PROTOCOLE

En plus du moteur de jeu, on doit fournir :

- une archive zip contenant tous les sprites
- un json contenant les paramètres d'initialisation, avec leur format (int, float, avec leur range, ou string (avec leur format) et leur description

I) DEROULEMENT

Les N joueurs sont identifiés par des numéros allant de 1 à N.

Le déroulement d'une partie se fait de la façon suivante :

1. On envoie les paramètres INITIALISATION au moteur de jeu
2. En boucle :
    a. le moteur de jeu nous envoie des INSTRUCTIONS
    b. on transfert les INSTRUCTIONS aux joueurs concernés
    c. on attend les ACTIONS des joueurs concernés
    d. a chaque réception d'ACTION, on la transmet au moteur

II) ETAT DU JEU

On considère que les moteurs de jeu seront déterministes en fonction de la phase d'initialisation et des actions des joueurs. Cela veut dire que si on relance une partie en fournissant les mêmes données d'initialisation et/ou le retour de celles-ci renvoyées par le moteur, ainsi que les mêmes actions des joueurs, on doit arriver au même état de la partie.

Ceci implique que toute donnée aléatoire doit être déterminée durant la phase d'initialisation, pour pouvoir être sauvegardée. Par exemple:

- Si on doit mélanger des cartes, on fournira l'ordre du tas en retour de l'initialisation.
- Si on a besoin de faire des lancés de dés, on fournira la seed du générateur aléatoire

Le moteur de jeu, a l'initialisation, fournira les données nécessaires pour retrouver l'état déterministe (dans un dictionnaire attaché à la clé "**initvalues**" de ses instructions suivant l'initialisation). On pourra réutiliser ces données pour recréer une partie identique.

III) REJOUER UNE PARTIE/ALLER À UN TOUR

Comme le moteur de jeu est déterministe, on peut recharger une partie à n'importe quel tour en :

1. lançant le moteur de jeu avec les valeurs fournies dans “**initvalues”**
2. renvoyant les actions des joueurs jusqu’au tour voulu ou jusqu'à la fin de la partie

C’est donc le serveur qui devra garder ces informations, pour pouvoir recharger une partie.

IV) MESSAGES

IV.1) INITIALISATION

Un json contenant les paramètres nécessaires au lancement d’une partie (nombre de joueurs, taille de la carte, etc…)

```
{
        “init”:{
                “players”:<nb joueurs”>,
                “<param1>”:”<val param1>”,
                “<param2>”:”<val param2>”,
                “<param3>”:”<val param3>”,
        …
        }
}
```

IV.2) INSTRUCTIONS

En cas de problème : un signalement d’erreurs
Sinon, 3 parties : l’affichage, les actions demandées, et l’état de la partie

```
{
   “errors” : [ <error list> ]
}
```

ou

```
{
   “displays”: [… ],
   “requested_actions”:[ … ],
   “game_state”:{ … }
}
```

IV.2.1) Erreurs

Les erreurs sont décrites chacune dans un dictionnaire, contenant au moins la clé “**type**”, indiquant le type d’erreur.

L’ensemble des erreurs est stocké dans une liste attachée a la clé “**errors**” des instructions envoyées par le moteur de jeu.

A) Général

A.1) BAD_FORMAT

En cas de format incorrect

```
{
   “type”:”BAD_FORMAT”
}
```

B) Lors de l’initialisation

B.1) MISSING_ARGUMENT

Pour un paramètre obligatoire manquant. Contient une clé “**arg**” indiquant le paramètre manquant, comme décrit dans le json descriptif.

```
{
  “type” : ”MISSING_ARGUMENT”,
  “arg” : “<missing argument>”
}
```

B.2) INCORRECT_VALUE

Pour un paramètre avec une valeur incorrecte. Contient :

- une clé “**arg**” indiquant le paramètre manquant, comme décrit dans le json descriptif.
- une clé “**value**” indiquant la valeur fournie

```
{
  “type” : ”INCORRECT_VALUE”,
  “arg” : “<missing argument>”,
  “value” : “<provided incorrect value>”
}
```

B.3) UNEXPECTED_ARGUMENT

Pour un paramètre non reconnu. Contient :

- une clé “**argname**” indiquant le nom du paramètre non attendu.
- une clé “**value**” indiquant la valeur fournie

```
{
  “type” : ”UNEXPECTED_ARGUMENT”,
  “argname” : “<missing argument>”,
  “value” : “<provided value>”
}
```

C) En réponse aux actions fournies

A chaque fois, on aura ici la clé “**player**”, indiquant le joueur concerné.

C.1) UNEXPECTED_ACTION

Quand on fournit une action non demandée (mauvais joueur ou mauvais type d’action)
On aura ici la clé “**action**”, avec en valeur l’action fournie (dictionnaire).

```
{
  “type” : ”UNEXPECTED_ACTION”,
  “player” : <player number>,
  “action” : “<involved action>”
}
```

C.2) MISSING_ACTION

Quand une action demandée est manquante
On aura ici la clé “**requested_action**”, avec en valeur l’action demandée (dictionnaire).

```
{
  “type” : ”MISSING_ACTION”,
  “player” : <player number>,
  “requested_ action” : <requestion action>
}
```

C.3) WRONG_ACTION

Quand une action demandée a une valeur incorrecte.
On aura ici :

- la clé “**action**”, avec en valeur l’action fournie (dictionnaire)
- la clé “**requested_action**”, avec en valeur l’action demandée (dictionnaire).
- la clé “**subtype**” qui indique en quoi l‘action est incorrecte :
    - **OUT_OF_ZONE** : clic en dehors d’une zone demandée
    - **WRONG_BUTTON** : clic avec un bouton non autorisé
    - **KEY_NOT_ALLOWED** : touche pressée non autorisée
    - **UNMATCHED_REGEX** : texte ne correspondant pas au pattern demandé
    - **TEXT_EMPTY** : texte vide
    - **TEXT_TOO_LONG** : texte trop grand

```
{
  “type” : ”WRONG_ACTION”,
  “subtype” : <subtype>,
  “player” : <player number>,
  “action” : <provided action>,
  “requested_ action” : <requested action>
}
```

VI.2.2) Affichage

L’affichage sera décrit par un dictionnaire (format JSON) reprenant les balises SVG et leurs paramètres habituels.

Si un affichage n’est pas fourni pour un joueur, on considère que son affichage reste inchangé par rapport au dernier fourni.

Le dictionnaire contiendra les clés :

- “**player**” pour indiquer le numéro du joueur impliqué
- “**width**” pour indiquer la largeur de l’affichage
- “**height**” pour indiquer la hauteur de l’affichage
- “**content**” pour indiquer le contenu de l’affichage

Le contenu reprend les valeurs des attributs des balises. Le type de chaque balise sera fourni par la clé “**tag**”.

Exemple :
Si on veut que l’affichage du joueur 1 corresponde au SVG suivant

```
<svg width="640" height="480">
  <rect x="10" y="10" width="620" height="460" fill="red" />
  <text x="320" y="240" fill="white">hello</text>
</svg>
```

On fournira le display :

```
{
  “player”:1,
  “width”:640,
  “height”:480,
  “content”:[
    {
      “tag”:”rect”,
      “x”:10,
      “y”:10,
      “width”:620,
      “height”:460,
      “fill”:”red”
    },
    {
      “tag”:”text”,
      “x”:320,
      “y”:240,
      “fill”:”white”,
      “content”:”hello”
    }
  ]
}
```

IV.2.3) Actions demandées

Les actions requises sont définies par une liste de dictionnaires. La plupart du temps, il ne devrait y avoir qu'une action demandée par tour, pour un seul joueur.

Chaque action aura forcément :

- la clé "**player**" pour indiquer le numéro du joueur a qui on demande une action
- la clé "**type**" pour indiquer le type d'action : CLICK, KEY, TEXT

A) CLICK

Pour demander un clic de souris.
On pourra optionnellement avoir en clé :

- "**zones**" : une liste de rectangles indiquant les endroits autorisés au clic (par défaut toute la zone de dessin). Chaque rectangle sera défini par un dictionnaire contenant 4 clés : "x", "y", "width" et "height". La description est similaire à un rectangle en SVG.
- "**buttons**" : une liste des boutons utilisables (par défaut, seulement clic gauche), décrits par les valeurs texte suivantes :
    - LEFT : clic gauche
    - RIGHT : clic droit
    - MIDDLE : clic milieu
    - DOUBLE : double clic
- "**confirm**" : une valeur booléenne pour demander à l'utilisateur si il est sûr de son choix (false par défaut)

```
{
   "type" : "CLICK",
   "player" : <player number>,
   "zones" : [...],
   "buttons":[...],
   "confirm":<true|false>
}
```

Exemple, si on veut demander au joueur 2 de cliquer uniquement dans les zones de 50x50 commençant en 0,0 ou en 100,200, avec uniquement les boutons gauches ou droits, et avec confirmation :

```
{
   "type" : "CLICK",
   "player" : 2,
   "zones" : [{
        "x":0,
        "y":0,
        "width":50,
        "height":50
    },{
        "x":100,
        "y":200,
        "width":50,
        "height":50
    }],
   "buttons":["LEFT", "RIGHT"],
```

```
    “confirm”:true
}
```

B) KEY

Pour demander l’appui sur une touche. Non sensible à la case.
On pourra optionnellement avoir en clé :

- “**keys**” : la liste des touches autorisées, sous forme d’une string (toutes par défaut)
- “**confirm**” : une valeur booléenne pour demander à l'utilisateur si il est sûr de son choix (false par défaut)

```
{
   “type” : “KEY”,
   “player” : <player number>,
   “keys”: “<keys list>,
   “confirm”:<true|false>
}
```

Exemple, si on veut demander au joueur 3 de presser sur Z,Q,S ou D sans confirmation :

```
{
   “type” : “KEY”,
   “player” : 2,
   “keys” : “ZQSD”
  }
```

C) TEXT

Pour demander la saisie d’un texte.
On pourra avoir optionnellement en clé :

- “**regex**“ : l’indication du format (tout par défaut)
- “**max_length**” : une taille max (64 par défaut)

```
{
   “type” : “TEXT”,
   “player” : <player number>,
   “regex”: “<regex to match>”,
   “max_length”:<max length>
}
```

Exemple, si le joueur 2 doit saisir une action parmi ATTACK et MOVE, suivi de deux coordonnées :

```
{
   “type” : “TEXT”,
   “player” : 3,
   “regex”: “^ATTACK|MOVE\s\d{1,2}\s\d{1,2}$”
}
```

IV.2.4) Etat de la partie

L’état de la partie contient :

- le score actuel de chaque joueur : une clé “**scores**” avec en valeur une liste d’entiers, le premier correspondant au premier joueur, le second au second joueur, etc…
- si la partie est terminée : une clé “**gameover**” avec en valeur un booléen qui indique si la partie est terminée (true) ou non (false)

```
{
   “scores” : [...],
   “gameover” : <true|false>
}
```