

Génie des Logiciels et des Systèmes: Modélisation, Vérification et Génération de Jeux

Kévin CARENOU

Thibault MEUNIER

Matthieu PERRIER

Sacha VANLEENE

15 décembre 2016

Introduction

Ce projet a pour objectif de modéliser un jeu via un langage propre, de s'assurer de sa faisabilité par un mécanisme de vérification automatique et d'offrir la possibilité d'y jouer.

Il s'agit d'un outil désirant simplifier le travail de son utilisateur. Lorsque celui-ci conçoit le jeu de demain, il est aidé par une syntaxe simple, des mécanismes de vérification, une exécution rapide dans une interface dédiée. C'est pourquoi chaque étape du développement a été pensée pour être utilisable sans friction.

Plusieurs jeux de démonstrations ont ainsi été créés afin de confronter nos idées à leur mise en pratique. Par souci de simplicité, ce rapport ne retiendra que l'exemple du jeu d'énigme décrit ci-dessous. "Le territoire est composé de trois lieux, un de début nommé Enigme et deux de fin qui représentent le succès, nommé Succès, et l'échec, Echec. Ces trois lieux sont qualifiés par leur nom. Le nombre de réponses possibles est représenté par un objet Tentative dont l'explorateur possède un nombre initial, par exemple 3. Le lieu Enigme contient une personne Sphinx qui est visible et obligatoire. Cette personne est qualifiée par le texte de la question. Son interaction contient un choix dont chaque action est qualifiée par les réponses possibles. L'action associée aux mauvaises réponses consomme un objet Tentative. L'action associée aux bonnes réponses donne une connaissance Réussite. Il existe un chemin obligatoire allant du lieu Enigme au lieu Succès dont la visibilité est conditionnée par la possession de la connaissance Réussite. Il existe un chemin obligatoire allant du lieu Enigme au lieu Echec dont la visibilité est conditionnée par la possession d'un nombre d'objet Tentative égal à 0."

Le projet s'est révélé ambitieux et le présent rapport s'attache à en montrer une version simplifiée mais fonctionnelle. Les divers choix que nous avons été amenés à prendre seront détaillés et expliqués. Les documents présentés sont le fruit d'une réflexion commune au membre du groupe et le résultat de dialogues répétés.

Sommaire

1	Syntaxe Textuelle	3
1.1	Langage Game	3
1.2	Metamodelle et contraintes statiques	3
2	Syntaxe Graphique	7
2.1	Point de vue Territoire	7
2.2	Point de vue Interaction	7
3	Verification en PetriNet	8
3.1	Transformation vers Petrinet	8
3.2	Terminaison du jeu	9
4	Generation de code	9
4.1	Initialisation du jeu	9
4.2	Code et Metamodelle	9
4.3	Execution	9

1 Syntaxe Textuelle

1.1 Langage Game

Pour modeliser son jeu, l'utilisateur devait disposer d'une syntaxe textuelle simple. Le choix d'un langage s'approchant d'une description naturelle nous a semble une bonne reponse a ce probleme. En limitant le nombre de mot cle et en laissant beaucoup de parametres facultatifs, l'utilisateur beneficie d'une grande liberte dans la realisation de son jeu. Les declarations s'eloigne d'une description ordonnee des donnees au profit d'instantiations masquees. Nous avons egalement eu le soucis de la factorisation du code en ne faisant pas repete a l'utilisateur des informations identiques. Ceci est gere par la creation dynamique de lien.

```

1 |jeu JeuEnigme dans Enonce avec Joueur difficulte facile
2 |territoire Enonce couvre Debut Succes Echec Enigme Initialisation BonChemin MauvaisChemin
3 |lieu Debut position depart
4 |lieu Succes position fin
5 |lieu Echec position fin
6 |lieu Enigme accueille Sphinx
7 |chemin Initialisation depuis Debut vers Enigme
8 |chemin BonChemin depuis Enigme vers Succes passage obligatoire visible selon BonneFin
9 |chemin MauvaisChemin depuis Enigme vers Echec passage obligatoire visible selon MauvaiseFin
10 |objet Tentative mesure 1
11 |connaissance Reussite
12 |explorateur Joueur possede 3 Tentative porte 3
13 |personne Sphinx passage obligatoire offre Question
14 |choix Question propose BonneReponse selon EncoreTentative offre FinQuestion, MauvaiseReponse offre Question selon EncoreTe
15 |choix FinQuestion position fin
16 |action BonneReponse possede Reussite
17 |action MauvaiseReponse consomme 1 Tentative
18 |condition BonneFin possede Reussite
19 |condition MauvaiseFin possede exactement 0 Tentative
20 |condition PremiereQuestion possede exactement 3 Tentative
21 //Voici un commentaire
22 |condition EncoreTentative possede plus 0 Tentative

```

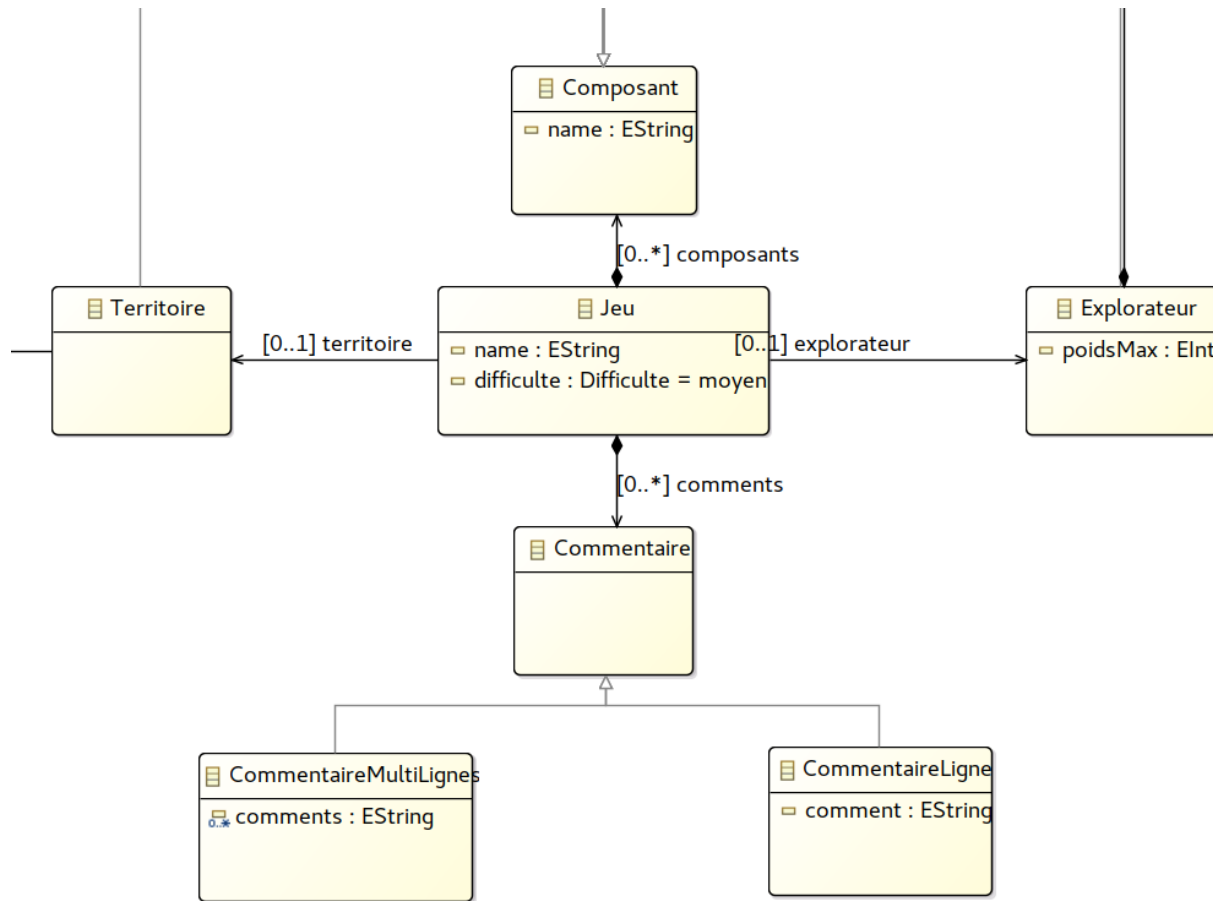
Un application directe de cette creation a la volee est les objets. L'utilisateur declare simplement un objet simple (ici Tentative) et un objet comprenant la quantite est cree a chaque fois que celui-ci est utilise (avec 3 Tentative par exemple, le modele cree un objet "3 Tentative") sans declaration explicite.

On note egalement la possibilite d'effectuer des commentaires. Ceux-ci sont egalement accessible au travers du modele decrit ci-apres. Il serait donc possible de permettre a un utilisateur avance d'agir sur la generation du code par l'intermediaire de macro (semblable a la JavaDoc). Les fichiers de declaration de jeu seront enregistres au format ".game".

1.2 Metamodelle et contraintes statiques









L'interet d'utiliser Xtext est de beneficier des outils de generation de modele. Les fichiers modeles seront regroupes sous l'extension ".gamemodel". Notre interet s'etant focalise sur la simplicite du langage, le modele genere est assez complexe et nous allons donc detaille seulement les parties principales.

Le jeu se presente de la facon suivante:



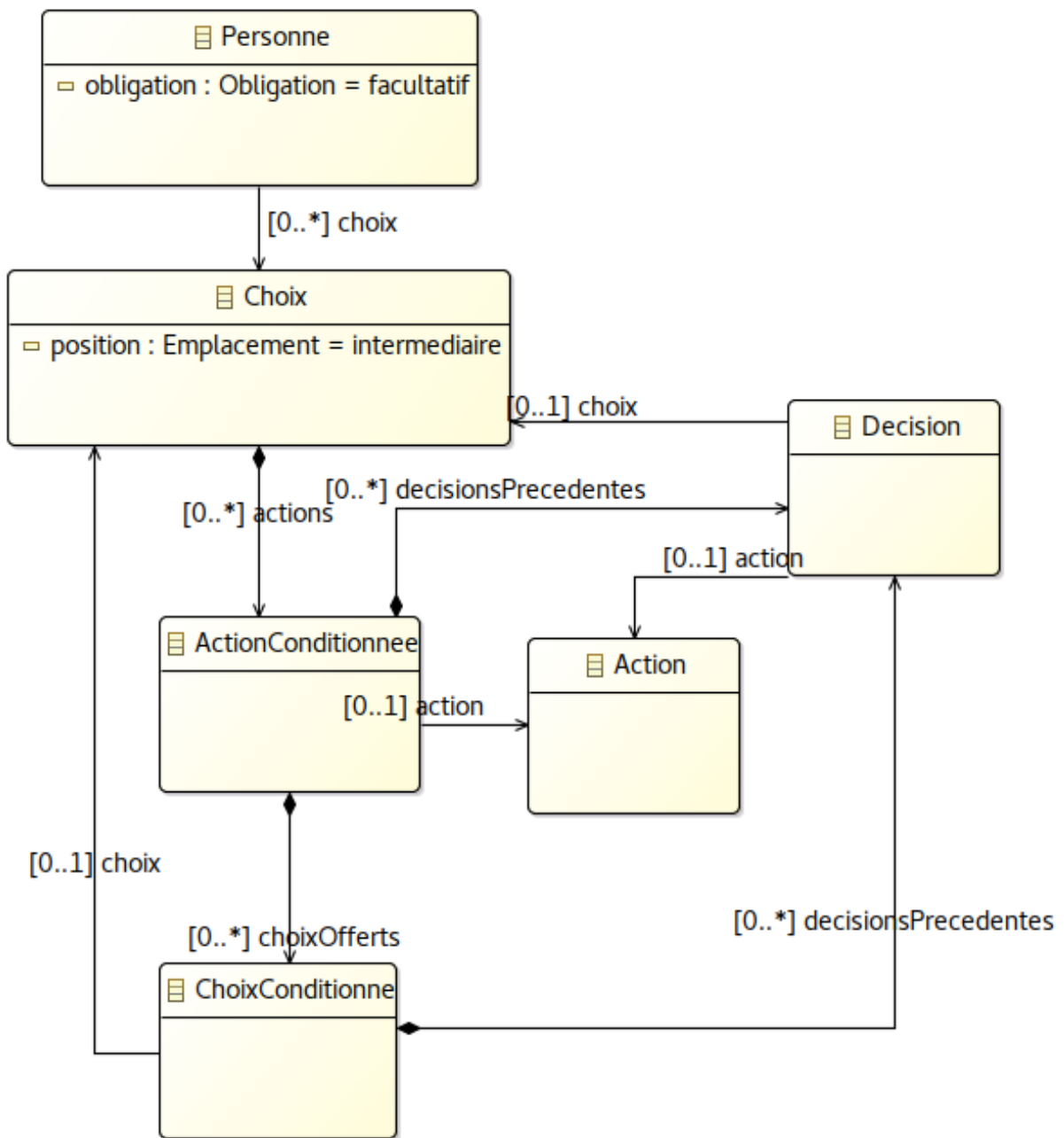
Les composants sont les objets declares explicitement par l'utilisateur. Il y a autant de composant que de declaration dans le fichier du jeu. Ils presentent tous un nom permettant de les identifie. Les objets dont l'instanciations est masquee ne sont pas considere comme des composants et on un identifiant unique genere par Xtext. Notons au passage que le territoire et l'explorateur sont des composants. Plusieurs peuvent avoir ete declares mais un seul de chaque sera lie au jeu.

L'un des points essentiels est l'absence de boolean dans le modele. Nous leur avons prefere l'emploi d'enumeration, plus explicite et modulable. Une relation binaire de prime abord peut se devoiler etre plus complexe et necessite la definition d'un troisieme etat.

<div> Difficulte</div> <div><ul style="list-style-type: none">- Moyen- Facile- Difficile</div>	<div> Visibilite</div> <div><ul style="list-style-type: none">- Visible- Invisible</div>	<div> Obligation</div> <div><ul style="list-style-type: none">- Facultatif- Obligatoire</div>
<div> Emplacement</div> <div><ul style="list-style-type: none">- Intermediaire- Depart- Fin</div>	<div> Ouverture</div> <div><ul style="list-style-type: none">- Ouvert- Ferme</div>	<div> Transformabilite</div> <div><ul style="list-style-type: none">- Immuable- Transformable</div>
<div> Comparateur</div> <div><ul style="list-style-type: none">- Egal- Superieur- Inferieur</div>	<div> Operateur</div> <div><ul style="list-style-type: none">- EtLogique- OuLogique- NonLogique</div>	

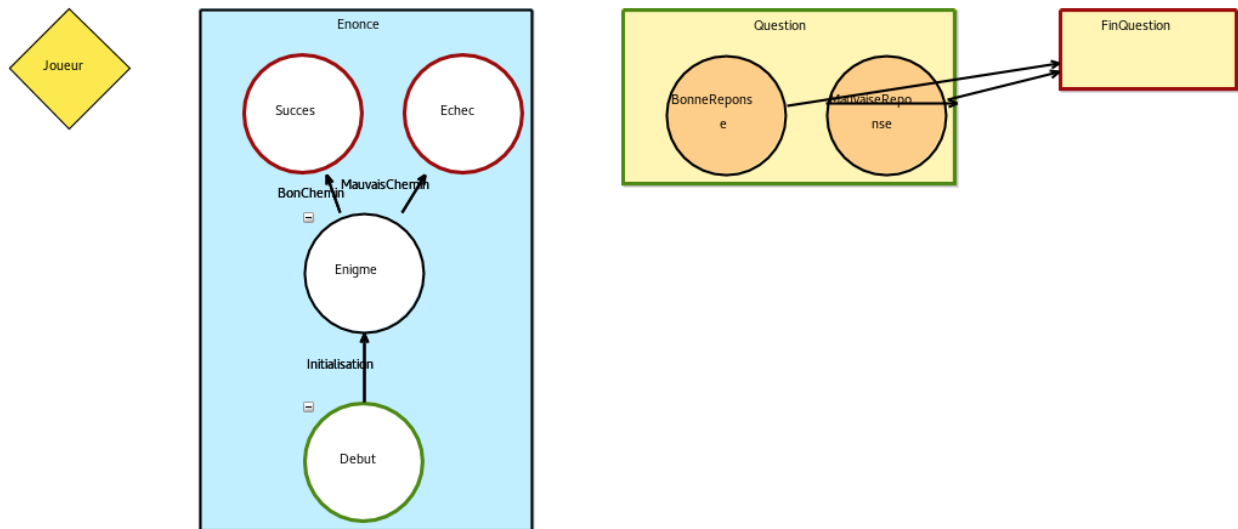
La variété des enumerations permet de decrir de multiples etats et offre une panoplie de possibilite au programmeur souhaitant implante le modele.

Pour la gestion des interactions, nous avons decide qu'une personne proposerait un choix contenant plusieurs actions possibles. Ces actions menerait a d'autre choix permettant de faire avancer la discussion.



2 Syntaxe Graphique

Comme pour la syntaxe textuelle, la syntaxe graphique c'est voulue simple. Elle ne permet donc pas de visualiser un jeu dans son ensemble mais seulement les points délicats que sont le territoire et les interactions.



2.1 Point de vue Territoire

Visualiser les lieux et leurs connections et un point essentiel à la progression du jeu. Les lieux de départ présentent un liséré vert, ceux de fin un rouge et les autres restent noirs. Il est également possible de retracer certains lieux afin d'avoir une vue plus complète de l'ensemble. L'utilisateur a également la possibilité d'ajouter des lieux ou des chemins directement depuis Sirius.

2.2 Point de vue Interaction

Les interactions furent plus techniques à implémenter, notamment du fait des multiples conditions sur ces objets. Une interaction se constitue par une suite de choix et d'action. Les actions étant proposées par les choix, elles sont dans ceux-ci, puis reliées aux choix vers lesquelles elles mènent.

3 Vérification en PetriNet

Afin de vérifier les contraintes dynamiques du jeu via l'outil Tina, il a été nécessaire de pouvoir réaliser un réseau de Pétri modélisant le plus précisément possible le jeu.

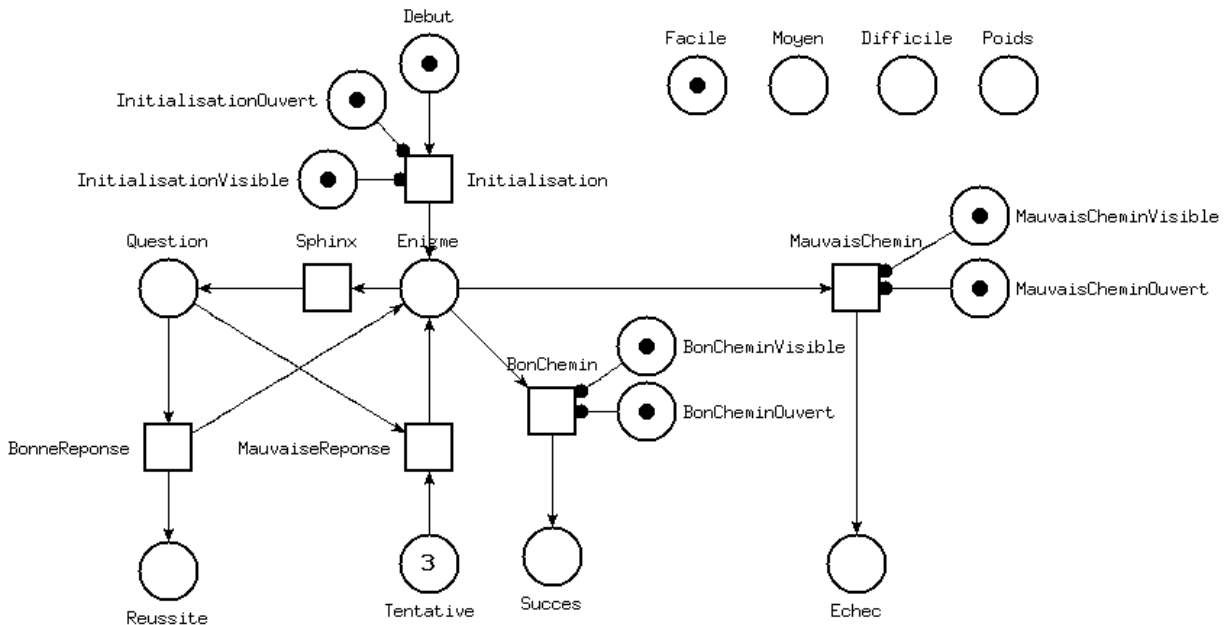
3.1 Transformation vers Petrinet

La transformation vers Petrinet s'est faite via l'outil ATL.

Pour contourner un problème d'instanciations non-maitrisées (par exemple, des objets présents dans des conditions se retrouvaient instanciés comme étant des objets réels), nous avons utilisé et couplé les *lazy rules*, les *called rules* et les sections impératives *do*. Ainsi, le modèle est parcouru manuellement et les éléments sont instanciés de façon maitrisée : deux éléments ne seront pas traités de la même façon selon leur utilité réelle déterminée par leur position dans le modèle.

De plus, pour représenter les conditions du type "*quantité d'un objet est supérieur / inférieure / égale à n*", nous avons utilisés la combinaison d'arc classiques et d'arcs inhibiteurs découverts lors du parcours de la documentation de l'outil *nd*.

Ainsi, bien que la transformation ne soit pas complète (les conditions ne sont pas présentes, la transformation des objets et le fait de poser un objet dans un lieu ne sont pas modélisés), elle permet d'obtenir une bonne représentation du jeu sous forme de réseau de Pétri.



3.2 Terminaison du jeu

Une transformation vers LTL a été créée dans le but de vérifier les contraintes dynamiques. Des templates ont été créés pour faciliter l'écriture de relations logiques.

Les contraintes dynamiques actuellement testés sont la terminaison possible du jeu ainsi que l'unicité de la présence dans un lieu (pas de dédoublement de l'explorateur).

4 Génération de code

Nous avons décidé d'implanter ce jeu en Java.

4.1 Initialisation du jeu

Le jeu s'insère directement dans un fichier exécutable contenant une unique méthode main et utilisant des packages codés préalablement. Le code est commenté pour permettre à un utilisateur de s'y retrouver plus facilement. Le choix de programmation pris a été de d'abord déclarer l'ensemble des objets, puis de créer les liens entre ces éléments. Ceci a évité de devoir utiliser des constructeurs et d'autres acrobaties dans le code Acceleo.

4.2 Code et Metamodelle

EMF offre des outils pour travailler directement avec un modèle en Java par l'intermédiaire de Factory. Nous n'avons pas retenu ce choix, le modèle généré par Xtext étant trop confus. L'absence d'interface, de classe abstraite ou même de valeur par défaut (sans passer par les traitements post-génération, complexe et non recommandé) nous a conduit à faire un modèle complet en Java.

Celui-ci suit les principes Modèle-Vue-Contrôleur. L'API est utilisable directement pour les utilisateurs aguerris, mais ceux-ci ne disposeront pas des outils de vérification.

4.3 Exécution

Après compilation du fichier généré par Acceleo, l'utilisateur arrive sur l'interface textuelle présentée page suivante. Celle-ci est sobre et offre à l'utilisateur de multiples propositions. Du fait de la modularité offerte par l'API, les utilisateurs désireux d'avoir une interface plus conviviale et visuelle peuvent laisser libre cours à leur imagination pour construire une vue graphique.

```
***** Objets Possedes *****
Id  Quantite Nom                                Poids
1)      3 Tentative                             0
*****
Poids Total : 0/3
*****
```

Que voulez vous faire ?

- (1) Lister les connaissances et les objets possédées
- (2) Lister ce qui est visible dans le lieu (personnes + objets)
- (3) Déposer un objet
- (4) Prendre un objet
- (5) Interagir avec une personne
- (6) Choisir un chemin
- (7) Transformer un objet

2

```
***** Objets Visibles *****
Il n'y a aucun objets
***** Personnes Visibles *****
Il n'y a personne
*****
```

Que voulez vous faire ?

- (1) Lister les connaissances et les objets possédées
- (2) Lister ce qui est visible dans le lieu (personnes + objets)
- (3) Déposer un objet
- (4) Prendre un objet
- (5) Interagir avec une personne
- (6) Choisir un chemin
- (7) Transformer un objet

3

```
Id  Quantite Nom                                Poids
1)      3 Tentative                             0
Choisissez un objet
```

Conclusion

Ce projet fut un projet ambitieux. L'objectif principal a été de rendre une version fonctionnelle fonctionnant sur des exemples simples, notamment celui présenté dans ce rapport. Ce dernier a été atteint à l'aide d'un dialogue permanent entre les membres du groupe. Des changements de modèle ont été opérés au fur et à mesure d'une compréhension plus fine du sujet et des caractéristiques techniques qui lui sont associées.

De la génération à l'exécution en passant par la vérification, nous avons fait face à de multiples difficultés. Documentation rare, interface utilisateur complexe, messages console cryptiques sont les principaux problèmes auxquels nous avons été confrontés. L'idée fut cependant plus stable que lors du bilan d'étude, chaque membre travaillant sur une partie spécifique et l'intégration des différents composants se faisant de manière plus progressive.

Nous laissons libre cours à votre imagination pour créer les jeux de demain.