3DWIGS

3D Web Interactive Game Studio

Manuel d'utilisation

Equipe Mini3D : Berlon Antoine, Bouzillard Jérôme, Chegham Wassim, Clergeau Thomas, Faghihi Afshin, Guichaoua Mathieu, Israël Quentin, Kien Emeric, Le Corronc Thibault, Le Galludec Benjamin, Le Normand Erik, Lubecki Aurélien, Marginier David, Sanvoisin Aurélien, Tolba Mohamed Amine, Weinzaepfel Philippe et Zadith Ludovic

Responsable Projet : Docteur Rémi Cozot

Année de réalisation : Année Universitaire 2010-2011

Ce manuel a été créé dans le but de faciliter la prise en main du logiciel auteur qu'est 3DWIGS. Pour rappel, 3DWIGS, pour 3D Web Interactive Game Studio est un programme de création de contenus 3D interactifs au sein de pages Web.

Dans un premier temps, il sera expliqué comment utiliser le langage de description que 3DWIGS met à votre disposition pour la description de vos mini-jeux. Dans une seconde partie, l'interface d'utilisation du logiciel sera décrite afin de faciliter sa prise en main. Et enfin, en annexe, vous trouverez un lexique du langage de description de 3DWIGS accompagné d'un exemple. Pour les utilisateurs les plus expérimentés, une description des fonctionnalités du moteur y sera également présente dans le cas où le langage de description fourni ne suffirait pas à la réalisation de votre projet vidéoludique.

L'équipe de développement de 3DWIGS vous remercie pour l'usage de son logiciel et vous souhaite une bonne réussite dans votre réalisation.

Table des matières

I. PRÉSENTATION ET PRISE EN MAIN DE L'INTERFACE DE 3DWIGS

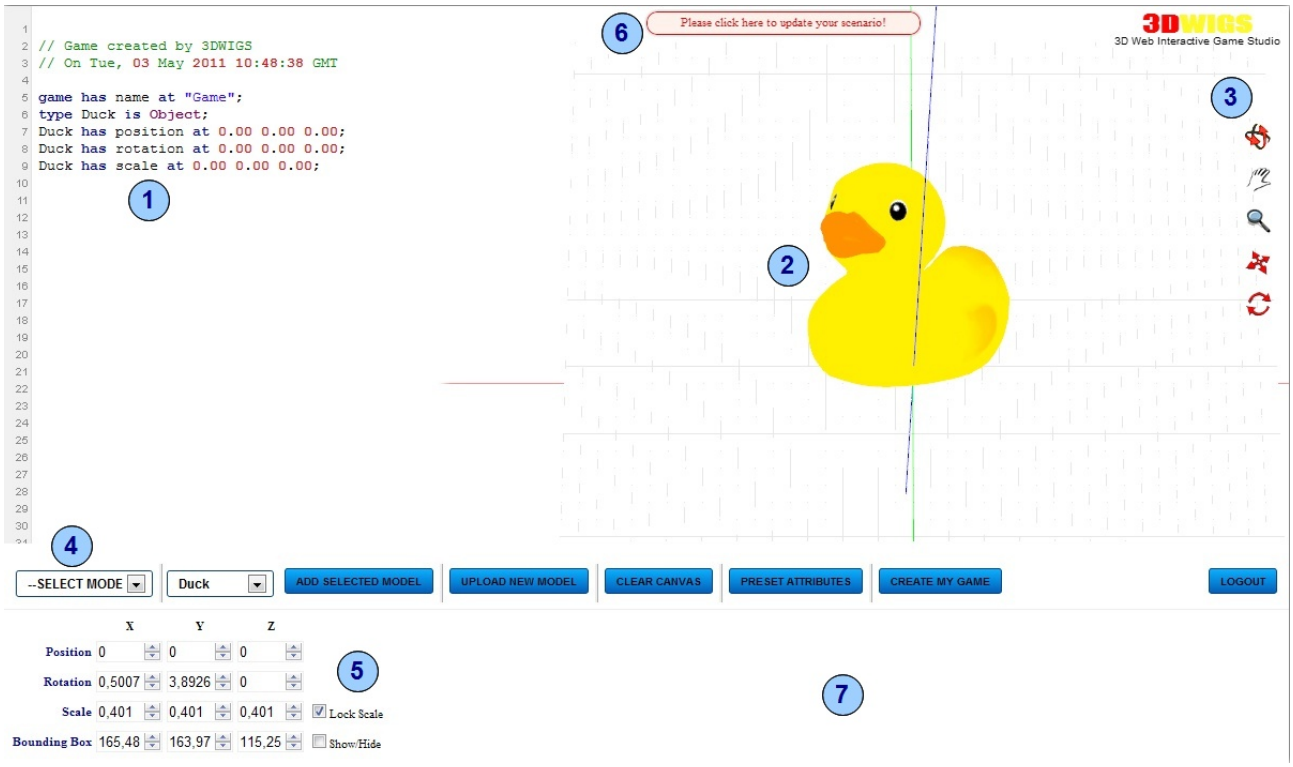


Illustration 1: Interface du logiciel 3DWIGS

1. La zone de code

Dans cette zone, l'utilisateur devra saisir sa description du jeu en respectant la grammaire 3DWIGS.

2. La zone d'affichage

La zone d'affichage est réservée à l'affichage des éléments du jeu. Lorsqu'un modèle est importé, il est inséré dans le contexte 3D et donc affiché dans cette zone.

3. Le menu latéral

Ce menu est associé avec la zone d'affiche. Il permet de (du haut vers le bas) :

- ✔ Effectuer une rotation autour d'un élément.
- ✔ Effectuer un déplacement de la camera suivant un plan.
- ✔ Effectuer un zoom avec la caméra.
- ✔ Déplacer l'élément sélectionné dans le contexte 3D.
- ✔ Effectuer une rotation de l'élément sélectionné autour de ses axes X, Y et Z.

4. Le menu principal

Ce menu permet à l'utilisateur de (de gauche à droite) :

✔ Sélectionner un modèle parmi ceux présents dans le contexte 3D, afin d'afficher ses propriétés dans la zone des propriétés.

✔ Sélectionner un modèle depuis la bibliothèque des modèles disponibles.

✔ Ajouter le modèle sélectionné depuis le menu b) dans le contexte 3D courant.

✔ Importer un nouveau modèle depuis le poste de l'utilisateur dans la bibliothèque 3DWIGS. Cet import déclenche l'ajout de l'élément importé dans le contexte 3D.

✔ Réinitialiser la session courante. ATTENTION ! Cette action efface tout le contenu de la base de données ainsi que la zone d'affichage. Cependant les modèles importés ne sont pas supprimés du serveur.

✔ Afficher la liste des modèles logiques ainsi que leurs attributs.

✔ Finalement ! Créer le jeu.

5. La zone des propriétés

Cette zone permet d'afficher les propriétés du modèle sélectionné. Elle permet également de modifier les valeurs de certains attributs de ce modèle.

6. Bouton de synchronisation

Ce bouton apparaît lors d'une modification du positionnement d'un model. Elle permet de changer les valeurs associées au model dans l'éditeur de texte et de sauvegarder le contenu du jeu en local.

7. La zone libre

Cette zone ne sert à rien pour l'instant. Mais pourrait servir à accueillir de nouvelles fonctionnalités à venir

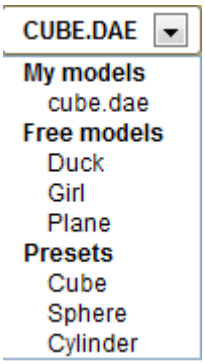
8. Tutoriel d 'utilisation de l'IHM

Manipulations à suivre pour créer un jeu via l'IHM :

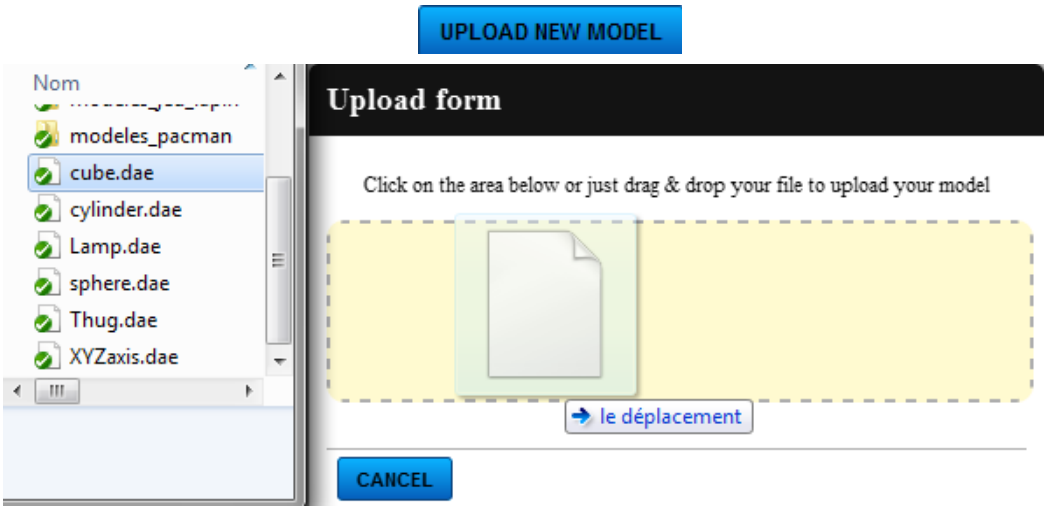
✔ On se connecte au site de 3DWIGS avec une version de navigateur web supportant HTML5 (Firefox, Chrome ..), aucune installation n'est requise.

✔ Après authentification, une fenêtre apparaitra vous demandant le nom de votre jeu, pour l'exemple ici nous mettrons « JeuDuLapin ».

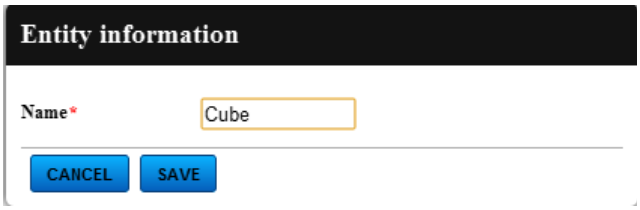
✔ Dans un premier temps vous devrez charger les modèles sur la scène et les placer. Pour cela vous pouvez utiliser des modèles déjà présents sur le site via le menu déroulant et les charger sur la scène avec le bouton :



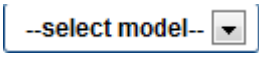
✔ Vous avez également la possibilité de charger des modèles externes (collada en .dae) via le bouton :



✔ Une fois le fichier collada (extension dae) glissé sur cette zone, une fenêtre apparaitra afin de renseigner le nom du modèle et celui-ci sera alors placé sur la scène avec des paramètres par défaut dans l'éditeur.



✔ Le modèle étant chargé sur la scène, vous pouvez le sélectionner via le menu déroulant de gauche :



et le déplacer en modifiant ses coordonnées en dessous.

Afin de sauvegarder vos modifications le plus souvent possible, n'oubliez pas de cliquer sur :

✔ Après avoir chargé et placé vos modèles, vous devrez également, dans la partie éditeur de texte, créer les règles associées au comportement désiré de votre jeu en utilisant le langage de 3DWIGS décrit dans la partie suivante.

II. DESCRIPTION ET APPRENTISSAGE DU LANGAGE DE DESCRIPTION

1. Premières notions de programmation avec 3DWIGS

a. Je commente, nous commentons, vous commentez

La description d'un jeu peut être relativement longue. En conséquence vous pouvez utiliser des commentaires qui vous permettront d'organiser votre code. Leur utilisation se fait à l'aide d'un double slash.

```
//Ceci est un commentaire, le logiciel ignorera cette ligne lors de la
//génération du jeu.
```

b. Vocabulaire

Dans 3DWIGS, l'ensemble des personnages ('Character'), des véhicules ('Vehicle') ou des obstacles ('Obstacle') sont appelés des types (annexe pour voir tous les types existants). Ils seront écrits avec une majuscule par la suite. Un élément de cet ensemble est appelé objet et on l'appelle par un identifiant. La vitesse de l'objet, sa couleur ou sa vie sont quant à eux, des attributs.

c. Les types primitifs

i. Les nombres

Les nombres utilisables dans 3DWIGS sont de deux natures : entier relatif ou décimal. Un décimal utilise le point comme séparateur, exemple pour pi : 3.14.

ii. Les booléens

Un booléen peut prendre uniquement deux valeurs : true ou false (pour vrai ou faux).

iii. Les chaînes de caractères

Une chaîne de caractères est une séquence de caractères comme "loup".

iv. Les identifiants

Un identifiant appartient à une sous-famille des chaînes de caractères. Le logiciel n'accepte que des identifiants commençant par une lettre et composés de caractères alphanumériques auxquels il faut ajouter le tiret - et le tiret bas _.

v. Les listes

Une liste est un ensemble d'objets regroupés ensemble et le nom de la liste est un identifiant.

vi. Les énumérations

Une énumération est un attribut ayant pour choix de valeur plusieurs identifiants. Par exemple, un booléen est une énumération primitive de deux valeurs ('true', 'false').

d. Les types prédéfinis

Vous avez à disposition des types prédéfinis qui se réduisent à des types comprenant un certain nombre d'attributs couramment utilisés, cf. liste en annexe. 3DWIGS utilise un langage qui lui est propre pour créer des jeux. Ce langage décrit les différents éléments d'un jeu. Tout comme n'importe quel langage de programmation, ce langage nécessite un petit apprentissage. Voici le détail de son utilisation.

2. Organisation du jeu

Un code écrit avec 3DWIGS est découpé en sept parties :

1. *Partie* ***Description du jeu***
2. *Partie* ***Déclaration de nouveaux types***
3. *Partie* ***Initialisations***
4. *Partie* ***Définitions***
5. *Partie* ***Commandes***
6. *Partie* ***Règles du jeu***
7. *Partie* ***Intelligence Artificielle***

Il faut que le code soit écrit dans cet ordre pour qu'il soit valide. Un ';' doit obligatoirement être mis à la fin de chacune de ces parties ou de leurs sous-parties. C'est une ponctuation qui doit être aussi naturelle que de mettre un point à la fin d'une phrase en français.

3. Description des phases de création d'un jeu

Afin de mieux vous expliquer, chaque partie sera illustrée par un exemple issu d'un jeu créé avec le logiciel. Le joueur contrôle un chasseur possédant un bazooka et devant tirer sur des lapins. Un lapin apparaît et suit un parcours prédéfini. Le chasseur doit le tuer en lui tirant dessus en un temps imparti. Ensuite un nouveau lapin apparaît, le joueur gagne au bout de 3 lapins tués.

a. Description du jeu

Cette partie permet de décrire le jeu de manière globale (classe game) :

```
game has [attribut] at [valeur de l'attribut];
```

Les attributs sont :

- ✔ name : le nom de votre jeu (chaîne de caractères).
- ✔ gravity : la gravité globale de l'environnement 3D (soit une valeur numérique pour une gravité verticale vers le bas, soit 3 valeurs numériques pour un vecteur).
- ✔ world : l'environnement de jeu (une énumération). Peut être soit 'generic' pour un environnement neutre, 'grid' pour un environnement découpé en cases et 'ribbon' pour un environnement linéaire adapté pour les jeux de type course.
- ✔ gridsize : la taille d'un carreau de la grille si l'attribut 'world' de 'game' est à 'grid' (valeur numérique à 1 par défaut).
- ✔ turnbased : le mode du jeu, tour par tour ou temps-réel (booléen à faux par défaut pour un jeu en temps réel).
- ✔ ranking : le classement de chaque joueur (liste de joueurs, vide par défaut).

Jeu :

```
game has gravity at 10, name at "Jeu du lapin";
```

b. Déclaration de nouveaux types

Cette partie permet de déclarer de nouveaux types issus d'un ou de plusieurs types primitifs du logiciel (voir en annexes) :

```
type [nom nouveau type] is [type primitif];
type [nom nouveau type] is [type primitif 1] and [type primitif 2] and
            [type primitif n];
```

Les types primitifs doivent exister et être différents s'il y en a plusieurs. Il est possible d'appeler un type que l'on vient de créer :

```
type Lapin is Character;
type LapinEnrage is Weapon and Lapin;
```

Les types primitifs 'Character' et 'Weapon' existent déjà. 'Lapin' existe aussi lorsque 'LapinEnrage' est créé, le code est valide.

'Lapin' et 'LapinEnrage' pourront servir dans les parties suivantes comme des type primitifs normaux. Les

attributs et spécificités de chaque type primitif sont copiés dans le type ainsi créé.

c. Initialisations

Cette partie permet de créer des entités et de les initialiser :

i. Création d'une entité

```
[nom de l'entité] is [type];
```

Un objet est créé à partir d'un type primitif, ou d'un type créé précédemment.

```
lapin is Lapin;
chasseur is Character;
bazooka is Weapon;
cam1 is Camera free; //les caméras servent à l'affichage de votre scène
```

Il est possible de spécifier que la représentation 3D de l'entité pourra être générée plusieurs fois en même temps en jeu. Si l'on voulait créer plusieurs lapins en même temps, il faudrait ajouter le mot-clé 'duplicable' :

```
[nom de l'entité] is [type] duplicable;
lapinMultipliable is Lapin duplicable;
```

ii. Déclaration d'un joueur ou d'une équipe

Les joueurs et les équipes sont essentiels pour déterminer un ou plusieurs gagnants (perdants) du jeu.

```
[nom du joueur] is Player;
[nom de l'équipe] is Team;
```

Un joueur peut être déclaré comme n'appartenant pas à une équipe spécifique.

```
[nom du joueur] is Player solo;
joueurHumain is Player solo;
```

Un joueur peut ensuite être déclaré en tant qu'humain ou intelligence artificielle, il faut alors préciser le nom de l'IA qui sera définie ensuite dans la partie ***IntelligenceArtificielle***.

```
joueurHumain has controller at "human";
joueurIA has controller at "ia1";
```

Une équipe est un regroupement de joueurs et/ou d'équipes.

iii. Initialisation des attributs d'une entité

```
[entité] has [attribut de l'entité] at [nouvelle valeur de l'attribut];
```

La valeur de l'attribut doit être du type attendu par l'attribut. Par exemple, on ne met pas de valeur numérique pour un attribut de type booléen.

```
Lapin has life at 1;
```

iv. Déclaration de la possession d'une entité par une autre :

```
[entité qui possède] has [entité possédée];
jeep has chasseur;
chasseur has bazooka;
```

Pour cet exemple, 'chasseur' va posséder 'bazooka', les transformations qui lui seront appliquées se répercuteront sur 'bazooka'.

Il est aussi possible de placer des joueurs (type Player) dans une équipe (type Team).

v. Déclaration d'une liste où seront regroupées plusieurs entités:

```
[entité qui possède] is list of [élément 1], [élément 2], [élément n];
listeArmesChasseur is list of bazooka;
```

d. Définitions

Cette partie permet de créer des bouts de code réutilisables dans la partie Commandes et Règles du jeu à l'aide de son nom.

```
definition [nom de la définition] means [corps de la définition];
```

Le corps d'une définition contient une ou plusieurs instructions séparées par des virgules.

```
definition [nom de la définition] means
      [instruction 1],
      [instruction 2],
      [instruction n];
```

Les instructions sont variées :

i. Appel d'une autre définition:

Une autre définition peut être appelée avec son nom mais cette définition doit avoir été déclarée avant.

```
definition genererLapin means
      generate lapinMultipliable;
definition generer2Lapins means
      genererLapin,
      genererLapin;
```

ii. Conditions

```
if [condition] then [corps de la condition] endif;
```

La condition est une opération de test (booléenne) :

- ✔ test numérique de l'attribut d'un objet (avec les opérateurs '=', '<', '>', '<=', '>=', !=) :

```
life of lapin >= 0 // la vie de 'lapin' doit être supérieure ou égale à 0
```

- ✔ test de l'attribut d'un objet non-numérique (avec l'opérateur '=') :

```
name of chasseur = "Le chasseur"
```

- ✔ test de l'état d'un objet :

```
lapin is dead // équivaut à : life of lapin <= 0
```

Pour appeler cette condition, 'lapin' doit avoir un attribut 'life'.

Ces états sont 'dead', 'alive', 'generated', 'effaced', 'touching' [nom d'entité], 'moving', 'waiting'

- ✔ test de la présence d'un élément dans une liste :

```
listeArmesChasseur contains bazooka
```

- ✔ test de la victoire ou de la défaite d'un joueur ou d'une équipe :

```
defeat of joueurHumain
victory of joueurIA
```

Il est possible de combiner plusieurs conditions à l'aide d'opérateurs logiques ('and', 'or', 'not') afin de n'en former qu'une :

```
(life of entite1 != 0 and (life of entite2 > 0 or (not entite3 is dead)))
```

Ici, pour que la condition soit validée, la vie de 'entite1' ne doit pas être égale à 0 (!=), mais aussi, la vie de 'entite2' doit être supérieure à 0 ou alors, 'entite3' doit être en vie.

Des parenthèses peuvent être utilisées afin de séparer les différents blocs de condition selon la priorité des opérateurs ou pour plus de lisibilité mais elles ne sont pas obligatoires dans ce cas.

Si la condition est validée, le corps de la condition s'exécute, sinon il est ignoré. Ce corps est aussi une suite d'instructions :

```
if [condition] then
      [instruction 1],
      [instruction 2],
      [instruction n]
endif
```

Il est possible d'ajouter d'autres instructions si la condition n'est pas validée :

```
      if [condition] then [corps n°1] else [corps n°2] endif
```

iii. Action

Une entité peut accomplir des actions :

- ✔ sauter avec une certaine force :
```
lapin jumps by 5
```
- ✔ attraper une autre entité :
```
jeep grasps chasseur //si le chasseur était sortit de la jeep
```
- ✔ ingérer une entité (la deuxième entité disparaît et la première la possède) :
```
chasseur ingests bazooka in inventaireArmes
```

Un inventaire doit avoir été déclaré avant (attribut 'inventories' dan Character).

- ✔ expulser une entité possédée par une autre avec une certaine force :
```
bazooka expels missile by 10
```
- ✔ effectuer une translation :
```
lapin moves forward by 5
```

Ces translations sont 'forward', 'backward', 'right', 'left', 'up', 'down'

- ✔ effectuer une rotation :
```
bazooka turns right by 20
```

Ces rotations sont 'clockwise', 'anticlockwise', 'right', 'left', 'up', 'down'

- ✔ accélérer si l'entité a une vitesse :
```
jeep accelerates by 10
```
- ✔ freiner si l'entité a une vitesse :
```
jeep brakes by 5
```

iv. Génération / effacement de la représentation 3D d'une entité

```
generate jeep
efface lapin
```

Si aucun objet de votre jeu n'est généré de cette manière, rien n'apparaîtra dans la scène 3D.

v. Attente

```
wait [valeur du temps] then [suite d'instructions] endwait
```

Permet d'attendre un instant donné puis d'exécuter les instructions qui sont à l'intérieur.

La valeur du temps peut être exprimée en minutes('min'), secondes('s'), millisecondes ('ms') ou images ('frames').
```
wait 5 s then ...
```

vi. Affectation

```
assign true to active of cam1 //assigne une valeur à l'attribut
add 5 to life of lapin //ajoute la valeur
sub 1 to life of lapin //soustrait la valeur
invert life of lapin with life of chasseur //intervertit les deux valeurs
```

vii. Activation / désactivation de commandes

```
activate commands
desactivate commands
```

Il est possible de n'activer / désactiver qu'une commande ou seulement le clavier ou la souris :
```
activate key A
activate mouse rClick
```

viii. Déclaration de la victoire d'une équipe ou d'un joueur :

```
victory of joueurIA
```

Un message peut être affiché :
```
victory of joueurHumain : "Vous avez gagné !"
```

ix. Arrêt du jeu :

```
game ends
game pause
```

Avec cette instruction, le jeu s'arrête. Si le jeu est en pause, une commande spécifique au logiciel permet de le reprendre. Un message peut être affiché :

```
game ends : "Le jeu est fini !"
```

x. Changement de tour d'un joueur :

Cette instruction n'est utile que si le mode tour par tour est actif (attribut 'turnbased' de game) :

```
nextturn joueurIA2 //dans le cas où 'joueurIA2' est aussi un chasseur
```

e. Commandes

Les définitions créées plus haut peuvent être assignées à des commandes afin de permettre au joueur (humain obligatoirement) d'interagir avec l'environnement 3D.

```
command for [nom du joueur] is
      [commande n°1] for [définition de la commande n°1],
      [commande n°2] for [définition de la commande n°2],
      [commande n] for [définition de la commande n];
```

(voir 'command' dans le lexique en annexe)

Les commandes assignées sont relatives aux touches du clavier,

Deux joueurs peuvent aussi se partager des commandes si le mode tour par tour est actif (attribut 'turnbased' de game) :

```
command for joueurHumain, joueurIA2 is
      key space for tirer; //'tirer' est une définition
```

f. Règles du jeu

Les règles du jeu contiennent des appels de définitions déjà créées auparavant et peuvent être appelées uniquement si le déclencheur devient valide au cours du jeu :

```
rule [déclencheur] then
      [nom de définition 1],
      [nom de définition 2],
      [nom de définition n];
```

Le déclencheur d'une règle est un événement qui se produit en cours de jeu à un moment précis. Il peut être :

- ✔ le début du jeu :
  ```
  game starts
  ```
- ✔ un test d'égalité pour l'attribut d'un objet :
  ```
  life of lapin becomes 0
  name of chasseur becomes "Le chasseur entraîné"
  ```
- ✔ un test de l'état changé d'un objet :
  ```
  lapin dies // pour ce déclencheur, 'lapin' doit avoir un attribut 'life'
  ```

Ces états sont 'dies', 'moves', 'touches' [nom d'entité], 'kills' [nom d'entité], 'gets', 'touched by' [nom d'entité], 'killed by' [nom d'entité]

- ✔ le test de la victoire ou de la défaite d'un joueur ou d'une équipe :
  ```
  defeat of joueurIA
  victory of joueurHumain
  ```
- ✔ aucun test (le contenu de la règle s'exécute tout le temps) :
  ```
  true
  ```

Deux règles du jeu ayant le même déclencheur seront appelées au même moment et leurs instructions s'exécuteront en parallèle.

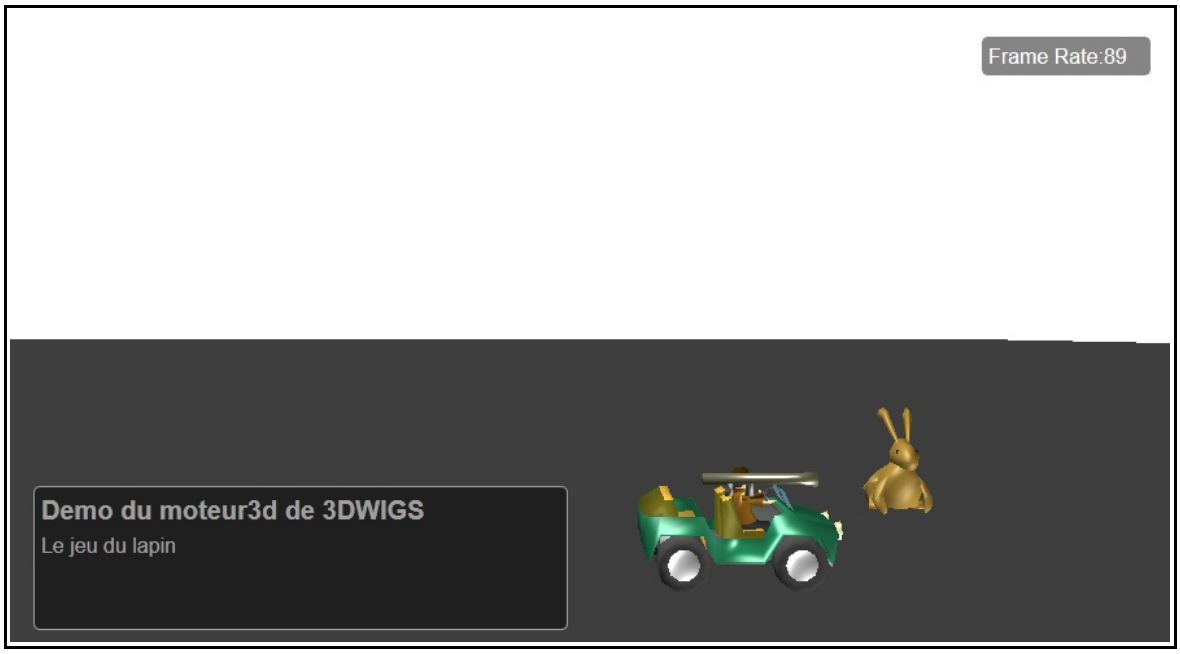
g. Intelligence Artificielle

Cette partie permet de décrire comment l'ordinateur va se comporter en tant que joueur. Une IA regroupe une série de règles du jeu.

```
ai [nom de l'ia] is
      rule [déclencheur] then [suite de définitions],
      rule [déclencheur] then [suite de définitions];
```

4. Résultat attendu

Voici une capture d'écran du jeu tel qu'il serait écrit en langage de 3DWIGS (voir en annexes pour la description du jeu complet) :



III. ANNEXE

Glossaire des mots-clés du langage de description de 3DWIGS

A

accelerates

*Partie **Définitions***

- ordonner à l'entité désignée d'accélérer avec une valeur (voir 'by') :

```
definition acc1 means
        sonic accelerates by 20; // sonic accélère de 20
```

activate

*Partie **Définitions***

- activer une commande ou toutes les commandes (voir 'disable', 'keyboard', 'mouse' et 'commands') :

```
definition activerTout means
        activate commands;
definition activerPauseSeulement means
        activate P; //active la touche P sur le clavier
```

add

*Partie **Définitions***

- ajouter une valeur à un attribut numérique (voir 'to' et 'sub') :

```
definition donnerBonus means
        add 30 to life of batman; // l'attribut 'life' de
// l'entité batman de type Character va augmenter de 30
```

ai

*Partie **Intelligence Artificielle***

- détailler le contenu d'une IA (voir 'is' et 'rule') :

```
ai ia_1 is
        rule /*déclencheur*/ then /*conséquences*/;
```

alive

*Partie **Définitions***

- dans une condition, tester si l'entité avec un attribut 'life' (ex : Character) ne l'a pas à 0 (voir 'if') :

```
if (perso is alive) // (voir 'is')
then /*conséquences*/
endif;
```

all

- désigner tous les objets 3D générés hérités de Empty

```
assign 100 at life of all; // (voir 'assign') va assigner 100 à
// l'attribut 'life' de tous les objets 3D qui en possèdent un
if life of all = 0 then /*conséquences*/ endif; // (voir 'if') va tester
// si tous les objets générés qui ont un attribut 'life' l'ont à 0
rule all moves then /*actions*/;
// (voir 'rule') teste si l'un des objets générés bouge
```

and

*Partie **Déclaration de nouveaux types***

- séparer les types lors de la déclaration d'un nouveau type (voir 'type' et 'is') :

```
type SuperPlombier is Plombier and Weapon and Vehicle;
// on peut ajouter autant de type que nécessaire avec 'and' du
// moment que les types existent déjà et ne soient pas redondants
```

*Partie **Définitions***

- combiner plusieurs sous conditions à l'intérieur d'une condition (voir 'if', 'or' et 'not') :

```
if (life of perso1 = 0 and life of perso2 >= 5)
// les conditions doivent être toutes les deux vraies
// pour pouvoir accéder à la conséquence (dans le then)
then /*conséquences*/
endif;
```

- définir les bornes d'un calcul ou d'une génération aléatoire (voir 'distance', 'angle', 'random' et 'between') :

```
distance between perso1 and perso2 > 10
angle between perso1 and perso2 > 10
random between 1 and 20
// ces valeurs peuvent être assignées ou testées dans une condition
```

angle

*Parties **Initialisations** et **Définitions***

- retourner la valeur de l'angle (sur l'axe z) formé par l'orientation entre deux objets :

```
definition calculAngle means
        if angle between perso1 and perso2 > 10
         then assign (angle between perso1 and perso2) to z of orientation
of perso3 // (voir 'assign')
        endif;
```

anticlockwise

*Partie **Définitions***

- ordonner à l'entité désignée d'effectuer une rotation dans le sens anti-horaire (trigonométrique) (voir 'turns', 'by' et 'clockwise') :

```
definition defL1 means
        avion turns anticlockwise by 10;
```

assign

*Partie **Définitions***

- affecter une valeur à un attribut, elle doit correspondre au type de l'attribut (voir 'to') :

```
definition assigner means
        assign true to active of camera1,
        assign 10 to life of perso1,
        assign "Nom" to name of perso2;
```

at

*Partie **Description du jeu***

- assigner une valeur (numérique ou non) à un attribut de game (voir 'has') :

```
game has name at "mon jeu";
```

*Partie **Initialisations***

- assigner une valeur (numérique ou non) à un attribut d'une classe ou d'un objet (voir 'has') :

```
Character has life at 100; // tous les objets de type Character
// commenceront avec l'attribut life à 100
perso1 has life at 10;
// la valeur fournie à l'attribut doit être de même type obligatoirement,
// ici, l'attribut 'life' est un nombre
perso2 has belonging at perso2;
// ici, l'attribut belonging est un Object (Character hérite de Object)
```

*Partie **Définitions***

- générer un objet à une position bien précise (voir 'generate') :

```
generate mario at 0 0 0;
```

B

backward

*Partie **Définitions***

- ordonner à l'entité désignée d'aller en arrière pour une translation (voir 'moves', 'by' et 'forward') :

```
definition bougerObjet means
        objet moves backward by 10; // l'objet va reculer de 10
definition bougerMario means
        mario moves backward by 10; // si mario (de type Character) a
// son attribut 'moveWithCamera' à true, il va se déplacer vers la caméra
```

becomes

*Parties **Règles du jeu***

- sert à déclencher des actions lorsque la valeur d'un attribut devient (ou ne devient plus) exactement une autre valeur (voir 'rule') :

```
rule value of jaugeForce becomes 10 then
        /*actions*/;
```

```
rule name of perso1 becomes "Nouveau nom" then
        /*actions*/;
```

between

*Parties **Initialisations** et **Définitions***

- définir les bornes d'un calcul ou d'une génération aléatoire (voir 'distance', 'angle', 'random' et 'and') :

```
distance between perso1 and perso2 > 10
angle between perso1 and perso2 > 10
random between 1 and 20
// ces valeurs peuvent être assignées ou testées dans une condition
```

brakes

*Partie **Définitions***

- ordonner à l'entité désignée de freiner d'une valeur (voir 'by') :

```
definition freiner means
        vaisseau brakes by 5; // la vitesse du vaisseau va décroître de 5
```

by

*Partie **Définitions***

- définit la valeur associée à un changement (voir 'jumps', 'moves', 'turns', 'accelerates', 'brakes' et 'expels') :

```
mario jumps by 10;
mario moves forward by 10;
jeep accelerates by 10;
jeep brakes by 10;
yoshi expels goomba by 10;
```

C

clockwise

*Partie **Définitions***

- ordonner à l'entité désignée d'effectuer une rotation dans le sens horaire (voir 'turns', 'by' et 'anticlockwise') :

```
definition defR1 means
        avion turns clockwise by 10;
```

command

*Partie **Commandes***

- déclarer la liste des commandes pour chaque joueur :

```
command for joueur1, joueur2 is
// les mêmes commandes sont autorisées pour deux joueurs si
// le mode tour par tour est actif (attribut 'turnbased' de game)
```

```
        mouse right for viserR,
      mouse left for viserL,
// (voir 'mouse') commande souris ('up', 'down', 'right', 'lClick',
// 'mClick', 'rClick', 'scrollUp' ou 'scrollDown')
        key enter for tirer;
// (voir 'key') commande clavier (lettre de 'A' à 'Z', 'up', 'down',
// 'left', 'right', 'space', 'escape' ou 'enter')
```

commands

*Partie **Définitions***

- activer / désactiver l'ensemble des commandes pour le clavier et la souris (voir 'activate', 'disable', 'keyboard' et 'mouse') :

```
activate commands
disable commands
```

contains

- dans une condition, vérifier si un objet appartient à une liste (voir 'list') :

```
if elements of inventory1 contains arme1 then
// l'attribut 'elements' de inventory1 est une liste d'objets
        /*conséquence*/
endif;
```

D

dead

*Partie **Définitions***

- dans une condition, tester si l'entité ayant un attribut 'life' (ex : Character) est mort (si son attribut 'life' a atteint 0) :

```
if (perso is dead) // (voir 'is')
then /*conséquences*/
endif;
```

defeat

*Partie **Définitions***

- déclarer l'échec d'un joueur dans la partie (voir 'victory') :

```
definition echec means
        defeat of joueurPrincipal : "Echec du joueur pricipal !",
// il est possible d'afficher un message
        defeat of joueurAllie;
```

definition

*Partie **Définitions***

- déclarer un ensemble d'actions réutilisables plusieurs fois dans des règles du jeu ou des commandes différentes (voir 'means') :

```
definition tirer means
// le nom 'tirer' pourra être appelé tel quel dans une règle du jeu
        bazooka expels missile by 100,
        wait 200 ms then
                generate missile
        endwait;
command for joueur1 is
        key space for tirer;  // appel de tirer dans la règle du jeu
rule /*déclencheur*/ then
        tirer;  // appel de tirer dans la règle du jeu
```

dies

*Partie **Définitions***

- déclarer la mort d'un objet de type Character :

```
rule mario touches zoneDeVide then
        mario dies;
```

- tester si l'objet de type Character vient juste de mourir :

```
rule mario dies then
        game ends;
```

disable

*Partie **Définitions***

- désactiver une commande ou toutes les commandes (voir 'activate', 'keyboard', 'mouse' et 'commands') :

```
definition desactiverTout means
        disable commands;
definition desactiverPauseSeulement means
        disable P; // désactive la touche P sur le clavier
```

distance

*Parties **Initialisations** et **Définitions***

- retourner la valeur de la distance entre deux objets (voir 'between') :

```
definition calculDistance means
        if distance between perso1 and perso2 > 10
        then perso2 dies
        endif;
```

down

*Partie **Définitions***

- ordonner à l'entité désignée d'aller en bas pour une translation (voir 'moves', 'by' et 'up') :

```
definition bougerHelico means
        helico moves down by 10;
// l'hélicoptère va descendre verticalement de 10
```

- ordonner à l'entité désignée d'effectuer une rotation vers le bas (voir 'turns', 'by' et 'up') :

```
definition incliner means
        avion turns down by 10; // l'avion va s'incliner vers l'avant
```

*Partie **Commandes***

- indiquer une commande clavier, touche flèche du bas (voir 'key') :

```
key down for /*action*/;
```

- indiquer une commande souris, déplacement du curseur vers le bas (voir 'mouse') :

```
mouse down for /*action*/;
```

duplicable

*Partie **Initialisations***

- Indique que l'objet peut être copié en plusieurs exemplaires lors d'un generate (voir 'generate') :

```
lapin is Character duplicable;
// dans les définitions ensuite :
definition genererPlusieurslapins means
        generate lapin on zone1;
        generate lapin on zone2;
// un deuxième lapin sera généré, ils sont issus du même objet
// mais ont leurs propres déplacement, valeurs d'attributs, etc
```

during

*Partie **Définitions***

- poursuivre une action pendant un certain temps (voir 'until') :

```
goomba moves left by 5 during 2 sec;
```

E

efface

*Partie **Définitions***

- effacer un objet de la scène 3D (il reste en mémoire) (voir 'generate') :

```
definition effacerPerso means
        efface perso1;
```

effaced

*Partie **Définitions***

- dans une condition, tester si l'entité n'a pas été générée ou n'est pas dans la scène 3D (voir 'if', 'generate' et 'generated') :

```
if (perso is generated) // (voir 'is')
then /*conséquences*/
endif;
```

else

*Partie **Définitions***

- le branchement conditionnel dans le cas où la condition du 'if' n'est pas respectée (voir 'if', 'then' et 'endif'). Il n'est pas obligatoire s'il ne s'avère pas nécessaire :

```
if /*condition*/
then /*conséquence 1 (condition respectée)*/
else /*conséquence 2 (condition non-respectée)*/
// si la condition n'est pas respectée :
// la conséquence 2 est exécutée à la place de la conséquence 1
endif;
```

endif

*Partie **Définitions***

- fermer une condition (voir 'if', 'then' et 'else'). Cela sert à déterminer jusqu'où la condition va faire effet :

```
if /*condition*/
then
        if /*condition*/
        then /*conséquence 1 (condition respectée)*/
                        // pas de else
        endif
else /*conséquence 2 (condition non-respectée)*/
// s'il n'y avait pas eu le 'endif', il aurait été impossible
// de déterminer à quelle condition appartenait ce 'else'
endif;
```

ends

*Partie **Définitions***

- terminer le jeu :

```
definition finJeu means
        game ends;
definition finJeuMessage means
        game ends : "Fin du jeu !"; // le message à afficher est facultatif
```

- arrêter un Chrono actif (voir 'starts' et 'pause') :

```
definition arrêtChrono means
        chrono1 ends : "Time's up." // le message à afficher est facultatif
```

*Partie **Règles du jeu***

- tester si un chrono vient juste de se terminer (voir 'starts') :

```
rule chrono1 ends then
        /*actions*/
```

endwait

*Partie **Définitions***

- fermer une attente (voir 'wait'). Cela sert à repérer jusqu'où l'attente va faire effet :

```
wait 5 s then
```

```
/*actions*/
endwait;
wait 5 s then
        wait 10 s then
        /*actions*/
        endwait,
        /*actions*/
endwait;
```

enter

*Partie **Commandes***

- indiquer une commande clavier, touche Entrée (voir 'key') :

```
key enter for /*action*/;
```

equip

*Partie **Définitions***

- équiper une entité de type Character avec un objet de son inventaire (voir 'previous' et 'next') :

```
definition equiper means
        perso1 equip arme1,
//l'attribut 'equipedObject' de perso1 devient arme1
        perso1 equip next, //l'attribut 'equipedObject' de perso1
// devient l'arme venant après arme1 dans la liste de son inventaire
        perso1 equip previous;
//l'attribut 'equipedObject' de perso1 redevient arme1
```

escape

*Partie **Commandes***

- indiquer une commande clavier, touche Echap (voir 'key') :

```
key escape for /*action*/;
```

expels

*Partie **Définitions***

- ordonner à un objet d'en lâcher un autre qui lui appartient (voir 'has', 'ingests' et 'grasps') :

```
definition tirer means
        batman expels missile by 20;
// (voir 'by') l'objet est expulsé avec une force de 20
// si le missile était n'était pas généré, il l'est avant d'être expulsé
definition lacher means
        batman expels missile by 0; // sans force, l'objet est juste lâché
```

*Partie **Règles du jeu***

- tester si un objet vient juste d'en relâcher un autre (voir 'gets') :

```
rules mario expels etoile then
        /*actions*/;
```

F

finished

*Partie **Définitions***

- dans une condition, tester si un Chrono est arrêté (voir 'if') :

```
if (chrono1 is finished) // (voir 'is')
then /*conséquences*/
endif;
```

first

*Partie **Initialisations***

- définir le point de vue de camera à la première personne, pour un jeu de type FPS par exemple (voir 'is') :

```
cam1 is camera first person;
```

*Parties **Définitions***

- désigner le premier élément d'une liste (voir 'list') :

```
first of list1
```

for

*Parties **Commandes***

- désigner le (ou les joueurs dans le cas d'un jeu en tour par tour) agissant sur la liste de commandes :

```
command for player1 is // un seul joueur
        /*commandes*/;
command for player1, player2 is // deux joueurs
        /*commandes*/;
```

- désigner l'action associée à une commande (voir 'command') :

```
key space for /*action*/;
```

forward

*Partie **Définitions***

- ordonner à l'entité désignée d'aller en avant pour une translation (voir 'moves', 'by' et 'backward') :

```
definition bougerObjet means
        objet moves forward by 10; // l'objet va avancer de 10
definition bougerMario means
        mario moves forward by 10; // si mario (de type Character)
// a son attribut 'moveWithCamera' à true, il va s'éloigner de la caméra
```

frame(s)

*Parties **Initialisations** et **Définitions***

- l'une des unités de temps disponibles, l'image (voir 'min', 's' et 'ms') :

```
chrono1 has end at 100 frames //la fin du chrono est à l'image 100,
// à 25 images par secondes, cela fait 4 secondes
```

free

*Partie **Initialisations***

- définir le point de vue de camera sans contraintes sur celle-ci (voir 'is') :

```
cam1 is camera free;
```

from

*Parties **Initialisations** et **Définitions*** (voir 'list' et 'remove')

- pour désigner le nom de la liste lorsqu'on lui retire un élément :

```
remove elt1 from liste1;
// l'élément doit exister et être dans la liste désignée
```

G

game

*Partie **Description du jeu***

- le mot-clé pour définir les attributs principaux de jeu. Ses attributs sont 'gravity', 'world', 'gridsize', 'name', 'turnbased' et 'ranking' :

```
game has gravity at 10, name at "Mon Jeu";
```

generate

*Partie **Définitions***

- faire apparaître un objet dans l'environnement 3D (voir 'game') :

```
definition depart means
        generate sol,
        generate zoneDepart,
        generate mario;
// il est possible de générer un objet à un endroit précis
definition genMario means
        generate sol at 0 0 20, // à une position bien précise
        generate zoneDepart on sol,
// au dessus d'un objet (position aléatoire ensuite)
        generate mario in zoneDepart;
// à l'intérieur d'une zone (position aléatoire ensuite),
// un objet duplicable peut être généré en plusieurs exemplaires
// d'un coup (voir 'duplicable')
definition genGoombas means
        generate 3 goombas on sol;
```

generated

*Partie **Définitions***

- dans une condition, tester si un objet est présent dans l'environnement 3D (voir 'generate') :

```
definition objetExiste means
        if objet1 is generated then
                /*conséquence*/
        endif;
```

generic

*Partie **Description du jeu***

- type d'environnement (map) général sans spécification spéciale (voir 'world', 'grid' et 'ribbon') :

```
game has world at generic;
```

gets

*Partie **Règles du jeu***

- tester si un objet vient juste d'en obtenir un autre par l'intermédiaire de ingests ou grasps (voir 'ingests', 'grasps' et 'expels') :

```
rules mario gets etoile then
        /*actions*/;
```

grasps

*Partie **Définitions***

- ordonner à un objet d'en attraper un autre pour le faire sien (voir 'has', 'ingests' et 'expels'):

```
definition entrerDansVoiture means
        voiture grasps perso1; // c'est la voiture qui attrape
// le personnage afin de le conduire quelque part
```

gravity

*Partie **Description du jeu***

- définir la gravité de l'environnement. la gravité peut être définie par un vecteur (trois chiffre float) ou un chiffre (gravité vers le bas) :

```
game has gravity at 1 1 1;
```

grid

*Partie **Description du jeu***

- type de d'environnement (map) découpé en cases de taille fixée par l'attribut gridsize (voir 'world', 'generic', 'ribbon' et 'gridsize') :

```
game has world at grid;
```

gridsize

*Partie **Description du jeu***

- définir la taille de chaque case dans le cas d'un jeu de type grille (voir 'world' et 'grid') :

```
game has gridsize at 12;
```

H

happens

- poursuivre une action jusqu'à ce qu'un événement se déclenche (voir 'until') :

```
mario moves forward until
        mario touches zone1 happens;
```

has

*Partie **Description du jeu***

- spécifier les attributs de 'game' (voir 'at') :

```
game has name at "mon jeu";
```

*Partie **Initialisations***

- accéder à un attribut existant d'une classe ou d'un objet pour le définir (voir 'at') :

```
Character has life at 100; // tous les objets de type
// Character commenceront avec l'attribut life à 100
perso1 has life at 10;
// l'objet perso1 de type Character commencera avec l'attribut life à 100
perso1 has life at life of perso2;
// l'attribut life de perso1 se basera sur l'attribut life de perso2
```

- déclarer un nouvel attribut n'appartenant pas déjà à une classe (voir 'at') :

```
Character has mon_attribut_1 at 100; // l'attribut nommé 'mon_attribut_1' n'existe pas dans la classe Character, tous les objets de type Character commenceront avec cet attribut initialisé à 20
perso1 has mon_attribut_2 at 20; // perso1 est de type Character,
// l'attribut nommé 'mon_attribut_2' n'existant pas dans
// cette classe, perso1 commencera avec cet attribut initialisé à 50
perso1 has mon_attribut_3 at "texte1"; // une chaîne de caractère peut
// être ajoutée (entre guillemets obligatoirement)
```

- faire qu'un objet s'approprie un autre objet et lui donne des contraintes de parenté :

```
perso1 has arme1; // l'objet perso1 s'approprie l'objet arme1
//(son attribut belonging devient 'true')
```

*Partie **Définitions***

- dans une condition, tester l'appartenance d'un objet pour un autre (voir 'if') :

```
if perso1 has arme1
then /*conséquence*/
endif; // l'objet perso1 s'approprie l'objet arme1
// (son attribut belonging devient 'true')
```

held

*Partie **Commandes***

- préciser qu'une commande s'effectue lorsqu'une touche du clavier est maintenue enfoncée (voir 'key', 'pressed' et 'released') :

```
command for player is
        key up held for /*action*/; // 'held' n'est pas obligatoire
```

I

if

*Partie **Définitions***

- exprimer un choix avec une condition (voir 'then', 'else' et 'endif') :

```
if /*condition*/ // si la condition est respectée les actions après le 'then' seront exécutées
then /*conséquence (condition respectée)*/
endif; // fin, condition respectée ou non
if /*condition 1*/
then /*(condition respectée)*/
        if /*condition 2*/ // d'autre conditions peuvent être imbriquées
        then /*conséquence (condition respectée)*/
        endif;
endif;
if /*condition*/ // si la condition est respectée les actions après le 'then' seront exécutées sinon ce sont celles après le 'else' qui le seront
then /*conséquence 1 (condition respectée)*/
else /*conséquence 2 (condition non-respectée)*/
endif;
```

in

*Partie **Initialisations***

- désigner une liste lorsqu'on y accède :

```
assign 5 to life of num 3 in list1 // la liste doit contenir au moins
// 3 éléments et les attributs appelés doivent exister
```

- désigner une liste lorsqu'on y place un nouvel élément (voir 'insert') :

```
insert arme1 in list1;
```

*Partie **Définitions***

- désigner une liste lorsqu'on y place un nouvel élément (voir 'insert' et 'ingests') :

```
insert arme1 in list1;
yoshi ingests ennemi1 in inventaireEnnemis;
```

- générer un objet à l'intérieur d'une zone (voir 'generate'):

```
generate mario in zoneDepart;
```

ingests

*Partie **Définitions***

- ordonner à un objet d'en prendre un autre qui ne lui appartient pas (voir 'has', 'grasps' et 'expels'), l'objet pris est effacé et mis dans un inventaire :

```
definition placeBook means
        perso1 ingests book1 in inventaireLibrairie;
```

insert

*Parties **Initialisations** et **Définitions***

- placer un objet dans une liste (voir 'list', 'in' et 'remove') :

```
insert arme1 in list1 num 2;
// (voir 'num') l'objet arme1 sera placé à la deuxième place dans la liste
// list1 (la dernière place si le nombre d'éléments est plus petit)
insert arme1 in list1;
// l'objet arme1 sera placé à la fin de la liste list1
```

invert

*Partie **Définitions***

- intervertir les valeurs de deux attributs (voir 'with') :

```
definition intervertir means
        invert speed of mario with speed of luigi;
```

is

*Partie **Déclaration de nouveaux types***

- déclarer un nouveau type (une nouvelle classe) (voir 'type') :

```
    type Plombier is Character; // le type doit déjà exister ou avoir été défini plus haut dans le code de la même manière
    type SuperPlombier is Plombier and Weapon; // on peut déclarer de nouveaux types qui sont composés de plusieurs déjà existant
```

*Partie **Initialisations***

```
mario is Plombier; // le type doit exister
```

*Partie **Définitions***

- tester l'état d'un objet selon son type dans une condition (voir 'if') :

```
if audio1 is started then /*conséquences*/ endif;
// (voir 'started') pour un objet de type Media
if perso1 is dead then /*conséquences*/ endif;
// (voir 'dead') pour un objet de type Character
```

*Partie **Commandes***

- déclarer une commande : se place après la déclaration (voir 'command' et 'for') :

```
command objet1 for player1 is definition1;
```

*Partie **Intelligence Artificielle***

- préciser le nom d'une règle du jeu (voir 'ai') :

```
ai nom_ia is /*conséquences*/ ;
```

J

jumps

*Partie **Définitions***

- ordonner à un personnage de sauter en indiquant la puissance (voir 'by') :

```
superGoomba jumps by 5;
```

K

key

*Partie **Commandes***

- déclarer une commande liée clavier (voir 'command' et 'mouse'), les commandes sont : lettre de 'A' à 'Z', 'up', 'down', 'left', 'right', 'space', 'escape' ou 'enter' :

```
command for joueur1 is
        key left for bougerJeepGauche,
        key right for bougerJeepDroite;
```

keyboard

*Partie **Définitions***

- activer / désactiver l'ensemble des commandes pour le clavier seulement (voir 'activate', 'disable', 'mouse' et 'commands') :

```
activate keyboard
disable keyboard
```

killed

*Partie **Règles du jeu***

- tester si une entité de type Character est tuée par une autre :

```
rule mario is killed by goomba then
        /*actions*/;
```

kills

*Partie **Définitions***

- déclarer qu'une entité a tué une autre entité de type Character :

```
mario kills goomba;
```

*Partie **Règles du jeu***

- tester si une entité tue une autre entité de type Character :

```
rule mario kills goomba then
        /*actions*/;
```

L

last

*Partie **Définitions***

- désigner le dernier élément d'une liste (voir 'list') :

```
last of list1
```

lclick

*Partie **Commandes***

- indiquer une commande souris, clic gauche (voir 'mouse') :

```
mouse lclick for /*action*/;
```

left

*Partie **Définitions***

- ordonner à l'entité désignée d'aller à gauche pour une translation (voir 'moves', 'by' et 'right') :

```
definition bougerObjet means
        objet moves left by 10; //l'objet va se décaler sur sa gauche de 10
definition bougerMario means
         mario moves left by 10; // si mario (de type Character) a son
attribut 'moveWithCamera' à true, il va se déplacer vers la gauche de l'écran
```

- ordonner à l'entité désignée d'effectuer une rotation vers la gauche (voir 'turns', 'by' et 'right') :

```
definition tournerMario means
        mario turns left by 10; // mario va tourner à gauche
```

*Partie **Commandes***

- indiquer une commande clavier, touche flèche de gauche (voir 'key') :

```
key left for /*action*/;
```

- indiquer une commande souris, déplacement du curseur vers la gauche (voir 'mouse') :

```
mouse left for /*action*/;
```

list

*Partie **Initialisations***

- regrouper et ordonner des éléments en créant une liste d'entités :

```
listeEnnemie is list of goomba1, goomba2, goomba3;
```

loop

*Partie **Initialisations***

- indiquer qu'un Media est interprété en boucle (voir 'is' et 'once') :

```
musiqueAmbiance is Media loop;
```

M

mclick

*Partie **Commandes***

- indiquer une commande souris, clic du bonton central (molette) s'il y en a un (voir 'mouse') :

```
mouse mclick for /*action*/;
```

means

*Partie **Définitions***

- déclarer une définition : se place après la déclaration :

```
definition bougerJeep means
        jeep moves forward by 10;
```

min

*Parties **Initialisations** et **Définitions***

- l'une des unités de temps disponibles, la minute (voir 'frame(s)', 's' et 'ms')

```
chrono1 has end at 1 min
```

mouse

*Partie **Définitions***

- activer / désactiver l'ensemble des commandes pour la souris seulement (voir 'activate', 'disable', 'keyboard' et 'commands') :

```
activate mouse
disable mouse
```

*Partie **Commandes***

- déclarer une commande liée à la souris (voir 'command' et 'key'), les commandes sont 'up', 'down', 'right', 'lClick', 'mClick', 'rClick', 'scrollUp' ou 'scrollDown' :

```
command for joueur1 is
        mouse down for seBaisser,
        mouse rclick for tirer;
```

moves

*Partie **Définitions***

- déclarer un déplacement (voir 'right', 'left', 'forward', 'backward' et 'turns') :

```
definition bougerObjet means
        objet moves backward by 10;
```

- tester si un objet vient juste de bouger :

```
rule mario moves then
        /*actions*/;
```

moving

*Partie **Définitions***

- dans une condition, tester si un objet est en train de bouger :

```
if mario is moving
then /*conséquence*/
```

```
endif;
```

ms

*Parties **Initialisations** et **Définitions***

- l'une des unités de temps disponibles, la milliseconde (voir 'frame(s)', 'min' et 's')

```
chrono1 has end at 750 ms
```

mute

*Partie **Définitions***

- désactiver le son d'un Media (voir 'on' et 'off') :

```
definition desactiverMusique means
        mute on mediaMusique; // désactive le son du média
        mute off mediaMusique; // active le son du média
```

muted

*Partie **Règles du jeu***

- dans une condition, tester si le son du Media a été coupé (voir 'mute', 'on' et 'off') :

```
if musiqueAmbiance is muted on // ou bien 'muted off'
then /*conséquence*/
endif;
```

N

name

*Partie **Description du jeu***

- définir le nom du jeu (voir 'game') :

```
game has name at "Mon jeu";
```

next

*Partie **Définitions***

- équiper une entité de type Character avec l'objet de son inventaire venant après l'objet déjà équipé (voir 'equip' et 'previous') :

```
definition equiper means
        perso1 equip next;
//l'attribut 'equipedObject' de perso1 devient l'objet suivant
```

nexturn

*Partie **Définitions***

- déclarer le tour du joueur suivant lors d'un jeu en tour par tour (voir 'game' et 'turnbased') :

```
nexturn joueur2;
```

not

*Partie **Définitions***

- dans une condition, exprimer la négation logique (voir 'and' et 'or'):

```
definition genererMario means
        if mario is not generated then // dans une condition
                generate mario
        endif;
definition destroyMario means
        if not (mario is not generated or mario is dead) then
// équivaut à "s'il est généré et en vie", il est possible
// d'en mettre autant que nécessaire
                destroy mario
        endif;
```

num

*Parties **Initialisations** et **Définitions***

- accéder à l'élément d'une liste à partir de sa position (voir 'list' et 'in') :

```
name of num 1 in listeDesPersos;
// accès à l'attribut nom du premier élément de la liste 'listeDesPersos'
insert arme1 in list1 num 2; // (voir 'insert')
remove num 1 from liste1; // (voir 'remove')
```

O

of

*Partie **Initialisations***

- énumérer le contenu d'une liste à sa création (voir 'is' et 'list') :

```
liste1 is list of perso1, perso2, perso3;
```

*Parties **Définitions** et **Règles du jeu***

- accéder en cascade aux attributs d'un objet ou d'une classe :

```
life of perso1
x of position of perso1
// x est un attribut de position et position est un attribut de perso1
```

- indiquer quelle Team ou quel Player est en condition de victoire ou de défaite (voir 'victory' et 'defeat') :

```
victory of team1;
defeat of player1;
```

off

*Partie **Définitions***

- réactiver le son d'un Media (voir 'mute' et 'on') :

```
definition activerMusique means
        mute off mediaMusique; // réactive le son du média
```

*Partie **Règles du jeu***

- dans une condition, tester si le son du Media a été réactivé (voir 'muted' et 'on') :

```
if musiqueAmbiance is muted off
then /*conséquence*/
endif;
```

on

*Partie **Définitions***

- désactiver le son d'un Media (voir 'mute' et 'off') :

```
definition desactiverMusique means
        mute on mediaMusique; // désactive le son du média
```

*Partie **Règles du jeu***

- dans une condition, tester si le son du Media a été coupé (voir 'muted' et 'off') :

```
if musiqueAmbiance is muted on
then /*conséquence*/
endif;
```

or

*Partie **Définitions***

- combiner plusieurs sous conditions à l'intérieur d'une condition (voir 'if', 'and' et 'not') :

```
if (life of perso1 = 0 or life of perso2 >= 5)
then /*conséquences*/
endif; //une seule des deux conditions doit être vraie pour pouvoir
// accéder à la conséquence (dans le then)
```

orientation

*Parties **Initialisations** et **Définitions***

- l'attribut d'un objet de type Empty (ou de ses dérivés), correspondant à son orientation dans l'environnement 3D selon son référentiel (le référentiel peut être un objet) (voir 'position' et 'size') :

```
objet1 has orientation at 90 0 180; // l'orientation de l'objet est de 90
// autour de l'axe x, 0 autour de l'axe y et 180 autour de l'axe z
```

P

pause

*Partie **Définitions***

- stopper le jeu temporairement :

```
definition pauseOn means
         game pause; // une fois le jeu mis en pause, il est possible de
reprendre la partie avec le menu contextuel
```

- stopper un Chrono temporairement (voir 'starts' et 'ends') :

```
definition stopperChrono means
        chrono1 pause;
```

- stopper un Media temporairement (voir 'play' et 'stop') :

```
definition stopperMedia means
        pause musiqueAmbiance;
```

paused

*Partie **Définitions***

- dans une condition, tester si un Media est arrêté temporairement (voir 'if') :

```
if musiqueAmbiance is paused
then /*conséquences*/
endif;
```

- dans une condition, tester si un Chrono est arrêté temporairement (voir 'if') :

```
if chrono1 is paused
then /*conséquences*/
endif;
```

person

*Partie **Initialisations***

- définir le point de vue de la camera à la première personne ou à la troisième personne (voir 'first' et 'third'):

```
cam1 is camera first person;
cam2 is camera third person;
```

play

- lancer ou relancer un Media arrêté (voir 'pause' et 'stop') :

```
definition jouerMedia means
        play musiqueAmbiance;
```

played

*Partie **Définitions***

- dans une condition, tester si un Media est actif (voir 'if') :

```
if musiqueAmbiance is played
then /*conséquences*/
endif;
```

position

*Parties **Initialisations** et **Définitions***

- l'attribut d'un objet de type Empty (ou de ses dérivés), correspondant à sa position dans l'environnement 3D selon son référentiel (le référentiel peut être un objet) (voir 'orientation' et 'size') :

```
objet1 has position at 0 10 20; // la position de l'objet est de 0
// sur l'axe x, 10 sur l'axe y et 20 sur l'axe z
```

pressed

*Partie **Commandes***

- préciser qu'une commande s'effectue lorsqu'une touche du clavier est juste pressée une fois (voir 'key',

'held' et 'released') :

```
command for player is
        key up pressed for /*action*/, // 'pressed' n'est pas obligatoire
        key up for /*action*/; // ne rien mettre équivaut à 'pressed'
```

previous

*Partie **Définitions***

- équiper une entité de type Character avec l'objet de son inventaire venant avant l'objet déjà équipé (voir 'equip' et 'next') :

```
definition equiper means
        perso1 equip previous;
//l'attribut 'equipedObject' de perso1 devient l'objet précédent
```

R

random

*Parties **Initialisations** et **Définitions***

- donner une valeur aléatoire entre deux valeur numériques, ces valeurs peuvent être des attributs (voir 'between' et 'and') :

```
random between 0 and 1 // donne soit 0 soit 1
random between life of perso1 and life of perso2
// les deux attributs doivent être numériques
```

ranking

*Partie **Description du jeu***

- établir un classement final des joueurs sous la forme d'une liste (voir 'game' et 'list') :

```
game has ranking at joueur1, joueur2;
```

rclick

*Partie **Commandes***

- indiquer une commande souris, clic droit (voir 'mouse') :

```
mouse rclick for /*action*/;
```

released

*Partie **Commandes***

- préciser qu'une commande s'effectue lorsqu'une touche du clavier est relâchée (voir 'key', 'pressed' et 'held') :

```
command for player is
        key up released for /*action*/; // 'released' n'est pas obligatoire
```

remove

*Parties **Initialisations** et **Définitions***

- pour retirer un élément d'une liste (voir 'list', 'from' et 'insert') :

```
remove num 1 from liste1;
// l'élément doit exister et être dans la liste désignée
remove first from liste1;
// (voir 'first') supprime le premier élément si la liste n'est pas vide
remove last from liste1;
// (voir 'last') supprime le dernier élément si la liste n'est pas vide
```

ribbon

*Partie **Description du jeu***

- type d'environnement (map) adapté aux jeux linéaires (voir 'world', 'generic' et 'grid') :

```
game has world at ribbon;
```

right

*Partie **Définitions***

- ordonner à l'entité désignée d'aller à droite pour une translation (voir 'moves', 'by' et 'left') :

```
definition bougerObjet means
        objet moves right by 10;
// l'objet va se décaler sur sa droite de 10
definition bougerMario means
         mario moves right by 10; //si mario (de type Character) a son
attribut 'moveWithCamera' à true, il se déplacera vers la droite de l'écran
```

- ordonner à l'entité désignée d'effectuer une rotation vers la droite (voir 'turns', 'by' et 'left') :

```
definition tournerMario means
        mario turns right by 10; // mario va tourner à droite
```

*Partie **Commandes***

- indiquer une commande clavier, touche flèche de droite (voir 'key') :

```
key right for /*action*/;
```

- indiquer une commande souris, déplacement du curseur vers la droite (voir 'mouse') :

```
mouse right for /*action*/;
```

rule

*Parties **Règles du jeu***

- définir une règles du jeu. Chaque règle a besoin d'un déclencheur et est déclenchée en parallèle. À l'intérieur, chaque action (nom de définition) est exécutée en série :

```
rule Game starts then
        /*action 1*/,
        /*action 2*/,
        /*action n*/;
```

S

s

*Parties **Initialisations** et **Définitions***

- l'une des unités de temps disponibles, la seconde (voir 'frame(s)', 'min' et 'ms') :

```
chrono1 has end at 10 s
```

save

*Partie **Définitions***

- enregistrer le jeu :

```
definition sauverJeu means
        save;
```

scrollDown

*Partie **Commandes***

- indiquer une commande souris, un cran de molette vers le bas (voir 'mouse') :

```
mouse scrollDown for /*action*/;
```

scrollUp

*Partie **Commandes***

- indiquer une commande souris, un cran de molette vers le haut (voir 'mouse') :

```
mouse scrollUp for /*action*/;
```

size

*Parties **Initialisations** et **Définitions***

- l'attribut d'un objet de type Empty (ou de ses dérivés), correspondant à sa taille selon son référentiel (le référentiel peut être un objet) (voir 'position' et 'orientation') :

```
objet1 has size at 1 2 3;
//la taille de l'objet est de 1 sur l'axe x, 2 sur l'axe y et 3 sur l'axe z
```

solo

*Partie **Initialisations***

- déclarer que joueur n'a personne d'autre que lui dans son équipe (voir 'is') :

```
player1 is Player solo; // le nom de l'équipe de 'joueur1' est 'joueur1'
```

space

*Partie **Commandes***

- indiquer une commande clavier, touche Espace (voir 'key') :

```
key space for /*action*/;
```

started

- dans une condition, tester si un Chrono est actif (voir 'if') :

```
if chrono1 is started
then /*conséquences*/
endif;
```

starts

*Partie **Définitions***

- lancer ou relancer un Chrono arrêté (voir 'pause' et 'ends') :

```
definition demarrerChrono means
        chrono1 starts;
```

*Partie **Règles du jeu***

- tester si le jeu vient juste de démarrer :

```
rule game starts then
        /*action de début du jeu*/;
```

- tester si un Chrono vient juste de démarrer :

```
rule chrono1 starts then
        /*action*/;
```

stop

*Partie **Définitions***

- arrêter un Media (voir 'play' et 'pause') :

```
definition stopperMedia means
        stop musiqueAmbiance;
```

stopped

*Partie **Définitions***

- dans une condition, tester si un Media n'est pas actif (voir 'if') :

```
if musiqueAmbiance is stopped
then /*conséquences*/
endif;
```

sub

*Partie **Définitions***

- soustraire une valeur à un attribut numérique (voir 'to' et 'add') :

```
definition prendreVie means
         sub 30 to life of batman; // l'attribut 'life' de l'entité batman
de type Character va baisser de 30
```

T

then

*Partie **Définitions***

- déclarer une condition (voir 'if') : se place après la déclaration :

```
if /*condition*/ then
then /*conséquences*/
endif;
```

- déclarer une attente (voir 'wait') : se place après la déclaration :

```
wait 5 s then
```

```
/*actions*/
endwait;
```

*Partie **Règles du jeu***

- déclarer une règle du jeu (voir 'rule') : se place après la déclaration :

```
rule /*déclencheur*/ then
        /*actions*/;
```

third

*Partie **Initialisations***

- définir le point de vue de camera à la troisième personne, pour un jeu de type plateforme par exemple (voir 'is'):

```
cam1 is camera third person;
```

to

*Parties **Initialisations** et **Définitions***

- désigner l'attribut d'un objet lors d'une opération mathématique ou d'une assignation (voir 'add', 'sub' et 'assign'):

```
definition toucherLapin means
    sub 1 to life of lapin1,
        assign "Lapin n°2" to name of lapin2;
```

touched

*Parties **Règles du jeu***

- tester si une entité entre en contact avec une autre ou avec une zone :

```
rule mario is touched by goomba then
        /*actions*/;
```

touches

*Parties **Règles du jeu***

- tester si une entité entre en contact avec une autre ou avec une zone :

```
rule mario touches goomba then
        /*actions*/;
```

touching

*Partie **Définitions***

- dans une condition, teste si une entité est en train d'en toucher une autre ou une zone (voir 'if' et 'is') :

```
if mario is touching zoneDepart
then /*conséquence*/
endif;
```

turns

*Partie **Définitions***

- déclarer une rotation (voir 'right', 'left', 'clockwise', 'anticlockwise' et 'moves') :

```
definition faireAvancerAiguille means
        aiguille turns clockwise by 10;
```

turnbased

*Partie **Description du jeu***

- définir le type de jeu : temps réel ou tour par tour (voir 'game') :

```
game has turn based at true;
```

type

*Partie **Déclaration de nouveaux types***

- créer un nouveau type d'entités issu d'un ou de plusieurs autres. Les objets ensuite créés avec ce type, auront les mêmes attributs que les types de base ont :

```
type EpeeFlingue is Sword and Weapon; // les objets créés avec
// ce nouveau type auront les attributs de Sword et de Weapon
type EpeeFlingueEnnemis is EpeeFlingue and Character;
// ce nouveau type aura les attributs de Sword, Weapon et Character
```

U

up

*Partie **Définitions***

- ordonner à l'entité désignée d'aller en haut pour une translation (voir 'moves', 'by' et 'down') :

```
definition bougerHelico means
        helico moves up by 10;
//l'hélicoptère va monter verticalement de 10
```

- ordonner à l'entité désignée d'effectuer une rotation vers le haut (voir 'turns', 'by' et 'down') :

```
definition incliner means
        avion turns up by 10; // l'avion va s'incliner vers l'arrière
```

*Partie **Commandes***

- indiquer une commande clavier, touche flèche du haut (voir 'key') :

```
key up for /*action*/;
```

- indiquer une commande souris, déplacement du curseur vers le haut (voir 'mouse') :

```
mouse up for /*action*/;
```

until

*Partie **Définitions***

- poursuivre une action jusqu'à ce qu'un événement se déclenche (voir 'happens' et 'during') :

```
mario moves forward until
        mario touches zone1 happens;
```

V

victory

*Partie **Définitions***

- déclarer la victoire d'un joueur dans la partie (voir 'defeat') :

```
definition victoire means
        victory of joueurPrincipal : "Victoire du joueur pricipal !",
// il est possible d'afficher un message
        victory of joueurAllie;
```

W

wait

*Partie **Définitions***

- arrêter une action pour un temps défini, les actions placées à l'intérieur s'effectuent après cette attente (voir 'endwait') :

```
wait 5 s then
/*actions*/
endwait;
wait 5 s then
        wait 10 s then
        /*actions*/
        endwait;
endwait;
```

waiting

*Partie **Définitions***

- dans une condition, tester si une entité est fixe (sa position est la même) :

```
if mario is waiting
then /*conséquences*/
endif;
```

with

*Partie **Définitions***

- intervertir les valeurs de deux attributs (voir 'invert') :

```
definition intervertir means
        invert speed of mario with speed of luigi;
```

world

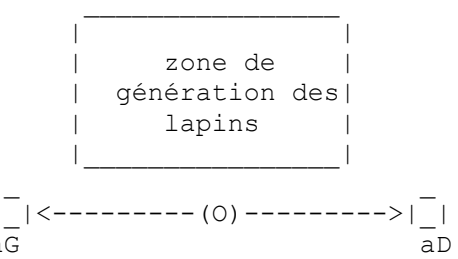
*Partie **Description du jeu***

- spécifier l'environnement du jeu (voir 'game', 'generic', 'grid' et 'ribbon') :

```
game has world at generic; // l'attribut 'world' est 'generic' de base
game has world at grid;
game has world at ribbon;
```

Exemple de jeu: Le jeu du lapin

C'est un jeu où le joueur contrôle un chasseur dans une jeep. Celui-ci possède un bazooka pour tirer sur des lapins. Un lapin apparaît et suit un parcours prédéfini. Le chasseur doit le tuer en lui tirant dessus en un temps imparti. Ensuite un nouveau lapin apparaît, le joueur gagne au bout de 3 lapins tués.




```
Les valeur des positions, orientations et tailles dans les
initialisations ne sont pas mises car elles relèvent de la disposition en
3D des éléments.
```

Description du jeu

```
      game has gravity at 10, name at "Jeu du lapin";
```

Déclaration de nouveaux types

```
      type Zobstacle is Zone and Obstacle;  // transparent + on ne
                                             // peut pas le traverser
      type Lapin is Character;
```

Initialisations

```
      player is Player solo;
      player has controller at "human";
      lapinEnnemi is Player solo;
      lapinEnnemi has controller at "ia1";

      Lapin has compteurPas at 0;
      lapin is Lapin;

      lapin has life at 1;
      lapin has compteurPas at 3;  //nouvel argument
      lapin has orientation at (random between 0 and 360)
            (random between 0 and 360)
            (random between 0 and 360);   //lorsque le lapin est généré

      cam1 is Camera free;

      chasseur is Character;
      chasseur has life at 5;

      jeep is Object;
      jeep has chasseur;

      arretGauche is Zobstacle;
      arretDroit is Zobstacle;

      zoneGenLapin is Zone;  //la zone ou va générer les lapins

      missile is Projectile duplicable;
```

```
      missile has damages at 100;
      missile has speed at 100;
      missile has lifeTime at 4 s;

      bazooka is Weapon;
      bazooka has shootInterval at 700 ms;
      // on va dire pour ce premier jeu que le chasseur n'a pas
      // de munitions

      chasseur has bazooka ;
      //aggrégation -> bazooka parenté à chasseur
      //            -> bazooka position relative, chasseur pos absolue

      bazooka has missile;
      bazooka has cam1;


      sol1 is Ground;

      compteurVagues is Counter;
      compteurVagues has currentValue at 0;
```

Définitions

```
definition  tirer  means
      bazooka expels missile by 100,
      wait 200 ms then
            generate missile
      endWait;


definition  reinitialisation  means
      generate chasseur at 4 5 34,
      generate sol1,
      generate bazooka,
      generate arretDroit,
      generate arretGauche,
      generate zoneGenLapin,
      add 1 to currentValue of compteurVagues,
      wait 2 s then
            generate lapin in zoneGenLapin
      endWait;


definition bougerJeep1 means
      jeep moves backward by 10;
definition bougerJeep2 means
      jeep moves forward by 10;

definition bougerBazooka1 means
      bazooka turns up by 10;
definition bougerBazooka2 means
      bazooka turns down by 10;
definition bougerBazooka3 means
      bazooka turns right by 10;
definition bougerBazooka4 means
      bazooka turns left by 10;

definition victoryPlayer means
```

```
    victory of player;
definition victoryLapinEnnemi means
    victory of lapinEnnemi;
definition gameEnds means
    game ends : "Le jeu est fini !";

definition subCompteurLapin means
    sub 1 to compteurPas of Lapin;

definition toucherLapin means
    efface missile,
    sub 1 to life of Lapin;
```

Commandes

```
command for player is
      key left for bougerJeep1,
      key right for bougerJeep2,
      mouse up for bougerBazooka1,
      mouse down for bougerBazooka2,
      mouse right for bougerBazooka3,
      mouse left for bougerBazooka4,
      key space for tirer;
```

Règles du jeu

```
rule game starts then
      reinitialisation;

rule currentValue of compteurVagues becomes 3 then
      victoryPlayer;

rule compteurPas of Lapin becomes 0 then
      victoryLapinEnnemi;

rule victory of Player then
      gameEnds;


rule lapin dies then
      reinitialisation;

rule lapin moves then
      subCompteurLapin;


rule missile touches Lapin then
      toucherLapin;
```

Intelligence Artificielle

```
//ensemble de règles à écrire pour obtenir une ia avancée
```

3. Utilisation du moteur 3D (pour programmeur averti)

3DWIGS (3D Web Interactive Game Studio) est une plateforme de création de jeux vidéo pour le web utilisant la technologie WebGL (via la librairie GLGE réalisée par **Paul Brunt(1)**) pour son rendu graphique. Pour ce faire, il

utilise un moteur 3D développé par une des équipes du groupe 3DWIGS.

Le moteur 3D n'est pas destiné à être utilisé directement par l'utilisateur du logiciel. Ses fonctions et méthodes sont implantées dans une couche inférieure du logiciel et sont appelées via la compilation du langage de description.

Cependant, il est possible de passer à travers afin d'améliorer son propre jeu en appelant directement les fonctions et méthodes du moteur.

I. Pré-requis

Pour utiliser le moteur3D, il faut vous munir:

- ✔ D'un navigateur web supportant le HTML5 (Mozilla Firefox-4+, Safari (uniquement sous MacOSX), Google Chrome-11+, Chromium). **Cependant, tous les tests ont été effectués sur Firefox 4 et Google Chrome 11**.
- ✔ De votre éditeur de texte ASCII favori (ex: Notepad++)

II. Utilisation

Avant de commencer à utiliser les fonctions du moteur, il faut indiquer le chemin d'accès à celles-ci afin de pouvoir les utiliser dans le fichier HTML.

Pour cela, il est impératif d'insérer la balise suivante dans le code:

<script type="text/javascript" src="js/moteur3d.js"></script>

(Attention au chemin d'accès du fichier .JS, dans notre cas, le fichier HTML se situe dans le même répertoire que le moteur 3D)

Il faut ensuite appeler la fonction M3D.MOTEUR.initialisation(xmlDoc, canvas, initFunctionName) avec:

- ✔ xmlDoc: le lien vers un document XML contenant l'initialisation d'une scène vide (exemple: level.xml à placer dans le dossier js)
- ✔ canvas: le nom du canvas où afficher la scène
- ✔ initFunctionName: le nom de la fonction qui place initialement les objets; le corps de la fonction initFunctionName peut être vide en cas d'initialisation d'une scène vide.

La partie moteur3d utilise la notion de graphe de scène. En effet la scène et les objets sont des éléments qui peuvent avoir des fils (objet, caméra, lumière). Lorsqu'une transformation est appliquée à un élément, elle est également appliquée à tous ses descendants. Il est également possible de créer des groupes qui ne représentent rien au niveau physique (tel un objet sans collada) mais qui permet de lier le comportement de différents objets.

Dans la scène, l'axe X est dirigé vers la droite, l'axe Y vers le haut et l'axe Z vers l'avant.

Les rotations sont faites dans des repères mobiles.

La partie moteur3d manipule directement les objets de la grammaire. Ceux-ci ont donc les attributs suivants :

- ✔ url : lien vers le modèle de l'objet
- ✔ id : identifiant de l'objet dans la scène (doit être unique)
- ✔ posX posY posZ : position de l'objet dans le repère du père
- ✔ rotX rotY rotZ : rotation de l'objet dans le repère du père
- ✔ sizeX sizeY sizeZ : échelle de l'objet dans le repère du père
- ✔ vitesse : tableau à 3 composants décrivant la vitesse dans le repère du père
- ✔ accélération : tableau à 3 composantes décrivant l'accélération dans le repère du père
- ✔ masse : masse de l'objet (pour l'application de la gravité)

Les éléments seront donc manipulés via leur identifiant : par défaut, l'identifiant de la scène est : 'mainscene'.

III. Manipulation des éléments de la scène

Les fonctions suivantes permettent de manipuler les différents éléments d'une scène.

M3D.MOTEUR.addObjet(gObject,testCollision,idParent) permet d'ajouter un objet à la scène, dont le père est l'élément d'identifiant idParent. Si ce dernier est nul, l'objet est attaché directement à la scène. Le modèle de l'objet est gObject.url (s'il est indéfini, un groupe est ajouté à la scène), il est positionné relativement aux attributs de position/rotation/échelle par rapport au père. Le booléen testCollision permet d'activer (si testCollision vaut true) ou non la détection de collision lors du placement de l'objet.

La fonction retourne un tableau :

- ✔ vide : si testCollision est à faux ou qu'aucune collision n'est détectée
- ✔ dont chaque élément est un tableau à 2 éléments dont le premier élément est l'objet ajouté et le second l'objet avec lequel il est en collision.

Si le tableau retourné est vide, l'objet a bien été ajouté à la scène, s'il est non vide, l'objet n'a pas été ajouté à la scène.

M3D.MOTEUR.removeObjet(idObject) permet de supprimer un objet ou un groupe dans la scène. L'argument idObject correspond à l'identifiant de l'élément que l'on souhaite supprimer. Attention, tous ses fils seront également supprimés. Cette fonction ne renvoie rien.

M3D.MOTEUR.changeParent(gObject,idParent) permet de changer le père d'un objet dans la scène. L'argument gObject représente l'objet de la grammaire dont on veut changer le père. L'argument idParent représente l'identifiant du nouveau parent (la scène si l'argument vaut null). Les attributs de transformation de gObject sont mis à jour. Cette fonction ne renvoie rien.

IV. Transformations géométriques des éléments de la scène

Les fonctions suivantes concernent les transformations géométriques appliquées à un objet dans la scène. Elles prennent toutes en argument gObject (l'objet de la grammaire à déplacer dont les attributs de transformation géométrique sont mis à jour après chaque transformation) et un booléen testCollision qui permet d'activer ou non les collisions lors du déplacement.

Chacune renvoie un tableau :

- ✔ vide si les collisions sont désactivées ou si aucune collision lors du déplacement n'est détectée
- ✔ dont chaque élément est un tableau à lui-même 2 éléments : le premier est un objet (le fils déplacé ou un de ses descendants) et le second l'objet autre dans la scène avec lequel il rentre en collision.
- ✔ si le tableau retourné est vide, la transformation géométrique a été effectuée avec succès. S'il est non vide, la transformation n'a pas été effectuée.

Les fonctions de déplacement et de rotation ont également un argument idRef. Il permet de définir dans le référentiel de quel objet (défini par son identifiant) sont données les valeurs de la transformation à effectuer. Si cet argument est nul, elles sont calculées dans le repère global.

M3D.MOTEUR.translate(gObject,tabVector,testCollision,idRef) permet de translater un objet de tabVector (tableau à 3 composantes) dans le référentiel donné par idRef en considérant ou non les collisions.

M3D.MOTEUR.setPosition(gObject,tabPos,testCollision,idRef) permet de placer un objet à la position tabPos (tableau à 3 composantes) dans le référentiel donné par idRef en considérant ou non les collisions.

M3D.MOTEUR.rotate(gObject,tabRot,testCollision,idRef) permet de faire tourner un objet d'angles tabRot (tableau à 3 composantes) dans le référentiel donné par idRef en considérant ou non les collisions.

M3D.MOTEUR.setAngle(gObject,tabRot,testCollision,idRef) permet de définir l'orientation (via tabRot : tableau à 3 composantes) d'un objet dans le référentiel donné par idRef en considérant ou non les collisions.

M3D.MOTEUR.mulScale(gObject,coefScale,testCollision) permet de multiplier l'échelle d'un objet par les coefficients dans coefScale (tableau à 3 composantes) en considérant ou non les collisions.

M3D.MOTEUR.setScale(gObject,coefScale,testCollision) permet de définir l'échelle d'un objet par les coefficients dans coefScale (tableau à 3 composantes) en considérant ou non les collisions.

V. Application des lois physiques

Le moteur3d permet l'application de règles de la physique à l'univers 3D créé : il est possible de définir une force constante applicable sur plusieurs objets (comme la gravité) décrite par 3 coefficients x, y et z et un coefficient supplémentaire de frottement général qui ralentit la vitesse des objets (multiplie la vitesse par 1-coefficient à chaque calcul de la scène).

La fonction d'application de la physique met à jour les variables vitesse et accélération pour tous les objets concernés.

M3D.MOTEUR.setGravite(newGravite) permet de définir le tableau à 3 composantes de la gravité.
Cette fonction ne renvoie rien.

M3D.MOTEUR.setCoefFrottement(newCoef) permet de définir un nouveau coefficient de frottement.
Cette fonction ne renvoie rien.

M3D.MOTEUR.applyPhysique(gObjList,gGObjList) applique les lois de la physique à différents objets de la grammaire. Les arguments gObjList et gGObjList sont tous deux des tableaux dont chaque élément est un objet de la grammaire. Le premier correspond à la liste des objets auxquels seront appliqués les lois de la physique sans la gravité, et le second avec la gravité.

La fonction renvoie un tableau dont la taille est la somme des tailles des deux tableaux en arguments.

Chaque élément de ce tableau contient un tableau :

- ✔ vide si l'objet a pu être déplacé sans collision (dans ce cas l'objet a bien été déplacé)
- ✔ dont chaque élément est un tableau à 2 éléments avec le premier qui est l'identifiant de l'objet qui a voulu être déplacé (ou l'un de ses descendants) et le second l'objet de la scène avec lequel il est en collision (si le

tableau est non vide, l'objet n'a pas été déplacé, il faut le translater du tableau à 3 composantes de l'attribut vitesse si on souhaite ignorer les collisions).

VI. Manipulation des caméras

De la même façon que l'on peut effectuer des transformations géométriques sur les objets ou les manipuler dans la scène, on peut le faire sur les caméras (attention une caméra ne peut avoir de fils).

M3D.MOTEUR.activeCamera(idCamera) permet de définir une caméra comme celle de la scène. L'argument idCamera représente l'identifiant de la caméra à activer. Cette fonction ne renvoie rien.

M3D.MOTEUR.addCamera(gCamera,idParent) permet d'ajouter une caméra dans la scène dont l'identifiant est donné par l'attribut id de gCamera. Les positions et orientations sont données via les attributs posX posY posZ orX orY orZ dans le référentiel du père donné par son identifiant idParent (si nul, le père est la scène). Cette fonction ne renvoie rien.

M3D.MOTEUR.removeCamera(idCamera) permet de supprimer une caméra uniquement si elle n'est pas active. Cette fonction renvoie un booléen indiquant si la caméra a été supprimée avec succès.

M3D.MOTEUR.translateCamera(gCamera,tabVector,idRef) fonctionne comme pour la fonction M3D.MOTEUR.translate mais avec gCamera objet de la grammaire correspondant à une caméra. De la même façon, il existe des fonctions M3D.MOTEUR.setPositionCamera(gCamera,tabPos,idRef), M3D.MOTEUR.rotateCamera(gCamera,tabRot,idRef) et M3D.MOTEUR.setAngleCamera(gCamera,tabRot,idRef).

VII. Autres outils

Il est possible de calculer la position, l'orientation et l'échelle dans le repère global de l'objet. Ces fonctions retournent chacune un tableau à 3 composantes.

M3D.MOTEUR.getGlobalPosition(idObject) retourne la position dans le repère global de l'élément représenté par son identifiant. De même pour l'orientation avec

M3D.MOTEUR.getGlobalOrientation(idObject) et pour l'échelle avec

M3D.MOTEUR.getGlobalScale(idObject).

Il est également possible de connaître l'écart entre deux objets selon un certain axe.

M3D.MOTEUR.ecart(idObject1,idObject2,dim) retourne l'écart entre les volumes englobant des objets donnés par leur identifiant. Si dim est compris entre 1 et 3 alors l'écart est donné entre le côté inférieur (gauche, bas, avant) du premier et le côté supérieur (droite, haut, arrière) du second, s'il est compris entre -3 et -1, il s'agit de l'écart entre le côté inférieur du premier et le côté supérieur du second.

(1) Le Britannique Paul Brunt est le créateur de la librairie GLGE. Il s'agit d'une librairie JavaScript facilitant l'utilisation du WebGL. Elle permet d'accéder directement à OpenGL ES2 et elle permet ainsi l'accès à l'utilisation de l'accélération matérielle des applications 2D/3D sans avoir besoin de télécharger aucun plugins.

Le but de GLGE est de masquer l'utilisation directe du WebGL au développeur web, qui peut alors se concentrer sur la création d'un contenu plus riche pour le web.

La documentation concernant cette librairie est disponible sur son site web : **http://www.glge.org/**