

//Début logique : boucle de rafraîchissement

refreshLoop ::= signalUpdateCounter **keyboardCommands** **mouseCommands**

//signalUpdateCounter est intégré dans la boucle de rafraîchissement et permet de mettre à jour

//les ressources de type temps, c'est à dire tous les compteurs.

//Des combinaisons de touches clavier déclenchent des signaux

keyboardCommands ::= (keystroke : signalSets)*

//A un clic souris est associé un ensemble de signaux

mouseCommands ::= (typeOfClick : signalSets)*

//Un signal fait appel au gestionnaire d'évènements lors de son activation

//On s'abstrait de la représentation machin d'un signal

signalSets ::= signal (| signal)*

//Gestionnaire d'évènements

eventsManager ::= signal (@ signal)* **instructions** (| signal **instructions**)*

//La sémantique d'une instruction est la même que dans un langage de programmation //

traditionnel, l'utilisation de variables est autorisée via les identificateurs des ressources

//La "fonction" resourceApply(expression) parle d'elle-même, mais pour assurer la //

compréhension, c'est un raccourci pour dire qu'on applique une expression e de calcul à la //

*ressource courante, ex : instructions ::= boisApply(+2*3) \equiv bois = bois + 6 (1)*

instructions ::= resourceApply(expression) | if **conditionnal** then **instructions** (& **instructions**)*

(else **instructions** (& **instructions**)*)? | **conceptsInstructions**

//Instructions intégrant totalement les notions ou concepts du jeu qu'il est utile de prédéfinir

conceptsInstructions ::= gameOver | pause | newGame | saveGame

conditionnal ::= testExpression (booleanOperator testExpression)*

booleanOperator ::= and | or

testExpression ::= expression comparisonOperator expression

comparisonOperator ::= < | > | <= | >= | == | !=

//Le premier opérateur définit la nature de la modification sur la ressource (ref (1))

expression ::= (arithmeticOperator value)+

arithmeticOperator = + | - | * | /

*//expression ::= value (arithmeticOperator value)**

*//arithmeticOperator = + | - | * | /*

//Les ressources sont directement utilisables dans les règles de calculs, il s'agit bien entendu //

d'une référence à leur valeur respective.

value = ressource | **constant**

constant ::= int | double

```
//Gestion des ressources
//Un signal est émis à chaque modification de la ressource (= à chaque utilisation de //
resourceApply(expression)
resourcesSets ::= name signal? (@ signal)* (timer|initValue)
//Géré par une table des symboles, un name est unique et peut éventuellement avoir été généré
//lors de la compilation de la première grammaire pour des attributs prédéfinis hérités
name ::= string
```

```
//Une valeur d'initialisation du timer par défaut est attribuée lors de la première compilation
timer ::= step initTimer
initValue ::= int | double
```

```
//Gestion des caméras et des entités
//Le "name" est le même que celui des ressources
camera ::= name position
//On s'abstrait de la représentation de la position pour le moment
position ::= vector | angle
```

```
entities ::= map with object+
```

```
//En considérant la carte comme une matrice de points dans l'espace sur laquelle est //
appliquée une texture unique
map ::= matrix texture
```

```
//Le "name" est le même que celui des ressources
object ::= name object3D parameters
parameters ::= coeffOfFriction = double weight = double speed = double position = vector
includingVolume = object3D isFixed = boolean isTraversable = boolean
object3D ::= model texture
model ::= [a-zA-Z]*.obj
texture ::= [a-zA-Z]*.mat
```

```
Moteur physique
physicsEngine ::= forces+ collision
forces ::= gravity | wind |...
//En considérant que les volumes englobants sont donnés par l'utilisateur
collision ::= object3D
gravity ::= vector
wind ::= vector
```

