Évaluation des outils auteurs de jeux sérieux, sans programmation

Mémoire de stage de Master sous la direction du Professeur Jean-Marc Labat

Bertrand MARNE

Septembre 2010

Sommaire

Remerciements

Je remercie

Jean Marc Labat, pour m'avoir immergé dans le domaine passionnant des *Serious Games* et pour son aide, ses encouragements et ses conseils.

Élisabeth Delozanne, pour m'avoir donné le goût de la recherche, pour son exigence et ses indispensables conseils en méthodologie de la recherche ainsi que tout au long de notre cursus de DU TICE/Master IFL.

Christophe Desimeur, pour son aide précieuse, notamment avec le test de *Multimedia Fusion Creator 2*.

Dans l'équipe *MOCAH* qui m'a accueilli pour ce stage, **Aissam Kenef** pour son écoute et son soutien, **Amel Yessad** et **Pradeepa Thomas** pour leurs conseils.

Et bien sûr, mon épouse **Anne-Zoé**, pour sa patience et ses encouragements (incluant de délicieuses tartes !)

I- Introduction

I.1- Contexte : une ingénierie naissante des jeux sérieux

Définir le terme de jeu sérieux (*serious game*, mais aussi *learning game*) peut se faire de bien des façons. Ainsi (Natkin, 2004) en donne la définition large suivante :

> *[…] l'utilisation des principes et des technologies des jeux vidéo pour des applications qui n'ont pas de caractère strictement ludique [...]*

Cette définition peut englober à la fois des jeux dont le cœur du *gameplay*[1] permet l'apprentissage et des jeux dont le gameplay sert de transition entre deux séquences d'apprentissage plus classiques (comme certains produits « ludo-éducatifs »). (Fabricatore, 2000) différencie ces deux façons de concevoir les jeux sérieux en parlant de *métaphore extrinsèque* pour ceux dans lesquels l'aspect ludique vient en complément de l'apprentissage et de *métaphore intrinsèque* pour ceux dans lesquels l'apprentissage est au cœur du gameplay. Ainsi, une définition plus restreinte des jeux sérieux, fondée sur la métaphore intrinsèque, serait d'après (Fabricatore, 2000) :

> *[…] a virtual environment and a gaming experience in which the contents that we want to teach can be naturally embedded with some contextual relevance in terms of the game-playing [...]*

Comme (Fabricatore, 2000), certains auteurs montrent que l'apprentissage peut-être facilité par l'utilisation de jeux sérieux. Ils expliquent par exemple que les jeux sérieux augmentent la motivation et l'engagement des apprenants (Prensky, 2004), améliorent leur persistance face aux tâches difficiles et leur envie de recommencer (Kearney and Pivec, 2007), voire permettent un apprentissage approfondi (Oblinger, 2004).

1 Le mot *jouabilité* a été très récemment choisi (JORF n°0167 du 22 juillet 2010 page 13542 texte n°105) comme traduction de « *gameplay* » par la commission générale de terminologie et de néologie du ministère de la Culture avec la définition suivante : « *Ensemble des possibilités d'action offertes à un joueur par un jeu vidéo ; par extension, qualité du jeu appréciée au regard de ces possibilités* ». Le terme *gameplay* tel que nous allons l'utiliser recouvre aussi d'autres sens : à la fois les mécaniques d'un jeu vidéo, ses règles, la façon d'y jouer, l'évolution de la difficulté dans le jeu, etc. soient toutes les facettes de la jouabilité (dans le sens français) et de sa conception.

Mais une des difficultés majeures de leur déploiement, c'est leur coût de développement (environ 15 000 € par heure de formation selon (Marfisi-Schottman et al., 2009)) qui est une problématique commune avec les EIAH (**E**nvironnements **I**nformatiques pour l'**A**pprentissage **H**umain) en général (Murray, 1999). Notamment dans le cas des jeux sérieux, les formateurs ou les concepteurs de cursus d'apprentissage, comme les *game designers*[1] sont rarement capables de mener un développement informatique complet aussi élaboré que celui d'un jeu vidéo. Ils doivent être assistés par une équipe de développeurs ce qui rend la réalisation longue, complexe et coûteuse.

Les outils auteurs sont des outils conçus pour faciliter le développement de logiciels. Cette expression est principalement utilisée dans deux domaines : la réalisation d'EIAH et celle de jeux vidéos. Ces outils cherchent à rendre possible la réalisation de jeux vidéos, ou d'EIAH à des auteurs qui sont pas ou peu informaticiens. Pour les systèmes de tuteurs intelligents, (Murray, 1999) définit l'objectif des outils auteurs ainsi :

> *The main goal of an ITS authoring system is to make the process of building an ITS easier. This ease translates into reduced cost and a decrease in the skill threshold for potential authors.*

Cette idée se démocratise même dans le grand public comme le montre la commercialisation de produits pour réaliser des jeux vidéos comme *Wario Ware Do It Yourself* pour Nintendo DS[2], *App Inventor for Android* (Google)[3], *UBI Art Framework* (UBI Soft)[4], MultiMedia Fusion Creator *2*[5] (dont il sera question lors de son évaluation p. 29). Tant dans le cadre des EIAH, que des jeux vidéos, les outils auteurs peuvent prendre des formes très diverses et intervenir à différents moments du développement du produit final. Ils peuvent être des éléments d'une chaîne d'ingénierie, comme des outils tout-en-un qui permettent la conception complète.

Cependant, il existe très peu de travaux sur la conception d'outils auteurs liant les deux univers des EIAH et des jeux vidéos. L'ingénierie des jeux sérieux reste à construire dans le but d'une industrialisation de leur conception (Burgos et al., 2008. Moreno-Ger et al., 2007. Marfisi-Schottman et al., 2009).

L'équipe *MOCAH* (**M**odèles et **O**utils en ingénierie des **C**onnaissances pour l'**A**pprentissage **H**umain) du LIP6 (**L**aboratoire d'**I**nformatique de **P**aris 6, l'Université Pierre et Marie Curie) est

1 L'expression « *game designer* » peut être traduite par *concepteur jeu*, mais le sens compris est parfois plus large que celui qui est donné à « *game designer* » : celui qui invente/conçoit le gameplay d'un jeu vidéo.
2 http://www.wariowarediy.com/
3 http://appinventor.googlelabs.com/about/
4 http://ubi-art.fr.ubi.com/
5 http://www.clickteam.fr/

impliquée dans le projet *SE-SG* (Méthodes et outils pour le développement de Serious Games) dont le but est de formaliser la conception des jeux sérieux notamment par la création d'outils auteurs (Yessad, Labat, and Kermorvant, 2010). C'est dans le cadre de cette équipe que j'effectue mon stage pour apporter un éclairage sur les outils auteurs déjà existants permettant de concevoir des jeux sérieux.

I.2- Objectifs du stage : exploration et évaluation d'outils auteurs

L'étude menée dans le cadre de mon stage se limite aux outils auteurs « sans programmation ». L'expression est à comprendre dans son acception la plus courante, c'est-à-dire *sans écrire de lignes de code*. En effet, deux phases se distinguent dans la programmation : une phase de *conception* pendant laquelle les tâches à faire effectuer au programme sont découpées en éléments logiques qui interagissent (conception algorithmique), et une phase de *codage* pendant laquelle ces éléments logiques et leurs interactions sont décrits dans un langage de programmation.

Fabriquer un programme aussi complexe qu'un jeu ou qu'un jeu sérieux nécessite une phase de conception logique (algorithmique). Par contre, le codage de cette logique ne se fera pas obligatoirement de façon textuelle (langage de programmation). Les outils visés par mon stage permettent de faire un codage plutôt visuel que textuel grâce à des interfaces en manipulation directe (du type WYSIWYG).

En m'inscrivant dans le thème plus vaste de l'ingénierie des jeux sérieux, j'ai donc fait une étude exploratoire des outils auteurs « sans programmation » existants, en essayant de savoir comment ils peuvent être utilisés et quels genres de jeux sérieux ils permettent de concevoir.

Après avoir situé mon travail dans quelques-unes des recherches sur les outils auteurs (d'EIAH et de jeux sérieux), je présenterai la méthodologie de mon évaluation des outils auteurs avant de donner les résultats afin de tenter de répondre à ces questions.

II- État de l'art : Scénariser plutôt que réaliser

Les études autour des outils auteurs de jeu sérieux sont encore peu nombreuses. Elles s'inscrivent dans les travaux sur l'ingénierie des jeux sérieux qui sont plus abondants. Nous allons donc voir quelles approches émergent de ces études et comment les outils auteurs s'y intègrent. Puis nous verrons plus spécifiquement quelles méthodes utiliser pour sélectionner des outils auteurs.

II.1- Une ingénierie naissante des jeux sérieux

Serious Game Conceptual Framework est un cadre de **conceptualisation** des jeux sérieux proposé par (Yusoff et al., 2009). Les auteurs distinguent plusieurs étapes dans la conceptualisation d'un projet de jeu sérieux qu'ils ont schématisées. Ces étapes sont reproduites dans la Figure 1.

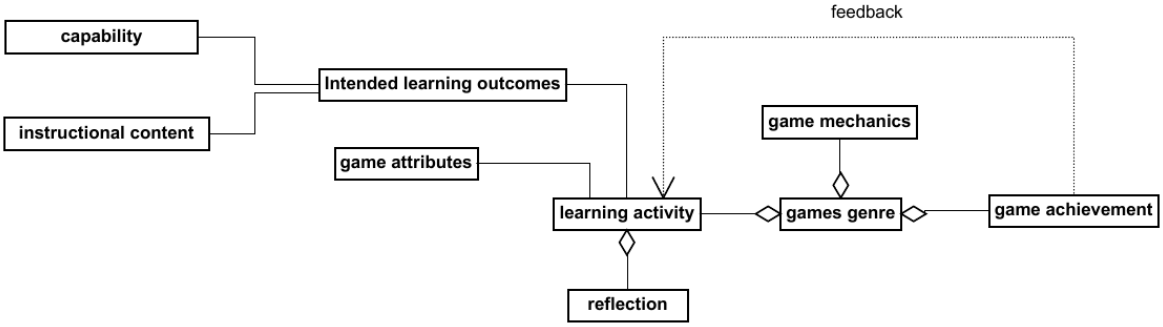


Figure 1 : Diagramme de classe du cadre conceptuel pour les jeux sérieux de Yusoff. Illustration tirée de (Yusoff et al., 2009)

Les compétences (décomposées en *instructional content* et *capability*) permettent de définir des objectifs pédagogiques (*Intended learning outcomes*) auxquels sont associés des attributs de jeu (*game attributes*). D'après (Yusoff et al., 2009) ces attributs sont conçus en s'appuyant sur la littérature pédagogique et les mécanismes d'apprentissage appliqués aux jeux vidéo. Cet aspect est précisé par (Capdevila Ibáñez, Boudier, and Labat, 2009) qui présentent un cadre conceptuel enrichi par le management des connaissances en s'appuyant sur l'expérience du développement du jeu sérieux *StarBank*[1]. Ils expliquent que typer les connaissances rassemblées est la façon de faire

1 http://starbankthegame.bnpparibas.com/

émerger les objectifs pédagogiques. Parmi ces objectifs pédagogiques, ceux qui sont considérés comme jouables seront intégrés au gameplay, les autres seront intégrés au scénario du jeu. La Figure 1 montre que d'après (Yusoff et al., 2009), c'est ainsi que l'activité d'apprentissage (*learning activity*) est définie. Les éléments de gameplay y sont intégrés grâce à une modélisation cognitive qui permet aussi de définir le genre du jeu (Capdevila Ibáñez, Boudier, and Labat, 2009). Selon (Yusoff et al., 2009), durant ces activités d'apprentissage des moyens et des temps doivent être prévus pour permettre une certaine métacognition, autorisant l'apprenant-joueur à prendre un recul suffisant pour réfléchir (*reflection*) sur ce qu'il a accompli et sur ce qui reste à accomplir (*game achievement*). Cet ensemble formé par l'activité d'apprentissage, les mécaniques de jeu et la métacoginition de l'apprenant-joueur déterminent le genre de jeu à concevoir (*games genre*). La Figure 1 montre donc des relations entre ces différents éléments qui sont cœur d'une ingénierie des jeux sérieux et qui doivent permettre de **conceptualiser** et de **scénariser** le jeu avant sa réalisation.

Pour compléter ces travaux sur la conceptualisation des jeux sérieux, (Huynh-Kim-Bang and Labat, [s.d.]) décrivent la création d'une bibliothèque de bonnes pratiques structurées en *design patterns* pour faciliter la conception des jeux sérieux en associant plaisir *(fun)* et pédagogie[1].

Ces éléments de **scénarisation** peuvent être replacés dans une chaîne de production de jeux sérieux : (Marfisi-Schottman et al., 2009) proposent d'utiliser la méthode industrielle japonaise des *5M* (**M**ain d'œuvre, **M**atériel, **M**atière, **M**éthode, **M**ilieu) pour concevoir des jeux sérieux. En se réappropriant (Yusoff et al., 2009) et (Capdevila Ibáñez, Boudier, and Labat, 2009), les auteurs expliquent qu'à partir des vœux du client une maquette est conçue en extrayant les connaissances, en définissant les objectifs pédagogiques et en identifiant des *moments didactiques* pour typer au mieux le scénario du jeu. Dans la maquette, les *moments didactiques* doivent être le plus possible associés à des *activités* (ex. : QCM, mots croisés, etc.) grâce à leurs métadonnées. Ensuite, cette maquette « storybordée » sous forme d'actes et de scènes et formalisée en XML est testée par des joueurs automatiques. Si elle passe ces tests, elle sert à la conception d'un prototype (souvent par des prestataires différents de ceux qui ont conçu la maquette) afin de faire un contrôle de cohérence et un débogage. Enfin, les prototypes techniquement valides sont testés par des utilisateurs réels et éventuellement ajustés par les experts en pédagogie qui avaient conçu les éléments du story-board. Cela peut donner naissance à de nouveaux prototypes à leur tour évalués. Dans cette étude, il est envisagé que la réalisation du jeu soit exécutée par des équipes extérieures aux concepteurs, il en était de même dans les papiers sur la conceptualisation(Yusoff et al., 2009. Capdevila Ibáñez,

1 Cette bibliothèque de *design patterns* est détaillée dans «Conception des spécifications d'un jeu sérieux pour l'évaluation des outils auteurs » p. 19

Boudier, and Labat, 2009). Par conséquent, peu d'études spécifiques aux jeux sérieux abordent cette phase de la **réalisation**.

<e-Game> (Moreno-Ger, Martinez-Ortiz, and Fernández-Manjón, 2005) est justement un projet qui s'intéresse au passage de la scénarisation au jeu lui-même. Les auteurs partent du constat que les outils de développement sont soit trop complexes (ex. : *DarkBasic*, *LUA*) soit trop basiques (ex. : *Game-Editor*[1]) pour que les concepteurs d'un scénario – pédagogues, game designers – puissent réaliser le jeu eux-mêmes. La solution que ces auteurs proposent est l'utilisation d'un **langage dédié** : *Domain Specific Language* ou *DSL (Moreno-Ger et al., 2006)*.

L'idée directrice est qu'un *langage dédié* spécifique d'un domaine peut être facilement compris par les experts de ce domaine. En se limitant au genre des jeux d'aventure et en s'appuyant sur des cas concrets, les auteurs ont choisi un développement incrémental partant du XML auquel ils ont ajouté au fur et à mesure les éléments dont les experts du domaine avaient besoin. Pour un langage dédié avec un domaine étroit (jeu d'aventure), il a suffi d'être déclaratif. Ainsi, la conversion du story-board écrit en langage naturel peut se faire tout simplement par l'inclusion des balises du langage dédié. Le document produit doit pouvoir être exécuté par un **moteur de jeu** pour être testé, et être directement modifiable par les concepteurs du scénario[2].

Dans une ingénierie des jeux sérieux, le **langage dédié** s'insère comme lien entre la **scénarisation** et la **réalisation**.

II.2- Des outils auteurs de jeux sérieux « sans programmation », mais fondés sur un langage dédié

Les outils auteurs de jeux sérieux qui ont bien été étudiés sont, comme dans le projet *<e-Game>*, des outils capables de produire un **langage dédié** qui sera ensuite exécuté par un **moteur**. Ces outils se placent donc souvent en amont de la **réalisation** du jeu, autour de la phase de **conceptualisation** et de **scénarisation** et permettent aux concepteurs pédagogiques et aux *game designers* d'éviter la réalisation d'un moteur de jeu.

Seuls certains se distinguent en proposant des *interfaces en manipulation directe* de codage évitant la rédaction textuelle de lignes de code ou même l'écriture dans le langage dédié lui-même. Nous allons évoquer trois de ces outils *sans programmation* : *IBIS*, *<e-Adventure>* (du projet *<e-Game>*) et *SeGAE*.

1 Ce logiciel est détaillé dans «Évaluation de Game-Editor » p. 31
2 Voir aussi la description de *<e-Adventure>* p. 10.

MOCAS (*MOtivationnal and Culturally Aware System*) est un système ludo-éducatif (Blanchard and Frasson, 2006). Développé à l'Université de Montréal, il permet de réaliser des scénarios de jeu d'exploration à la première personne. L'exemple illustré par le papier est un jeu permettant d'étudier la mythologie grecque. La capture d'écran du jeu proposée par les auteurs montre bien le genre de « gameplay » envisagé.

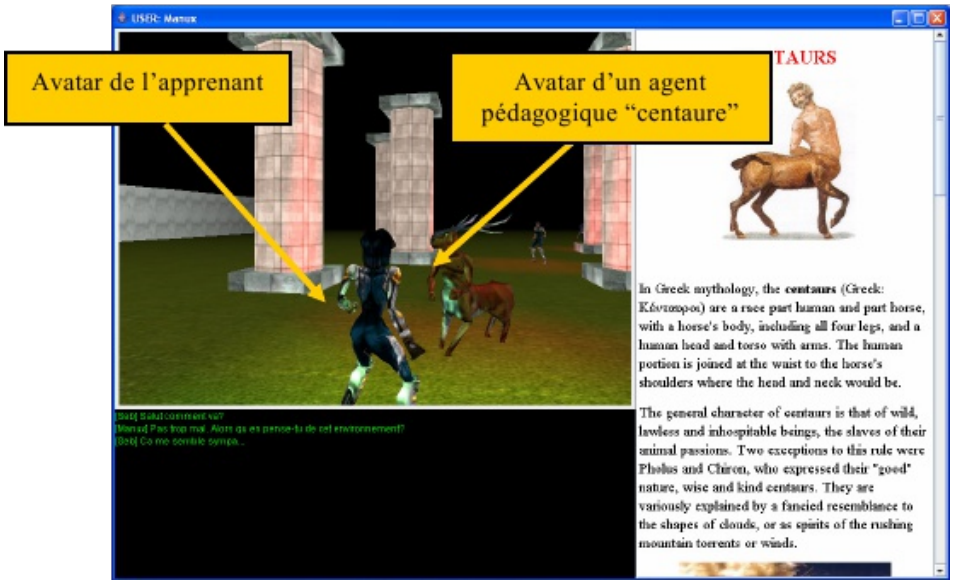


Figure 2 : Capture d'écran d'un jeu conçu avec *MOCAS*. Tirée de (Blanchard and Frasson, 2006)

Il y a trois zones dans la Figure 2. La zone dans laquelle le joueur déplace en 3D son avatar à la recherche des agents pédagogiques conversationnels. À droite une zone dans laquelle les connaissances contextuelles du cursus sont décrites. Enfin, en bas, la zone de conversation. L'apprentissage se fait par le dialogue avec les agents pédagogiques, et la consultation des connaissances contextuelles au cours de l'exploration.

Pour permettre aux experts pédagogiques de développer le scénario (des cours) sans programmer, (Blanchard and Frasson, 2006) proposent des interfaces graphiques d'édition : *Intuitive Builder for Intelligent Systems* (*IBIS*). Selon des codes établis, le dédale de la zone de jeu peut être conçu directement à partir d'une image de carte fabriquée avec n'importe quel éditeur d'image. Le plus complexe est d'élaborer la base de connaissances et les interactions avec les agents pédagogiques. *IBIS* propose donc un module de représentation des connaissances sous forme de graphe conceptuel (Figure 3) car les auteurs estiment comme (Paquette et al., 2006) que c'est une façon accessible de représenter l'information. Dans ce graphe, les nœuds sont les concepts qui sont reliés entre eux par des liens d'inclusion et de précédence. Seules les « feuilles » de ce graphe en arbre peuvent contenir les *ressources didactiques* ou les *points de validation*. Cet outil produit un fichier XML dans un langage dédié qui sera interprété par le moteur du jeu.

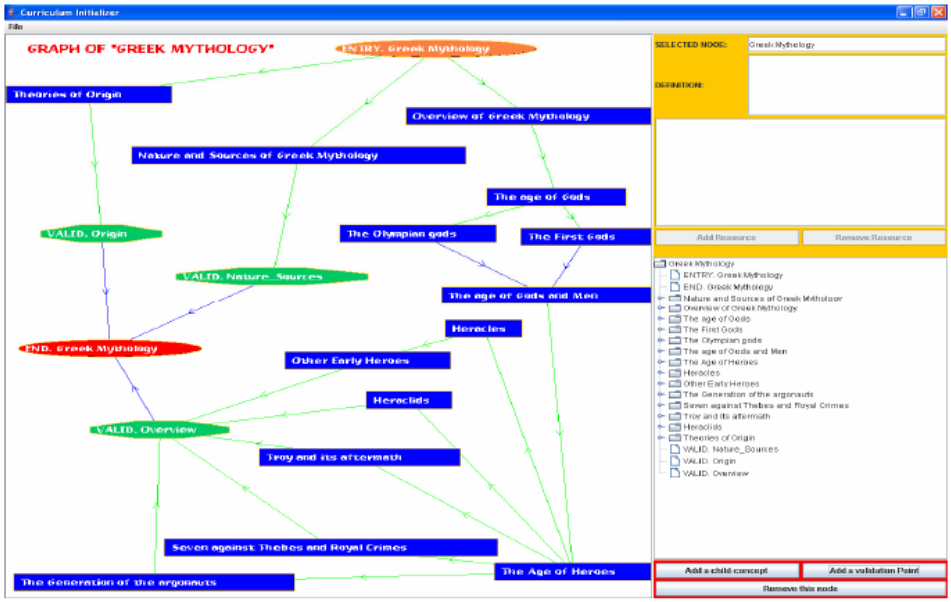


Figure 3 : Édition du curriculum dans *IBIS*. Tiré de (Blanchard and Frasson, 2006).

- Les rectangles représentent les concepts principaux.
- Les ovales oranges et rouges sont respectivement les points d'entrée et de fin.
- Les hexagones sont les points de validation.

Pour les agents pédagogiques conversationnels, *IBIS* permet seulement de choisir leur aspect et de leur affecter une partie du curriculum conçu dans l'interface d'édition afin de limiter l'étendue des concepts sur lesquels ils peuvent converser. Mais l'interface ne permet pas de gérer finement le contenu et les mécanismes de cette conversation.

IBIS est donc un exemple d'outil auteur permettant de scénariser pour *MOCAS*. Bien que, selon l'idée de la métaphore intrinsèque (Fabricatore, 2000), *MOCAS* ne soit pas vraiment un jeu sérieux mais plutôt un produit ludo-éducatif, *IBIS* donne des pistes sur les moyens de créer une base de connaissances dans une ingénierie des jeux sérieux.

<*e-Adventure*> qui se place dans le projet <*e-Game*> est par contre bien un projet d'outil auteur pour les jeux sérieux au sens de la métaphore intrinsèque. Il s'agit notamment d'une interface graphique qui assiste les auteurs de scénarios et produit directement un programme codé dans le langage dédié de <*e-Game*>, exécutable par le moteur du jeu. Cette interface graphique propose même un accès intégré à l'ensemble éditeur graphique et moteur/interpréteur du langage dédie (Burgos et al., 2008). Dans <*e-Adventure*>, le développement du langage et son implémentation sont intimement liés à la conception du moteur de jeu. Un *loader* permet de transformer les informations du langage dédié en informations utilisables par le moteur de jeu (Moreno-Ger et al., 2006).

<e-Adventure> permet de construire des jeux sérieux *point-and-click.* C'est-à-dire des jeux dans lesquels l'apprenant-joueur agit en pointant des éléments sur l'écran et en les cliquant. Les exemples les plus connus de jeux *point-and-click* sont *Myst* (jeu à la première personne) ou *The Secret Of Monkey Island* (jeu à la troisième personne). Le langage dédié de *<e-Adventure>* décrit des *scènes* correspondantes aux écrans de jeu dans lesquelles l'apprenant-joueur peut se déplacer et interagir, soit avec des Personnages Non Joueurs (*PNJ*), soit avec des objets. Les PNJ sont capables d'avoir des conversations arborescentes avec le joueur. Le joueur peut avoir des actions sur les objets comme prendre, utiliser, regarder, etc. Ces objets ont aussi des états (activé, ouvert, visité, etc.) que le joueur peut faire changer et qui donnent la dynamique du jeu. Ces éléments sont implémentés directement dans le langage. Ils sont conçus pour être facilement écrits et lus par des utilisateurs non informaticiens.

Toutefois, l'interface graphique cherche à éviter aux auteurs l'édition directe du XML. Cette interface permet d'éditer le storyboard grâce à des modules qui sont spécialisés dans les éléments de scénario : scènes (Figure 4), objets, PNJ, conversations (Figure 5), etc.

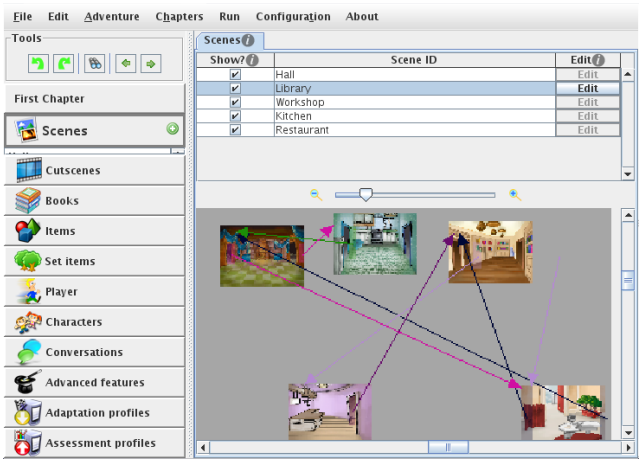


Figure 4 : Relations entre les scènes dans *<e-Aventure>*

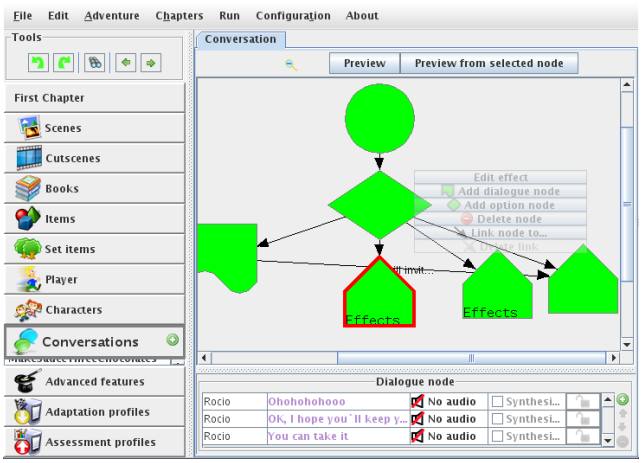


Figure 5 : Arbre de conversation dans *<e-Adventure>*

Dans le laboratoire LIP6 de l'UPMC qui m'accueille pour mon stage, *SeGAE* (**Se**rious **G**ame **A**uthoring **E**nvironment) est développé pour un genre de jeu similaire (Yessad, Labat, and Kermorvant, 2010). Conçu pour le jeu sérieux de rôle *Blossom Flowers*[1], *SeGAE* permet aux concepteurs pédagogiques d'éditer des scénarios de missions dans une interface graphique. Cette interface graphique est générée à partir d'un modèle de mission du jeu, et produit une sortie dans un langage dédié qui pourra être interprétée et exécutée par le moteur du jeu. Alors que l'interface de *<e-Adventure>* permet essentiellement une programmation déclarative (écriture d'un story-board *arborescent), SeGAE* permet de faire une programmation **événementielle**. Dans *SeGAE*, un

1 http://www.ktm-advance.com/viewProject_fr.php?id=108

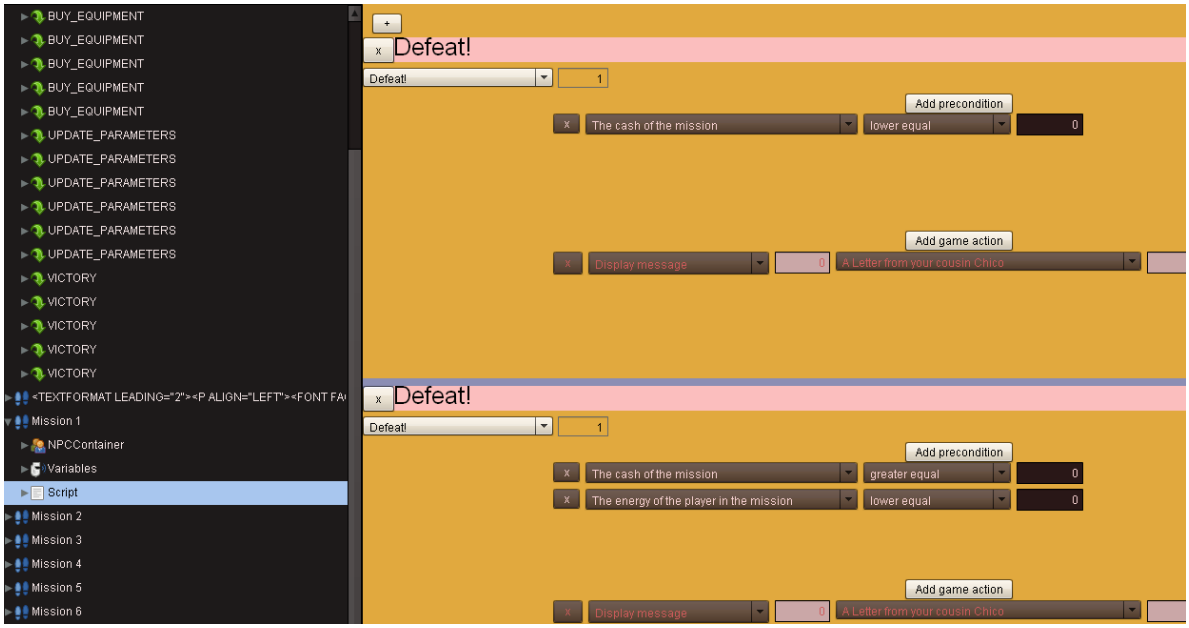


Figure 6 : Programmation événementielle dans *SeGAE*

événement dans le jeu a des pré-conditions sous la forme « `variable1 comparaison variable2` ». Où `variable1` est un paramètre du jeu, `comparaison` peut être une égalité ou une inégalité, `variable2` est soit un autre paramètre du jeu, soit une valeur fixée par les auteurs. Lorsque les pré-conditions sont remplies, elles déclenchent des post-actions. Par exemple la capture d'écran de *SeGAE* (Figure 6), montre que pour qu'il y ait défaite dans la mission 1, la **pré-condition** est que la variable « `The cash of the mission` » soit inférieure ou égale (« `lower equal` ») à `0`. Cela entraîne une **post-action** qui est l'apparition d'un message (« `A Letter from your cousin Chico` »). Cette programmation événementielle permet d'introduire une grande diversité dans les missions qui sont construites avec *SeGAE*.

SeGAE et *<e-Adventure>* facilitent la réalisation du jeu, mais ni sa conceptualisation, ni sa scénarisation, contrairement à *IBIS* (Figure 3 p. 10) qui permet de modéliser les connaissances d'un domaine. Leur utilisation exige donc d'avoir déjà effectué le travail de conceptualisation en suivant par exemple le cadrage proposé par (Yusoff et al., 2009) que nous avons vu (Figure 1 p. 6). Tous ces outils reposent sur un *langage dédié* qui doit être interprété et exécuté par un *moteur de jeu* qui existe déjà. Ils ne permettent pas de modifier ce moteur ou d'en créer un autre sans programmation. En conséquence, les genres de jeux réalisables avec ces outils fondés sur des langages dédiés *(MOCAS, <e-Adventure>* et *SeGAE*) sont très peu nombreux. Il manque à la fois une forme de généricité de ces langages dédiés à l'ensemble des genres de jeux, et des interfaces graphiques capables de coder des interactions plus variées.

II.3- Des méthodes pour tester des outils auteurs

Pour réaliser des jeux sérieux complets, dont le moteur n'a pas encore été conçu, il faut donc se tourner vers d'autres outils auteurs. Quelles méthodes pour évaluer ces outils ? Tom Murray a réalisé deux études successives très approfondies sur les outils auteurs de tuteurs intelligents (Murray, 1999) et (Murray, 2003) dont certains éléments méthodologiques peuvent être réutilisés pour sélectionner des outils auteurs de jeux sérieux.

Après avoir différencié les outils de type *shell* avec codage textuel des outils auteurs à interface de *manipulation directe* type WYSIWYG comme ceux qui nous intéressent, il pose deux questions fondamentales pour bien choisir parmi eux : Dans quel but utiliser un outil auteur ? (« *Authoring tool goals* »). Qui sont les utilisateurs ? (« *Who are the authors?* ») Parmi les réponses que Murray donne à la question « Dans quel but utiliser les outils auteurs ? » certaines s'appliquent très bien aux outils auteurs de jeux sérieux :

- Pour diminuer les efforts (temps et coût de développement, ressources techniques utilisées)
- Pour diminuer le seuil de compétences nécessaires pour les auteurs. Et proposer une courbe d'apprentissage progressive.
- Pour aider les auteurs à construire leur logiciel sur de bonnes bases pédagogiques : modélisation des connaissances, modélisation pédagogique, définition des objectifs, modélisation cognitive, etc. (Capdevila Ibáñez, Boudier, and Labat, 2009) Et sur de bonnes bases informatiques : qualité de la programmation, modularité, réutilisabilité, etc.
- Pour faire du prototypage rapide. Soit dans une phase de développement comme nous l'avons vue détaillée par (Marfisi-Schottman et al., 2009)[1]. Soit dans le cadre de la recherche pour tester des hypothèses sur les jeux sérieux sans entrer dans des développements au long cours.

D'après Murray, la question « Qui sont les utilisateurs des outils auteurs ? » ne trouve pas facilement de réponses, mais ouvre vers d'autres questions plus précises qu'il sera nécessaire de se poser :

- Quel est le niveau de compétences des auteurs en ingénierie logicielle ?
- Quelles sont les compétences des auteurs en création multimédia ?
- Quelles sont les compétences des auteurs en création pédagogique ?
- Combien de temps ont les auteurs pour apprendre à utiliser l'outil auteur et développer leur EIAH (système de tuteur intelligent ou jeu sérieux) ?

1 Voir la méthode d'industrialisation de la conception des jeux sérieux par les 5M p. 7

– Est-ce un seul auteur ou une équipe d'auteurs (travail collaboratif) ?

Tom Murray fait émerger de ces questionnements quelques principes qui doivent être suivis par les outils auteurs pour atteindre leurs buts. Les interfaces graphiques doivent permettre des manipulations directes (WYSIWYG, clics, glisser-déposer, etc.) pour rendre l'édition rapide et intuitive. Mais aussi pour permettre un retour (*feedback*) efficace permettant aux auteurs de tester leurs modifications de façon visuelle. Ces interfaces doivent comporter les outils de base de tout éditeur : *Undo*, copier-coller, rechercher/remplacer, etc. L'outil auteur doit être adaptable aux différents utilisateurs : à la fois des auteurs expérimentés qui sauront l'utiliser et concevoir un projet *ex nihilo*, et à la fois aux néophytes à partir d'un exemple concret qu'ils enrichissent au fur et à mesure. L'interface doit proposer des moyens de visualisation variés de ce qui a été construit. De sorte que tout au long du développement, les auteurs qui ne sont pas experts en informatique puissent avoir une vision globale de l'ensemble de leur projet. Tout en permettant de bâtir un projet modulaire, dont les morceaux seront réutilisables dans des contextes différents. L'outil auteur doit pouvoir être étendu selon les besoins des auteurs. En proposant par exemple un système de scripts, spécifique et facile d'apprentissage, pour rajouter simplement de nouvelles fonctionnalités. Pour s'intégrer dans un *workflow* complexe, l'outil auteur doit avoir une gestion des droits et des outils d'administration de ces droits. Ces outils d'administration peuvent aussi donner accès à des journaux d'utilisation, voire à une gestion des versions (comme *Subversion* ou *Git*) et à diverses statistiques sur les contenus.

La liste est longue et bien qu'il s'agisse de critères de choix pour des outils auteurs de tuteurs intelligents, ils peuvent servir d'éléments d'évaluation pour les outils auteurs d'autres types d'EIAH comme les jeux sérieux.

Nous l'avons vu, les outils auteurs dédiés aux jeux sérieux qui existent déjà sont principalement axés sur la phase de scénarisation du jeu, et sur la modélisation avec un langage dédié de ce scénario en un story-board exécutable. Il y a donc une lacune concernant les outils capables de construire des moteurs de jeu, ou des jeux sérieux complets.

Mais, est-il possible de rendre la fabrication de jeux sérieux complets accessible aux non-programmeurs comme les experts pédagogiques ou les *game designers* ? Grâce à ces critères d'évaluation j'ai élaboré une méthodologie me permettant d'explorer et de chercher d'autres outils, d'étudier comment ils s'utilisent et quels genres de jeux sérieux ils permettent de concevoir.

III- Méthodologie

Évaluer les outils auteurs est une tâche délicate. D'autant que, nous le verrons, les outils consacrés aux jeux (pas spécialement sérieux) sont assez nombreux. Il m'a donc été rapidement nécessaire de construire une liste de critères de sélection et d'évaluation de ces outils. Après avoir présenté ces critères, nous verrons comment j'ai fait la sélection des outils. Puis comment j'ai testé les logiciels sélectionnés. En particulier, quels sont les spécifications et le scénario de conception que j'ai utilisés pour ces évaluations approfondies.

III.1- Critères d'évaluation retenus

Les critères de Murray exposés dans « Des méthodes pour tester des outils auteurs » (p. 13) furent une base pour tenter d'élaborer une liste d'éléments d'évaluation (Murray, 2003). La plupart de ces critères sont en fait des critères *d'utilisabilité*, j'ai donc entrepris de les réorganiser en les réinsérant dans les 8 critères ergonomiques élaborés par (Bastien and Scapin, 1993) : *Guidage*, *charge de travail*, *contrôle explicite*, *adaptabilité*, *gestion des erreurs*, *homogénéité/cohérence*, *signifiance des codes et dénominations*, *compatibilité*. Cette synthèse est schématisée dans la Figure 7.

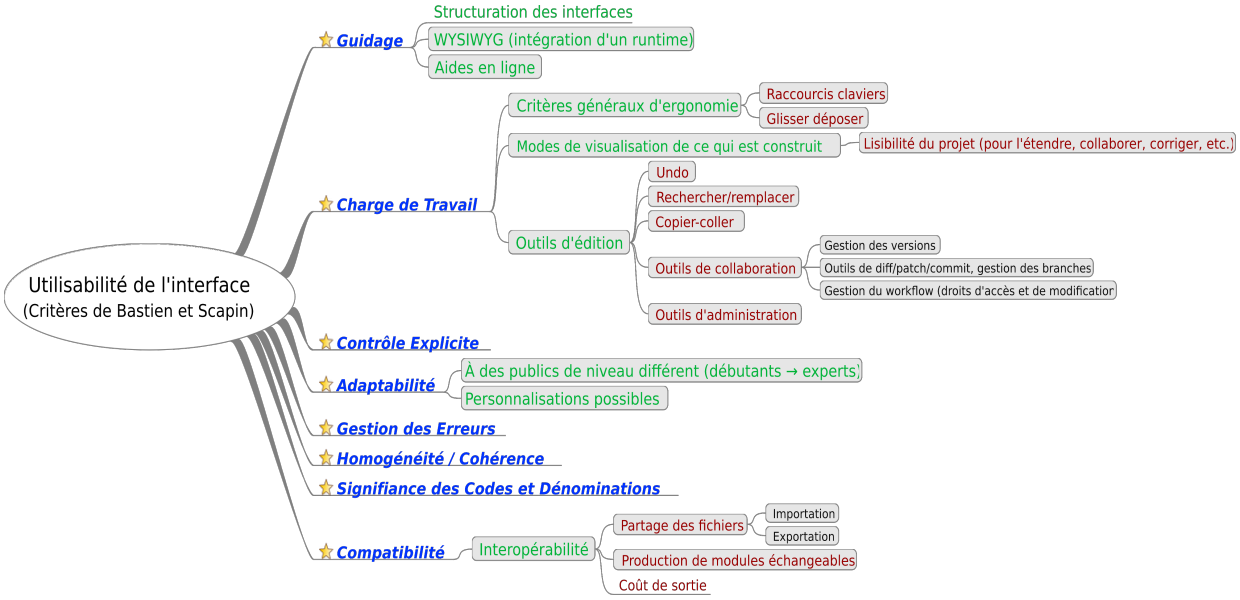


Figure 7 : Synthèse des critères d'évaluation de Murray (encadrés à fond gris) et des critères ergonomiques de Bastien et Scapin (en gras italiques, précédés d'une étoile)

J'ai rajouté quelques items à ceux de Murray, il s'agit de critères souvent utilisés dans le choix des logiciels en général. D'autres critères de ce type ne pouvaient être placés ici, notamment les questions d'**accessibilité** qui ne sont pas directement liées à l'interface humain-machine. C'est pourquoi j'ai créé une série de critères pour l'accessibilité et qui englobe ceux de Bastien et Scapin (Figure 8).

Figure 8 : Critères d'**accessibilité** pour l'évaluation des outils auteurs de jeux sérieux

J'ai aussi élaboré quatre autres séries de critères. L'une de ces séries (Figure 9) vient compléter ceux de la langue vus ci-dessus : **documentation & support**. Et notamment du type de documentation disponible pour le logiciel et du type de support disponible.

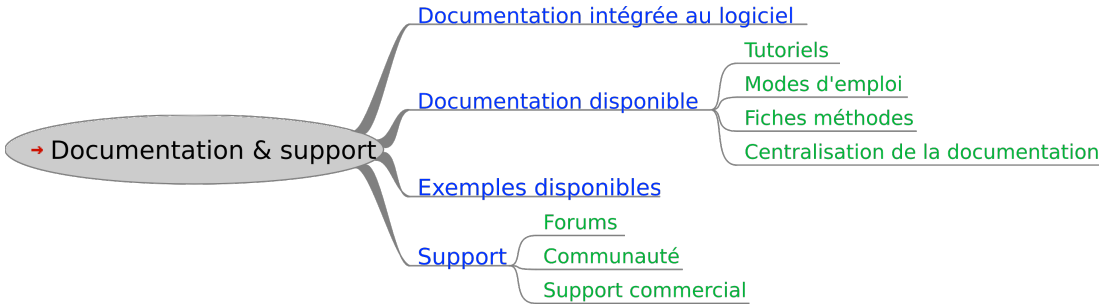


Figure 9 : Critères liés à la **documentation** et au **support** pour évaluer les outils auteurs de jeux sérieux

Une série de critères est assez classique pour les ingénieurs. Elle concerne la viabilité du logiciel dans le système d'information de la structure dans laquelle il s'insère. Notamment les questions de fiabilité, de durée de vie ou de complexité de déploiement sur les postes de travail (Figure 10). J'ai ajouté à ces critères de **maintenabilité** la notion d'extensibilité (par des scripts) issue des critères de Murray.

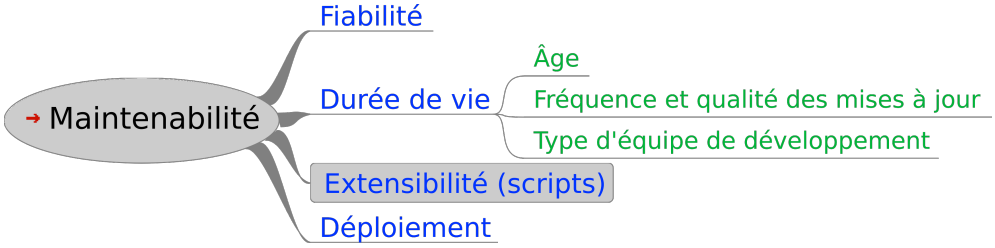


Figure 10 : Critères de **maintenabilité** attendus pour outil auteur de jeux sérieux

La série de critères présentée Figure 11 est spécifique des outils auteurs et de leur capacité à proposer une **programmation visuelle** dans une interface de manipulation directe. En particulier de proposer des outils complémentaires comme des *briques* pré-programmées assurant les fonctions classiques des jeux sérieux. Murray recommandait aussi la présence de fonctions permettant aux auteurs de concevoir et d'utiliser des modèles (*templates*), ainsi que la présence d'outils de débogage. C'est donc dans cette série que je les ai ajoutés.

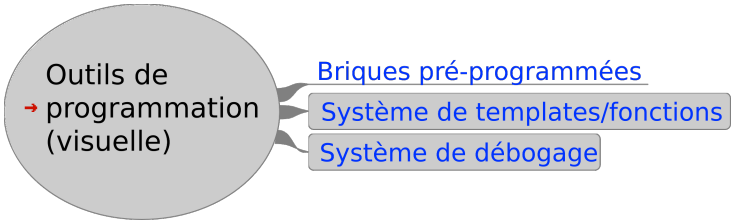


Figure 11 : Critères liés à la **programmation** notamment **visuelle** dans les outils auteurs de jeux sérieux

La dernière série de critères d'évaluation est propre aux EIAH (Figure 12). Elle reprend les critères de Murray présentés plus haut[1] manquants dans les autres séries. Il s'agit de la présence d'**outils de conceptualisation**, surtout en vue d'une modélisation des connaissances et d'une modélisation pédagogique comme dans *MOCAS*[2].

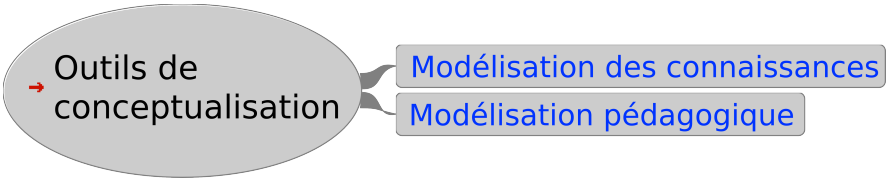


Figure 12 : Critères liés aux moyens de **conceptualisation** dans les outils auteurs de jeux sérieux

Ces séries de critères sont rassemblées dans la Figure 22 en annexe p. 44.

III.2- Choix des logiciels testés

Il existe peu d'outils auteurs dédiés aux jeux sérieux. Nous en avons déjà parcouru quelques-uns pour constater qu'ils ne permettaient pas la construction d'un jeu entier, car ils reposaient sur des moteurs de jeu déterminés et limités en fonctionnalités[3]. Pour trouver de nouveaux logiciels permettant de créer des jeux sérieux de tous les genres, j'avais donc le choix entre les outils auteurs d'EIAH, et les outils auteurs de jeux (tout court). Comme d'après la métaphore intrinsèque de (Fabricatore, 2000) les jeux sérieux sont avant tout des jeux, j'ai choisi de me tourner vers les outils

1 Voir les critères de Murray retenus pour l'évaluation des outils auteurs de jeux sérieux dans « Des méthodes pour tester des outils auteurs » p. 13
2 Voir la présentation de *MOCAS* p. 9
3 Voir la présentation de quelques outils auteurs de jeux sérieux dans « Des outils auteurs de jeux sérieux « sans programmation », mais fondés sur un langage dédié » p. 8

auteurs dédiés aux jeux plutôt qu'à ceux consacrés aux EIAH. Sur internet j'ai trouvé de très nombreuses références d'outils auteurs dédiés aux jeux. C'est à partir des outils suggérés par mon laboratoire et en m'aidant du catalogue du site *Ambrosine*[1] que j'en ai identifié plusieurs dizaines qui étaient considérés « sans programmation ». Il a donc fallu faire une première phase de tri, avant de pouvoir tester en profondeur les meilleurs outils pour créer des jeux sérieux.

III.2.1- Une évaluation en largeur : une sélection lors d'un parcours de l'ensemble des outils trouvés

L'évaluation « en largeur » s'oppose à une évaluation « en profondeur ». Elle a consisté à tester chaque outil rencontré pendant un temps très court, une heure environ, et à choisir de le conserver ou non. De cette façon grandement réduite, la liste des logiciels passe par une seconde itération de tests, un peu plus longs : quelques heures. Et ainsi de suite avec des tests d'une demi-journée, d'une journée, de quelques jours. Au cours de ces tests, j'ai inspecté rapidement les interfaces en m'appuyant notamment sur les 8 critères de (Bastien and Scapin, 1993) (exposés p. 15). J'ai parcouru les principales fonctionnalités des logiciels à la recherche de deux éléments : des briques pré-programmées pour faciliter la réalisation, de moyens visuels pour programmer la logique du jeu (tests, boucles, événements, affectations, etc.). Au cours des tests les plus longs, j'ai essayé de faire le tour des fonctionnalités présentes en étudiant aussi la documentation disponible et en évaluant son abondance et sa qualité.

Les outils auteurs qui ressortent de cette sélection « en largeur » sont de plusieurs genres. Certains sont limités par le gameplay qu'ils autorisent à l'image de *<e-Adventure>*[2]. D'autres permettent la conception de n'importe quel type de jeu. Parmi le premier groupe, il y a ceux qui permettent de faire des jeux d'aventure de type *point-and-click* : *<e-Adventure>* et *Adventure Game Studio. RPG Toolkit* permet de faire des jeux de rôle (*Role Playing Game*). *Scrolling Game Development Kit* est spécialisé dans les jeux d'action (avec un défilement de l'écran). *Silent Walk FPS Creator* permet de faire des jeux en 3D à la première personne (*First Person Shooter*). Dans le groupe des outils auteurs plus généralistes, il y a : *Construct*, *Multimedia Fusion Creator 2*, *Game-Editor*, *Game Maker 8*, *Sim's Carnival Game Studio* et *Sharendipity*. En annexe p. 45 se trouve un descriptif rapide de chacun de ces logiciels.

1 http://www.ambrosine.com/index.php
2 Voir la description détaillée de *<e-Adventure>* p. 10

Il faut noter que j'ai choisi d'exclure les outils proches de *Scratch*[1] (comme *Scratch* lui-même, *StarLogo TNG*[2] ou *App Inventor for Android* évoqué en introduction p. 4) qui permettent une programmation visuelle proche du *Logo* en utilisant des briques comme celles des *Legos* (Resnick et al., 2009). Dans ces logiciels, s'il n'y a pas écriture de ligne de code, la programmation visuelle proposée reste très proche d'une programmation textuelle : les mots-clés des instructions étant remplacés par des objets graphiques (Figure 13). J'ai jugé qu'ils étaient au-delà de la limite des outils « sans programmation ».

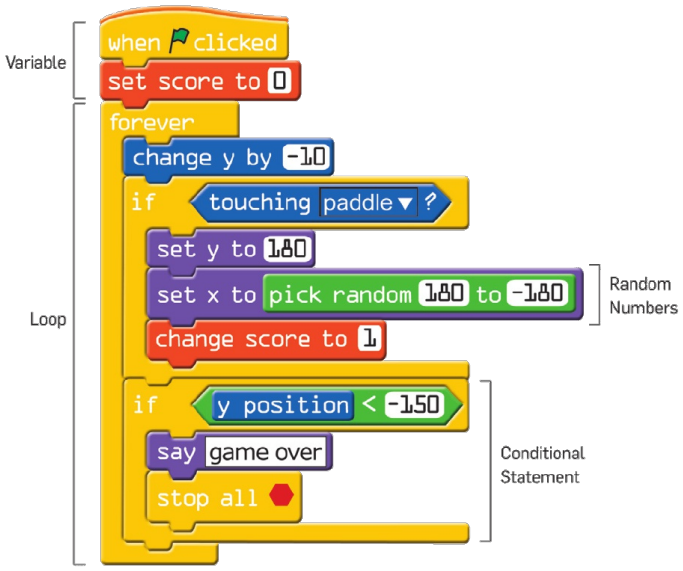


Figure 13 : Exemple de programmation avec *Scratch*. Illustration tirée de (Resnick et al., 2009)

III.2.2- Une évaluation en profondeur : la réalisation d'un jeu sérieux avec chacun des outils sélectionnés

Pour faire une évaluation en profondeur de chacun de ces outils auteurs sélectionnés, il faut les tester dans une situation réaliste d'utilisateur normal. Il est donc vraiment nécessaire de réaliser un jeu sérieux avec eux. Pour cette réalisation, j'ai dû concevoir un scénario de jeu sérieux adapté (voir ci-après). Il n'était pas possible de créer ou de tester plusieurs scénarios dans le temps de mon stage, c'est pourquoi je n'ai retenu qu'un seul genre de gameplay et donc écarté certains outils auteurs en fonction des genres de jeux qu'ils permettaient de créer. Par conséquent, la liste des outils auteurs sélectionnés présentée ci-dessus a été réduite à six : *Construct*, *Multimedia Fusion Creator 2*, *Game-Editor*, *Game Maker 8*, *Sim's Carnival Game Studio* et *Sharendipity*.

III.3- Conception des spécifications d'un jeu sérieux pour l'évaluation des outils auteurs

Le but de mon stage n'était pas de concevoir un scénario de jeu sérieux. Mais, pour tester les outils auteurs en profondeur, il m'a été nécessaire d'en construire un, avec plusieurs objectifs. L'objectif principal était que le scénario soit crédible, et permette d'aboutir à la réalisation d'un jeu réellement utilisable avec des apprenants : le jeu doit rester intéressant et prenant pour les joueurs tout en ne

1 http://scratch.mit.edu/
2 http://education.mit.edu/drupal/starlogo-tng

sacrifiant pas à l'apprentissage. Le second objectif important de la conception du scénario est d'élaborer un jeu dont la réalisation va rendre possible le test de beaucoup de fonctionnalités des outils auteurs. Enfin il était indispensable que le scénario ne soit pas trop long à développer afin d'avoir suffisamment de temps dans mon stage à consacrer à son implémentation dans les outils auteurs.

Devant l'abondance des outils auteurs dans le domaine des jeux d'aventure *point-and-click*, l'idée initiale avait été d'en écrire un sur le thème d'une enquête de police scientifique s'appuyant sur des compétences de génétique et de microbiologie. Après plusieurs jours de conception, il est apparu que les ramifications d'un tel scénario étaient telles que leur développement aurait pris la majeure partie de la durée de mon stage. J'ai donc choisi d'abandonner cette piste (et les outils auteurs qui vont avec) pour me concentrer sur un scénario plus rapide à développer.

J'ai plutôt choisi de développer un jeu sur des compétences en immunologie. Le cursus visé est celui de 3e, et notamment la majeure partie du troisième chapitre des programmes scolaires de Sciences de la Vie et de la Terre (S.V.T.) sur le « *Risque infectieux et protection de l'organisme* ». En annexe p. 48 se trouve l'extrait du programme scolaire concerné.

Pour déterminer le genre de gameplay à développer, j'ai tenté de trouver les *designs patterns* (Huynh-Kim-Bang and Labat, [s.d.]) les plus adaptés à ce cursus. Le système immunitaire est complexe et en réponse à « *How to design game interactions to make understand a complex system?* », *DP-Navigator* (Huynh-Kim-Bang and Labat, [s.d.])[1] suggère « *Make build or modify the system* » et notamment l'utilisation des micromondes (*Microworld Interaction*). Mais aussi « *How to make learn high-level knowledge?* », « *Use intensive action phases to make practice and use less intensive phases to make reflect.* » (voir aussi (Kiili, 2007)) qui incite à l'utilisation d'un gameplay permettant des temps d'action et des temps de réflexion. Parmi les autres *designs patterns* quelques-uns ont aussi paru très adaptés. *Pavlovian Interactions* (« *How to make memorize simple but numerous elements.* », « *Provide repetitive cases where elements are basically used.* ») va permettre aux apprenants de mémoriser les nombreuses associations entre microbes et défenses. La mise en place d'un tutoriel (*Tutorial*), puis d'une liberté de choix dans les éléments joués (*Freedom of Pace* « *How to make a progression adapted to users?* », « *Create a network of choices and let the user choose her pace.* ») permettra une progression adaptée aux joueurs/apprenants. L'utilisation de *niveaux de jeu* reprenant des concepts des niveaux antérieurs, mais présentant une difficulté plus

1 *DP-Navigator* permet de partager et de naviguer dans les *designs patterns* pour jeux sérieux : *Exploring Design Patterns in Serious Games (Project SE-SG)*, http://seriousgames.lip6.fr/DesignPatterns/. les *designs patterns* y sont présentés sous forme d'un diptyque « question-réponse ».

grande pourra à la fois renforcer la motivation du joueur et son apprentissage (Kearney and Pivec, 2007). Des écrans d'information pourront donner les informations scientifiques/de jeu à connaître pour interagir (*Informative Loading Screens* « *How to transmit small chunks of information?* », « *Use loading screens to provide random chunks of information.* »). Enfin, un retour du jeu adapté permettra au joueur de comprendre ce qu'il réussit et d'amorcer une métacognition (*Quick Feedbacks* « *How to help users during action phases?* », « *During action phases, provide just small and immediately useful information to guide users.* » ; et *Fun Rewards* « *How to give incentives for users to progress in the game?* », « *Display and give rewards based on pleasure.* »).

Un genre de jeu correspond particulièrement à ces *designs patterns*. Il s'agit du genre de jeu *tower defense*[1] (voir Figure 14). Il s'agit d'un jeu de stratégie en temps réel particulier dans lequel le joueur doit préparer ses défenses afin de résister à un certain nombre de vagues successives d'ennemis aux compétences particulières. La distinction entre les phases d'action et de préparation pourrait être particulièrement propice à une réflexion de l'apprenant/joueur (Kiili, 2007). Un modèle particulier de jeu *tower defense* correspond particulièrement bien aux *designs patterns* énoncés ci-dessus, il s'agit de *Bubble Tanks*[2] (voir Figure 14), qui présente en plus l'avantage de mieux simuler un environnement ouvert comme le corps et laisser une plus grande liberté aux apprenants-joueurs.

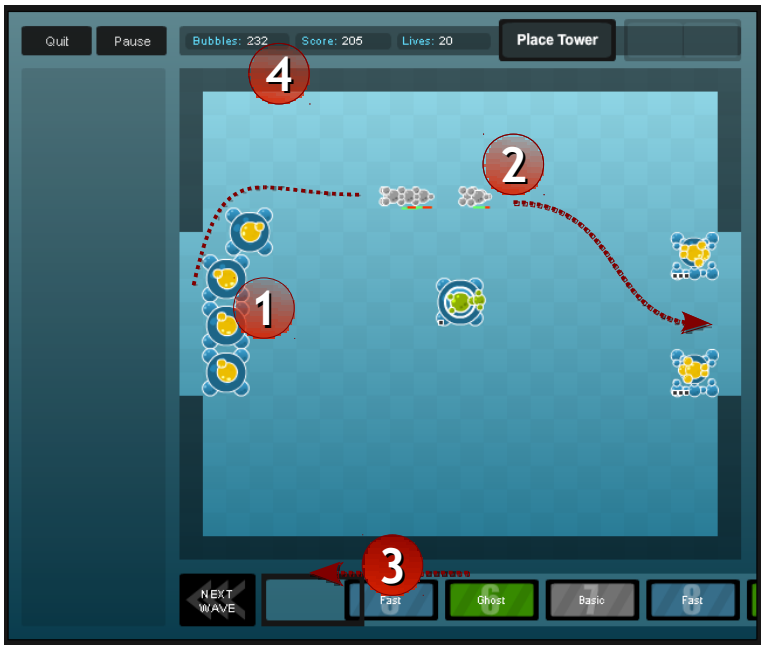


Figure 14 : Exemple de gameplay *tower defense* (capture d'écran de *Bubble Tanks* d'Armor Games http://armorgames.com/play/4962/bubble-tanks-tower-defense)
- 1 : des tours de défense placées par le joueur
- 2 : des ennemis qui tentent de traverser le terrain
- 3 : les types d'ennemis à venir
- 4 : les scores

La métaphore intrinsèque (Fabricatore, 2000) est que le territoire à défendre est le corps humain, les défenses (les tours) placées par le joueur sont le système de défense de l'organisme et les ennemis à détruire sont les microbes.

1 Exemple typique de jeu *tower defense* : http://www.miniclip.com/games/canyon-defense/fr/. Exemple de jeu sérieux *tower defense* : http://www.semconstellation.fr/spip.php?breve89

2 http://armorgames.com/play/4962/bubble-tanks-tower-defense

En m'appuyant principalement sur mon expérience de professeur de S.V.T., mais aussi sur des travaux de similaires de (Clements, Pesner, and Shepherd, 2009) (conception d'un *tower defense* sur les défenses immunitaires) et sur le gameplay de *Bubble Tanks*, j'ai écrit le scénario du jeu intitulé « *Défenses Immunitaires* ».

Dans le jeu *Défenses Immunitaires*, il faut donc empêcher les infections en détruisant les microbes qui tentent de s'infiltrer dans l'organisme. L'apprenant-joueur a une phase de préparation durant laquelle il place les défenses (symbolisées par des tours) adaptées aux vagues de microbes qu'il sait qu'il va rencontrer (l'écran de chargement ainsi qu'un panneau sur l'aire de jeu lui donnent une vision prédictive des vagues de microbes. Voir aussi la Figure 16 p. 23 et la Figure 15). La gestion de la santé (consommée par chaque ajout d'une « tour » de défense) et du parcours des microbes est centrale durant cette phase préparatoire, ainsi bien sûr que l'adéquation défense/microbe. Quand le joueur se sent prêt, il lance lui-même l'infection et provoque ainsi l'entrée des microbes par vagues successives. L'apprenant-joueur peut alors contrôler si le dispositif de défense est adapté aux microbes et l'ajuster si nécessaire, dans la limite de la santé restante (sachant que chaque microbe passant au travers des défenses réduit la santé). L'ajustement se fait tout au long de la longue phase d'infection et permet donc au joueur de bien comprendre quelles sont les bonnes combinaisons entre les microbes et les moyens de défense. À la fin de chaque niveau un bilan permet au joueur de savoir si l'objectif est atteint et avec quel niveau d'efficacité. Une description détaillée du scénario se trouve en annexe p. 50.

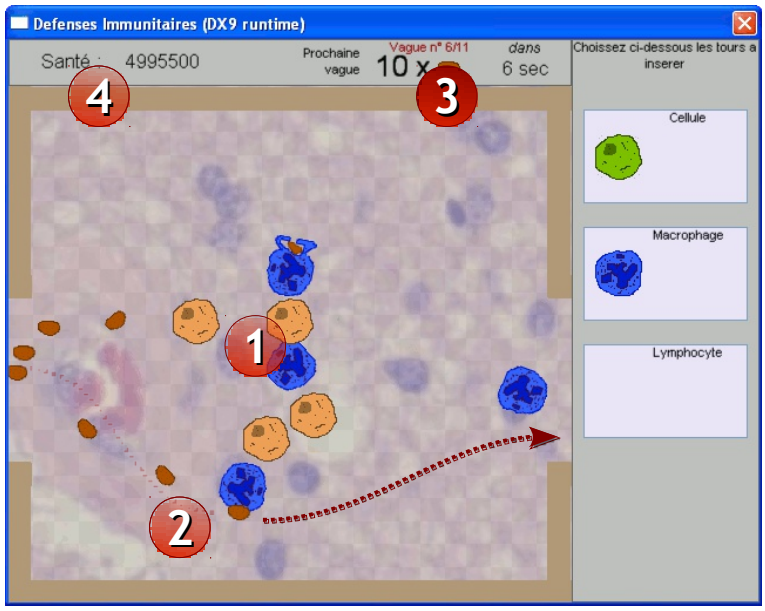


Figure 15: Exemple d'implémentation du scénario de «*Défenses Immunitaires*» (avec le logiciel *Construct* voir p. 27)
- 1 : les cellules de défense placées par le joueur
- 2 : des microbes qui entrent dans l'organisme
- 3 : les types de microbes à venir
- 4 : la santé restante

Les objectifs pédagogiques visés par *Défenses immunitaires* sont de **connaître** les différents types de microbes et les différents types de défenses de l'organisme. **D'associer** les bonnes défenses immunitaires aux bons microbes. De **comprendre** le principe des techniques médicales associées à l'immunité, et le principe des déficiences immunitaires. **D'adopter** un comportement avisé en cas

d'infection. Il est prévu que l'apprentissage s'effectue en deux temps : Au début de chaque niveau par la présentation dans l'écran de chargement des informations de jeu (*Informative Loading Screen* (Huynh-Kim-Bang and Labat, [s.d.])), pendant le jeu par la mise en pratique de ces indications de gameplay (qui sont aussi des informations scientifiques). Un renforcement de l'apprentissage doit s'effectuer lors de la répétions des actions (*Pavlovian Interactions* (Huynh-Kim-Bang and Labat, [s.d.])), la possibilité de refaire un niveau – réussi ou non – et la réutilisation dans chaque niveau de concepts des niveaux précédents (Kearney and Pivec, 2007). Il faudra réfléchir à l'intégration de ce jeu dans un cursus d'apprentissage de niveau 3e afin d'éviter la formation de conceptions erronées dues aux compromis faits à la jouabilité[1], et pour favoriser la métacognition et la synthèse des connaissances par les apprenants.

Ainsi défini, j'ai considéré que le modèle de connaissances de *Défenses Immunitaires* est en grande partie dans les écrans de chargement (dont l'intégralité du texte est lisible en annexe p. 57), le modèle pédagogique est dans le gameplay d'association tour/ennemi et dans la succession des niveaux rejouables. Le modèle de l'apprenant et l'interface devront être conçus grâce aux outils auteurs.

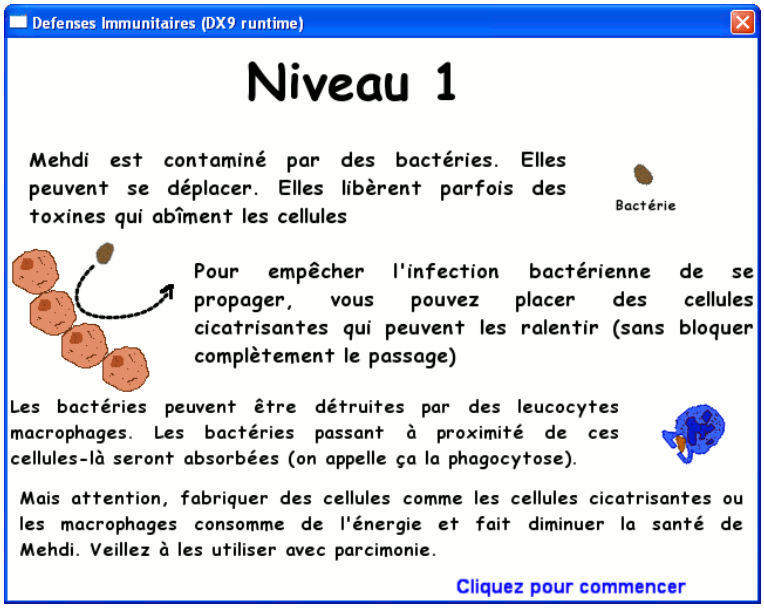


Figure 16 : Exemple d'écran de chargement donnant des informations au joueur pour le premier niveau

1 Par exemple les dimensions relatives des cellules de défenses et des microbes ne peuvent pas être respectées pour permettre le gameplay : en réalité les cellules sont au moins 10 à 10 000 fois plus grosses que les microbes, et dans le jeu ce facteur sera d'environ 2 à 5.

Dès le début des phases de tests avec les outils auteurs il est apparu insuffisant d'avoir seulement un scénario, mais nécessaire d'avoir des spécifications complètes et détaillées du jeu. Par exemple, expliciter clairement quelles sont les propriétés de chaque tour : leur coût, leur vitesse de réaction, leur champ d'action, leur niveau de mise à jour, et le coût de cette mise à jour, etc. Autre exemple, définir clairement quels sont les éléments à initialiser au début de chaque niveau (voir les tableaux 1,2 et 3 de la Figure 17).

Niveau	**Santé**	**Argent**	**Mise à jour des tours possibles**		
			Cellules cicatrisantes	**Macrophages**	**Lymphocytes**
N° du niveau (ex.: 7)	Valeur de la santé (ex.: 5000)	Quantité d'argent (ex.: 5000)	De 0 à 3 : – 0 : rien – 1 : cellules de base – 2 : cellules muqueuses – 3 : cellules souches	De 0 à 3 : – 0 : rien – 1 : niveau 1 – 2 : niveau 2 – 3 : niveau 3	De 0 à 3 : – 0 : rien – 1 : niveau 1 – 2 : niveau 2 – 3 : niveau 3

Tableau 1 : Spécifications générales d'un niveau

Type de tour	**Position x**	**Position y**
De 0 à 3 : – 1 : cellule cicatrisante – 2 : macrophage – 3 : lymphocyte	Position horizontale en multiples de 20 pixels	Position verticale en multiples de 20 pixels

Tableau 2 : Spécification de la déclaration des tours déjà placées

Vague	**Type**	**Nombre**	**Vitesse**	**Couleur**	**Latence**
N° de la vague de microbes	Type de microbe dans la vague : – 1 : bactérie – 2 : virus	Nombre d'ennemis dans la vague	Vitesse des microbes de 1 à 5	6 couleurs spécifiques possibles : – 2 pour les bactéries – 5 pour les virus	Temps en secondes entre cette vague et la suivante

Tableau 3 : Spécification des vagues de microbes

Figure 17 : Spécifications pour initialiser un niveau de *Défenses Immunitaires*

La conception d'un scénario étant partiellement suffisante il a fallu produire un ensemble de modèles (modèle des tours, modèle des niveaux) qui regroupés avec le scénario constituent une spécification du jeu. Quelques autres détails de cette spécification sont donnés dans le scénario de réalisation détaillé en annexe p. 63 et présenté ci-après.

III.4- Scénario de réalisation de Défenses Immunitaires

Comment implémenter le jeu *Défenses Immunitaires* de façon à tester le mieux possible les outils auteurs sélectionnés ? Les spécifications ont déjà été conçues de façon à utiliser des mécanismes de jeu plutôt simples et répandus, mais il était nécessaire de construire un **scénario de réalisation** pour tester progressivement les outils auteurs.

J'ai donc trié parmi les fonctionnalités du jeu celles qui paraissaient les plus simples à programmer afin de constituer une chronologie de réalisation. Deux types de critères ont permis ce tri : l'utilisation de capacités connues et reconnues des outils auteurs (glanées dans les documentations et sur leurs forums) et la simplicité de conception algorithmique de la fonctionnalité du jeu concernée. Il faut aussi que l'ordre des fonctionnalités construites reste logique pour obtenir à chaque étape quelque chose d'exécutable. Ce tri n'a pas entièrement été fait dès le départ. Il a été considérablement affiné au cours des tests de réalisation avec les outils auteurs. Notamment lorsque la spécification du jeu a été précisée, mais aussi lorsque des obstacles de conception algorithmique sont apparus.

Voici les étapes chronologiques raffinées[1] du scénario de réalisation de *Défenses Immunitaires* :

– Étape 1 : Construire l'aire de jeu (détails en annexe p. 64)
– Étape 2 : Faire parcourir l'aire de jeu par une bactérie (détails en annexe p. 64)
– Étape 3 : Tester des obstacles sur les trajets (détails en annexe p. 64)
– Étape 4 : Tester l'introduction des macrophages (détails en annexe p. 64)
– Étape 5 : Construire une vague de bactéries (détails en annexe p. 64)
– Étape 6 : Faire parcourir l'aire de jeu par une vague de virus (détails en annexe p. 64)
– Étape 7 : Tester l'introduction des lymphocytes et de leurs anticorps (détails en annexe p. 65)
– Étape 8 : Des anticorps qui agglutinent les microbes reconnus (détails en annexe p. 65)
– Étape 9 : Rajouter un début de gestion de la santé (détails en annexe p. 65)
– Étape 10 : Interface simple de rajout des tours (détails en annexe p. 65)
– Étape 11 : Afficher des informations sur les objets (détails en annexe p. 66)
– Étape 12 : Fabrication d'un niveau de jeu (détails en annexe p. 66)
– Étape 13 : Externaliser les paramètres du niveau dans un fichier de configuration (détails en annexe p. 66)

Le temps du stage n'a pas permis l'implémentation complète des spécifications du jeu dans les outils auteurs. D'autres étapes de conception étaient prévues pour cela :

1 Ces étapes sont aussi détaillées en annexe p. 63

- Ajout d'un système de spécificité des microbes (couleurs spécifiques)
- Mise en œuvre de l'agglutination sélective par les anticorps en fonction de la couleur des microbes
- Afficher les informations sur les tours installables
- Raffinement des éléments du jeu pour se rapprocher des spécifications (tours destructibles, mise à jour des tours, etc.)
- Extraire des résultats une modélisation de l'apprenant
- Évaluer ces objectifs en fonction de la réussite et des objectifs du niveau
- Permettre aux utilisateurs de construire leurs propres niveaux

D'autres éléments du scénario décrit en annexe p. 50 n'ont pas été détaillés, ni dans la spécification ni dans le scénario de réalisation par manque de temps.

IV- Résultats de l'évaluation

Durant le temps de ce stage, je n'ai pu commencer à réaliser le jeu sérieux *Défenses Immunitaires* qu'avec deux outils auteurs. J'ai choisi ceux qui avaient les meilleurs exemples de jeu du genre *tower defense* à étudier. Trois outils auteurs sortent du lot par la qualité des exemples de *tower defense* disponibles : *Multimedia Fusion Creator 2*, *Construct* et *Game-Editor*. Bien que je n'aie travaillé à la réalisation du jeu sérieux qu'avec les deux derniers, mon évaluation porte bien sur ces trois outils. Pour l'évaluation de *Multimedia Fusion Creator 2*, je me suis appuyé à la fois sur une inspection approfondie du logiciel et sur les travaux de Christophe Desimeur. Étudiant en DU « TICE » à l'Université Pierre et Marie Curie, Christophe a choisi comme projet de fin de formation de réaliser une partie du jeu sérieux *Défenses Immunitaires* avec *Multimedia Fusion Creator 2*. C'est donc grâce au suivi de ses travaux et à plusieurs entretiens au cours et à la fin de ceux-ci que j'ai pu compléter mes observations sur cet outil auteur avec celles de Christophe.

Pour chacun des logiciels, la présentation de l'évaluation détaillée en annexe p. 67 se fait en suivant les critères retenus lors de la mise en place de la méthodologie[1]. Je vous présente ci-après chaque logiciel accompagné du bilan de son évaluation, avant de faire la synthèse de l'étude de ces trois outils auteurs.

IV.1- Évaluation de Construct

IV.1.1- Présentation du logiciel et de l'implémentation du jeu

Construct[2] est le premier outil auteur de jeux vidéo testé. C'est un logiciel libre (sous licence *GPL*) réalisé par trois étudiants. Il fonctionne exclusivement sous *MS-Windows* et nécessite impérativement la présence de *DirectX 9* ou plus, ainsi qu'une carte graphique assez performante (au moins 64Mo de mémoire vidéo, mais 256Mo minimum sont conseillés). La stabilité de *Construct* et des jeux produits avec dépend beaucoup de la puissance de la carte graphique et de

1 Voir « Critères d'évaluation retenus » p. 15
2 http://www.scirra.com/

l'implémentation des fonctions de *DirectX 9* dans son pilote. Donc, testé sur un ordinateur portable doté d'une carte graphique assez minimale, *Construct* n'a pas brillé par sa stabilité.

Ce logiciel a comme particularité d'être un environnement de développement complet et qui se veut entièrement en manipulation directe (c'est-à-dire sans utilisation d'outils de type *shell*, ou d'écriture de lignes de code). Il contient notamment un éditeur d'images, une bibliothèque de *briques* de gameplay, une interface de manipulation de ces objets et de leurs instances dans des calques et une interface de programmation événementielle.

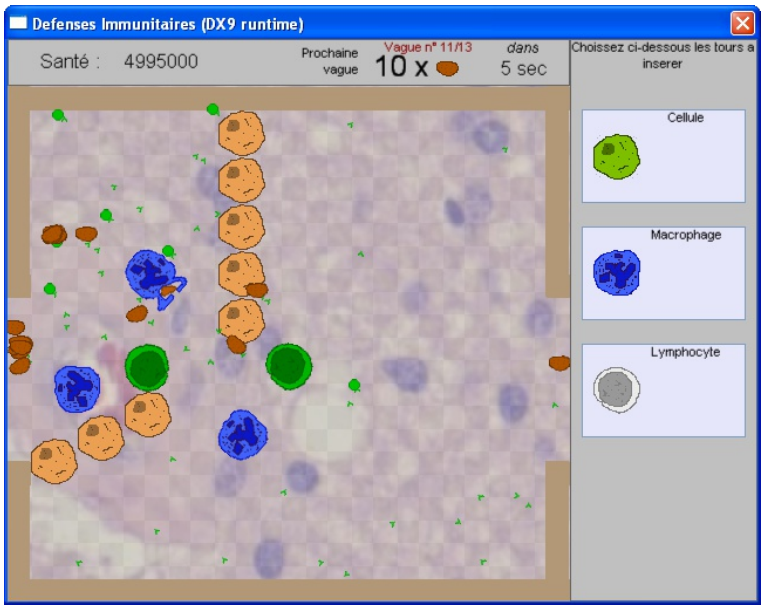


Figure 18 : Exemple de réalisation du niveau 7 du jeu sérieux *Défenses Immunitaires* avec *Construct*. Les **virus** (petits ronds verts) et les **bactéries** (ovales bruns) interagissent avec des **lymphocytes B** (cellules vertes), leurs **anticorps** (petits « Y » verts) et des **cellules phagocytaires** (cellules bleues).

Comme c'est le premier outil sur lequel j'ai travaillé à la réalisation de *Défenses Immunitaires*, c'est aussi l'outil sur lequel elle a été la plus aboutie. Mais ce n'est pas la seule raison, c'est aussi parce que l'apprentissage de l'utilisation de cet outil auteur est facile et qu'il s'est montré assez pratique à l'usage. Avec *Construct*, l'étape 13 du scénario de réalisation (Étape 13 : Externaliser les paramètres du niveau dans un fichier de configuration, p. 66) a été atteinte et presque entièrement terminée (voir Figure 18).

Les détails de l'évaluation de l'Utilisabilité de l'interface graphique et accessibilité du logiciel, des Documentation et support, des Outils de programmation du logiciel et de la Maintenabilité du logiciel sont en annexe à partir de la p. 67.

IV.1.2- Bilan pour *Construct*

Construct est un outil auteur qui manque un peu de maturité, notamment pour assurer sa stabilité. Mais il permet néanmoins une programmation réellement graphique grâce à plusieurs points forts : son interface plutôt bien pensée et organisée, une programmation événementielle bien guidée et de très nombreuses briques de gameplay qui épargnent aux auteurs de jeux des efforts de conception et de réalisation. Ces trois aspects garantissent une courbe d'apprentissage très progressive mettant la réalisation de jeux à la portée de ceux qui ne veulent pas écrire de lignes de code. Il reste néanmoins

un prérequis nécessaire : savoir construire la logique du fonctionnement du jeu et l'avoir fait en amont.

Construct peut-il être un outil auteur de jeu sérieux ? Avec le temps, il aurait sans doute été possible de terminer le jeu *Défenses Immunitaires* entièrement avec *Construct*. Cependant la finition du jeu ne serait pas suffisante et des bogues subsisteraient car certains éléments du jeu (*pathfinding*, lecture du fichier de configuration, agglutination des microbes, système de spécificité, etc.) sont très difficiles à programmer avec *Construct*, car les briques adaptées ne sont pas là. Mais cette réalisation reste possible. Toutefois, *Construct* ne possède aucune brique spécifique aux jeux sérieux. Ainsi les diverses modélisations classiques d'un tel EIAH (modélisation du domaine de connaissance, la modélisation pédagogique et celle de l'apprenant) ne sont pas prévues et vont dans tous les cas nécessiter de gros efforts de programmation dans ce logiciel.

IV.2- Évaluation de Multimedia Fusion Creator 2

IV.2.1- Présentation du logiciel et de l'implémentation du jeu

Multimedia Fusion Creator 2 (*MMF 2*)[1] est un outil auteur de logiciels multimédias édité par la société Clickteam et décliné en trois versions : *The Game Factory 2* à 49 €, *Multimedia Fusion 2* à 99 € et *Multimedia Fusion Developer 2* à 299 €. La version la moins chère n'a pas de *runtime* indépendant, la version intermédiaire oblige à avoir un tatouage sous forme d'écran surgissant indiquant que le logiciel est conçu avec *MMF 2*. La version la plus chère ne possède pas ce tatouage, elle est la plus complète et la plus extensible. L'évaluation présentée ici a été effectuée sur une version de démonstration également disponible sur le site web de l'éditeur. Il s'agit d'une démonstration de la version la plus chère, mais ne pouvant pas produire de logiciel indépendant (les logiciels conçus doivent s'exécuter dans l'éditeur). *MMF 2* fonctionne sous *MS-Windows*. Les logiciels réalisés avec peuvent fonctionner sous *MS-Windows*, mais aussi en *Java*, en *Flash* (extension commerciale). *MMF 2* existe en français et en anglais (bien que certains éléments ne soient pas traduits en français).

Les auteurs de *Construct*[2] revendiquent s'être abondamment inspirés de *MMF 2*, il y a donc des principes similaires dans les deux logiciels. Ainsi, *MMF 2* permet aussi une programmation sans écrire de lignes de code dans une interface graphique en manipulation directe.

1 http://www.clickteam.fr/
2 *Construct* est présenté p. 27

Christophe Desimeur a travaillé dans le cadre de son projet de DU à l'Université Pierre et Marie Curie à la réalisation de *Défenses Immunitaires* avec *MMF 2*. Il s'est appuyé sur les spécifications partielles que je lui ai transmises. Il n'a pu suivre exactement le scénario de réalisation car il a été élaboré au fur et à mesure. Le jeu qu'il a conçu est à l'étape 5 (Étape 5 : Construire une vague de bactéries, p. 64) du scénario de réalisation, avec en plus la gestion de la santé et de l'interface de jeu.

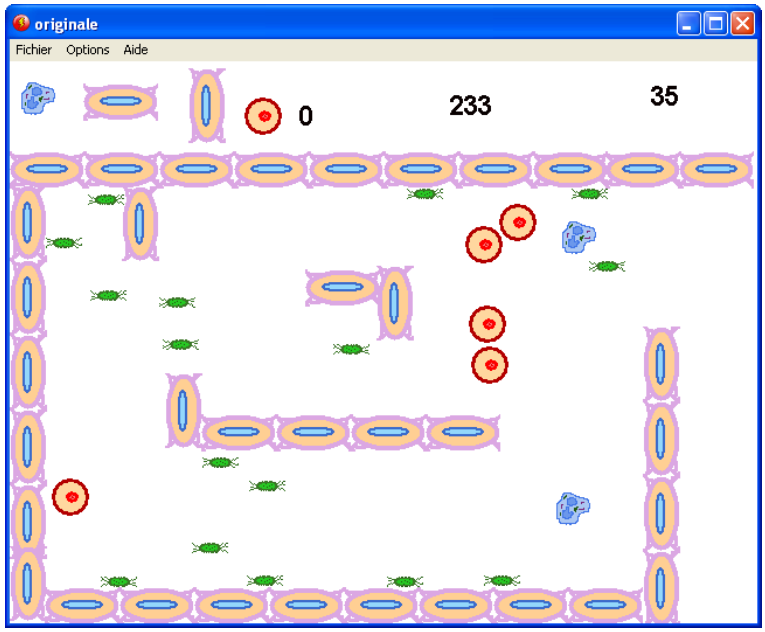


Figure 19 : Capture d'écran de la réalisation de *Défenses Immunitaires* par Christophe Desimeur avec *MMF 2*. Les **bactéries** (ovales verts) se déplacent et sont retenues par les **cellules** (les grands ovales sont les *cellules épithéliales*, et les ronds à dominante rouge les *cellules souches*) et absorbées par les **macrophages** (cellules bleues).

Les détails de l'évaluation de l'Utilisabilité de l'interface graphique et accessibilité du logiciel, des Documentation et support, des Outils de programmation du logiciel et de la Maintenabilité du logiciel sont en annexe à partir de la p. 77.

IV.2.2- Bilan pour *Multimedia Fusion Creator 2*

MMF 2 est un outil auteur qui permet de concevoir des jeux sans programmation. Ses points forts sont le système de programmation événementielle et les très nombreuses briques de gameplay, en particulier en intégrant celles de la communauté des utilisateurs. La documentation interne et externe au logiciel est abondante et pour tous les niveaux. Cet ensemble permet aux néophytes comme aux utilisateurs expérimentés de se lancer dans la réalisation de jeux avec cet outil, sans écrire de ligne de code, et en leur garantissant une courbe d'apprentissage très progressive. Bien sûr, il reste nécessaire d'avoir en amont conçu la logique du jeu, car *MMF 2*, pas plus que *Construct*, ne proposent d'outil permettant la scénarisation.

MMF 2 peut-il être un outil auteur de jeu sérieux ? Sa stabilité et l'abondance de contributions externes lui donnent une certaine crédibilité. Mais, la conception d'un jeu tel que *Défenses Immunitaires* dans *MMF 2* reste néanmoins assez longue et difficile. Christophe Desimeur évalue la durée de ce travail à 3-4 mois/homme de développement. En outre, *MMF 2* ne possède aucune

brique spécifique au développement des jeux sérieux. Rien n'est prévu pour y modéliser le domaine de connaissances, l'apprenant ou la pédagogie.

IV.3- Évaluation de Game-Editor

IV.3.1- Présentation du logiciel et de l'implémentation du jeu

Game-Editor[1] est un logiciel libre (*GPL* v3[2]) pour créer des jeux vidéo. Le logiciel a été conçu par une très petite équipe : selon le gestionnaire de version des sources du logiciel, seuls deux contributeurs ont vraiment participé à son développement depuis 24 mois[3]. Une des particularités de *Game-Editor* est d'être multiplateforme. L'outil auteur fonctionne sous *MS-Windows*, *GNU/Linux* et *Mac OS X*, et permet de concevoir des jeux pour ces trois plateformes et pour *iPhone*, *iPad*, *Pocket PC*, *Handheld PC*, *GP2X* et les appareils fonctionnant avec *Windows Mobile*.

Game-Editor n'est pas tout à fait comme *Construct*[4] et *Multimedia Fusion Creator 2*[5] dans la mesure où la réalisation des jeux avec cet outil auteur nécessite presque toujours l'édition de lignes de code dans un langage de script dont la syntaxe est proche du *C*. Est-ce vraiment un outil auteur « sans programmation » ? Dans une certaine mesure, il est possible de créer des jeux qu'en éditant des expressions très simples de type arithmétique, semblables à celles qui existent dans les deux autres outils auteurs[6]. Mais, l'évaluation en profondeur consistant à réaliser *Défenses Immunitaires* montre qu'il est impossible de réaliser un jeu de ce genre-là sans passer par l'écriture de scripts complexes.

Ainsi, la réalisation avec *Game-Editor* n'a pas été très avancée et a seulement atteint l'étape 5 (« Étape 5 : Construire une vague de bactéries » p. 64) du scénario de réalisation.

1 http://game-editor.com/Main_Page
2 Selon les auteurs de *Game-Editor*, l'une des conséquences de l'utilisation de la version 3 de la *GPL* est que les logiciels fabriqués avec *Game-Editor* doivent rester libres. Pour les utilisateurs qui souhaitent commercialiser leur travail, il y a une licence payante de *Game-Editor* : http://game-editor.com/Buy
3 http://sourceforge.net/apps/trac/game-editor/log/?action=stop_on_copy&mode=stop_on_copy&rev=202&stop_rev=&limit=900
4 « Évaluation de Construct » p. 27
5 « Évaluation de Multimedia Fusion Creator 2 » p. 29
6 Expressions arithmétiques saisies dans *Construct* : Figure 24 en annexe p. 69, et *Game-Editor* : Figure 34 en annexe p. 79.

Les détails de l'évaluation de l'Utilisabilité de l'interface graphique et accessibilité du logiciel, des Documentation et support, des Outils de programmation du logiciel et de la Maintenabilité du logiciel sont en annexe à partir de la p. 83.

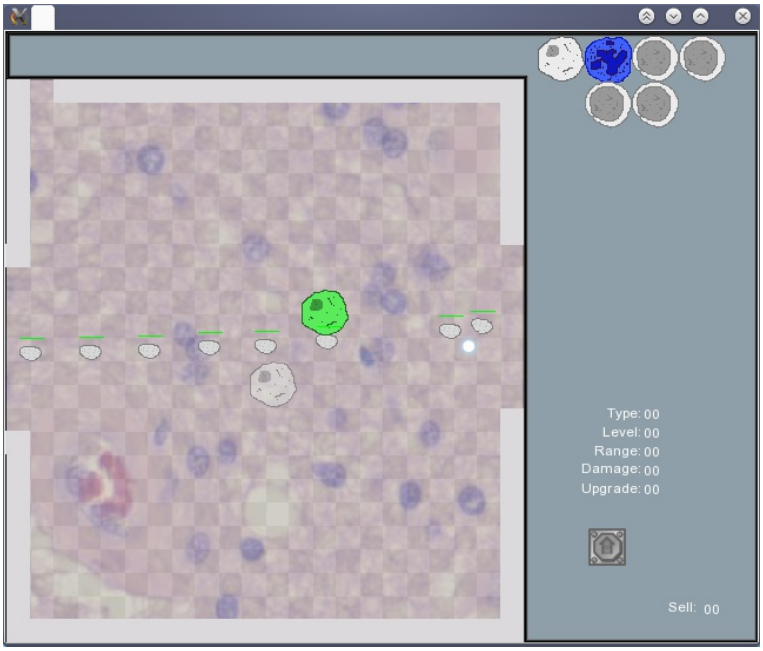


Figure 20 : Exemple de réalisation de *Défenses Immunitaires* avec *Game-Editor*. Les vagues de **bactéries** (petits ovales) parcourent l'aire de jeu en évitant les **cellules**.

IV.3.2- Bilan pour *Game-Editor*

Game-Editor n'aurait pas dû faire partie de cette étude sur les outils auteurs « sans programmation »[1]. Si lors de la pré-séléction des outils auteurs la prédominance du rôle de l'*éditeur de script* ne m'est pas apparue, je me suis rapidement rendu compte, lors de l'évaluation en profondeur[2], que son utilisation abondante serait indispensable pour réaliser *Défenses Immunitaires*. Pourquoi avoir quand même fait cette évaluation ? D'une part parce qu'elle a été la dernière, ne me laissant pas le temps d'adopter un autre logiciel. D'autre part parce que certains exemples de jeux réalisés avec *Game-Editor* montrent que des utilisateurs de niveau professionnel font confiance à cet outil auteur.

Mon objectif était devenu de savoir si les qualités de *Game-Editor* permettaient aux néophytes en programmation textuelle de l'utiliser avec un temps de prise en main raisonnable. Le bilan de mon évaluation est que la courbe d'apprentissage commence trop abruptement pour ce type de public. Notamment à cause des défauts de son interface et du manque de *briques de gameplay* qui oblige à la programmation textuelle dans l'*éditeur de script*. Les utilisateurs de *Game-Editor* seront plutôt des programmeurs chevronnés, à l'aise avec la syntaxe du *C*, mais qui souhaitent alléger une partie des tâches de réalisation grâce à cet outil. Toutefois, il leur faudra une bonne conception logique en amont, notamment pour isoler les éléments qui seront pris en compte par les rares briques de gameplay de *Game-Editor* et les éléments qui devront être réalisés avec les scripts.

1 Voir le sens donné dans mon travail à l'expression « sans programmation » dans « Objectifs du stage : exploration et évaluation d'outils auteurs » p. 5.

2 Modalités détaillées de l'évaluation en profondeur : « Une évaluation en profondeur : la réalisation d'un jeu sérieux avec chacun des outils sélectionnés » p. 19.

Game-Editor peut-il être un outil auteur de jeux sérieux ? Ce n'est certainement pas un outil pour ceux qui ne veulent pas faire de programmation textuelle. Cependant, l'intégration de l'éditeur de script permet d'aller plus loin que les autres outils auteurs dans la complexité et dans la spécificité du programme conçu. En prenant le temps de maîtriser l'interface du logiciel et le langage de script de *Game-Editor*, il serait probable de réaliser entièrement *Défenses Immunitaires*. Pourtant, à l'instar des autres outils auteurs évalués ici, *Game-Editor* ne propose aucune fonctionnalité dédiée au développement des EIAH et permettant la modélisation des connaissances, de la pédagogie ou de l'apprenant.

IV.4- Bilan global des évaluations en profondeur

Construct, *Multimedia Fusion Creator 2* et *Game-Editor* ont comme d'autres outils auteurs rencontrés[1] deux points communs principaux. D'une part, ces logiciels proposant une programmation graphique s'appuient sur le paradigme de la **programmation événementielle** plutôt que séquentielle. Ce paradigme rend accessible l'utilisation d'**événements** spécifiques des jeux vidéos (ex. : collisions, visée, étapes d'un mouvement, etc.) auxquels sont adossées des **actions.** D'autre part, ces outils auteurs ont justement des bibliothèques d'**événements**, d'**actions** et d'**objets** spécifiques des jeux vidéos : les **briques de gameplay**.

Il ressort de l'évaluation de ces trois logiciels et de la pré-sélection d'autres que la qualité des outils auteurs de jeux dépend d'un côté de la quantité et de la pertinence des briques de gameplay, de l'autre de la qualité de l'interface de programmation événementielle. Des trois outils évalués en profondeur, c'est *Construct* qui propose l'interface de programmation événementielle la mieux réussie. Même si l'interface de *Multimedia Fusion Creator 2* se révèle très efficace, il lui manque les **comportements** qui simplifient fortement la programmation dans *Construct*[2]. Du côté des briques de gameplay il est difficile de départager les deux logiciels qui ont chacun leurs manques et leurs qualités spécifiques. Toutefois, *Multimedia Fusion Creator 2* qui fédère une communauté bien plus grande que *Construct*, bénéficie d'un nombre très élevé de contributions qui atténuent probablement les manques du logiciel. *Game-Editor*, nous l'avons vu, ne possède ni beaucoup de briques de gameplay, ni une interface de programmation événementielle adaptée à une programmation graphique. Cependant, cela reste un outil crédible pour les développeurs chevronnés en programmation textuelle, pour lesquels il simplifiera un certain nombre de tâches comme la gestion de l'affichage, la mise en place de la gestion des événements, la gestion des héritages, etc.

1 ex. : *SeGAE* p. 11, et les outils-auteurs pré-sélectionnés décrits en annexe p. 46.
2 Voir le principe des *comportements* de *Construct* en annexe p. 72.

Les trois outils auteurs testés sont aussi caractérisés par des manques. D'abord, il leur manque des fonctionnalités de *conceptualisation*. Il n'y a rien dans ces logiciels pour concevoir un scénario, contrairement à *MOCAS*[1]. Nous avons vu aussi qu'ils s'adressaient à des auteurs ayant déjà soigneusement conçu la logique de leurs jeux avant d'en entamer la réalisation. Les erreurs conceptuelles peuvent survenir rapidement dans une programmation événementielle qui se ferait sans une conception en amont. Par exemple en associant des actions contradictoires aux mêmes événements. Cette nécessité d'avoir une conception bien faite avant d'entamer la réalisation avec ces outils auteurs est d'autant plus grande qu'ils pêchent aussi par deux autres aspects. D'une part, les programmes conçus dans leur interface de programmation événementielle sont *difficiles à relire* pour y déloger les erreurs. D'autre part, ils proposent des outils de *débogage peu convaincants*, n'exploitant pas leur interface de programmation événementielle.

Pour finir sur les manques de ces trois outils, aucun ne possède de fonctions liées au travail collaboratif, ni même à la gestion des versions du travail d'un développeur unique. C'est un gros handicap, car nous avons vu que la production des jeux sérieux était un travail d'équipe[2]. Si ces logiciels étaient plus interopérables en permettant notamment de travailler facilement sur des fragments de jeu, cette difficulté serait estompée. Mais nous avons aussi vu que les formats de fichiers n'y sont pas interopérables et que l'échange et la manipulation de données dans des formats standards comme le XML n'y étaient pas simples à mettre en œuvre.

Alors, ces outils auteurs ne sont pas faits pour réaliser des jeux sérieux ? L'examen de ces trois-là montre que la réalisation d'un jeu sérieux *complet* avec eux, si elle est possible, ne permettra pas d'atteindre des objectifs de qualité, de coût de développement et de travail collaboratif tels qu'ils sont envisagés dans un développement industriel[2].

Je peux néanmoins imaginer de les insérer dans une ingénierie des jeux sérieux à plusieurs niveaux. D'abord en amont de la réalisation. Lors de la phase de conceptualisation pour un prototypage rapide qui serait réalisé par les équipes d'experts pédagogiques ou de game-designers. Toujours en phase de scénarisation, ces outils auteurs pourraient servir à présenter les spécifications du jeu sérieux aux équipes de développement : les spécifications rédigées ne sont pas toujours suffisamment explicites pour les programmeurs, les accompagner d'une démonstration jouable peut permettre d'éviter certaines ambiguïtés et de diminuer les coûts de développement. Enfin, ces outils auteurs peuvent aussi être utilisés pour réaliser des moteurs de jeux, capables de lire des fichiers de missions ou de niveau rédigés dans un langage dédié. L'état de l'art montre que l'effort de

1 Voir la méthode de modélisation des connaissances de *MOCAS* avec *IBIS* p. 9.
2 Voir la méthode d'industrialisation du développement des jeux sérieux par les 5M p. 7.

recherche se porte essentiellement sur des outils auteurs de jeux sérieux qui formalisent des scénarios et des missions grâce à des langages dédiés exécutés par des moteurs de jeu[1]. Les outils auteurs étudiés ici peuvent, à condition de progresser dans la manipulation du XML, servir à construire ces moteurs de jeu capables d'exécuter ces langages dédiés. Cela ouvre la porte de la réalisation de nouveaux genres de jeux en réutilisant les outils auteurs de scénarisation de mission qui existent déjà.

Nom	***Construct***	***Multimedia Fusion Creator 2***	***Game-Editor***
Utilisabilité de l'interface graphique	Très bonne utilisabilité (Guidage, charge de travail, adaptabilité et gestion des erreurs)	Bonne utilisabilité (Charge de travail, adaptabilité et gestion des erreurs)	L'interface devient réellement utilisable pour les programmeurs chevronnés
Documentation	– En anglais. – Assez complète et claire. – Pas dans le logiciel	– En anglais et en français. – Très complète et claire. – Éparse.	– En anglais – Très complète – Très riche dans le logiciel
Fonctionnalités centrales	– Programmation entièrement graphique possible. – Système de comportement associé aux objets. – Nombreuses briques de gameplay.	– Programmation entièrement graphique possible. – Nombreuses briques de gameplay. – Visualisation globale du programme sous forme de tableau. – Production de jeux en *Java* et en *Flash*	– Système de script qui permet d'aller loin dans l'élaboration. – Production de jeux multiplateformes
Fonctionnalités manquantes	– Stabilité du logiciel. – Outils de débogage adaptés. – Briques capables de traiter le XML. – Production de jeux pour d'autres plateformes que *MS-Windows/Direct X*. – Outils de conception/scénarisation	– Association de comportements aux objets. – Outils de débogage adaptés. – Brique de *pathfinding*. – Outils de conception/scénarisation	– Interface graphique de conception de scripts – Outils de débogage adaptés. – Outils de conception/scénarisation
Maintenabilité	– Peu mature – Scriptable (*Python*) et extensible (le SDK est disponible)	– Très stable – Extensible (le SDK est disponible)	– Très stable
Détails	En annexe p. 67	En annexe p. 77	En annexe p. 83

Figure 21 : Récapitulatif des caractéristiques principales des trois outils-auteurs de jeux évalués en profondeur

1 Voir « Des outils auteurs de jeux sérieux « sans programmation », mais fondés sur un langage dédié » p. 8.

V- Conclusion

L'objectif de mon stage était de faire une étude exploratoire des outils auteurs « sans programmation » capables d'aider à la réalisation des jeux sérieux. Cette exploration avait plusieurs objectifs, comme de découvrir leurs concepts sous-jacents de programmation graphique, de savoir quels genres de jeux sérieux peuvent être construits avec, et d'envisager la place ces outils-auteurs dans une ingénierie des jeux sérieux qui tend progressivement vers l'industrialisation.

L'état de l'art m'a permis de dresser un panorama de cette ingénierie et de repérer les occasions d'y utiliser des outils-auteurs. À l'image d'*IBIS*, ces outils sont déjà utilisés pour formaliser les connaissances à enseigner et, comme avec *<e-Adventure>* et *SeGAE* utilisés pour modéliser des scénarios de missions ou de niveaux. Par contre, la possibilité d'utiliser des outils auteurs pour *réaliser* les jeux et des moteurs de jeux semble inexplorée.

La prospection des outils auteurs de jeu, dans le but savoir s'ils pouvaient s'insérer dans la phase de réalisation d'un jeu sérieux, m'a conduit à scénariser puis à dresser les spécifications du jeu *Défenses Immunitaires.* J'ai accompagné ces spécifications, spécialement conçues pour tester des outils auteurs, de critères d'observation précis afin de proposer une évaluation en profondeur.

Avec cette évaluation en profondeur, j'ai mis en évidence que les outils auteurs testés sont fondés sur une interface de *programmation événementielle* dans laquelle s'insèrent des *briques de gameplay* préprogrammées. La qualité cette interface graphique de programmation et la pertinence des briques de gameplay sont des éléments centraux dans la qualité d'un outil-auteur de jeu. Je remarque aussi que ces outils auteurs se caractérisent par l'absence d'outils de conceptualisation. Ils ne proposent ni de moyens pour élaborer un scénario de jeu, ni d'interface pour assister à la conception de logique algorithmique du jeu. Si ces outils ont pour but de faciliter le travail des « non-programmeurs », il me paraît nécessaire qu'ils incorporent ces fonctionnalités. Parmi les autres manques, l'absence de fonctionnalités dédiées au développement d'EIAH est moins surprenante puisqu'en l'absence de logiciels mixtes, j'ai fait le choix de tester des outils auteurs de jeux plutôt que des outils auteurs d'EIAH.

Avec ces manques et ces qualités, je pense que les outils-auteurs de jeux peuvent s'intégrer dans l'ingénierie des jeux sérieux à plusieurs niveaux : le prototypage, la formulation de spécifications, et la conception de moteurs de jeux capables d'interpréter des scénarios ou des missions. Ce dernier point permet d'esquisser une ingénierie dans laquelle les outils se succèdent et interagissent : les outils comme *IBIS* permettent de construire les modèles, les outils comme *SeGAE* de faire du *level-design* (conception de missions) dans un langage dédié, les outils auteurs de jeux de réaliser les moteurs qui exécutent ces langages.

Pour une intégration plus large dans le processus de conception et de réalisation des jeux sérieux, il me semble nécessaire d'étoffer ces outils. Il s'agirait de les enrichir avec de nouvelles *briques* permettant l'échange de données (en XML, par exemple *IMS-LD*, *SCORM*, etc.), le travail collaboratif, la production de modèles (de connaissances, de méthodes pédagogiques), l'analyse de traces du joueur, la conception de scénarios et, pour les ouvrir au public de développeurs le plus large possible, la modélisation algorithmique.

Bibliographie

Alvarez J., Djaouti D., Jessel J., Methel G., Molinier P. « Morphologie des jeux vidéo ». In : *Hypertextes, hypermédias. Collaborer, échanger, inventer: expériences de réseaux*. *Collaborer, échanger, inventer : expériences de réseaux, 29, 30 et 31 octobre 2007*. Hammamet, Tunisie : Lavoisier, 2007. p. 277–294. ISBN : 978-2-7462-1891-8.

Bastien J. M. C., Scapin D. L. *Critères Ergonomiques pour l'Évaluation d'Interfaces Utilisateurs (version 2.1)*. Le Chesnay : INRIA, 1993. (Programme 3. Intelligence artificielle, Systèmes cognitifs et Interaction homme-machine).

Blanchard E., Frasson C. « Faciliter la création d'environnements virtuels d'apprentissage s' inspirant des jeux vidéo ». In : *International Symposium on Technologies of Information and Communication in Education for Engineering and Industry (TICE2006)*. *International Symposium on Technologies of Information and Communication in Education for Engineering and Industry (TICE2006), Toulouse, France*. Toulouse, France : [s.n.], 2006.

Burgos D., Moreno-Ger P., Sierra J. L., Fernández-Manjón B., Specht M., Koper R. « Building Adaptive Game-Based Learning Resources: The Marriage of IMS Learning Design and <e-Adventure> ». *Simulation & Gaming*. 2008, Vol. 39, p. 414–431.

Capdevila Ibáñez B., Boudier V., Labat J. « Knowledge Management Approach to Support a Serious Game Development ». In : *Proceedings of the 2009 Ninth IEEE International Conference on Advanced Learning Technologies* [En ligne]. *The 9th IEEE International Conference on Advanced Learning Technologies*. Riga, Latvia : IEEE Computer Society, 2009. p. 420-422. Disponible sur : < http://portal.acm.org/citation.cfm?id=1582708.1582954 > (consulté le 27 June 2010) ISBN : 978-0-7695-3711-5.

Clements P., Pesner J., Shepherd J. « The Teaching of Immunology Using Educational: Gaming Paradigms ». In : *Proceedings of the 47th Annual Southeast Regional Conference* [En ligne]. *the 47th Annual Southeast Regional Conference*. Clemson, South Carolina : ACM, 2009. p. 1-4. Disponible sur : < http://dx.doi.org/10.1145/1566445.1566474 > (consulté le 2 April 2010) ISBN : 978-1-60558-421-8.

Fabricatore C. « Learning and Videogames: an Unexploited Synergy ». In : *2000 AECT National Convention - a recap. 2000 AECT National Convention*. Long Beach, CA : Secaucus, NJ : Springer Science + Business Media, 2000.

Huynh-Kim-Bang B., Labat J. *Design Patterns in Serious Games: how to mix fun and pedagogy?* [En ligne]. Disponible sur : < http://seriousgames.lip6.fr/DesignPatterns/ >

Kearney P. R., Pivec M. « Recursive Loops of Game-Based Learning: a Conceptual model. ». In : *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2007* [En ligne]. *World Conference on Educational Multimedia, Hypermedia and Telecommunications (EDMEDIA) 2007*. Vancouver, Canada : AACE, 2007. p. 2546-2553. Disponible sur : < http://www.editlib.org/p/25731 >

Kiili K. « Foundation for problem-based gaming ». *British Journal of Educational Technology* [En ligne]. 2007, Vol. 38, n°3, p. 394-404. Disponible sur : < http://dx.doi.org/10.1111/j.1467-8535.2007.00704.x > (consulté le 24 February 2010)

Marfisi-Schottman I., Sghaier A., George S., Prévôt P., Tarpin-Bernard F. « Vers une industrialisation de la conception et de la production des Serious Game ». In : *Actes de l'Atelier «Jeux Sérieux: conception et usages». 4ème conférence francophone sur les Environnements Informatiques pour l'Apprentissage Humain*. Le Mans, France : Atief, lium, 2009. p. 75-84.

Moreno-Ger P., Burgos D., Sierra J. L., Manjón B. « A Game-Based Adaptive Unit of Learning with IMS Learning Design and ». In : *Creating New Learning Experiences on a Global Scale* [En ligne]. *Second European Conference on Technology Enhanced Learning, EC-Tel 2007*. Crete, Greece : Springer, 2007. p. 247-261. Disponible sur : < http://dx.doi.org/10.1007/978-3-540-75195-3_18 > (consulté le 22 February 2010) ISBN : 9783540751946.

Moreno-Ger P., Martinez-Ortiz I., Fernández-Manjón B. « The <e-Game> Project: Facilitating the Development of Educational Adventure Games ». In : *Proceedings of the IADIS International Conference on Cognition and exploratory learning in digital age (CELDA 2005). Cognition and exploratory learning in the digital age (CELDA 2005). Porto, Portugal: IADIS*. Porto, Portugal : [s.n.], 2005. ISBN : 9728924054.

Moreno-Ger P., Martınez-Ortiz I., Sierra J. L., Fernández-Manjón B. « Language-Driven Development of Videogames: The <e-Game> Experience ». In : *Entertainment Computing - ICEC 2006: Proceedings (Lecture Notes in Computer Science ... Applications, incl. Internet/Web, and HCI). 5th International Conference in Entertainment Computing (ICEC 2006)*. Cambridge, UK : Springer, 2006. ISBN : 9783540452591.

Murray T. « Authoring Intelligent Tutoring Systems: An Analysis of the State of the Art ». *International Journal of Artificial Intelligence in Education*. 1999, Vol. 10, n°1, p. 98–129.

Murray T. « An Overview of Intelligent Tutoring System Authoring Tools: Updated Analysis of the State of the Art ». In : *Authoring Tools for Advanced Technology Learning Environments*. Pays-Bas : Dordrecht ; Boston ; London : Kluwer Academic Publishers, cop. 2003, 2003. p. 491–544. ISBN : 978-1402017728.

Natkin S. *Jeux vidéo et médias du XXIe siècle*. Paris : Vuibert, 2004. 112 p. ISBN : 2711748448, 9782711748440.

Oblinger D. G. « The Next Generation of Educational Engagement ». *Journal of Interactive Media in Education*. 21 May 2004, Vol. 8, n°8, p. 2.

Paquette G., Léonard M., Lundgren-Cayrol K., Mihaila S., Gareau D. « Learning Design based on Graphical Knowledge-Modelling ». *Journal of Educational Technology and Society*. 2006, Vol. 9, n°1, p. 97-112.

Prensky M. *Digital Game-Based Learning*. [s.l.] : McGraw-Hill, 2004. ISBN : 0071454004.

Resnick M., Maloney J., Monroy-Hernández A., Rusk N., Eastmond E., Brennan K., Millner A., Rosenbaum E., Silver J., Silverman B., Kafai Y. « Scratch: Programming for All ». *Communications of the ACM* [En ligne]. November 2009, Vol. 52, n°11, p. 60–67. Disponible sur : < http://dx.doi.org/10.1145/1592761.1592779 >

Yessad A., Labat J., Kermorvant F. « SeGAE: a Serious Game Authoring Environment ». In : *To appear in Proceeding of The 10th IEEE International Conference on Advanced Learning Technologies. 10th IEEE International Conference on Advanced Learning Technologies*. Sousse, Tunisia : [s.n.], 2010.

Yusoff A., Crowder R., Gilbert L., Wills G. « A Conceptual Framework for Serious Games ». In : *Proceedings of the 2009 Ninth IEEE International Conference on Advanced Learning Technologies*. *The 9th IEEE International Conference on Advanced Learning Technologies*. Riga, Latvia : IEEE Computer Society, 2009. p. 21-23. ISBN : 978-0-7695-3711-5.

Annexes

Table des matières des annexes

I- Critères d'évaluation pour les jeux sérieux

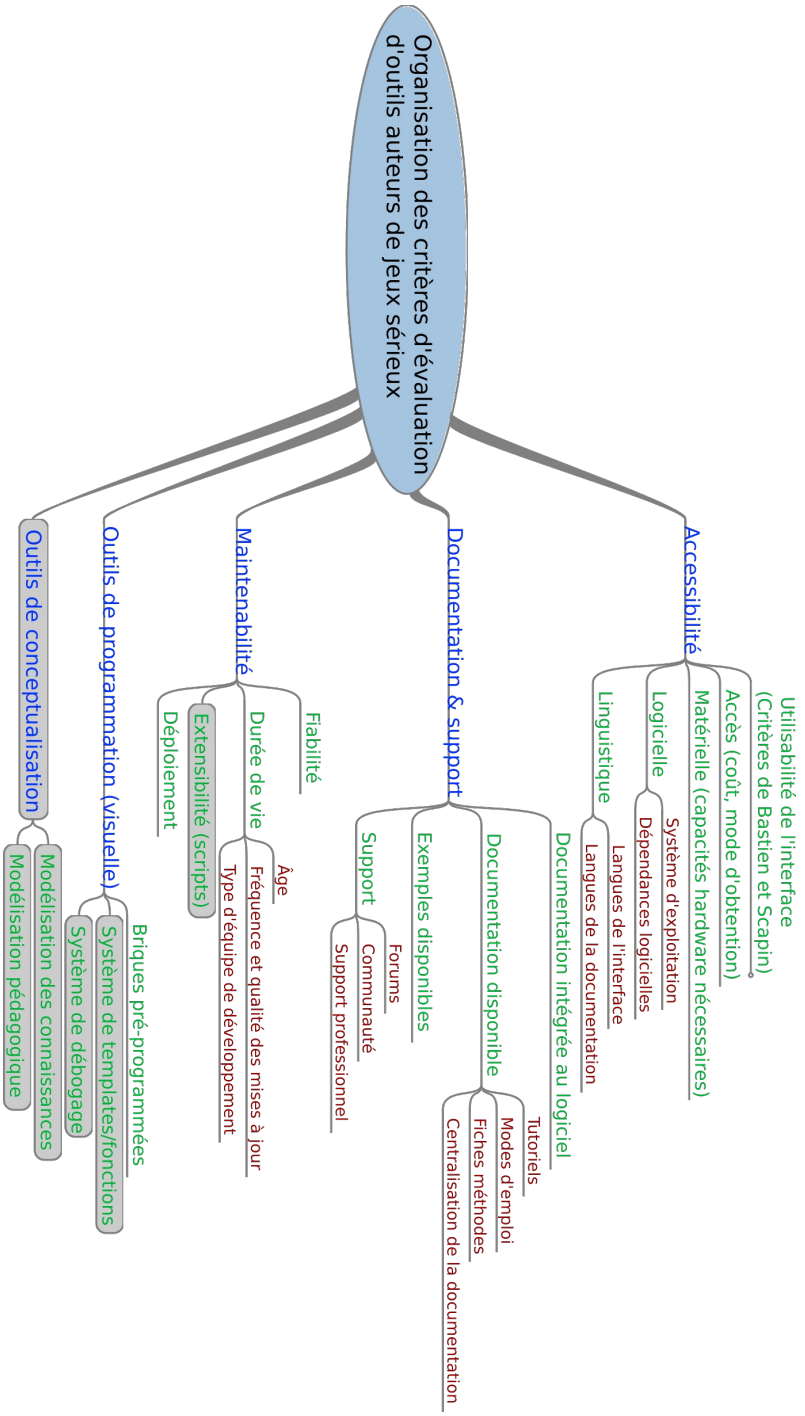


Figure 22 : Arbre des critères d'évaluation de jeux sérieux retenus. Les critères encadrés à fond gris sont issus des travaux de (Murray, 2003). Le détail des critères d'*Utilisabilité de l'interface* sont visibles sur la Figure 7 p. 15.

Plus de détails dans « Critères d'évaluation retenus » p. 15.

II- Outils auteurs pré-sélectionnés

Le tableau ci-après, découpé en deux parties, résume les principales caractéristiques des outils auteurs pré-sélectionnés pour leur évaluation (voir aussi « Une évaluation en largeur : une sélection lors d'un parcours de l'ensemble des outils trouvés » p. 18).

Le premier tableau p. 46 présente les outils-auteurs généralistes. Le second p. 47 présente les outils auteurs dédiés à un genre de jeu.

Nom du logiciel	***Construct***	***Multimedia Fusion Creator 2***	***Game-Editor***	***Game Maker 8***	***Sim's Carnival Game Creator***	***Sharendipity***
URL du site	http://www.scirra.com/	http://www.clickteam.com/fr/	http://game-editor.com/	http://www.yoyogames.com/gamemaker/	http://www.simscarnival.com/	http://www.sharendipity.com/
Système d'exploitation	MS-Windows avec DirectX 9	– Logiciel : MS-Windows – Jeux : MS-Windows, Java, Flash	– Logiciel : MS-Windows, GNU/Linux et Mac OS X – Jeux : MS-Windows, GNU/Linux, Mac OS X, diverses plateformes mobiles (détail p. 31)...	MS-Windows	MS-Windows (Flash)	En ligne (Flash)
Licence	*GPL*	Payant (de 49 à 299 €)	*GPL*	Gratuit / Payant (version Pro)	Gratuit	Gratuit
Genre de gameplay	Tous	Tous	Tous	Tous	Tous (?)	Tous (?)
Commentaires	Voir p. 27	Voir p. 29	Voir p. 31	– Interface assez cohérente – Documentation complète et claire. Communauté active – Certains exemples convaincants – Nécessite d'écrire des scripts pour les interactions élaborées	– Interface peu modulable et un peu lente (Flash) – Élaboration d'interactions complexes difficile – Documentation de bonne qualité – Possibilité de partager ses créations et de modifier les créations partagées par les autres – Édité par Electronic Arts	– Interface difficile à comprendre et peu modulable (Flash) Élaboration d'interactions complexes difficile – Documentation un peu faible – Possibilité de partager ses créations et de modifier les créations partagées par les autres

Nom du logiciel	***<e-Adveture>***	***Adventure Game Studio***	***RPG Toolkit***	***Scrolling Game Development Kit***	***Silent Walk FPS Creator***
URL du site	http://e-adventure.e-ucm.es/	http://www.adventuregamestudio.co.uk/	http://www.toolkitzone.com/toolkit.php	http://gamedev.sourceforge.net/index.shtml	http://www.silentworks.hu/index.php
Système d'exploitation	MS-Windows, GNU/Linux, MacOS X (java)	MS-Windows avec .Net 2, anciennes versions pour GNU/Linux et Mac OS X	MS-Windows	MS-Windows avec DirectX 9	MS-Windows
Licence	*LGPL*	Gratuit	*GPL*	*GPL*	Gratuit/10$ (sans logo sur les jeux)
Genre de gameplay	*Point-and-click*	*Point-and-click*	*RPG* (jeux de rôles)	Jeux d'action avec défilement d'écran (plate-forme, tir, etc.)	*FPS* (Jeu de tir à la première personne)
Commentaires	Voir p. 10	– Interface complète, structurée et assez ergonomique – Documentation très complète et de qualité. Communauté très active – Exemples très convaincants, dont certains de qualité professionnelle	– Interface agréable et bien organisée – Documentation abondante et communauté extrêmement active et centralisée – Exemples convaincants – Un *scripting* très simple est nécessaire	– Interface parfois un peu brouillon, mais permet de faire des choses élaborées – Documentation un peu faible – Exemples peu convaincants	– L'interface est cohérente et pratique – La documentation est claire et bien illustrée – Permet de faire plutôt des jeux de labyrinthes à énigmes que des jeux de tir

III- Défense de l'organisme dans les programmes de 3e

Extrait de la partie du programme de Sciences de la Vie et de la Terre (paru au Bulletin officiel spécial n°6 du 28 août 2008) concernée par le jeu sérieux *Défenses Immunitaires*. Les parties en gras sont celles traitées par le jeu.

L'organisme est constamment confronté à la possibilité de ***pénétration de micro-organismes (bactéries et virus) issus de son environnement****.*

Ils se transmettent de différentes façons d'un individu à l'autre directement ou indirectement. Ils franchissent la peau ou les muqueuses : c'est la contamination. Après contamination, les microorganismes se multiplient au sein de l'organisme : c'est l'infection.

Ces risques sont limités par la pratique de l'asepsie et par l'utilisation de produits antiseptiques*. L'utilisation du préservatif permet de lutter contre la contamination par les microorganismes responsables des infections sexuellement transmissibles (IST) notamment celui du SIDA.*

Des antibiotiques appropriés permettent d'éliminer les bactéries. Ils sont sans effet sur les virus*.*

L'organisme reconnaît en permanence la présence d'éléments étrangers grâce à son système immunitaire*.*

Une réaction rapide – la phagocytose, réalisée par des leucocytes – permet le plus souvent de stopper l'infection*.*

D'autres leucocytes, des lymphocytes spécifiques d'un antigène reconnu *se multiplient rapidement dans certains organes, particulièrement les ganglions lymphatiques.*

Les lymphocytes B sécrètent dans le sang des molécules nommées anticorps, capables de participer à la neutralisation des microorganismes et de favoriser la phagocytose*.*

Une personne est dite séropositive pour un anticorps déterminé lorsqu'elle présente cet anticorps dans son sang.

Les lymphocytes T détruisent par contact les cellules infectées par un virus.

Les réactions spécifiques sont plus rapides et plus efficaces lors de contacts ultérieurs avec l'antigène.

La vaccination permet à l'organisme d'acquérir préventivement et durablement une mémoire immunitaire relative à un microorganisme déterminé grâce au maintien dans l'organisme de nombreux leucocytes spécifiques.

Une immunodéficience acquise, le SIDA, peut perturber le système immunitaire.

Un test permet de déterminer si une personne a été contaminée par le VIH.

IV- Scénario du jeu sérieux *Défenses Immunitaires*

IV.1- Gameplay :

Principalement inspiré de : http://armorgames.com/play/4962/bubble-tanks-tower-defense

Chaque niveau de jeu est caractérisé par un tableau délimité en 24x22 cases (Largeur*Hauteur), dont d'une bordure d'une case percée par une ou plusieurs entrées et sorties (de 7 cases chacune). Les microbes entrent par vagues plus ou moins grosses, plus ou moins espacées : ce sont les infections. Dans un même niveau, les infections de plus en plus importantes (nombre de microbes) se suivent. Tous les microbes d'une même infection (vague) sont identiques. Il faut empêcher les microbes d'aller de l'entrée à la sortie. Dans chaque case le joueur peut déposer une défense de type *tour* (qui mesure 2x2 cases, et qui agit sur les cases environnantes). Avant et pendant les infections, il faut faire correspondre la bonne défense au bon microbe, en fonction de :

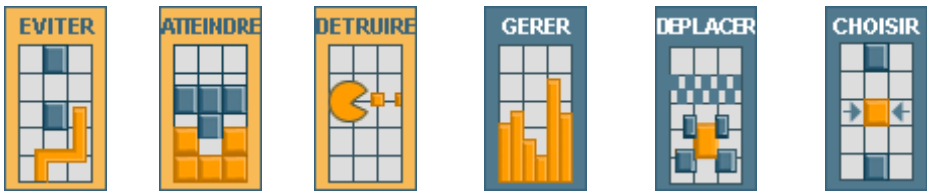
- son type (bactérie, virus),
- des antigènes qu'il porte (une couleur correspond à un antigène)
- et des moyens à disposition (« santé », « argent ») pour agir.

En veillant à ce que ces microbes ne franchissent pas le tableau (sortie) sinon la santé diminue. Si la santé devient nulle, la partie est perdue.

Le joueur peut préparer le tableau de jeu en plaçant ses défenses avant de lâcher les vagues d'infections. Il pourra aussi faire des modifications pendant les infections. Il a une vision sur les infections futures (pour pouvoir anticiper).

La stratégie du joueur est de placer les bonnes défenses aux bons endroits pour se protéger contre toutes les vagues d'infections. En gérant les budgets « santé » et « argent ».

Type de jeu selon le modèle G/P/S (Alvarez et al., 2007) :



IV.2- Éléments du jeu :

IV.2.1- Variables dynamiques :

IV.2.1.a- La santé :

Quantité non nulle au début. **Dès que la santé est nulle → perdu**.

- Qui **augmente** au fur et à mesure très lentement.
- Qui **diminue fortement** si un microbe passe à travers les défenses.
- Qui **diminue** lors de la création ou de l'utilisation de défenses.

IV.2.1.b- L'argent :

Quantité nulle au début. Nécessaire pour se soigner.

- **Augmente** si de la recherche médicale a abouti à la création de soins (*antiseptiques*, *antibiotiques*, *antiviraux*, *vaccins*, *thérapie génique antibactérienne*)
- Plus il y a de soins trouvés **plus l'augmentation est rapide**
- Utiliser un soin **diminue** la quantité d'argent.

IV.2.1.c- Les couleurs :

Elles expriment la spécificité des microbes et des moyens de défense.

- Elles symbolisent les antigènes portés par les microbes (et donc leur vulnérabilité aux défenses).
- Elles symbolisent la vulnérabilité des cellules aux virus (de même couleur).

IV.2.2- Les microbes (« ennemis ») :

Caractérisés par leur type (*Bactérie/Virus*) et leurs *antigènes* (affichés sous forme de couleurs) permettent à certaines défenses de les reconnaître.

IV.2.2.a- Les bactéries :

- **Caractéristiques** : grosses, et ont plusieurs antigènes (couleurs), mais un commun (couleur commune des bactéries)

- **Effet** : libèrent des *toxines* qui diminuent la « vie » des cellules.
- **Détruites par** : les *macrophages*, les *antiseptiques*, les *antibiotiques* (plus ou moins spécifiques), et même par un type de *virus* (ceux qui ont la couleur commune des bactéries) issus de la *thérapie génique*.

IV.2.2.b- Les virus :

- **Caractéristiques** : petits (trop petits pour les *macrophages*), ont un seul *antigène* (couleur). Certains virus (VIH, grippe) sont polymorphes : ils changent *d'antigènes* (couleur) régulièrement.
- **Effet** : peuvent infecter les cellules (cicatrisantes ou leucocytes) de la même couleur, ils les tuent alors après un délai en libérant plus des virus du même type. Cas particulier, un type de virus peut agir sur les bactéries : ceux de la *thérapie génique*.
- **Détruits par** : les *macrophages* (si et seulement si ils ont été agglomérés par des *Lymphocytes B* spécifiques avant), les *antiviraux* spécifiques.

IV.2.3- Les défenses :

Sont de deux types : de type « **tour** » et de type « **globale** ».

Les **tours** agissent localement (cases autour d'elles), les **globales** agissent sur l'ensemble du tableau de jeu :

IV.2.3.a- Défenses de type « tour » :

Les cellules de cicatrisation :

- **Caractéristiques** : très peu coûteuses en santé. Ont une couleur (au hasard ou choisie par le joueur ?) qui correspond à leur sensibilité aux virus de la même couleur.
- **Effet** : sont un mur, elles canalisent les microbes. Note : le joueur ne peut pas totalement fermer le chemin.
- **Niveaux** :
  - 1 : cellules simples
  - 2 : cellules muqueuses → elles libèrent du mucus qui ralentit les microbes
  - 3 : cellules souches → peuvent être transformées en *leucocytes*
- **Détruites par** : les *toxines* (en grande quantité). Les *virus* spécifiques (même couleur) qui les infectent.

Les leucocytes macrophages :

- **Caractéristiques** : un peu coûteux en santé. Ont tous la même couleur (sensibles aux mêmes virus). Insensibles aux toxines.
- **Effet** : phagocytent (mangent et détruisent) les gros éléments qui passent dans leur rayon d'action. C'est-à-dire les *bactéries* et les *virus agglutinés* par des *anticorps* (sécrétés par les *lymphocytes B*)
- **Niveaux** :
  - de 1 à 3 : augmentation du rayon d'action et de la vitesse d'absorption
- **Détruits par** : les *virus* qui leur sont spécifiques (même couleur).

Les leucocytes de reconnaissance :

- **Caractéristiques** : un peu coûteux en santé. Ont une couleur (mais dans le jeu, aucun virus n'a la même couleur).
- **Effet** : apprennent les *antigènes* (couleurs) des microbes. Ils sont nécessaires pour fabriquer des *Lymphocytes B et T* fonctionnels.
- **Niveaux** :
  - de 1 à 3 : augmentation du rayon d'action et de la vitesse de reconnaissance
- **Détruits par** : les *toxines* (lentement).

Les leucocytes Lymphocytes B :

- **Caractéristiques** : assez coûteux en santé. Ont deux couleurs :
  - couleur externe → tous la même, elle définit leur *sensibilité aux virus*.
  - Couleur interne → antigènes qu'ils savent reconnaître et agglutiner, c'est le joueur qui la choisit parmi celles disponibles (en fonction de ce que les Leucocytes de reconnaissance ont appris). Sans couleur interne (qui peut-être définie plus tard), le Lymphocyte B est inactif.
- **Effet** : produisent des *anticorps* spécifiques (couleur donnée) : ils agglutinent les microbes ayant les mêmes *antigènes* (couleur interne). L'agglutination rend les microbes plus lents et plus faciles à détruire par les *macrophages* (les *macrophages* ne peuvent manger les *virus qu'agglutinés*).
- **Niveaux** :
  - de 1 à 2 : augmentation de la quantité *d'anticorps* produits
  - 3 : Lymphocyte B mémoire → Il a deux effets :

- permet de poser des *lymphocytes B* du même type pour un coût (santé) plus faible
- *séropositivité* : permet de fabriquer des *Lymphocytes T et B* de la même couleur interne (donc capable *d'agglutiner* les mêmes antigènes) dès le début des niveaux suivants (sans qu'il y ait besoin de *Leucocytes de reconnaissance*)

- **Détruits par** : les *toxines* (lentement), les *virus* qui leur sont spécifiques (même couleur).

Les antibiotiques :

- **Caractéristiques** : coûtent un peu d'argent et nécessitent d'être au niveau de *recherche médicale n°2*. Peuvent avoir un mélange de couleur plus ou moins grand selon leur niveau. Coûtent aussi en santé.
- **Effet** : Détruisent les *bactéries* plus ou moins vite (voir niveaux). À chaque destruction la santé baisse un peu.
- **Niveaux** :
  - 1 : *antibiotiques de spectre large*. Ils ont une seule couleur (la couleur commune des bactéries), c'est-à-dire agissent sur toutes les bactéries, mais sont peu efficaces (lents).
  - 2 : *antibiotiques de spectre intermédiaire*. Ont une couleur de plus. Ils agissent sur les bactéries qui ont au moins cette couleur. Ils sont plus rapides.
  - 3 : *antibiotiques à spectre étroit*. Ils ont 2 couleurs (en plus de la couleur commune). Ils n'agissent que sur les bactéries qui ont cette combinaison de couleurs. Ils sont très rapides.
- Pas destructibles, mais **ont une durée de vie** dans l'organisme. Après quoi ils disparaissent.

Les antiviraux :

- **Caractéristiques** : coûtent beaucoup d'argent et nécessitent d'être au niveau de *recherche médicale n°3*. Ont une ou plusieurs couleurs (ex. trithérapies). Coûtent aussi en santé.
- **Effet** : Détruisent les *virus* auxquels ils sont spécifiques (même couleur). À chaque destruction, la santé baisse un peu.

- Pas de niveau
- Pas destructibles, mais **ont une durée de vie** dans l'organisme. Après quoi ils disparaissent.

IV.2.3.b- Défenses de type « globale » :

Les leucocytes Lymphocyte T :

- **Caractéristiques** : assez coûteux en santé. Mettent du temps à être fabriqués et libérés (ce temps peut-être réduit s'il y a *séropositivité*). Sont spécifiques (ont une couleur choisie par le joueur quand il demande la fabrication), mais nécessitent qu'il y ait eu reconnaissance (par des *Leucocytes de reconnaissance* ou grâce à la *séropositivité*).
- **Effet** : libérés sous forme de vague après leur fabrication, ils tuent les *cellules* (*cicatrisantes* ou *leucocytes*) infectées par les *virus* de même couleur partout sur le tableau de jeu.
- Pas de niveau
- pas destructibles. Leur **effet est ponctuel**.

Les antiseptiques :

- **Caractéristiques** : Coûtent peu en argent. Nécessitent le niveau de *recherche médicale n°1*. N'ont pas de couleur.
- **Effet** : libérés sous forme de vague, ils tuent 80 % des *bactéries* qui sont proches des zones d'infection. Certaines bactéries sont plus résistantes (taux de destruction plus faible).
- Pas de niveau
- Pas destructibles. Leur **effet est ponctuel**. Le joueur ne peut enchaîner leur utilisation, il faut attendre le **rechargement**.

Thérapie génique antibactérienne :

- **Caractéristiques** : Coûte beaucoup d'argent. Nécessité le niveau de *recherche médicale n°5* (le plus élevé). Ont deux couleurs (la couleur commune des bactéries+une couleur choisie par le joueur).
- **Effet** : libère une vague de *virus inoffensifs*, capables de détruire entre 99 % et 100 % des *bactéries* de mêmes couleurs.

- Pas de niveau
- Pas destructibles. Leur **effet est ponctuel**. Le joueur ne peut enchaîner leur utilisation, il faut attendre le **rechargement** (très très lent).

Le Vaccin :

- **Caractéristiques** : Assez coûteux. Nécessite le niveau de *recherche médicale n°4*. A une couleur, choisie par le joueur.
- **Effet** : fait acquérir immédiatement la *séropositivité* spécifique d'un antigène (couleur) : elle permet donc de fabriquer des *Lymphocytes B et T* à moindre coût et sans avoir besoin des *Leucocytes de reconnaissance*. Cette *séropositivité* est définitive (jusqu'à la fin du jeu).
- Pas de niveau
- Pas destructible. L'**effet est permanent**.

IV.2.4- La recherche médicale :

Elle permet développer des **soins**. Elle prend (du temps) et coûte de l'argent à partir du niveau n°2. Mais quand un soin est développé, il rapporte aussi de l'argent progressivement.

Les niveaux sont :

- 1 : les **antiseptiques**. Sont vite développés, mais rapportent peu.
- 2 : les **antibiotiques**. Ont besoin du niveau n°1 pour être développés. Sont plus lents à concevoir. Rapportent un peu plus.
- 3 : Les **antiviraux**. Ont besoin du niveau n°2 pour être développés. Sont plus lents à concevoir. Rapportent un peu plus.
- 4 : les **vaccins**. Ont besoin du niveau n°3 pour être développés. Sont vraiment lents à concevoir. Rapportent beaucoup.
- 5 : la **thérapie génique antibactérienne**. A besoin du niveau n°4 pour être développée. Est vraiment très très lente à concevoir. Rapporte énormément.

V- Pages de tutoriel de *Défenses Immunitaires*

Ces pages s'affichent comme écran de chargement avant chaque niveau du jeu *Défenses Immunitaires* et présentent à l'apprenant-joueur à la fois des indications de gameplay et des informations scientifiques[1]. Chaque page-écran est introduite par un titre, suivie d'un petit texte dans lequel les mots-clés sont mis en emphase.

Table des écrans de présentation des niveaux

1 Voir aussi « Conception des spécifications d'un jeu sérieux pour l'évaluation des outils auteurs » p. 19.

Indications préliminaires du jeu

Mehdi s'est écorché.

La **contamination** commence, car des **microbes** entrent par la petite blessure

Vous devez empêcher **l'infection** de s'aggraver en détruisant les **microbes** avant qu'ils ne se propagent dans tout l'organisme

Pour cela, placez des *défenses* pour empêcher les **microbes** de traverser l'écran

Quand les **microbes** parviennent à sortir de l'écran, la **santé** de Mehdi diminue. Il ne faut pas qu'elle atteigne 0 où vous aurez perdu...

Niveau 1

Mehdi est contaminé par des **bactéries.** Elles peuvent se déplacer. Elles libèrent parfois des toxines qui abîment les cellules

Pour empêcher l'infection bactérienne de se propager, vous pouvez placer des **cellules cicatrisantes** qui peuvent les ralentir (sans bloquer complètement le passage)

Les bactéries peuvent être détruites par des **leucocytes macrophages**. Les bactéries passant à proximité de ces cellules-là seront absorbées (c'est la *phagocytose*).

Mais attention, fabriquer des cellules comme les **cellules cicatrisantes** ou les **macrophages** consomme de l'énergie et fait diminuer la **santé** de Mehdi. Veillez à les utiliser avec parcimonie.

Niveau 2

Certaines **cellules cicatrisantes** peuvent fabriquer un liquide gluant (le mucus) qui ralentit les bactéries. Ce sont les **cellules muqueuses**

D'autres **cellules cicatrisantes** peuvent changer de type et de devenir des **leucocytes macrophages** si besoin. Ces cellules qui peuvent changer de type sont les **cellules souches**

Vous pouvez augmenter le niveau des **cellules cicatrisantes** placées pour empêcher les microbes d'avancer. Elles deviennent alors des **cellules muqueuses** au niveau 2 et des **cellules souches** au niveau 3

Augmenter le niveau des **cellules cicatrisantes** placées fait diminuer la **santé** de Mehdi, pensez-y !

Niveau 3

Il y a de plus en plus de **bactéries !**

Heureusement, nous pouvons aussi aider Mehdi grâce à la médecine !

Les **antiseptiques** sont des produits médicaux qui détruisent les **bactéries**. Ils s'appliquent sur les plaies

Vous pouvez appliquer une vague **d'antiseptique** sur la plaie de Mehdi pour limiter le nombre de **bactéries**

Mais attention, vous avez peu **d'argent** et chaque application d'antiseptique vous coûtera un peu

Niveau 4

La *contamination* continue de plus belle !

Vous pouvez faire progresser la médecine en investissant dans la **recherche médicale**

Les chercheurs sont sur le point de trouver comment fabriquer des **antibiotiques**

Ils seront capables de détruire les **bactéries** autour d'eux

Terminer ces recherches vous coûtera un peu **d'argent**. Chaque **antibiotique** posé aussi...

Lancez la **recherche médicale** dès que vous en avez les moyens, et aidez-vous des **antibiotiques** pour vaincre l'infection

Niveau 5

Devant l'ampleur de la contamination, les **antibiotiques** pourraient ne pas être assez efficaces...

Les chercheurs ne sont pas loin de pouvoir en produire des plus puissants. Malheureusement, ils ne fonctionnent que sur les bactéries de la même couleur ! Ce sont les **antibiotiques à spectre étroit**

Si vous financez encore la **recherche médicale**, vous pourrez disposer de ces **antibiotiques** plus efficaces. Pensez à faire attention à leur *spécificité*.

Vous remarquerez que la quantité **d'argent** augmente plus vite quand vous financez la **recherche médicale**. C'est normal, les nouveaux soins découverts sont vendus et cela vous enrichit.

Niveau 6a

Les vagues de contamination sont très désormais très nombreuses...

Les **lymphocytes B** sont des *leucocytes*, comme les **macrophages**. Ces cellules permettent de défendre l'organisme en libérant des **anticorps** qui ralentissent les microbes en les **agglutinant** et les rendent plus faciles à **phagocyter** par les **macrophages**

Les **anticorps** sont spécifiques d'un seul type de **microbe !** Un **lymphocyte B** ne peut produire des **anticorps** que contre un seul type de **microbe** (reconnu à ses **antigènes** représentés par leur couleur)

Pour l'instant l'organisme de Mehdi sait fabriquer peu de types **d'anticorps**. Choisissez bien les **lymphocytes B** que vous placez.

Niveau 6 b

D'autres types de **bactéries** débarquent ! Elles ont des **antigènes** encore inconnus du corps de Mehdi

Ses **lymphocytes B** ne savent pas fabriquer des **anticorps** contre ces nouvelles **bactéries**

Vous pouvez placer des **leucocytes de reconnaissance**, qui apprennent les **antigènes** des nouveaux microbes pour que le corps puisse fabriquer des **anticorps** adaptés.

Dès que Mehdi sait fabriquer des **anticorps** contre un **antigène,** on dit qu'il est *séropositif* pour cette maladie (Si si !)

Placez astucieusement des **leucocytes de reconnaissance** pour que le corps de Mehdi apprenne à se défendre avec des **anticorps** adaptés à ces nouvelles **bactéries**

Niveau 7

Comme si les bactéries ne suffisaient pas ! Voici de nouveaux microbes : les **virus !**

Au contact des *cellules*, ils s'y introduisent parfois et la forcent alors à fabriquer d'autres **virus** avant d'éclater

Ils sont portés par les courants dans le corps de Mehdi

Trop petits, ils ne peuvent pas être **phagocytés** (absorbés) par les **macrophages** sans avoir été auparavant **agglutinés** avec des **anticorps !**

En plus, les **antiseptiques** et **antibiotiques** ne sont pas efficaces sur eux !

Bon courage !

Niveau 8

Il va falloir bien anticiper, car il se présente à la fois des vagues de **bactéries** et de **virus...**

Placez bien vos **leucocytes de reconnaissance**, **macrophages** et **lymphocytes B**

Contre les **bactéries**, vous pourrez aussi avoir recours aux **antiseptiques** et aux **antibiotiques**

Avec tout ça vous devriez vous en sortir... ou pas !

Niveau 9

Vous avez réussi à arriver là... bravo...

Mais il y a désormais encore plus de **microbes** et ils sont encore plus variés !

Heureusement vous pouvez utiliser un nouveau type de **lymphocytes :** les **lymphocytes T** (comme Tueurs ! :-))

Quand un **antigène** est connu par Mehdi (a été reconnu par les **leucocytes de reconnaissance**), les **lymphocytes T** permettent d'en détruire une bonne part. Cool !

Mais, libérer les **lymphocytes T** dépense beaucoup de **santé** et n'est possible que de temps en temps.

Je ne crois pas que vous pourrez y arriver sans... Même avec, j'en doute.

Niveau 10

Vous avez sans doute remarqué que vous pouviez augmenter le **niveau** des *cellules* que vous placez (les **cellules cicatrisantes**, les **macrophages**, les **lymphocytes B** et les **leucocytes de reconnaissance**).

Un troisième niveau de **lymphocytes B** arrive ! Les **lymphocytes mémoires**...

Ils permettent à Mehdi de devenir définitivement séropositif contre un **antigène :** plus besoin de **leucocytes de reconnaissance** quand il est **séropositif** contre un **antigène**, il saura toujours fabriquer des **anticorps** contre eux avec ses **lymphocytes B**

Chouette !

Niveau 11

Étant donné que la situation ne fait qu'empirer, un peu d'aide de la **recherche médicale** ne serait pas de trop !

Justement, les chercheurs nous informent qu'ils sont sur le point de fabriquer des **antiviraux !**

Ça va coûter cher à mettre au point, mais c'est prometteur !

Ça devrait permettre d'éliminer une bonne partie des **virus** d'un type donné...

Donc, vous devriez sans doute investir dans la **recherche médicale**

Niveau 12

Là...

Hem...

C'est un sacré **virus** qui apparaît : il ne s'attaque pas à n'importe quelles *cellules*, mais aux *leucocytes*, soit celles qui *défendent l'organisme* (**leucocytes de reconnaissance**, **macrophages**, **lymphocytes**)

Bonne chance...

Niveau 13

Vous vous en sortez, mais ça pourrait ne pas durer...

Heureusement les chercheurs sont sur le point de mettre au point le vaccin.

Le vaccin permettra de rendre Mehdi **séropositif** pour un **antigène** sans avoir besoin de **leucocyte de reconnaissance** ou de **lymphocyte mémoire**

Il sera donc tout le temps capable de fabriquer des **anticorps** contre cet **antigène**

Pratique!... Mais finir cette **recherche médicale** risque de coûter assez... cher !

Niveau 14

Voilà de nouveaux **virus** très casse-pieds...

Ils changent **d'antigène** tout seuls...

Niveau 15

Une dernière avancée de l'équipe de **recherche médicale !**

Ils sont sur le point de finir d'inventer des techniques de **thérapie génique**

C'est très complexe et très très cher !

Il s'agit de fabriquer nos propres **virus** inoffensifs, mais qui attaquent les **bactéries** et les *cellules infectées* par des **virus** pour les empêcher de nuire.

Niveau 16

Contamination par le VIH, virus du SIDA !

Ce **virus** change **d'antigène,** mais en plus s'attaque aux **lymphocytes !**

VI- Étapes de réalisation de *Défenses Immunitaires*

L'évaluation en profondeur des outils-auteurs auteurs testés au cours de mon stage repose sur la réalisation du jeu *Défenses Immunitaires*. Pour que cette réalisation permette au mieux d'évaluer les outils, elle a été scénarisée par étapes chronologiques[1].

Tables des étapes de réalisation

1 Voir la description de ce travail : « Scénario de réalisation de Défenses Immunitaires » p. 25.

Étape 1 : Construire l'aire de jeu

La spécification établit que l'aire de jeu fait 24x22 cases (Largeur*Hauteur), dont une bordure épaisse d'une case percée par une ou plusieurs entrées et sorties (de 7 cases chacune).

Étape 2 : Faire parcourir l'aire de jeu par une bactérie

Les bactéries sont un premier type de microbe. Elles peuvent se déplacer en évitant des obstacles. Il faut donc simuler cette trajectoire entre l'entrée et la sortie. D'après la spécification, la trajectoire doit paraître aléatoire, car c'est plus réaliste, mais les bactéries doivent aussi atteindre la sortie suffisamment vite pour une meilleure jouabilité. Il faut donc utiliser (ou implémenter) une fonction de dessin d'objets mobiles (*sprites*) et une fonction de recherche de trajectoire (*pathfinding*).

Étape 3 : Tester des obstacles sur les trajets

Vérifier que les obstacles (tours) sont bien évités par les bactéries et que les trajectoires sont bien calculées (et que les *sprites* restent dans l'aire de jeu).

Étape 4 : Tester l'introduction des macrophages

Les macrophages sont des cellules capables de détruire les bactéries par contact (les macrophages absorbent les bactéries par phagocytose). Dans le jeu les macrophages sont symbolisés par des tours au comportement similaire (destruction par contact). Il faut donc utiliser ou implémenter une fonction de détection de collision. En cas de collision, modifier l'affichage : animation de la phagocytose, disparition de la bactérie.

Étape 5 : Construire une vague de bactéries

Dans les *tower defense*, une vague est une succession de plusieurs ennemis identiques. Il faut donc construire une succession de bactéries.

Étape 6 : Faire parcourir l'aire de jeu par une vague de virus

Les virus sont des microbes qui n'ont pas de déplacement propre, ils sont portés par les courants de fluides internes. Selon la spécification il faut créer un courant qui entraîne les virus de haut en bas de l'aire de jeu avec une légère « gravité » vers la sortie de l'aire pour favoriser la jouabilité. Sur les obstacles (bord de l'aire de jeu ou tours), les virus doivent rebondir. Il faut donc trouver un système

pour faire suivre un courant à une vague de virus, provoquer des rebonds lors des collisions, et introduire une légère gravité vers la sortie de l'aire.

Étape 7 : Tester l'introduction des lymphocytes et de leurs anticorps

Les lymphocytes sont des cellules qui produisent des anticorps contre les microbes qu'ils sont capables de reconnaître. La spécification indique que les lymphocytes sont des tours qui émettent rythmiquement des couronnes de 8 anticorps dès que des microbes qu'ils reconnaissent sont dans l'aire de jeu. Les anticorps ont des trajectoires linéaires et rebondissent sur les obstacles. Ils ont une durée de vie après laquelle ils disparaissent. Il faut donc détecter la présence de microbes spécifiques, créer le déplacement des anticorps (en gérant leurs collisions et leurs rebonds)

Étape 8 : Des anticorps qui agglutinent les microbes reconnus

Les anticorps agglutinent les microbes. Les microbes agglutinés sont facilement phagocoytés par les macrophages (et notamment les virus qui ne peuvent être phagocytés sans être agglutinés). Lors de la collision des microbes (spécifiques) avec les anticorps, leur trajectoire doit changer (trajectoire commune puisque les éléments sont agglutinés) et les microbes doivent changer de statut (devenir phagocytables).

Étape 9 : Rajouter un début de gestion de la santé

La santé, affichée à l'écran augmente très lentement et diminue dès que les microbes passent la sortie. La spécification indique aussi qu'un *feedback* doit être donné au joueur à chaque fois que de la santé est perdue. Il faut implémenter l'affichage d'un compteur et en modifier la valeur en fonction de la collision avec la sortie.

Étape 10 : Interface simple de rajout des tours

Les tours sont les lymphocytes et les macrophages que les utilisateurs déposent sur l'aire de jeu. La spécification indique que les tours sont placées sur une grille de 12x11 cases. Que le dépôt d'une tour retire de la santé (en fonction du type de tour) et que le *pathfinding* doit être testé à chaque fois.

Il faut donc une fonction de sélection de tour, de placement des tours et de vérification de l'ouverture du chemin des microbes. Il s'agit donc de mettre en place les premières fonctions interactives du jeu.

Étape 11 : Afficher des informations sur les objets

Lors d'un clic sur les objets du jeu (tours, ennemis), l'information doit s'afficher. Cela complète les fonctions d'interface du jeu.

Étape 12 : Fabrication d'un niveau de jeu

Un niveau de jeu est défini par un certain nombre de paramètres comme les tours possibles à placer ou encore le nombre, le type et l'ordre des vagues d'ennemis (paramètres illustrés Figure 17 p. 24). Pour fabriquer un niveau, il faut aussi prévoir plusieurs phases de jeu : l'affichage de l'écran d'information du niveau. La phase de préparation pendant laquelle l'apprenant/joueur choisit quelles tours placer. Et la phase d'action qu'il déclenche quand il se sent prêt et qui fait défiler les vagues d'ennemis dans les défenses qu'il a disposées. Puis l'écran final qui fait le bilan de l'action du joueur (santé restante).

Il faut donc au préalable prévoir l'initialisation des paramètres du niveau et la réalisation du passage des vagues successives. Puis mettre en œuvre les trois phases du jeu.

Étape 13 : Externaliser les paramètres du niveau dans un fichier de configuration

Pour faciliter l'édition des niveaux, voire leur conception par un outil externe, il est prévu d'externaliser tous les paramètres d'un niveau dans un fichier XML. Il faut donc implémenter des fonctions de lecture de ce fichier et d'initialisation des variables du niveau en fonction.

VII- Détails de l'évaluation de *Construct*

L'outil auteur *Construct* est présenté p. 27. Ci-dessous se trouvent les détails de son évaluation selon les critères retenus[1].

VII.1- Utilisabilité de l'interface graphique et accessibilité du logiciel

L'interface de *Construct* visible sur la Figure 23 s'articule autour de plusieurs zones. Une zone

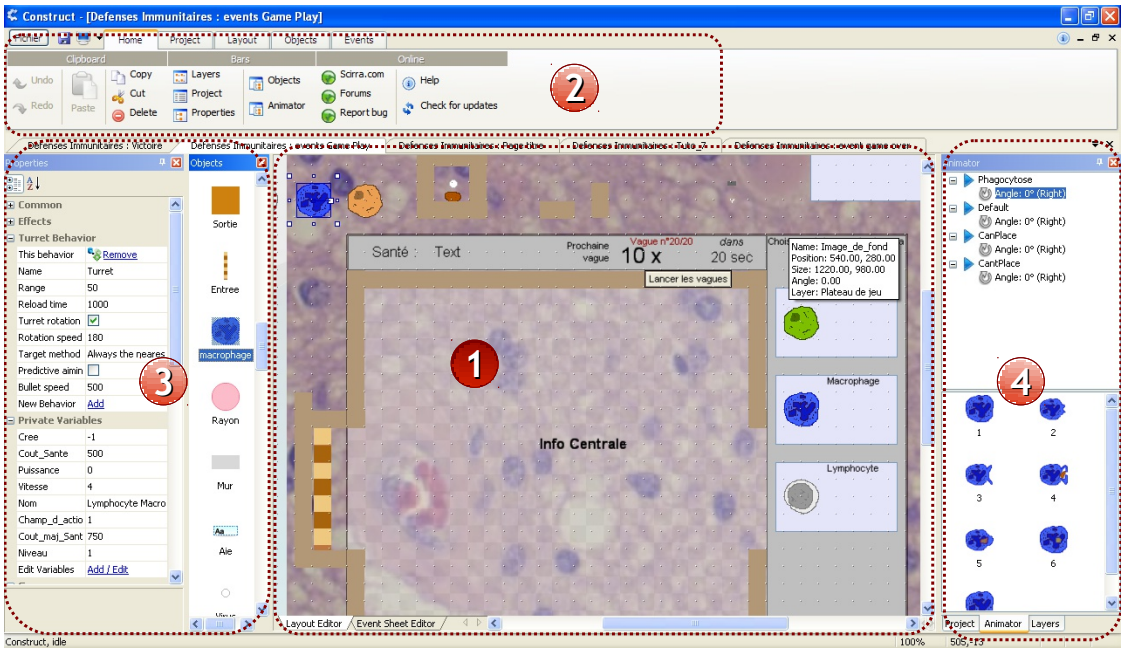


Figure 23 : Aspect général de l'interface de l'outil auteur *Construct*. Les pointillés rouges délimitent les zones qui sont numérotées :

- Zone 1 : zone d'édition, ici le canevas des objets graphiques (*Layout Editor*)
- Zone 2 : bandeau des outils (copié-collé, sauvegarde, *undo*, etc.)
- Zone 3 : panneau des propriétés des objets
- Zone 4 : panneau de gestion du projet, des calques et des animations. Ici la gestion de l'animation des *macrophages* de *Défenses Immunitaires*

centrale d'édition permettant soit de travailler sur les objets graphiques et leur position sur l'aire de jeu, soit sur la programmation événementielle. Cette zone centrale est entourée en haut par un bandeau d'outils contextualisés, à gauche par un panneau d'édition des propriétés des objets du jeu,

1 Les critères d'évaluation sont présentés dans : « Critères d'évaluation retenus » p. 15.

à droite par un panneau permettant de travailler sur plusieurs éléments : les modules du projet, les animations des objets graphiques et les calques de l'aire de jeu. L'ensemble des textes est en anglais, il n'existe pas de traduction.

Le logiciel sera familier aux utilisateurs de *MS-Windows* puisqu'il en utilise le *toolkit* graphique standard dont l'apparence est reconnue. L'interface est donc homogène et reprend ainsi les éléments d'ergonomie de l'interface graphique de *MS-Windows*. *Construct* propose des thèmes pour cette interface reprenant les aspects visuels d'applications connues (*MS-Windows XP*, *MS-Office*, *MS-Visual Studio*, etc.) qui ont tous une très bonne lisibilité. La seule autre personnalisation de l'interface possible est de changer la position des différents panneaux les uns par rapport aux autres ou de les rendre flottants.

La conception d'un jeu avec cet outil s'articule autour d'éléments d'interface centraux. Le canevas permet de créer, configurer et agencer les éléments graphiques du jeu. La Figure 23 illustre ces éléments graphiques pour *Défenses Immunitaires* : ce sont l'aire de jeu, ses bordures, les éléments d'interface (sélection des tours, scores, etc.) et les différents *sprites* (objets mobiles) utilisés par le jeu. L'autre élément d'interface central est celui de la programmation événementielle qui permet de construire la logique du jeu. Il est complété par un formulaire de sélection de comportements des objets du jeu. Ce découpage de l'interface en zones comme celles de l'édition du canevas, de la programmation des événements, des propriétés des objets, etc. (voir Figure 23) diminue la charge de travail de l'utilisateur. Ce découpage offre une bonne concision et une faible densité informationnelle des éléments d'interaction dans chacune de ces zones de travail.

L'édition dans le canevas des objets graphiques fonctionne comme avec un éditeur de schéma WYSIWYG : il est possible de créer des objets, de leur affecter une forme, puis de les placer sur l'aire de jeu. Soit à leur place définitive, soit à une place provisoire sachant qu'ils pourront être replacés plus tard par le programme élaboré avec l'éditeur d'événements. Les objets y sont simples à dupliquer puisque par copier-coller se créent de multiples instances d'un même objet, sachant que les instances héritent des propriétés de l'objet dont elles sont issues. Une fonction d'annulation existe et permet de corriger ses erreurs. L'aire de jeu pouvant être une vue restreinte de ce qui est affiché sur le canevas (pour permettre des défilements par exemple), l'auteur peut à tout moment provoquer la compilation du jeu par le *runtime* inclus et donc vérifier son travail.

La programmation du jeu se fait dans l'éditeur d'événements par le biais de formulaires avec un guidage très équilibré des utilisateurs. Ces formulaires de type *Wizard*, accompagnent l'utilisateur par une suite de propositions qui doivent être acceptées pour passer à la suivante. Dans ces

formulaires, l'utilisateur est guidé dans ses choix, soit par des listes catégorisées (Figure 25), soit par des icônes, soit par des champs dont la couleur change en fonction de la validité des expressions qui y sont données (Figure 24). Cette interface s'adapte aussi aux utilisateurs expérimentés en leur proposant un champ de recherche qui réduit les éléments proposés ou avec la possibilité de court-circuiter une partie des clics en écrivant directement des expressions qui sont aussi éditables avec le formulaire.

Figure 24 : Dans ce morceau de formulaire de *Construct*, l'expression « `Anticorps.Y-` » incomplète est signalée par un léger fond rouge, alors que l'expression correcte « `Anticorps.X-5` » a un fond légèrement vert..

Par exemple, pour renseigner une position par rapport à un objet (un anticorps), il faut cliquer sur l'icône de l'anticorps dans la liste des objets (zone 2 de la Figure 25) ce qui provoque l'apparition d'un nouveau

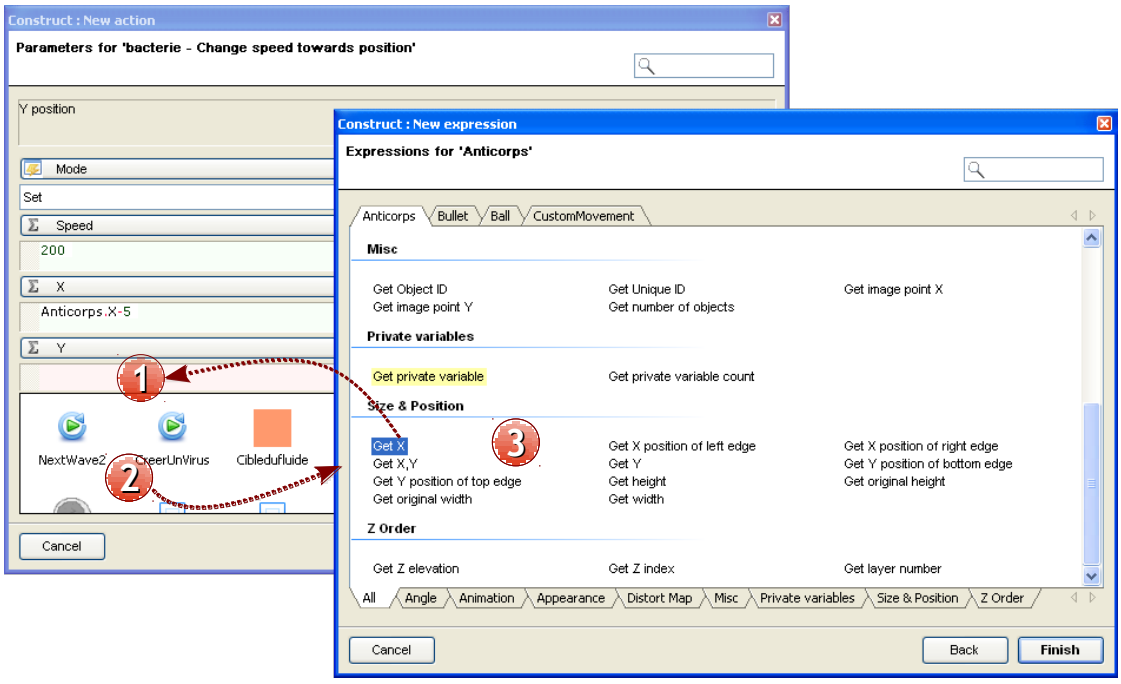


Figure 25 : Exemple de *guidage*, de *groupement par la localisation* et de *prise en compte de l'expérience de l'utilisateur* (Bastien and Scapin, 1993) dans l'interface de *Construct*.

- L'utilisateur est guidé dans ses choix par des listes (zones 2 et 3)
- Dans la zone 3, les items de la liste sont groupés par type (par des onglets et par des groupes)
- L'utilisateur qui sait ce qu'il cherche dans la liste peut utiliser le formulaire de recherche (en haut à droite)
- L'utilisateur expérimenté peut aussi écrire directement « `Anticorps.Y` » dans la zone 1

formulaire où il est possible de choisir parmi les nombreuses propriétés de l'objet : dans la catégorie « *Size & Position* » l'utilisateur peut choisir « *Get Y* » (zone 3 de la Figure 25) ce qui remplit le formulaire avec « `Anticorps.Y` » (zone 1 de la Figure 25). Si l'utilisateur est expérimenté et qu'il

connaît déjà l'expression correspondante, il peut taper directement « `Anticorps.Y` » dans le formulaire initial (zone 1 de la Figure 25).

Le travail dans cette interface de programmation permet de produire une sorte de listage des événements qui vont régir le jeu. Comme cette liste peut rapidement devenir très longue, les concepteurs de *Construct* ont prévu des moyens de grouper les événements et de replier ces groupements (Figure 26). Malheureusement, les glisser-déposer et copier-coller qui permettent ces regroupements ont un fonctionnement assez erratique qui alourdit beaucoup le travail des auteurs de jeux. De même, les annulations (*Undo*) donnent des résultats assez étranges et ne fonctionnent pas toujours. Les sauvegardes fréquentes sont donc indispensables. Heureusement, parmi les nombreux raccourcis clavier de *Construct*, `Ctrl-S` est bien dédié à la sauvegarde rapide.

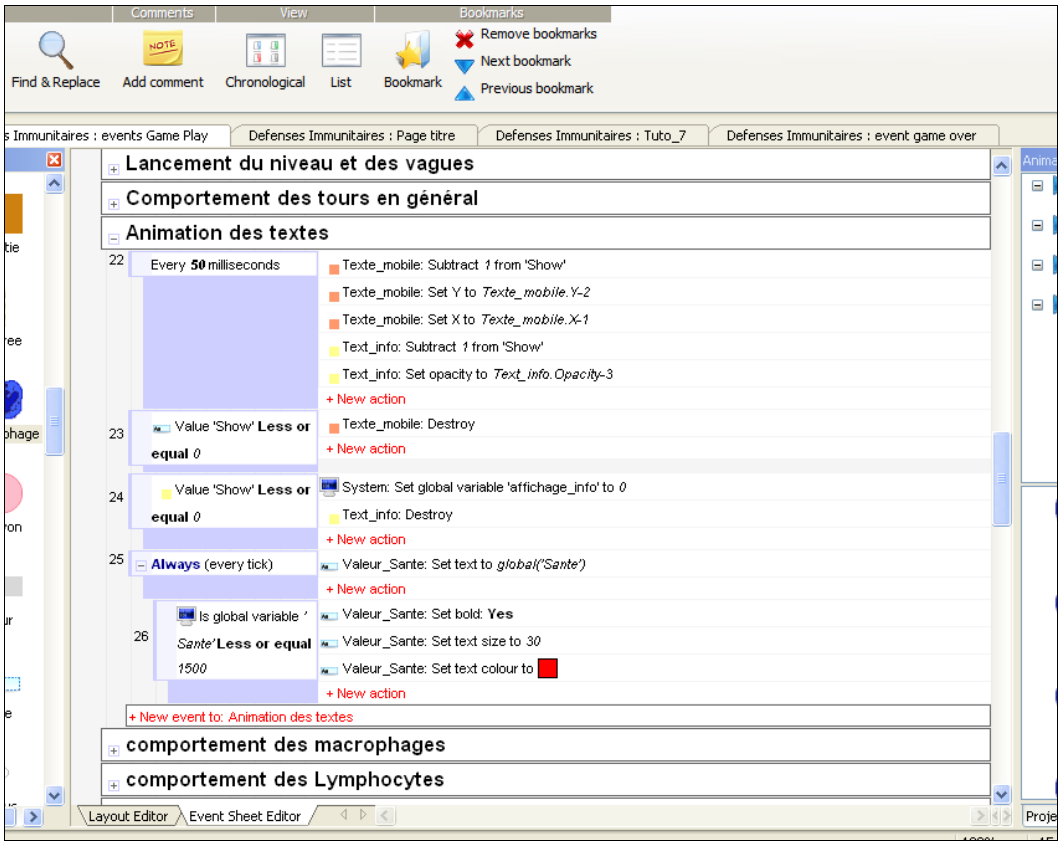


Figure 26 : Dans *Construct*, les événements sont listés les uns au-dessus des autres et peuvent être groupés. Le groupe « *Animation des textes* » est ici déplié, alors que les autres sont repliés.

Cet outil auteur pose d'autres problèmes d'accessibilité. Par exemple, *Construct* ne propose aucun outil de travail collaboratif ou de gestion de version. En outre, comme les fichiers de *Construct* sont binaires, il n'est pas possible d'utiliser un gestionnaire de versions externe (comme *Subversion* ou *GIT*). *Construct* ne se distingue pas non plus par son d'interopérabilité. Les jeux produits sont limités à *MS-Windows* avec *DirectX 9*. Et à cause de son format de fichier binaire, le logiciel ne

permet pas d'importer et d'exporter tout ou partie de son travail dans un format interopérable ou vers un autre logiciel du même type. Enfin, les outils de manipulation de données de *Construct* sont tellement frustres qu'il très ardu d'utiliser des informations venant d'un fichier texte. Il est par exemple très difficile de concevoir un programme capable de lire le contenu de balises XML pour configurer un niveau (voir plus en détail la lecture d'un fichier de configuration p. 75).

Donc, si des auteurs choisissent de commencer un projet avec *Construct* et qu'ils doivent par la suite changer d'outil, le développement devra être repris de zéro, car cet outil auteur ne permettra l'échange d'aucun élément. Ainsi, le *coût de sortie* d'un développement avec *Construct* est énorme.

VII.2- Documentation et support

Construct se caractérise par un gros défaut qui est l'absence d'aide dans l'interface du jeu. Si les formulaires ont des intitulés clairs et que certains ont parfois un petit texte explicatif pour annoncer leur effet (visible Figure 27 p. 72), il n'y aucune bulle d'aide, ni aucun accès à une aide contextuelle comme il en existe souvent dans les logiciels (souvent associée à la touche `F1`). Donc l'apprentissage de l'utilisation du logiciel ne pourra se faire qu'à tâtons ou en utilisant les documentations, tutoriels et supports disponibles sur internet.

Heureusement, la documentation disponible en ligne est assez riche et très simplement accessible sur le site de *Construct*[1]. Elle est composée de tutoriels pour les débutants, d'une FAQ (*Frequently Asked Questions*), et de modes d'emploi centralisés dans un Wiki. L'ensemble de ces ressources est en anglais.

Les tutoriels sont comme souvent sous forme de pas-à-pas et sont suffisamment bien conçus pour qu'un néophyte ait une bonne vision du fonctionnement de *Construct*. Notamment le tutoriel pour les grands débutants : *Ghost Shooter tutorial*[2] qui permet d'avoir d'excellentes bases. Le mode d'emploi principal référence la plupart des fonctions (mais pas tout à fait toutes !) et donne dans la plupart des cas des explications détaillées avec parfois des captures d'écran. Par contre, à part les tutoriels, il n'existe pas d'entrée thématique dans la documentation, ainsi il n'y a pas de *fiches-méthode* pour apprendre à accomplir certains types de tâches.

Si les auteurs de jeux ne trouvent pas dans la documentation en ligne, ils peuvent essayer le forum. La communauté qui l'anime n'est pas très grande (une centaine de personnes, dont seulement une

1 http://www.scirra.com/help/
2 Le texte du tutoriel : http://www.scirra.com/tutorials/ghostshooter/tutorial.zip, la vidéo du tutoriel : http://scirra.com/phpBB3/viewtopic.php?p=2319, et enfin le jeu qui est fabriqué avec : http://www.scirra.com/construct/demos/ghostshooter_cap.zip

dizaine quotidiennement actifs), mais les réponses sont assez rapides (quelques jours) et plutôt détaillées et exactes. En outre, ces forums sont une assez bonne base de connaissances pour compléter les lacunes de la documentation.

L'ensemble de ces éléments de documentation est centralisé et il n'est pas nécessaire de s'éparpiller sur le web pour le trouver. Il en est de même avec les exemples et extensions de *Construct*. Certains sont proposés directement sur le site, mais la plupart peuvent être trouvés dans les forums. Ces exemples sont nombreux et illustrent les différents genres de gameplay. Ainsi, il m'a été indispensable d'étudier à fond l'exemple le plus abouti de *tower defense* disponible *td.2.cap*[1] pour commencer à travailler sur la réalisation de *Défenses Immunitaires*.

VII.3- Outils de programmation du logiciel

Après avoir parcouru le *Ghost Shooter Tutorial*, il est très simple de commencer à travailler avec *Construct*. Grâce à l'interface décrite plus haut (p. 67), l'utilisateur est guidé pour arriver à réaliser sa programmation. Cette programmation repose sur plusieurs éléments comme les **objets,** les **comportements**, les **événements** et les **actions**. Dans *Construct* les jeux sont composés **d'objets** qui peuvent être des *sprites*, des textes, une surveillance du clavier et de la souris, des médias divers comme du son ou de la vidéo, des éléments d'interface comme des boutons ou des boites d'information, des structures de données comme un tableau, une chaîne de caractères ou un fichier, des éléments graphiques particuliers comme des particules ou des effets de lumière, etc. Il est possible d'attribuer à ces objets des propriétés comme leur nom, leur position, et des propriétés spécifiques comme la couleur, la taille des polices, etc. Il y a des propriétés particulières qui peuvent être attribuées à un

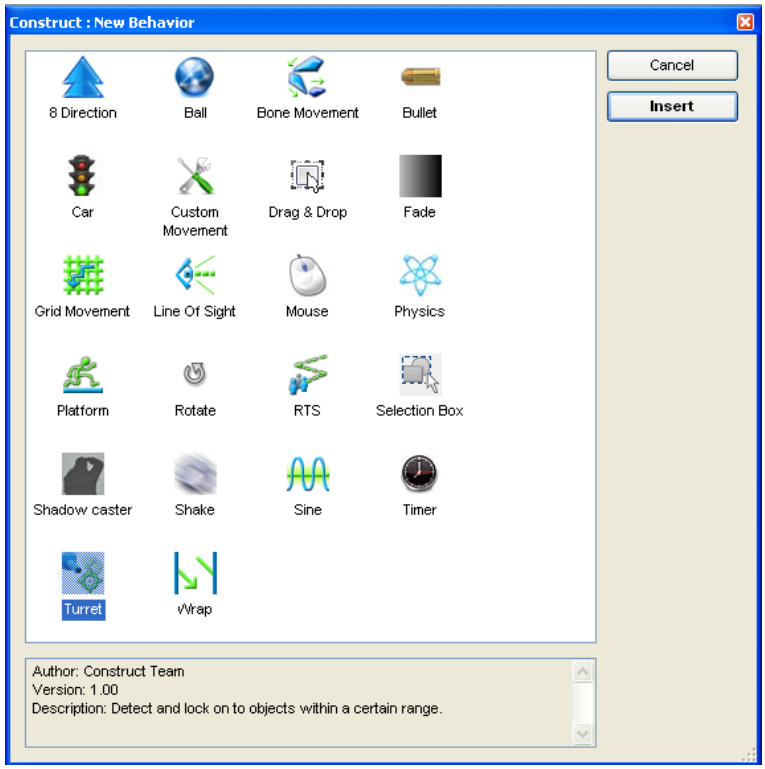


Figure 27 : *Comportements* proposés par *Construct* pour l'objet *sprite*. Le comportement « *Turret* » est décrit : « *Detect and lock on to objects within a certain range* ».

1 Exemple de *tower defense* : http://www.scirra.com/forum/download/file.php?id=372

objet, ce sont les **comportements**. Ce sont des mécaniques souvent spécifiques des jeux. Sur la Figure 27 il y a la liste des comportements pour les *sprites*, notamment le comportement « *Turret* ». Lorsqu'un *sprite* a ce comportement, il se tourne automatiquement vers les cibles qui lui ont été désignées à leur entrée dans son champ d'action. Et dès qu'une latence définie (rechargement de la tour) est écoulée, le comportement déclenche un événement de tir. Les auteurs du jeu peuvent programmer les conséquences de cet événement dans l'interface de programmation événementielle.

Tous les comportements ne déclenchent pas d'événement (par exemple, « *Sine* » visible Figure 27 provoque simplement un déplacement sinusoïdal), mais ils introduisent tous du dynamisme dans les objets auxquels les utilisateurs les rattachent.

C'est l'interface de programmation événementielle qui permet d'attribuer des **actions** aux **événements**. Par exemple de modifier l'animation du *sprite* du macrophage lorsqu'une bactérie passe à sa portée. Le « programmeur » du jeu attribue un comportement *Turret* au macrophage qui permet de déclencher automatiquement un événement « *On shoot* » (tir) lorsque la bactérie est proche. Puis il indique à *Construct* que dès le « tir » du macrophage, l'animation de la phagocytose doit se déclencher (Figure 28). L'ensemble est fait par une suite de formulaires de type *Wizard* (voir

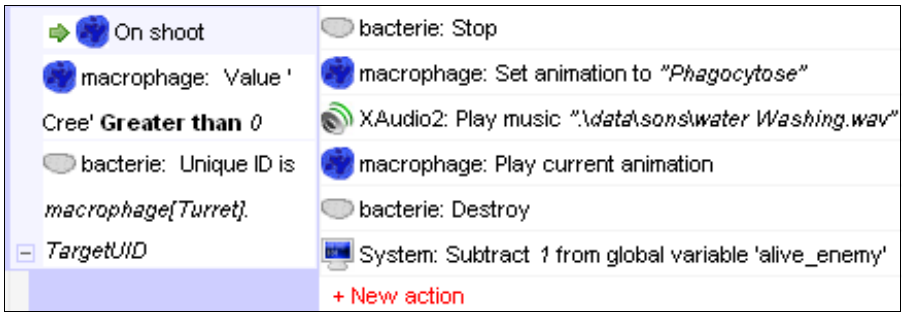


Figure 28 : Programmation de l'événement « *On shoot* » dans *Construct*.

aussi p. 68 l'inspection de l'interface de programmation). Dans un premier temps, le « programmeur » choisit dans une liste l'événement auxquel il veut affecter des actions. Puis il peut affecter des actions les unes après les autres (en cliquant sur « *New action* » visible sur la Figure 28) par le même type de formulaires : choix des objets du jeu sur lesquels se fera l'action, choix des types d'action à faire sur ces objets, paramétrage de ces actions (Figure 29).

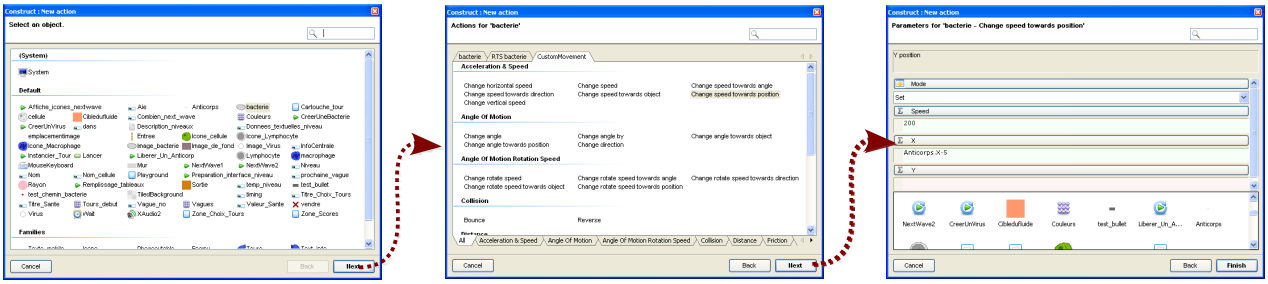


Figure 29 : Les trois étapes pour éditer une action dans *Construct* :
- choisir l'objet sur lequel se fera l'action (ici une bactérie)
- choisir le type d'action sur cet objet (lui donner une vitesse par rapport à une position)
- paramétrer l'action (donner une vitesse de 200 en direction de l'anticorps qui a agglutiné la bactérie). À propos de ce paramétrage voir aussi l'inspection de l'interface d'édition des événements p. 68)

En conséquence de ce guidage très poussé, les erreurs de programmation et les messages d'erreur sont rares. C'est l'accompagnement des actions de l'utilisateur par l'interface qui permet d'éviter les actions erronées et donc ces erreurs. Toutefois, elles ne permettent pas d'empêcher la conception d'une logique du jeu incorrecte, c'est-à-dire qui ne conduit pas à l'effet désiré, voire à des comportements incohérents du jeu qui peuvent même le faire planter.

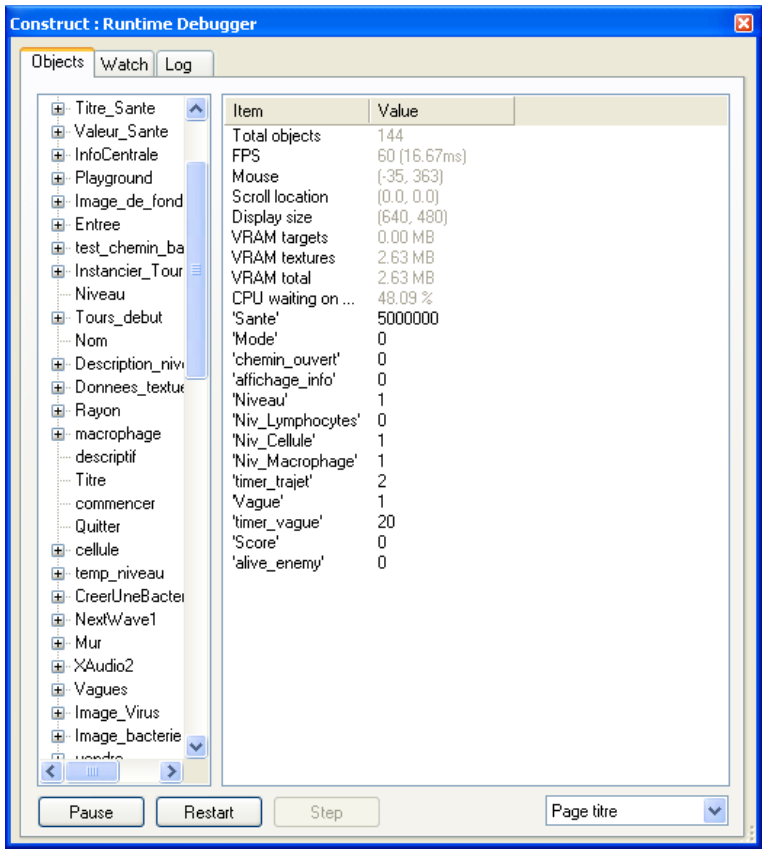


Figure 30 : La fenêtre de débogage de *Construct* donne la liste des pointeurs internes du jeu et leurs valeurs.

Il existe bien dans *Construct* une interface de débogage, elle donne les valeurs des différents pointeurs internes du *runtime* pendant que le jeu s'exécute (Figure 30). Les données fournies par cette fenêtre sont peu exploitables, car très complexes et assez déconnectées de la programmation événementielle. Les utilisateurs préféreront donc se pencher sur le listage des événements et des actions pour y rechercher les problèmes. Et ce n'est pas chose facile, car sur un jeu complexe, les événements sont très nombreux et reposent sur beaucoup de conditions. Or, la lecture d'un programme complexe dans cette interface est difficile et peu pratique (La Figure 26 p. 70 montre une copie d'écran d'un morceau de programme). Il aurait fallu

pouvoir sortir un listage imprimé des événements pour pouvoir analyser globalement le programme et ce n'est pas possible avec *Construct.* Les outils pour identifier et corriger les erreurs ne sont pas adaptés.

Mis à part les problèmes pour débusquer les erreurs, programmer dans cette interface s'avère assez simple. Mais cela n'est pas toujours vrai. Pour que cette programmation reste simple, il y a deux prérequis. D'abord, il faut avoir bien conçu la logique du jeu en amont. L'interface de *Construct* ne facilite pas la conception algorithmique, en particulier parce qu'elle ne propose pas de vue d'ensemble efficace. Les néophytes qui souhaitent se lancer directement dans la réalisation en espérant pouvoir faire cette conception au fur et à mesure ne seront pas guidés pour qu'elle soit cohérente et bien faite. Ils risquent de buter assez vite sur des problèmes de fonctionnement de leur jeu. L'autre prérequis est de ne pas avoir à construire des comportements nouveaux ou des actions et des événements qui ne sont pas dans les bibliothèques de *Construct*. Le logiciel propose énormément de *briques de gameplay* parmi ces bibliothèques de comportements, d'événements et d'actions qui sont très riches. Mais il peut arriver que ces briques ne couvrent pas tous les besoins. Par exemple, il n'existe pas encore de brique pour lire un fichier XML. Il faut donc concevoir la lecture du fichier avec les outils de programmation de *Construct.* Comme ces outils sont plutôt imaginés pour faire interagir des briques de jeu ils se montrent assez peu souples pour d'autres usages : le « programmeur » du jeu est très vite amené à contourner l'interface graphique en éditant du code textuel à la main et en utilisant des fonctions du noyau de *Construct* auxquelles il n'accède pas par l'interface graphique. Dans la Figure 31 il y a le code suivant :

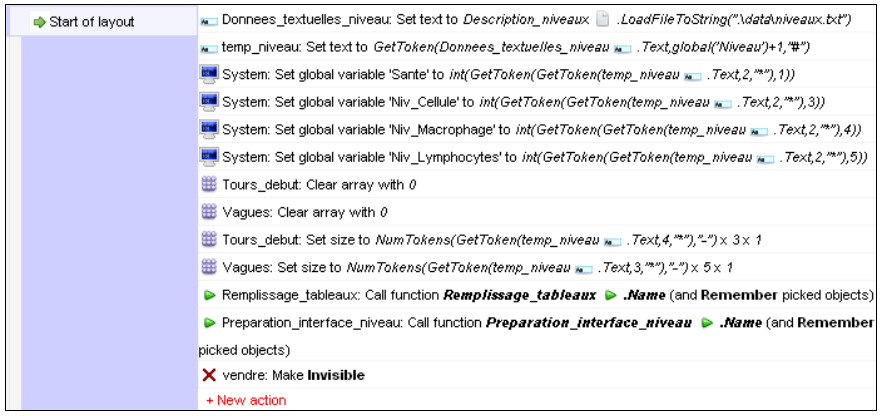


Figure 31 : Exemple de codage textuel dans *Construct* pour pallier aux manques de certaines briques : la lecture d'un fichier de configuration textuel.

« `int(GetToken(temp_niveau.Text,2,"*"),1))` », il s'agit d'une expression qui permet de récupérer le nombre entier situé après la seconde étoile (`*`) de la chaîne `temp_niveau`. Les fonctions

qui y sont écrites `(int` et `GetToken),` documentées dans le forum de *Construct* ne sont pas accessibles par l'interface graphique du logiciel.

L'interface de programmation événementielle est donc très pratique tant que les « programmeurs » utilisent les *briques de gameplay* existantes, qu'ils ont bien pensé leur jeu avant de commencer sa réalisation et qu'ils ne tentent pas de concevoir des gameplays trop élaborés qui demanderont l'édition de code textuel ou l'utilisation des outils de bas niveau de *Construct* (comme les fonctions, les tableaux indexés, etc.)

VII.4- Maintenabilité du logiciel

Construct est un logiciel très jeune. Sa dernière version stable est numérotée 0.99.91 (du 13 juillet 2010) et est considérée comme une *Release Candidate* de la version 1.0. Pourtant le défaut majeur de ce logiciel est sa stabilité, notamment sur les configurations ayant des petites cartes graphiques. Lors de son utilisation, les sauvegardes très fréquentes ont été indispensables.

L'équipe de développement est assez réduite et non professionnelle (les auteurs sont des étudiants), mais le rythme de mise à jour est assez rapide (une version stable tous les trois mois et de nombreuses versions de développement dans l'intervalle). Les travaux de cette équipe sont complétés par des contributeurs qui proposent des greffons (car *Construct* a un *SDK* et de la documentation pour en étendre les fonctionnalités) ou des scripts en *Python* qui complètent les bibliothèques de briques.

Pour conclure sur la maintenabilité de *Construct* qui n'est vraiment pas son point fort, il n'y a pas d'outil prévu pour le déployer et déployer ses mises à jour sur plusieurs postes.

Le bilan de l'évaluation de *Construct* est fait en p. 28.

VIII- Détails de l'évaluation de *MMF 2*

L'outil auteur *Multimedia Fusion Creator 2* (*MMF 2*) est présenté p. 29. Ci-dessous se trouvent les détails de son évaluation selon les critères retenus[1].

VIII.1- Utilisabilité de l'interface graphique et accessibilité du logiciel

Comme *Construct* s'en inspire, l'interface de *MMF 2* lui ressemble beaucoup. Il y a approximativement les mêmes zones : une zone d'édition, un bandeau d'outils contextualisés, un panneau pour les propriétés du jeu et un panneau pour la gestion du projet global (Figure 32).

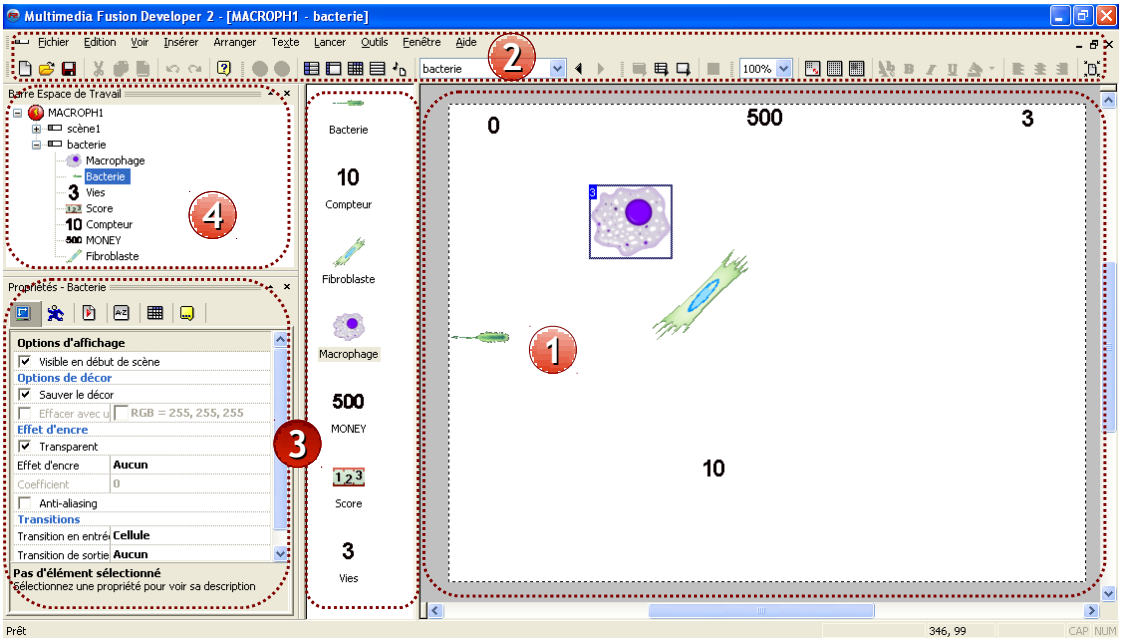


Figure 32 : Aspect général de l'interface de l'outil auteur *MMF 2*. Les pointillés rouges délimitent les zones qui sont numérotées :

- Zone 1 : zone d'édition, ici le canevas des objets graphiques (éditeur de scènes)
- Zone 2 : bandeau des outils (copié-collé, sauvegarde, *undo*, etc.)
- Zone 3 : panneaux des propriétés des objets
- Zone 4 : panneau de gestion du projet

1 Les critères d'évaluation sont présentés dans : « Critères d'évaluation retenus » p. 15.

C'est le *toolkit* d'interface graphique de *MS-Windows* qui est utilisé avec ses éléments d'ergonomie. Par exemple, les panneaux sont déplaçables, et il y a des raccourcis clavier qu'il est possible de personnaliser.

Dans *MMF 2* il y a le même découpage des activités que dans *Construct* qui permet d'alléger le travail des auteurs de jeu grâce à une plus grande concision : un **éditeur de scène** WYSIWYG pour positionner les objets graphiques et un **éditeur d'événements** qui sert à programmer le jeu.

Si l'éditeur de scène est assez comparable au *Layout Editor* de *Construct*[1], l'interface proposée par MMF 2 pour l'éditeur d'événements varie sensiblement. Dans *Construct*, l'insertion de nouveaux événements se fait par une suite de formulaires de type *Wizard* cohérents, où l'information est structurée et lisible[2]. La Figure 33 montre que dans *MMF 2*, l'interface est moins homogène, ce qui alourdit la programmation.

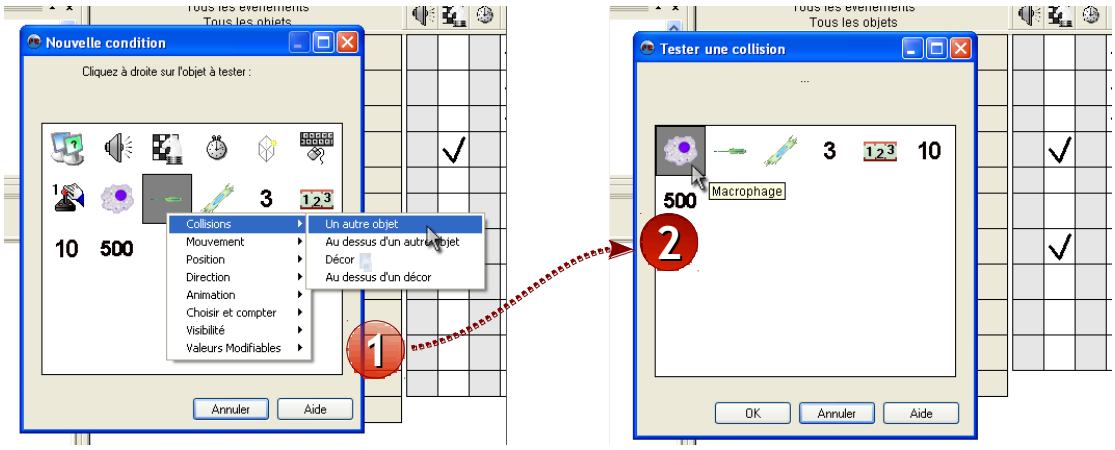


Figure 33 : Exemple de manque de cohérence dans l'interface de *MMF2* : l'insertion d'un événement de collision entre une bactérie et un macrophage.
- La première étape est de choisir l'objet qui déclenche l'événement par clic *gauche* (zone 1).
- Puis, par un clic *droit*, l'utilisateur choisit quel événement : ici la collision (dans le menu dépliant de la zone 1). Et le sélectionne par un clic *gauche*.
- Dans un nouveau formulaire, l'utilisateur choisit par un clic *gauche* l'objet qui interagit dans la collision (zone 2).

Lorsque l'utilisateur travaille sur les actions consécutives d'un événement il est parfois amené à faire des éditions complexes pour lesquelles *MMF 2* le guide sur le même modèle que *Construct*[3] : l'utilisateur néophyte est guidé par le biais d'une combinaison formulaire/menus contextuels comparables à ceux de la Figure 33. L'expérience de l'utilisateur chevronné est prise en compte puisqu'il peut saisir directement l'expression textuelle désirée. Toutefois, la protection contre les erreurs n'est pas automatique et c'est à l'utilisateur de demander la vérification la validité des expressions saisies (Figure 34).

1 Le *Layout Editor* de *Construct* est visible sur la Figure 23 p. 67.
2 Voir la description des formulaires de *Construct* p. 69 et p. 74.
3 Exemple de guidage de l'utilisateur dans *Construct* : Figure 25 p. 69.

Figure 34 : Exemple de *guidage*, *prise en compte de l'expérience utilisateur* et *protection contre les erreurs* (Bastien and Scapin, 1993) dans l'interface de *MMF 2*.

- L'expression « `X Left("Fibroblaste")` » peut être obtenue par des formulaires/menus (Figure 33) en cliquant sur « *Récupérer les données d'un objet* ».
- L'expression peut aussi être saisie à la main par les utilisateurs qui la connaissent.
- Cliquer sur « *Vérifier l'expression* » permet d'afficher qu'il y a une « *Erreur de syntaxe* » (le « + » n'est suivi de rien).

Une fois que les événements et les actions sont insérés en nombre, *MMF 2* propose deux types de visualisation qui permettent d'avoir une vision d'ensemble du programme conçu. La visualisation

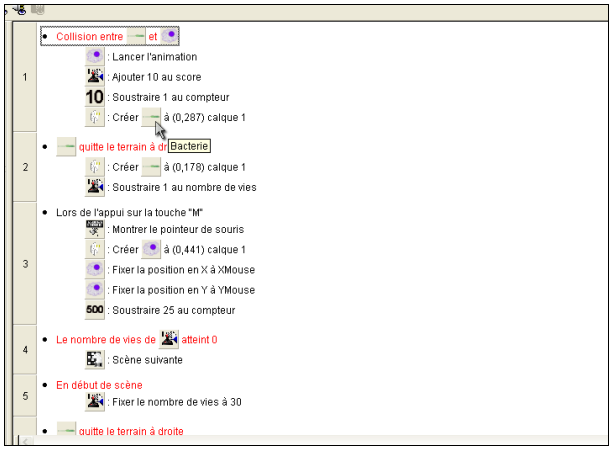


Figure 35 : Présentation des *événements* et des *actions* qui leur sont rattachées sous forme de **liste** dans *MMF 2*. Cette présentation permet de voir le détail des interactions.

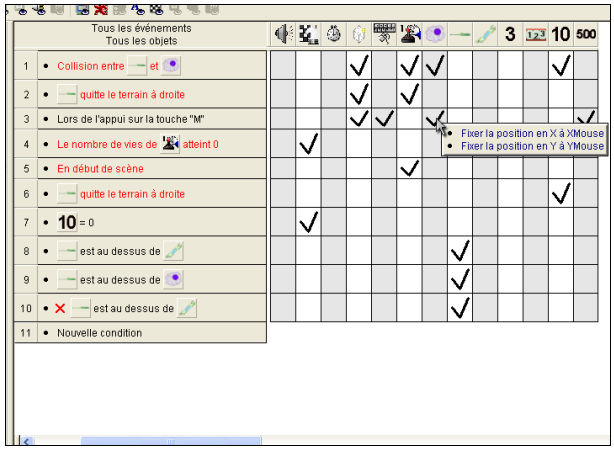


Figure 36 : Présentation des *événements* (lignes) et des *objets* (colonnes) sur lesquels les *actions* se font (présence de coches) sous forme de **tableau** dans *MMF 2*. Cette présentation permet une vision globale des interactions. Un survol des coches par la souris permet aussi de connaître le détail des actions.

sous forme de **liste,** proche de celle de *Construct*[1], permet de voir le détail de chaque action programmée pour chaque événement (Figure 35). La présentation du programme sous forme de **tableau** (Figure 36) met en regard les événements (en lignes) et les objets sur lesquels les actions agissent (en colonnes). Même s'il faut faire apparaître une infobulle pour lire le détail des actions,

1 Visualisation des événements sous forme de liste dans *Construct* : Figure 26 p. 70

cette façon de représenter l'information permet de mieux comprendre le programme dans sa globalité. Il aurait été encore plus agréable de pouvoir regrouper ces événements comme c'est le cas dans *Construct*[1].

Les ressemblances entre *Construct* et *MMF* 2 ne s'arrêtent pas là. Comme son cadet, *MMF* 2 ne permet ni le travail collaboratif, ni la gestion des versions, ni l'interopérabilité. Il repose sur un format de fichiers binaire propriétaire qui n'est lu par aucun autre logiciel et ne permet pas les imports-exports. Par contre, *MMF 2* possède de nombreuses extensions parmi lesquelles certaines permettent de lire et de manipuler des fichiers XML. Le *coût de sortie* d'un projet commencé avec *MMF 2* est donc très élevé. Les auteurs devront refaire toute la programmation. Mais s'ils ont utilisé des fichiers de paramétrage en XML, ils pourront les réutiliser même si leurs développements futurs se font sans *MMF* 2.

VIII.2- Documentation et support

MMF 2 possède de la documentation contextuelle intégrée au logiciel : il y a des infobulles sur beaucoup d'items, et l'utilisateur peut aussi accéder à une aide adaptée à tout moment grâce au raccourci traditionnel `F1`. Logiciel commercial vendu dans une boite, *MMF 2* possède aussi un mode d'emploi papier auquel nous n'avons pas pu accéder en testant la version de démonstration téléchargée. Ce ne fut pas réellement un problème pour prendre en main le logiciel, car la documentation embarquée, accompagnée de son tutoriel suffit pour faire comprendre aux néophytes les tenants et les aboutissants du logiciel.

Sur le web il y a aussi de nombreux autres tutoriels et documentations. Ces informations sont éparpillées sur les nombreux sites des développeurs utilisant *MMF* 2 et sur des sites communautaires. Certains de ces sites sont référencés par l'éditeur de *MMF 2* en particulier sur leurs pages web en anglais[1]. Cette communauté est très active, et l'éditeur tente de la fédérer par des conventions, l'édition d'un magazine[2] ou des pages dédiées sur son site[3]. Bien sûr, il y a aussi des forums, en français[4] et en anglais[5] qui sont tous les deux très actifs.

Dans cette galaxie de sites, il est parfois difficile de trouver l'information car elle manque de centralisation. Grâce aux moteurs de recherche, les utilisateurs pourront toutefois retrouver ces

1 Le site en anglais est beaucoup plus complet que le site français : http://www.clickteam.com/website/usa/
2 http://www.clickteam.com/website/usa/klikdisc
3 http://www.clickteam.com/website/usa/user_sites et http://www.clickteam.com/website/usa/creations
4 http://www.clickteam.com/epicentre/
5 http://www.clickteam.com/epicenter/

pages web. C'est aussi de cette façon qu'ils pourront trouver de nombreux exemples de jeux réalisés avec *MMF 2* avec leurs fichiers sources à étudier[1].

Bien que *MMF 2* soit édité par une entreprise, elle ne semble pas apporter un support spécifique et à l'instar de *Construct* ou *Game-Editor*, ce support se fera par le biais des forums.

VIII.3- Outils de programmation du logiciel

Dans la même veine que *Construct*, *MMF 2* repose sur une **programmation événementielle** et des **briques de gameplay**. Par contre, dans *MMF 2* la notion de *comportement* des objets n'existe pas en tant que telle : les utilisateurs peuvent affecter un ou plusieurs **mouvements** à chaque objet, mais pas d'autres types de comportements. Par conséquent, s'ils souhaitent un comportement complexe du genre *pathfinding* (recherche de chemin) comme il en existe un dans *Construct*[2], ils devront le programmer eux-mêmes dans l'interface d'édition des événements. Heureusement, dans cet exemple précis, ils pourront trouver de nombreuses contributions de *pathfinding* sur internet, y compris dans les packs officiels de l'éditeur. Mais elles restent complexes à utiliser pour les « programmeurs » débutants qui ne maîtrisent pas le logiciel.

Les *briques de gameplay* de *MMF 2* sont donc principalement les types d'**objets** (Figure 37) qui

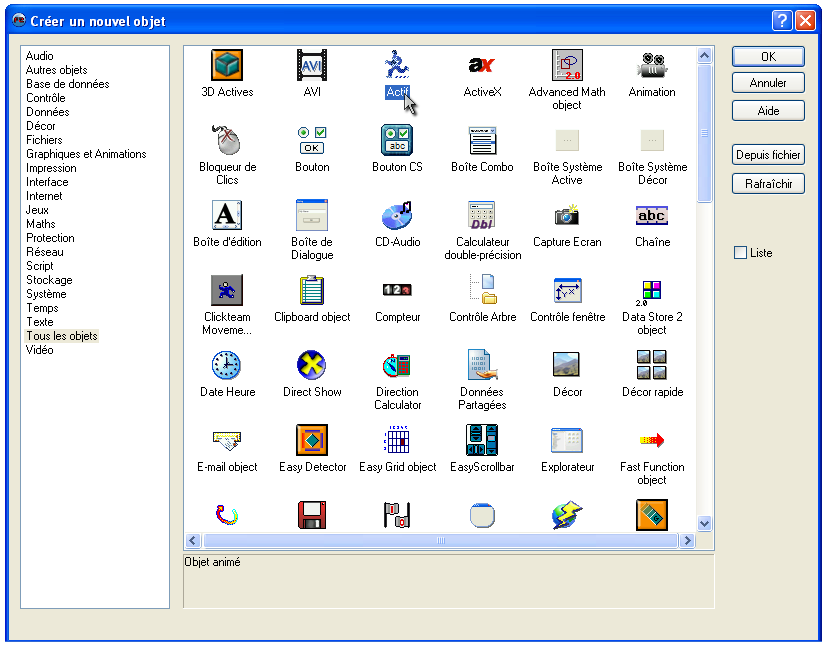


Figure 37 : Liste des objets de *MMF 2*. Bien que cette liste soit très grande dans un jeu du genre *tower defense*, ce sera surtout des objets « *Active* » (qui sont en fait des *sprites*) qui seront utilisés.

sont d'ailleurs beaucoup plus nombreux que ceux de *Construct*, et les types d'**événements** et

1 Sources d'un exemple de jeu *tower defense* : http://www.clickteam.com/epicenter/ubbthreads.php?ubb=showflat&Number=163615

2 « *Grid Movement* » visible sur la Figure 27 p. 72.

d'**actions** sur ces objets. Il y a sur internet de nombreuses contributions qui ajoutent encore à la grande diversité de *briques* proposées par cet outil auteur. C'est vraiment l'une des grandes forces de *MMF 2*.

La méthode de programmation est la même que celle de *Construct* qui s'est inspiré de *MMF 2*[1]. Ainsi, la même logique de programmation pourra être implémentée sans problème dans les deux outils, en tenant compte de leurs différences de *briques de gameplay*.

VIII.4- Maintenabilité du logiciel

Bien que je n'aie pas personnellement pu tester *MMF 2* autant que *Construct*, il semble bien plus stable, car je n'ai eu aucun plantage. *MMF 2* est un logiciel mature commercialisé depuis longtemps et descendant d'une longue lignée d'outils auteurs pour créer des jeux sérieux. Cette longue lignée est aussi un handicap puisque certaines *briques de gameplay* erronées (comme « *mouvement huit directions* ») dans *Klik & Play* (l'ancêtre de *MMF 2*) ont dû être conservées erronées par compatibilité ascendante.

Le bilan de l'évaluation de *MMF 2* est fait en p. 30.

1 Voir la description de la programmation dans *Construct* p. 73.

IX- Détails de l'évaluation de *Game-Editor*

L'outil auteur *Game-Editor* est présenté p. 31. Ci-dessous se trouvent les détails de son évaluation selon les critères retenus[1].

IX.1- Utilisabilité de l'interface graphique et accessibilité du logiciel

Lors de son premier lancement, l'interface de *Game-Editor* semble très dépouillée (un écran noir et une barre de menus). Cet aspect est trompeur, car pour manipuler les propriétés des **acteurs** qui sont

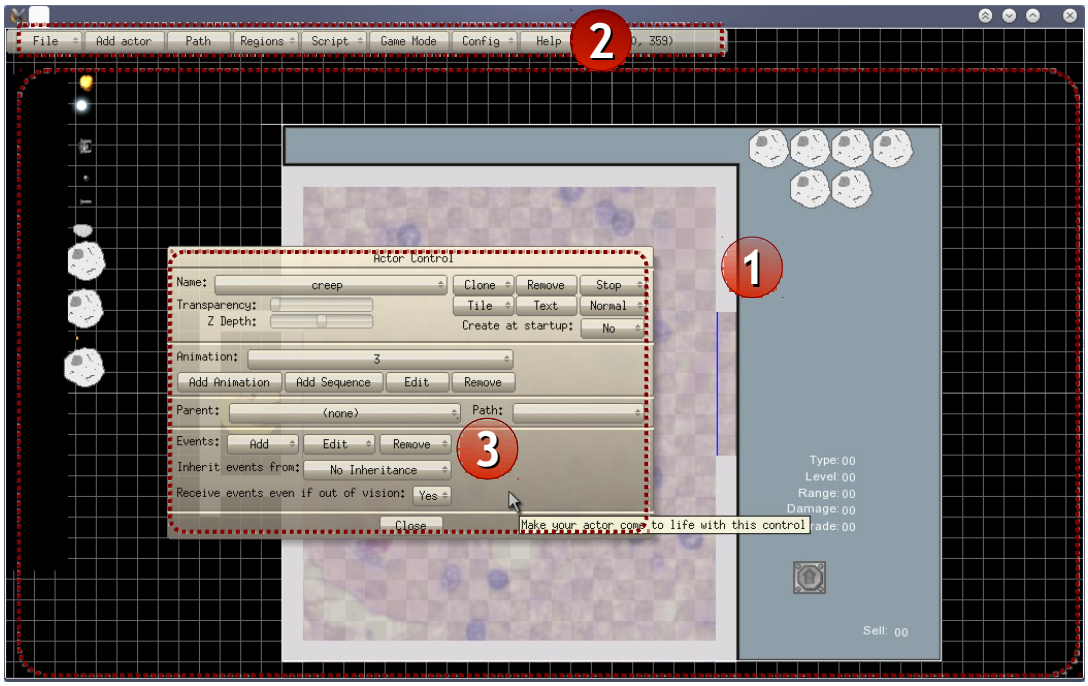


Figure 38 : Aspect général de l'interface de l'outil auteur *Game-Editor*. Les pointillés rouges délimitent les zones qui sont numérotées :

- Zone 1 : zone de manipulation des *acteurs* (objets du jeu)
- Zone 2 : bandeau de menu (non contextuel)
- Zone 3 : fenêtre de l'*Actor Control*. L'ensemble de la programmation se fait à partir de celle-ci. (non contextuelle)

1 Les critères d'évaluation sont présentés dans : « Critères d'évaluation retenus » p. 15.

les objets du jeu, il est nécessaire de passer par la fenêtre d'*Actor Control*. Et elle a au contraire une densité informationnelle très forte.

Une zone de menu permet de faire les actions les plus simples comme charger, sauver, trouver de l'aide, ajouter des acteurs, configurer, etc. Il existe aussi, comme dans *Construct* et *Multimedia Fusion Creator 2*, une zone dans laquelle les auteurs peuvent organiser les objets graphiques (les **acteurs**) à la façon d'un éditeur WYSIWYG. La programmation est entièrement faite dans la fenêtre d'**Actor Control** qui comme son nom l'indique permet de contrôler toutes les propriétés liées aux acteurs : leur **apparence**, leurs liens de **parenté** et leur héritage, les **événements** et les **actions** auxquels ils sont rattachés (Figure 38). Bien que l'accès à l'*Actor Control* se fasse à partir des objets (clic droit, puis *Actor Control*), cette fenêtre n'est pas contextuelle. Elle permet d'accéder à l'ensemble des acteurs du jeu : elle est le moyen central de tout travail de programmation dans *Game-Editor.* L'ensemble des textes de l'interface est en anglais.

Ces choix d'organisation ainsi que le *toolkit* graphique spécifique de l'interface rendent l'ensemble assez difficile à prendre en main. D'autant que certains éléments attendus essentiels, comme les raccourcis clavier (`Ctrl-S`, `Ctrl-C`/`V`, `Suppr`, etc.), ne sont pas implémentés (seul `Ctrl-Z` pour annuler les déplacements des acteurs sur le canevas l'est). En plus d'une trop forte densité d'informations, le regroupement des fonctions de programmation dans l'*Actor Control* est la source de beaucoup de problèmes d'utilisabilité. Le premier, bien visible sur la Figure 39, c'est la lisibilité. Bien que les items soient groupés, il est très difficile de savoir quoi chercher et où chercher. Seuls les utilisateurs les plus expérimentés et qui auront acquis des automatismes sauront utiliser rapidement une telle interface. Il y a notamment une différence fondamentale entre les boutons ornés de l'icône « ▲ » et les autres. Les premiers sont des *menus* et leur utilisation est centrale dans l'*Actor Control*. Par exemple dans la Figure 39, ils donnent accès aux événements déjà programmés pour l'acteur « *creep* » : « *Collision* », « *Create Actor* » et « *Draw*

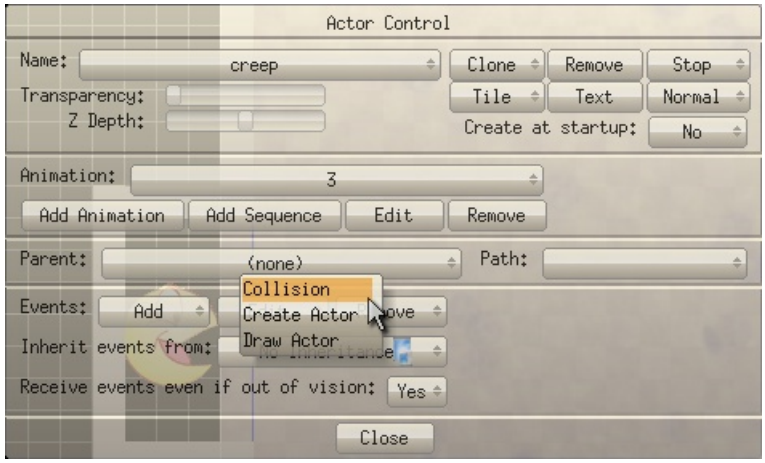


Figure 39 : Détail de l'*Actor Control* de *Game-Editor*. Le *guidage* n'est pas optimal. La *charge de travail* plutôt forte en raison de la *densité informationnelle*. Certains *codes sont peu signifiants*, par exemple le sigle « ▲ » signifie que le bouton est un menu déroulant. La différence entre les deux types de boutons est infime et pourtant fondamentale dans la programmation de *Game-Editor* (Bastien and Scapin, 1993).

Actor ». Les autres boutons (sans icône) peuvent soit provoquer l'ouverture d'une fenêtre de paramétrage fille, soit produire un effet. Et ce, le plus souvent sans confirmation (il n'y a pas de bouton annuler sur la Figure 39). La succession des fenêtres filles peut d'ailleurs être longue pour certains travaux. Par exemple, si l'utilisateur se trouve face à la Figure 39 et qu'il choisit de cliquer sur « *Collision* », il lui sera ensuite proposé un menu pour déterminer s'il veut agir sur la *réponse physique* du moteur ou sur l'*éditeur de script*. S'il choisit l'éditeur de script, il devra dans une nouvelle fenêtre fille paramétrer la collision (comment ?, avec qui ?, combien de fois ?, etc.) et confirmer. Pour avoir enfin accès à la dernière fenêtre fille : celle de l'éditeur de script. L'interface manque donc à la fois de lisibilité, d'organisation, de cohérence et d'affordance. Son utilisation demande par conséquent un apprentissage plutôt difficile, car long.

L'**éditeur de script** (Figure 40) tire un peu son épingle du jeu de l'utilisabilité de l'interface. Il propose une *coloration syntaxique* qui facilite la lecture des scripts complexes. Il utilise un système de *guidage* de l'utilisateur qui permet aussi d'éviter d'avoir à mémoriser toutes les commandes textuelles et toutes les variables du jeu. Cela diminue la *charge de travail*. Enfin, l'**éditeur de scripts** propose un système de chargement/sauvegarde des scripts qui pourront ainsi être réutilisés dans d'autres travaux réalisés avec *Game-Editor*.

Figure 40 : L'**éditeur de scripts** de *Game-Editor*. Les scripts (ici la création d'une nouvelle tour) peuvent être complexes. La **coloration syntaxique** et les **menus** (situés en bas de la fenêtre) augmentent la *lisibilité*, *guident* l'utilisateur et diminuent sa *charge de travail* (Bastien and Scapin, 1993).

Ces chargements/sauvegardes de scripts sont les seules fonctions d'interopérabilité de *Game-Editor*. Comme *Construct* et *Multimedia Fusion Creator 2*, cet outil auteur ne permet aucun échange de ce qui a été construit avec d'autres logiciels. Le *coût de sortie* sera donc assez élevé pour des utilisateurs ayant commencé leur projet avec *Game-Editor*. Ils pourront cependant récupérer les routines conçues dans leurs scripts et d'éventuels fichiers de paramètres dont ils auraient programmé les « parseurs » avec un script dans *Game-Editor* (qui ne possède pas de fonction native pour manipuler les données d'un fichier de configuration).

IX.2- Documentation et support

Game-Editor a une interface peu utilisable, mais sa prise en main reste possible pour un néophyte grâce à son abondante documentation interne. Toute cette documentation est en anglais. Chaque bouton, chaque curseur, chaque menu de *Game-Editor* est accompagné d'une infobulle d'aide qui détaille grandement la fonction sous-jacente (Figure 41). Cette seule documentation sous forme

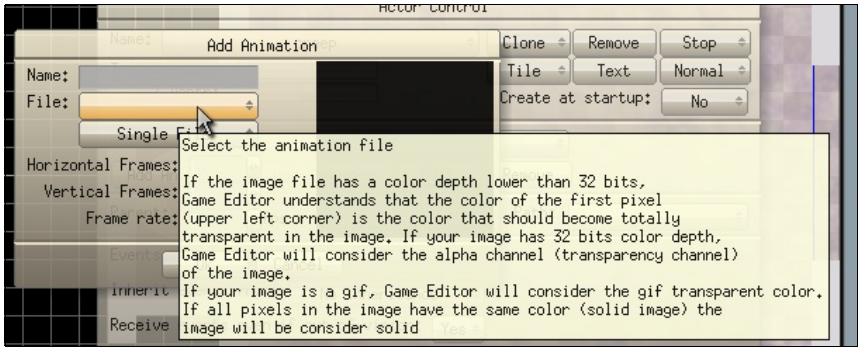


Figure 41 : Les **infobulles** de *Game-Editor* sont très détaillées et compensent en partie les problèmes d'utilisabilité de l'interface. Ici, l'infobulle détaille le choix du fichier contenant l'animation d'un acteur : le format de fichier à utiliser pour concevoir l'animation est décrit dans la bulle.

d'infobulles compense en partie l'aspect abrupt de l'interface. Le logiciel est aussi livré avec des tutoriels accessibles par le menu « *Help* ». Ce sont des *screencasts*, soit des séquences enregistrées (et commentées par des textes incrustés) d'utilisateurs effectuant des actions. Ces tutoriels sont soit des pas-à-pas complets (et trop longs) de réalisation d'un jeu. Soit, plus utiles, des pas-à-pas thématiques qui nous enseignent des méthodes de développement avec *Game-Editor*. Par exemple : Comment gérer une collision, comment suivre un acteur lors du défilement du décor, comment détecter l'utilisation du clavier, etc. Il est indispensable de prendre le temps d'en suivre quelques-uns pour comprendre comment prendre en main ce logiciel. Il semble aussi nécessaire d'accompagner cet apprentissage par la lecture de l'abondante documentation sur le site web de

Game-Editor[1] qui détaille les fonctionnalités et guide[2] les utilisateurs néophytes. Sur ce même site web, se trouvent d'autres tutoriels[3] et de nombreux exemples de jeux[4] que les utilisateurs peuvent étudier.

Game-Editor possède aussi une petite communauté fédérée par le forum officiel du site[5]. Il est la source exclusive de support pour les utilisateurs qui peuvent y poser des questions (quelques membres très actifs répondent vite), s'en servir comme base de connaissances et y trouver d'autres exemples à étudier[6].

IX.3- Outils de programmation du logiciel

Game-Editor propose une programmation événementielle comme *Construct* et *Multimedia Fusion Creator 2*. Chaque **acteur** peut déclencher une série **d'événements** qui déclenchent à leur tour des **actions.** Les **acteurs** ne sont pas typés, mais sont néanmoins bien caractérisés par leur forme (boutons de la zone « *Animation* » sur la vue de l'*Actor Control* Figure 39 p. 84), par leurs mouvements (selon un chemin ou grâce à une brique de *pathfinding A**) et leur parenté qui s'accompagne d'héritages de propriétés et d'événements (Figure 39 p. 84).

Les **événements** sont donc associés aux acteurs (zone « *Events* » de l'*Actor Control* sur la Figure 39 p. 84) et peuvent être hérités des acteurs parents. Certains événements, comme les collisions ou les étapes des mouvements, sont dans les *briques de gameplay* de *Game-Editor*. Mais cela ne représente qu'une dizaine de possibilités. Il est donc souvent nécessaire d'utiliser l'éditeur de script pour en programmer d'autres.

Les **actions** sont dans le même cas. Quelques-unes proviennent des briques de gameplay de *Game-Editor*, les autres devront être réalisées avec l'éditeur de script.

L'**éditeur de script** (Figure 40 p. 85) est donc un élément central de la programmation dans *Game-Editor*. Il n'est pas possible de programmer un jeu sérieux comme *Défenses Immunitaires* sans y avoir abondamment recours. Cet outil auteur ne se met donc pas à la portée des utilisateurs qui ne veulent pas saisir de ligne de code. D'autant que le débogage sera essentiellement un travail de relecture de ce code.

1 http://game-editor.com/Help
2 http://game-editor.com/docs/program_overview.htm
3 http://game-editor.com/Tutorials
4 http://game-editor.com/Games
5 http://game-editor.com/forum/
6 Exemple de jeu *tower defense* dont le fichier source est disponible : http://game-editor.com/forum/viewtopic.php?f=6&t=8183

IX.4- Maintenabilité du logiciel

Game-Editor s'est montré très stable au cours des tests. Développé depuis 2002, ce logiciel semble assez mature pour être utilisé sans risque de perdre tout ou partie de son travail. En conséquence de cette grande stabilité, les mises à jour sont rares et le rythme de développement réduit. C'est sans doute aussi la conséquence de la taille de l'équipe de développement que nous évoquions dans la présentation du logiciel p. 31.

Ne prévoyant pas le travail collaboratif, *Game-Editor* ne permet pas non plus le déploiement sur plusieurs postes.

Le bilan de l'évaluation de *Game-Editor* est fait en p. 32.

Résumé

Ce mémoire fait la synthèse de l'évaluation des outils auteurs de jeux sérieux sans programmation menée dans le cadre d'un stage de Master 2 à l'Université Pierre et Marie Curie.

Cette étude a pour objectif d'explorer les outils auteurs de jeu pour évaluer leur capacité à concevoir des jeux sérieux ou à participer à une ingénierie des jeux sérieux.

Un état de l'art de l'ingénierie des jeux sérieux permet de constater que la conception et la modélisation des jeux sérieux ont déjà été étudiées. Notamment, que les outils auteurs sans programmation sont surtout consacrés à la modélisation des scénarios et missions de jeux sérieux dans des langages dédiés qui seront exécutés par des moteurs préconçus.

En s'appuyant sur les travaux de Murray, l'étude détaille la méthodologie employée pour les évaluations des outils auteurs de jeux : les critères d'évaluation, les étapes de sélection, mais aussi la construction d'un scénario de jeu sérieux destiné à les tester en profondeur.

Les évaluations sont détaillées et permettent de dessiner un modèle général de ces outils. Ils sont fondés sur des briques de gameplay qui implémentent des objets, des comportements, des événements et des actions spécifiques des jeux vidéos, et sur une programmation événementielle dans une interface graphique en manipulation directe. Avec leurs qualités et leurs défauts, il semble que ces outils puissent être utilisés pour du prototypage, la formalisation de spécifications pour des développeurs, ou même la réalisation de moteurs de jeu capables d'exécuter des fichiers de scénario ou de mission conçus par ailleurs.