WGE : Moteur de règles

Maxime COLIN

27 janvier 2011

Résumé

Ce document va présenter le fonctionnement des règles du jeu et les choix adoptés pour les décrire et les configurer. On distinguera la partie *configuration* qui permettra de configurer les données du jeu de la partie *règles* qui permettra de définir les règles du jeu.

Note Ce document est non définitif et est amené à évoluer.

1 Configuration

1.1 Définition

Le fichier de configuration contiendra les données du jeu tel que les différentes unité disponible, leur caractéristiques, les paramètres d'une partie, etc.

1.2 Format

Le fichier sera écrit en JSON.

```
{ "config" :

"something" : {

... configuration de something ...

}

}
```

1.3 Les données configurable

Il faut tout d'abord décrire les différentes données du jeu qui seront configurable. Nous allons commencer par configurer les données basique d'une partie, comme le nombre de joueur, la durée des tours, etc.

```
"game" : {
"name" : "Fightly" //nom du jeu
"player" : "4", //nombre de joueur
"tour" : "60", //durée d'un tour
"type" : "..." //type de la partie
}
```

Ensuite, la carte. On peut définir une carte par sa géométrie, c'est à dire la forme des cases (carré, hexagonale, losange), sont nombre de case en largeur et en hauteur.

```
"map" : {
"geometry" : "square",
"width" : "50",
"height" : "50",
}
```

Les unités.

```
"units" : {
{
"name" : "soldat", //nom de l'unité
"hp" : "100", //point de vie
"move" : "1", //nombre de case de déplacement
"reach" : "1" //porté d'attaque
"resistance" : "fire" //resistance(s)
"attacks" : { //liste des attaques
{ "name" : "", "type" : "", "reach" : "inherit", "power" : "" }
}
},
{
"name" : "cavalier",
"hp" : "200",
"move" : "3",
"reach" : "1"
}
}
```

2 Règles

2.1 Format

Le fichier de règles sera écrit en JSON. Voici un exemple de règle basique.

```
{ "rules" :

"myAction" : {
"if" : { condition },
"do" : { conclusion1, conclusion2 }
},

"anotherActon" : {
"if" : { condition1, condition2 },
"do" : { conclusion }
}

}
```

Les noms des règles "myAction" et "anotherAction" représentent les actions utilisateurs. Ensuite la règles comporte deux partie : "if" qui liste les conditions à réalisé pour valider la règle, et "do" qui liste des actions à réaliser lorsque la règle est validé.

2.2 Les conditions

Les conditions pourront être des appels à des tests existants fournit avec le moteur de jeu et dans le cas d'une utilisation avancé, elles pourront être des tests écrit à la main et faisant appel à des méthodes d'accès aux données du jeu.

Un exemple d'utilisation de tests existant :

```
"attaquer" : {
"if" : { "estAPorter" },
"do" : { conclusion }
},
```

Ici, "estAPorter" fait appel a un test préexistant vérifiant que l'unité attaqué est à porté de l'unité attaquante.

Cette règle aurait également pu être écrit avec un test. "Est-ce que la distance de la cible est plus petit que la porté de l'unité ?" :

```
"attack" : {
"if" : { "left" : "target.distance", "op" = "<", "rigth" : "unit.getReach" },
"do" : { conclusion }
},
```

Dans le cas où plusieurs conditions sont données, un ET logique est appliqué entre chaque condition.

```
"if" : { condition1, condition2 }
```

Pour réalisé un OU logique entre conditions, on créer deux règles avec le même nom. La première satisfaite sera appliquée.

2.3 Les conclusions

Les conclusions sont les actions à réalisé lorsque les conditions sont validé. Il s'agit d'un appel à une fonction du jeu.

```
"moveUnit" : {
"if" : { conditions },
"do" : {
{ "call" : "unit.move", "on" : { "caseDeDestination" } }
}
}
```

"call" permet de donner une fonction de callback, ici *unit.move* et "on" de préciser les paramètres de cette fonction, ici *caseDeDestination*.