Jérémy TREMBLAY
Antoine VITON
Ugo VIGNON
Adrien COUDOUR
Maxime WISSOCQ

# Scriptage des interactions et énigmes – Préambule

## I - Idée générale

L'objectif ici est de scripter notre jeu : définir des règles permettant de modifier le jeu depuis *l'extérieur (*ie. À l'extérieur du programme), pour pouvoir ainsi spécifier une série d'actions qui va s'enclencher après la réalisation d'un ou plusieurs conditions. Par exemple, ouvrir une porte et faire apparaître un objet après l'activation d'un levier, réaliser des cinématiques, etc. Nous avons besoin d'un tel système pour la réalisation de nos énigmes, qui sera ainsi beaucoup plus simples à réaliser, et à intégrer dans le jeu.

## II - Identification des différents scripts

Pour scripter notre jeu, nous avons réalisé un travail de conception jusqu'ici de haut niveau : pour commencer nous avons lister les types de script que nous pouvons créer, regroupés par catégories :

### 1. Scripts dis de Scène.

Correspond aux scripts réalisés a l'entrée et à la sortie d'une carte, autrement dit, dans lors des transitions de cartes. On peut citer :

- Les scripts lors de l'entrée de la carte par le joueur
- Les scripts qui se produisent lors de la sortie de la carte par le joueur

Ils seront utilisés rarement, surtout pour initialiser certaines choses lors des transitions.

### 2. Scripts dits des entités

Correspond aux scripts qui seront réalisés par l'intermédiaire de conditions qui seront vérifiées sur les entités. On peut noter :

- Les scripts réalisés à la création de nos entités
- Les scripts réalisés lors d'une interaction du joueur (lorsque le joueur réalise une action, un déplacement,…)
- Les scripts réalisés lors de la collision de plusieurs entités
- Les scripts réalisés lors de la mise du jour (qui seront exécutés chaque *frame*).

Il s'agit ici du type de scripts que nous comptons le plus utiliser. Lors de certaines actions, collisions où conditions (comme en vérifiant la vie du personnage par exemple), certains de ces scripts vont s'enclencher.

### 3. Scripts dits triggers

Scripts réalisés par l'intermédiaire d'un trigger, qui se déclenche lorsque certaines conditions sont remplies, lors de l'entrée et de la sortie d'une entité dans une certaine zone. On peut citer :

- Les scripts lors de l'entrée dans une zone par une entité
- Les scripts lors de la sortie d'une zone par une entité

On compte les utiliser pour déclencher l'apparition des monstres, de pièges, et éventuellement d'éléments d'interactions à l'entrée d'une salle ou d'une zone particulière. On peut également le faire à la sortie du joueur.

On souhaite ainsi réaliser des scénarios, qui se décheront dans certains cas de figure. Quand un scénario se produira, une série d'action se déroulera, conformément au script qui sera rédigé. Il s'agit ici du but théorique souhaité.

## III - Le support de rédaction

En ce qui concerne la zone où ces règles seront écrites et où ces scripts seront définis, nous avons pensé naturellement à un fichier texte qui sera lu par notre programme et où les scripts se dérouleront lorsque les conditions qui ont été définies seront validées.

Les fichiers texte ne sont pas très *user-friendly* pour les utilisateurs lambda non-programmeurs, et il possèdent l'inconvénient qu'ils sont moins explicites qu'une interface graphique (dans laquelle il sera possible de vérifier chaque ajout de l'utilisateur l'un après l'autre, là où le fichier texte sera lu en une seule fois).

Cependant, nous ne voulions pas réaliser une interface graphique complexe, l'idée est de pouvoir rapidement ajouter des scénarios qui se produiront dans certaines conditions, et qui réaliseront certaines actions. Nous utiliserons donc un fichier texte pour commencer, à voir par la suite si nous complexifierons le processus.

## IV – Les énigmes à incorporer

L'idée est donc de réaliser des énigmes, voire plus généralement des scripts qui s'exécuteront dans certaines conditions. On pense ainsi à pouvoir positionner un levier à un endroit précis de la carte, ce qui permettra de réaliser une série d'actions, comme par exemple, ouvrir une porte et faire apparaître un coffre. On peut imaginer verrouiller une porte lorsque le joueur entre dans une salle et faire apparaître une grande quantité d'ennemis autour de lui. On a ainsi un ensemble de *conditions* qui permettent l'exécution d'un ensemble *d'actions* lorsque ces conditions sont validées.

L'idée maintenant est de lister toutes ces actions, afin de pouvoir cibler et donc limiter les choses qu'il sera possible de faire sur le jeu. Nous sommes obligés de fixer une limite à ce qu'il est possible de faire, car sinon la complexité augmente énormément et il n'est pas possible de toutes manières de réaliser toutes les actions que l'utilisateur souhaite.

On a donc convenu des actions suivantes, et j'ai ajouté entre parenthèses des cas d'utilisation de ces actions, afin de mieux voir dans quel contexte elles pourraient servir (ces actions sont volontairement vagues) :

- Détruire un élément (casser un mur ou des jarres)
- Remplacer un élément (changer une porte verrouillée en porte ouverte)
- Faire apparaître un élément (une potion, une clé, un coffre en précisant son contenu, un escalier…)
- Modifier les statistiques d'une entité (baisser les dégâts d'un ennemi, régénérer la vie du joueur, téléporter le joueur, déplacer une entité…)
- Faire apparaître une entité (des ennemis, des projectiles)
- Faire disparaître une entité (supprimer les projectiles et ennemis à l'écran)
- Bloquer les contrôles lors de certains moments, inverser les contrôles
- Appliquer des effets visuels (comme la caméra qui tremble par exemple)

Je pense que les actions citées ci-dessus couvrent 90 % des cas de figure qui risquent de survenir dans notre jeu. Ils sont vagues et permettent d'exprimer plusieurs actions qui elles seraient plus spécifiques.

Cependant pour plus de spécificité, nous avons pensé à laisser à l'utilisateur spécifier plus en détail sur quels éléments il souhaitait réaliser les actions. Par exemple, il lui serait possible d'appeler l'action « Faire apparaître une entité » en précisant qu'il souhaite faire apparaître un ennemi, de type poursuiveur (qui va venir foncer sur le joueur), dans un rayon de 100 pixels autour du joueur.

# V – Les conditions

Pour que ces actions se déclenchent, il faut qu'un certain nombre de conditions soient satisfaites. On se réfère alors aux différents types de scripts présentés au début.

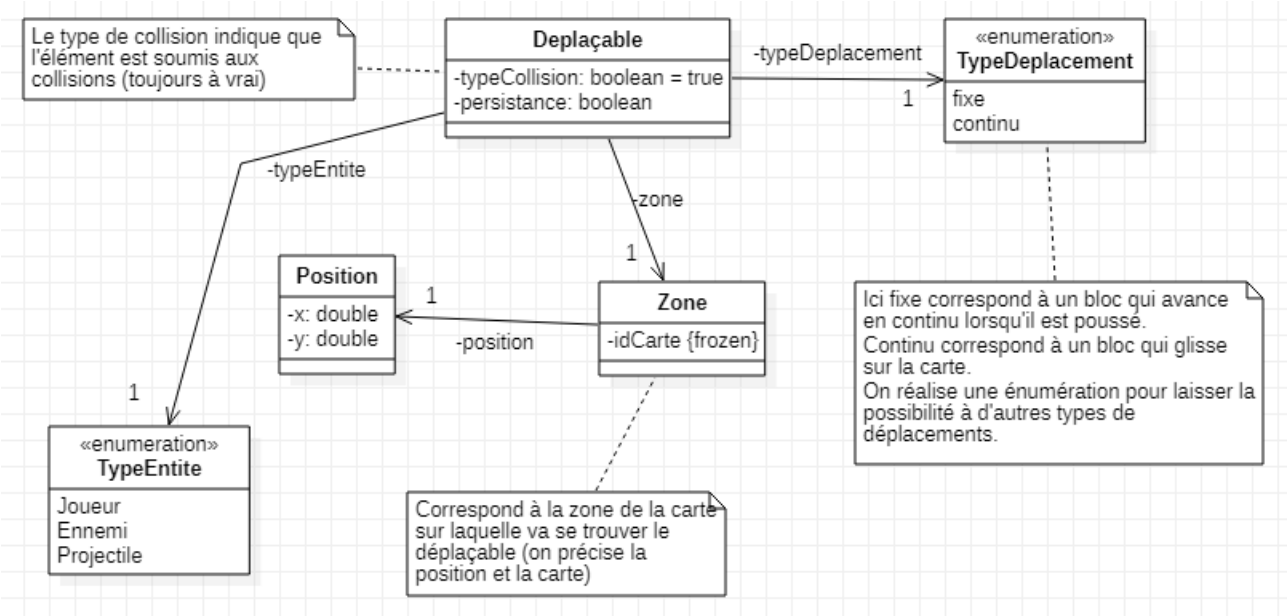
Afin d'être plus précis, on veut laisser la possibilité à notre utilisateur qu'il puisse *choisir* ces conditions. Il doit ainsi pouvoir déclencher sa série d'action lorsque le joueur se trouve en position ($x$, $y$) dans la carte $w$, et lorsqu'il a une vie inférieure à $z$.

Dans la théorie, cela serait le scénario idéal. Mais chaque condition ajoutée comme possibilité dans le fichier vient complexifier sa mise en œuvre dans le programme. Bien évidemment, nous n'allons pas commencer à ajouter des affectations ou des variables qui viendraient encore plus complexifier le processus.

Nous avons définis ces conditions de manière simplistes. Nous avons réfléchis dans quel cas une série d'action pourrait se produire : lorsqu'un levier est activé, lorsqu'un mur est cassé ou quand un bloc est poussé.

En ce qui concerne le bloc poussé, nous avons imaginé pouvoir préciser dans le fichier de script un élément qui peut-être *déplacé* comme ayant une position (il faut qu'on puisse savoir à quel endroit celui-ci se trouve sur la carte). Pour cela, il serait possible de spécifier une zone, qui se composerait d'un identifiant correspondant à la carte sur laquelle se trouve le *déplaçable* ainsi qu'une position ($x$, $y$) (en pixel ou en tuiles). Il serait possible de préciser quel type d'entité peut pousser ce déplaçable (un joueur ? Un ennemi ? Un projectile?), ainsi que l'information permettant de savoir si la position de ce *déplaçable* est persisté (c'est-à-dire, est-ce qu'on doit sauvegarder sa position lors d'un changement de carte?). Enfin, il doit être possible de savoir si cet élément « glisse » sur le sol une fois poussé (comme sur de la glace) où s'il avance que par à-coups (lorsqu'on le pousse il avance régulièrement).

On se retrouve alors avec quelque chose comme cela :




Ainsi dans l'idéal, on aimerait paramétrer dans le fichier de script l'endroit sur laquelle va se trouver le déplaçable (autrement dit, la *zone*), ainsi que le type de déplacement que celui-ci aura, s'il sera persisté ou non, ainsi que le type d'entité qui pourra le déplacer. Seul le paramètre de collision restera toujours bloquant.

On aimerait faire quelque chose de similaire avec des *actionnables* (correspond à des leviers, boutons ou prismes), et des *destructibles* (qui vont disparaître dans certaines conditions).

La condition, ce serait en réalité l'action qui va se produire associée au nom de l'élément (ce n'est pas clair du tout et j'ai du mal à l'expliquer). Si par exemple un déplaçable est déplacé par le joueur (quand celui rentre en collision avec), alors la condition sera validée, et toutes les actions définies en dessous seront exécutées.

Si un *actionnable* est actionné (par exemple la collision d'un coup d'épée du personnage entre en collision avec le levier, ou le personnage marche sur un bouton, ou un projectile touche un prisme), alors les actions suivantes seront évaluées (ouvrir une porte, faire apparaître des ennemis, etc).

On imagine une chose similaire pour un *destructible* (quand il est détruit, alors les actions se produisent).

On peut ajouter d'autres conditions qui seraient basées sur les points de vie du personnage, sa position (des *triggers*, avec des détections lors de l'entrée ou de la sortie d'une une zone), et bien d'autres choses encore.

# VI – Un exemple pour mieux comprendre

Un exemple vaut parfois mieux que mille mots, et donc voici ce qu'on a imaginé comme fichier texte, qui préciserait ainsi différentes règles avec les conditions associées.

```
//Ici on a la condition : un élément déplaçable est créé en interne dans le programme sur la carte 2 à la position 43, 23, qui ne peut être poussé que par des joueurs, qui avance de manière fixe et qui est sauvegardé au changement de carte.
DEPLACABLE ZONE 1 43 23 TYPE_ENTITE Joueur TYPE_DEPLACEMENT Fixe PERSISTANCE Vrai
//Si cette condition est vérifiée, autrement dit si ce déplaçable est déplacé, alors on exécute les actions définies après.
        APPARITION_ENTITE ZONE 1 12 13 TYPE_ENTITE Ennemi TYPE_COMPORTEMENT Poursuiveur
        REMPLACER_ELEMENT TYPE_ELEMENT Porte
        MODIFIER_STATISTIQUES TYPE_ENTITE Joueur STATISTIQUE PointsDeVie VALEUR 100

//Si le joueur entre dans la zone rectangulaire de coordonnées (34, 56) à (38, 58) sur la carte 3, alors on exécute les actions
POSITION TYPE_DETECTION Entree TYPE_ENTITE Joueur ZONE 2 34 56 38 58
         APPARITION_ENTITE ZONE 4 12 13 TYPE_ENTITE Projectile TYPE_COMPORTEMENT DeplacementLigneDroite DIRECTION Droite
         APPARITION_ENTITE ZONE 4 15 18 TYPE_ENTITE Projectile TYPE_COMPORTEMENT DeplacementLigneDroite DIRECTION Gauche

//Déclare un levier actionnable que par le joueur lorsqu'il effectue une attaque à une certaine position (32, 56) sur la carte 5, non persisté, et ne pouvant pas être actionné de nouveau (« bloquant »)
ACTIONNABLE ZONE 4 32 56 TYPE_ACTIONNABLE Levier TYPE_ENTITE Joueur COLISION ZoneAttaque TYPE_COLISION Bloquant
         TELEPORTATION_ENTITE TYPE_ENTITE Joueur ZONE 1 32 12
```

Des questions restent encore en suspend avec cette manière de fonctionner : avec une implémentation similaire, il va y avoir un très grand nombre d'énumérations différentes (une pour les types d'ennemis, pour les types d'entité, pour les types d'éléments, pour les types de déplacements, pour les types d'actionnables…). De plus, à chaque type d'ennemi ajouté ou chaque nouveau comportement, il faudra ajouter d'autres énumérations...

De plus, la lecture du fichier va être la véritable difficulté de ce projet : les arguments risquent d'être placés dans le désordre, il faudra retrouver les bons, les remettre dans le bon ordre, les parser… J'ai du mal à voir comment faire très précisément, même si je me doute que le patron interprète pourrait nous aider à lire ces données.

L'avantage de ce scriptage est que cela nous permettrait de directement pouvoir créer des scénarios et énigmes différentes en fonction du nombre de joueurs dans le multijoueur par exemple.

Il reste encore des problèmes au niveau du placement des leviers, et autres déplaçables (ou caisses). Pour instancier ces éléments, il me faut connaître leurs dimensions, il faut donc inscrire en dur quelque part les dimensions des images (donc des données de la vue qui peuvent être amenées à changer), afin que les détections sur les zones de collisions se fassent correctement. Il faut également préciser en dur les coordonnées de chaque élément dans un fichier texte, mais je ne vois pas comment le faire autrement.

Peut-être partons nous dans une mauvaise direction, j'espère avoir été un minimum clair dans mes explications.