valeur du switch devra être affichée au-dessus ou en-dessous ou à côté du switch.

Cependant, si le type de la valeur du switch est "color" alors aucun texte ne sera affiché autour du switch, mais sa couleur changera en fonction.

Exemple :

Liens

Un élément "lien" doit être relié (ou pas) à un autre élément de type "lien", en glissant le doigt de l'un vers l'autre.

Affichage :

Ils doivent être organisés en deux colonnes : pour X éléments de type "lien", les X/2 premiers éléments doivent apparaître sur la colonne de gauche, et les X/2 éléments sur la colonne de droite.

Un élément doit afficher sa valeur ainsi qu'un rond permettant de faire le lien. Si le type de la valeur du lien est "color" alors aucun texte ne doit être affiché, mais le rond doit être de la bonne couleur.

Exemple :