Chapitre 8

Logiciel de mesure de la difficulté

Sommaire

Dans ce chapitre, nous présentons un logiciel développé pour étudier la difficulté d'un gameplay. Nous y avons implanté le modèle de difficulté ainsi que le modèle de scénario décrits dans les chapitres **??** et **??**. Nous allons tout d'abord présenter l'objectif de ce logiciel, puis son interface graphique. Pour finir, nous décrirons son architecture logicielle.

8.1 Objectif

Nous avons développé ce logiciel afin de pouvoir exploiter les modèles de difficulté et de scénario dans le cadre de différents gameplays. Nous souhaitons à la fois évaluer la faisabilité d'un outil de mesure de la difficulté, et définir plus précisément l'intégration logicielle du modèle de scénario et de calcul de la difficulté. En implantant les différents modèles et algorithmes définis dans les sections précédentes, nous sommes en mesure de les manipuler directement à partir d'une interface graphique, ainsi que de les confronter à des données réelles.

Le logiciel de mesure de la difficulté doit fournir l'interface nécessaire pour :

- spécifier le scénario d'un jeu vidéo, c'est à dire permettre de construire et de manipuler l'hypergraphe de scénario,
- spécifier une méthode de reconnaissance des capacités, de manière à pouvoir calculer l'algorithme `Difficulte ??`,
- calculer et afficher l'état du modèle de scénario en fonction de la lecture d'une trace,
- calculer et afficher la courbe de difficulté d'un gameplay.

Dans la prochaine section nous décrivons tout d'abord l'algorithme général de fonctionnement du logiciel. Par la suite, nous décrirons l'interface graphique du logiciel de mesure de la difficulté. Dans la section suivante, nous en présentons l'architecture logicielle.

8.2 Algorithme général de fonctionnement du logiciel

Nous décrivons à la figure 8.1, l'enchainement des différentes étapes qui permettent de calculer la courbe de difficulté à laquelle chaque joueur s'est trouvé confronté.

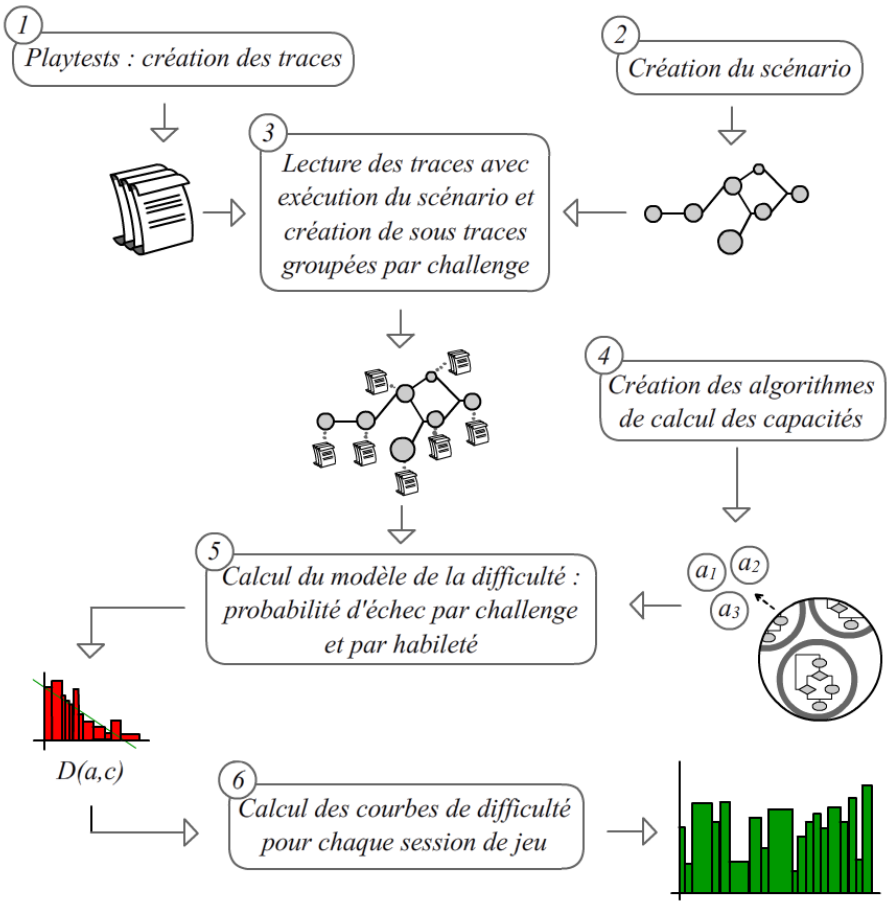

FIGURE 8.1 – Fonctionnement du logiciel

Les étapes 1,2 et 4 de la figure 8.1 sont réalisées par l'expérimentateur. L'étape 1 consiste à recueillir les traces d'évènements qui décrivent les sessions de jeu de plusieurs joueurs. Le moteur de jeu doit donc être pourvu d'un code capable d'enregistrer les évènements utiles et de générer ainsi une trace par session de jeu, c'est à dire un ensemble de fichiers XML que le designer fournit au logiciel d'analyse de la difficulté. L'étape 2 consiste à décrire le scénario du jeu, c'est à dire l'enchainement des différents challenges, à l'aide de l'éditeur d'hypergraphe de scénario du logiciel. L'étape 4 consiste à décrire, pour chaque capacité que le game designer souhaite mesurer, un algorithme capable de déterminer si le joueur tente d'atteindre le but à court terme correspondant, et s'il y parvient. Les étapes 2 et 4 seront plus amplement décrites dans les sections suivantes.

Au cours de l'étape 3, le logiciel va rejouer chaque trace, en recalculant l'état du scénario à chaque nouvel évènement. De cette manière, le logiciel segmente la trace de chaque joueur

en sous-traces, chaque sous-trace correspondant à l'exécution d'un challenge particulier. A l'issue de l'étape 3, chaque challenge est associé à un ensemble de sous traces, qui décrivent les multiples tentatives de validation pour le groupe de joueurs étudié.

L'étape 5 permet au logiciel de calculer le modèle de difficulté. Pour chaque challenge, le logiciel va fournir l'ensemble des sous traces aux algorithmes de calcul des capacités [1]. Chaque sous trace est ainsi notée en fonction de l'habileté montrée par le joueur. Les sous traces sont ensuite regroupées par classe de niveau. A partir de la fréquence de sous-traces aboutissant à un échec, le logiciel détermine la probabilité d'échec pour chaque niveau.

L'étape 6 utilise le modèle de difficulté pour calculer la courbe de difficulté de chaque session de jeu. L'étape 5 a permis de noter les capacités de chaque joueur pour chaque sous trace. Nous pouvons donc connaitre, pour chaque session de jeu, la suite de challenges qu'a tenté le joueur et les capacités dont il a fait preuve. Nous pouvons donc obtenir une mesure plus précise de sa performance, non pas une valeur binaire d'échec ou de réussite, mais la probabilité d'échec en fonction de son comportement. Le logiciel trace donc une courbe de difficulté à partir des probabilités d'échec successives [2].

Dans la prochaine section, nous décrivons de quelle manière le designer peut accomplir chacune de ces étapes à l'aide du logiciel, en présentant plus précisément son interface.

8.3 Interface

La section suivante présente l'interface du logiciel de calcul de la difficulté d'un gameplay.

1. Voir algorithme *Niveau* section **??**. Attention, ce que nous appelons sous-traces ici sont les traces fournies par la fonction *trace*, que l'agorithme *Niveau* sub-divise encore.
2. Voir algorithme *TraceCourbe* section **??**

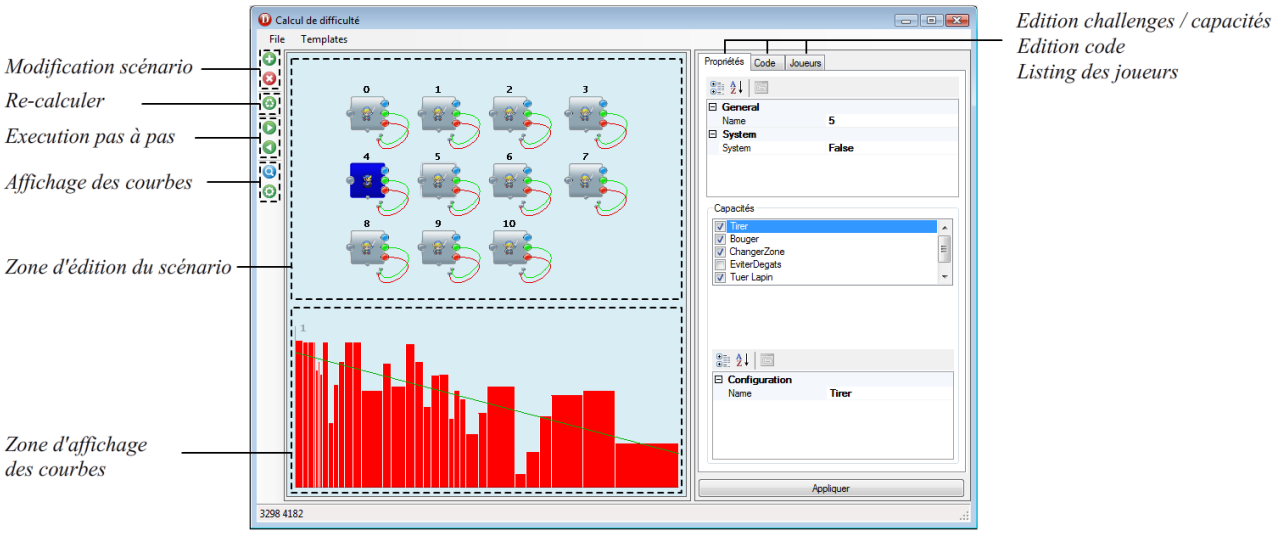

FIGURE 8.2 – Logiciel de calcul de la difficulté

La figure 8.2 présente la fenêtre principale du logiciel. Seule la page d'édition des propriétés des capacités et des challenges est visible, et masque de fait les pages d'édition du code et la liste de joueurs.

8.3.1 Edition du scénario

Le logiciel de calcul de la difficulté permet de créer et d'éditer le scénario d'un jeu vidéo, sous forme d'un hypergraphe. Les challenges sont représentés sous forme de boites rectangulaires (Figure 8.3). En fonction de son état, la couleur du challenge varie : bleu si actif, rouge pour l'échec et vert une fois validé. Les zones circulaires font office de point d'attaches pour les liens entrants et sortants. En particulier, la couleur des zones d'attache des liens sortants permet de connaître l'état testé par le lien, c'est à dire l'étiquette fournie par la fonction λ_L (sec. **??**), à savoir $\{actif, valide, echec\}$.

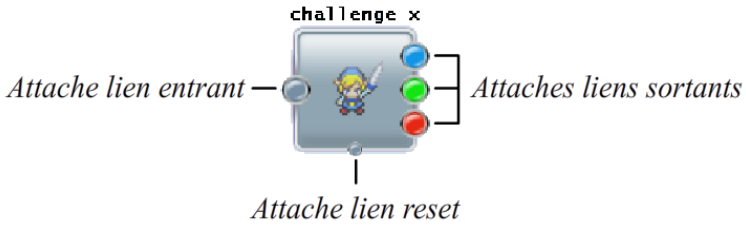

FIGURE 8.3 – Représentation d'un challenge

La couleur de la zone d'attache est reportée sur le lien, et un lien en pointillé indique une négation. La figure 8.4 présente un lien qui n'est satisfait que si le challenge A est valide, et le challenge B n'est pas en échec.

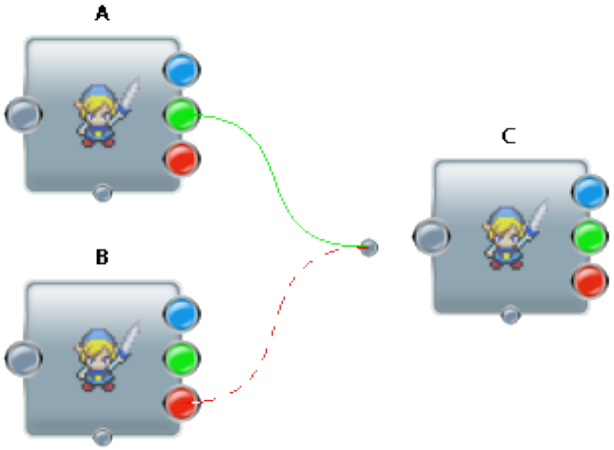

FIGURE 8.4 – Exemple de lien entre challenges

C ne s'activera que si A est valide et B n'est pas en échec.

Les différents challenges peuvent être déplacés et supprimés de même que l'ensemble des liens. L'utilisateur a la possibilité de sélectionner la trace d'un joueur particulier et de l'exécuter en mode pas à pas, de manière à visualiser son parcours.

8.3.1.1 Mise à jour de l'état du jeu

La programmation de la mise à jour de l'état du jeu en fonction des évènements reçus par le moteur de jeu, comme présenté section **??**, est une tâche que doit accomplir le designer et qui dépend de chaque gameplay. Le code de mise à jour des variables de jeu est un code dynamique, qui doit être chargé par l'application à son exécution. Nous avons choisi d'utiliser le langage *lua*, et intégré un petit éditeur de texte dans l'application de calcul de la difficulté. Le listing suivant présente un exemple de code de mise à jour des variables d'état du jeu.

```
1 GArmeJoueur = "";
2
3 // MISE A JOUR ETAT DU JEU
4 // Appel a chaque nouvel evenement
5 // code : code de l evenement
```

```
6  // nbValues : nombre de parametres lies a l evt
7  // values : tableau de parametres lies a l evt
8  function UpdateEtatJeu(code, time, nbValues, values )
9  if code == 11 then
10   GArmeJoueur = values[0];
11 end
12 return
13 end
```

La variable *GArmeJoueur* décrit l'arme qu'utilise actuellement le joueur (préfixée G pour rappeler sa portée globale à l'ensemble du code Lua). Le seul moyen pour le moteur de scénario de connaitre ce choix consiste à recevoir l'évènement numéro 11. Le code du listing se charge de récupérer la bonne valeur dans la trame pour l'assigner à la variable *GArmeJoueur*. Ce code de mise à jour est appelé par le moteur de scénario à chaque lecture d'un nouvel événement de la trace, ce qui permet de maintenir à jour l'état du jeu sous formes de variables Lua, exploitables dans n'importe quelle autre partie du code.

8.3.1.2 Préconditions des challenges

Les transitions entre les différents états d'un challenges sont soumises à diverses préconditions, comme présenté section **??**. Ces précondition sont également codées en lua. Le listing suivant présente par exemple une précondition d'activation :

```
1  // Test d activation
2  // Retourner si le plotpoint peut s activer
3  function IsActif23( )
4
5  if GArmeJoueur == "Rocket Launcher" then
6    return 1
7  end
8
9  return 0
10 end
```

Le challenge ne pourra s'activer que si les conditions propres à ses lien entrants sont satisfaites, mais également si la fonction `IsActif23` retourne la valeur 1, c'est à dire si le joueur est armé du *Rocket Laucher*. Lorsque le designer clique sur un challenge, le code des conditions est chargé dans la fenêtre d'édition pour que le designer puisse le modifier à loisir.

8.3.2 Edition des algorithmes de calcul des capacités

L'étape 4 de l'algorithme 8.1 correspond à l'édition des algorithmes de calcul des capacités. Le designer peut mesurer autant de capacités qu'il le souhaite, qui seront listées sur l'écran principal de l'application, figure 8.2. Pour chaque capacité, le designer doit écrire un algorithme, chargé de comptabiliser le nombre de fois où le joueur tente d'atteindre le but à court terme associé à la capacité, ainsi que le nombre de fois où il y arrive effectivement. Le listing suivant présente un exemple de code de calcul d'une capacité, ici la capacité de viser précisément.

```
1  //-----------------------------------------------
2  //-- VARIABLES
3  //-----------------------------------------------
4  local nbReu = 0;
5  local nbTry = 0;
6  local nbTryPending = 0;
7  local lastTime = 0;
8
9  //-----------------------------------------------
10 //-- RECONNAISSANCE
11 //-----------------------------------------------
12 //-- Nouvel event
13 //-- Donne un nouvel event de la trace.
14 //-- Retourne nombre de nouvelles tentatives puis de nouvelles
       reussites
15 //-- code : code de l evenement
16 //-- nbValues : nombre de parametres lies a l evt
17 //-- values : tableau de parametres lies a l evt
18 function NewEvent1(code, time, nbValues, values )
19 if code == 32 and (values[0] == "True" or values[1] == "True") then
20   nbTry = nbTry+1;
21   nbTryPending = nbTryPending +1;
22   lastTime = time;
23 end
24
25 if code == 02 and nbTryPending > 0 then
26   nbTryPending = nbTryPending -1
27   nbReu = nbReu+1;
28 end
29
30 if time - lastTime > 5 then
31   nbTryPending = 0;
```

```
32 end
33
34 return nbReu ,nbTry ;
35 end
36
37 //-- Restart
38 //-- Raz de la detection de capacites
39 function ResetStrat1( )
40 nbReu = 0;
41 nbTry = 0;
42 nbTryPending =0;
43 lastTime = 0;
44 end
```

L'évènement 32 correspond à un tir du joueur, et ses deux valeurs correspondent au fait que lorsque le joueur tire, il peut soit voir un ennemi, soit être pris en chasse par un ennemi. Ceci nous permet de ne pas tenir compte des tirs que le joueur effectue au hasard, pour le simple plaisir d'utiliser son arme, mais tient uniquement compte des tirs utiles. L'évènement 2 correspond à un dégât subit par un joueur ennemi. L'algorithme évalue ainsi le nombre de tirs utiles du joueur, ainsi que le nombre d'impacts correspondants à ces tirs.

Le designer doit écrire, pour chaque capacité qu'il souhaite mesurer, l'algorithme permettant de compter réussites et tentatives, à partir de la suite d'évènements d'une trace. Lua est un langage de script simple et efficace pour cette tache, directement accessible à partir de l'éditeur intégré à l'application. Le designer doit uniquement déclarer ses variables et remplir le corps des trois fonctions : leur prototype et les commentaires sont automatiquement générés à la création du challenge, afin de minimiser et de cadrer au maximum cette partie du design.

Une fois l'étape 5 de la figure 8.1 atteinte, le designer est en mesure de visualiser les probabilités d'échec en fonction des capacités du joueur. La figure 8.5 montre un exemple calculé par le logiciel à partir de données d'expérience, pour la capacité spécifiée précédemment, c'est à dire la précision de tir du joueur. On remarque que plus le joueur est précis, plus sa probabilité d'échec diminue.

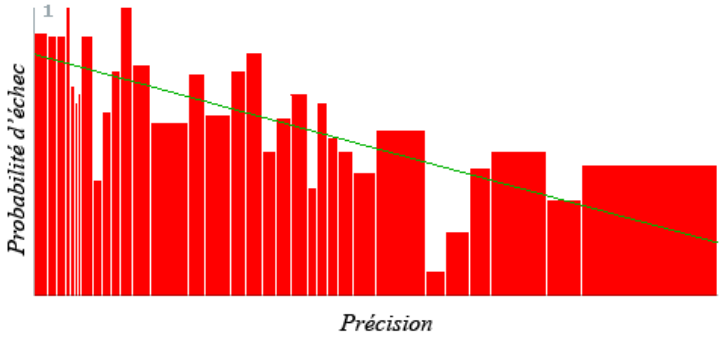

FIGURE 8.5 – Probabilité d'échec en fonction de la précision au tir.

En abscisse, la précision du joueur (probabilité de toucher la cible), et en ordonnée, la probabilité d'échec. La droite verte en est la régression linéaire ($r = -0.72$, $p = 0.000002$)

Dans notre logiciel, nous modélisons la probabilité d'échec en fonction des capacités par la droite de régression linéaire calculée sur les données. Le coefficient de corrélation linéaire nous permet ainsi d'évaluer la qualité de cette régression, et de mesurer la force du lien qui relie la mesure de capacité à la probabilité d'échec du joueur. Le coefficient de corrélation nous permet de classer une capacité : plus sa valeur absolue est élevée et plus la mesure de capacité permet de prédire la difficulté du jeu. De même le coefficient de corrélation ainsi que sa significativité nous permettent de déterminer si le modèle réussit à prédire la difficulté du jeu : nous rejetons systématiquement les corrélations inférieures à 0.3 ou dont la significativité dépasse 0.1 [3].

8.3.3 Courbe de difficulté

Une fois l'étape 6 de la figure 8.1 atteinte, le designer est en mesure d'étudier les courbes de difficulté de chaque session de jeu. A partir de la liste des joueurs, le designer peut consulter l'ensemble des courbes tracées et des mesures de capacités de chaque joueur. Le calcul de la difficulté est réalisé à partir de la moyenne des difficultés prédites pour chaque capacité, pondérée par la valeur absolue de leur coefficient de corrélation. Toute capacité dont le coefficient de corrélation est inférieure à 0.3 ou la significativité supérieure à 0.1 sont écartées du calcul. Si aucune capacité n'est suffisamment significative, cette partie de

3. Nous partons du principe, dans ce prototype, qu'un modèle linéaire suffit à décrire la relation entre capacité et difficulté. Mais d'autres modèles peuvent être envisagés, comme par exemple un réseau de neurone entrainé sur la moitié des données et évalué sur l'autre.

la courbe n'est pas tracée et elle est remplacée par un fond rouge, comme le montre la figure 8.6.

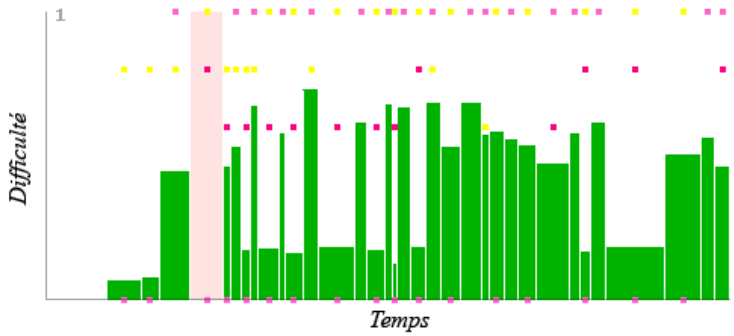

FIGURE 8.6 – Courbe de difficulté.

Les parties en rouges correspondent à des challenges pour lesquels le moteur ne peut prévoir le difficulté, par manque de valeurs ou parce qu'aucune capacité ne décrit précisément la difficulté du jeu.

Sur la courbe de difficulté sont reportées diverses informations supplémentaires. Nous avons incorporé, dans nos traces, différentes valeurs issues de questionnaires soumis au joueur à la fin de chaque challenge. Les points jaunes indiquent ainsi le niveau de plaisir rapporté par le joueur, les points magentas la difficulté rapportée par le joueur. Les points roses indiquent le résultat de chaque challenge.

8.4 Synthèse

Dans ce chapitre, nous avons présenté le prototype d'un logiciel de mesure de la difficulté. Ce logiciel permet de réaliser diverses tâches, représentées graphiquement figure 8.1. Le designer dispose d'un éditeur de scénario, qui lui permet de modéliser la suite de challenges que le joueur peut rencontrer, ainsi que la logique qui décrit leur écoulement. Une fois cet outil intégré à un moteur de jeu, cette description pourrait être intégrée plus finement au reste du code du moteur afin de faciliter le travail du designer. Le game designer dispose ainsi d'une représentation graphique de l'enchainement des challenges du jeu, qui lui permet de visualiser le parcours du joueur et d'accéder rapidement et intuitivement aux différentes mesures réalisées.

L'intégration du langage de script Lua permet au designer de décrire à la fois le fonctionnement de l'hypergraphe de scénario ainsi que les algorithmes de calcul des capacités du

joueur. A partir de ces algorithmes, le logiciel est en mesure de mesurer la difficulté, c'est à dire la probabilité d'échec en fonction des capacités du joueur. Notre modèle de difficulté permet, dans une dernière étape, de construire la courbe de difficulté de chaque session de jeu.

Ce logiciel pourrait être amélioré de diverses manières. Par exemples, la logique du scénario décrite par le designer est ici sous exploitée. La logique de déroulement du scénario dépend en effet de deux types de variables : les variables du modèle de scénario, entièrement calculées par ce dernier, et les variables décrivant l'état du jeu, dont la valeur n'est pas prévisible. L'état des conditions de validation et d'échec des challenges peuvent être prédites à l'aide du modèle de difficulté, et l'état des liens est entièrement calculé par le moteur. Seules les conditions d'activation des challenges peuvent s'appuyer sur des variables d'état du jeu dont la valeur reste imprévisible. En conséquence, si le designer construit un scénario sans utiliser de conditions d'activation, *le déroulement du scénario devient entièrement prévisible à partir du calcul de satisfaction des liens et du modèle de difficulté*. Un tel scénario permettrait de prévoir les parcours les plus probables de certains joueurs, par exemple en fonction de leur capacités, et par conséquent la courbe de difficulté la plus probable dans ce cas. Ce type de fonctionnalité exploite pleinement la logique de scénario afin de fournir au designer des informations particulièrement utiles.

En l'état, ce prototype offre néanmoins une première vision instructive de la difficulté d'un gameplay et de son lien avec le comportement du joueur. Ce logiciel nous a permis de réaliser diverses expérimentations, que nous présentons dans le chapitre suivant.

Bibliographie