

Modélisation et optimisation des décisions dans la conception de jeux

Introduction

Cet article est le premier d'une série sur l'application de techniques de modélisation et d'optimisation des décisions dans la conception de jeux. Il présente le parcours de la conception d'un jeu et précise les endroits où ces techniques de modélisation et d'optimisation se révèlent utiles.

De la recherche, pas d'itérations !

La plus grande partie de la conception d'un jeu (*game design*) est un processus de *recherche*. Le concepteur évalue un grand nombre de configurations possibles afin de résoudre un problème de conception – déterminer la manière de connecter les pièces d'un donjon, les caractéristiques et capacités des différents types d'agents du jeu, les « nombres magiques » qui gouvernent l'efficacité des unités dans un système de combat, voire la combinaison de caractéristiques que le jeu aura dès le début.

Tout comme un personnage dirigé par une intelligence artificielle utilisera un système de recherche de chemin pour naviguer dans le monde du jeu, la conception implique une certaine navigation dans un espace abstrait constitué de toutes les configurations possibles en prenant une configuration de base et en la modifiant de manière itérative : le concepteur regarde attentivement une partie de la conception – le système de combat, une partie du monde, un arbre des technologies dans un jeu de stratégie, etc. – et tente de trouver une manière de l'améliorer en changeant cette configuration.

Les concepteurs aiment utiliser le terme « itération » pour décrire ce procédé, mais « recherche » le décrirait mieux. La vérité est que, lors de l'« itération », le concepteur procède à des expériences avec le jeu en cours de développement. En utilisant son expérience, il effectue de petites modifications qui transformeront la configuration actuelle en une nouvelle qui – tout du moins, il l'espère – remplira mieux les critères.

Ces « itérations » ne ressemblent généralement pas aux changements linéaires qui apparaissent dans les « itérations » propres au code informatique ; elles ressemblent bien plus à une recherche à travers un labyrinthe, avec beaucoup de tournants brusques et, parfois, des retours en arrière. Ces déplacements mènent plus près de l'objectif, mais sans donner l'impression franche d'avoir amélioré le jeu. Parfois, une décision qui devait améliorer le jeu a des défauts qui n'étaient pas prévus, il est alors nécessaire de revenir en arrière.

La conception de jeux est une discipline extrêmement difficile : la situation est similaire à une pièce sombre, remplie d'objets pointus, à travers laquelle il est extrêmement difficile de naviguer sans danger à l'écart des sentiers battus. Le voyageur subit presque toujours les mêmes douloureuses blessures, en particulier s'il se déplace trop vite. Malheureusement, il dispose de peu d'outils pour éclairer cette pièce sombre, de très peu de techniques bien définies et disciplinées pour effectuer cette recherche.



Cette pièce noire est la raison pour laquelle il est nécessaire d'« itérer » : il n'est pas toujours possible de connaître les ramifications d'une décision tant qu'elle n'est pas tentée. En d'autres mots, le concepteur doit **chercher** (même Will Wright en parle, en faisant référence à une « recherche dans un espace de solutions », au cours de sa conférence à [la GDC 2004](#)).

Par conséquent, la phase de conception limite fortement la productivité de l'équipe de développement et est une source majeure d'imperfections : elle concentre les risques pour le développement d'un jeu. Un très grand nombre d'équipe se trouve limitée par de mauvaises décisions de conception, des déferlements de créativité non concertés, de lents glissements dans les fonctionnalités, une mauvaise perception du marché cible ou d'autres problèmes concernant la conception qui ont mené à des problèmes de qualité du produit.

Avec tous les dangers qui rôdent lors de l'expérimentation de la conception du jeu, il n'est pas étonnant que de nombreux éditeurs et développeurs s'opposent au risque sous toutes ses formes, en préférant explorer les zones proches de genres, de licences, de conventions bien établies et explorée en long et en large, plutôt que de courir les risques si bien connus de l'innovation avec des retours relativement inconnus. Explorer la pièce sombre leur semble simplement trop risqué.

Comment changer cette attitude ? Au lieu de simplement *éviter* l'innovation, il serait mieux de trouver des méthodes pour améliorer les compétences de conception, de construire des outils pour que l'innovation soit plus sûre et plus efficace

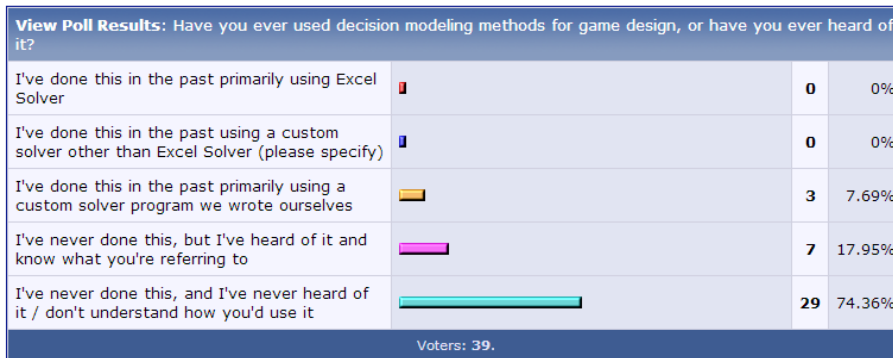
À propos de cette série

Cet article est le premier d'une série qui présentera la *modélisation des décisions*, un ensemble d'outils pour décomposer les décisions en des modèles formels qui peuvent être explorés pour trouver la meilleure solution.

La modélisation et l'optimisation des décisions sont des techniques fréquentes dans le *management*, la finance, la planification avancée de projets et bien d'autres domaines encore, afin d'améliorer le processus décisionnel. Ces techniques impliquent de résoudre les difficiles problèmes de décision et d'optimisation en recherchant les solutions possibles bien plus vite que ne le pourraient des humains.

Malgré tous les bénéfices potentiels, la modélisation et l'optimisation des décisions semble relativement inconnue parmi les concepteurs de jeu de l'industrie. Un sondage récent sur un forum populaire a montré qu'à peine vingt-cinq pour cent des participants avaient entendu parler de la modélisation des décisions : à peine huit pour cent l'avaient déjà utilisée. Une étude similaire sur

Facebook, envoyée directement aux concepteurs, a donné des résultats presque identiques avec un nombre proche de répondants.



Utilisée correctement, la modélisation des décisions peut améliorer de manière significative un certain nombre d'aspects du processus de conception :

- elle peut aider à optimiser la configuration de systèmes spécifiques ou à trouver les valeurs optimales des paramètres du jeu ;
- elle peut apporter un éclairage sur des décisions comme les combinaisons de fonctionnalités à inclure dans le jeu ;
- elle peut aider à modéliser les décisions qu'un joueur peut prendre, particulièrement en termes d'identification des stratégies dominantes.

Cette série fournira des exemples dans chacune de ces trois catégories.

Définition

Qu'est-ce que la modélisation des décisions ? En des termes simples, **la modélisation des décisions est le processus de simulation d'une décision et de recherche automatique de sa solution.**

Ainsi, la première étape est de définir une sorte de décision de conception du jeu, de tenter de prendre en compte tous les facteurs d'intérêt qui l'influencent, puis de construire un modèle qui représente de manière précise la décision, en définissant des variables d'entrée (de *décision*) et une seule variable de sortie. Ensuite, un algorithme se charge d'effectuer une *recherche* pour les valeurs optimales des variables de décision qui produisent la meilleure sortie.

Si tout se passe bien, ce procédé permet d'explorer bien plus de solutions possibles qu'à la main ou par l'imagination. Il ne peut cependant pas s'appliquer à tous les cas : cette technique est appropriée pour une série de problèmes, mais pas tous. Pour ceux-là, le résultat est souvent bien meilleur, arrive plus rapidement et, dans certains cas, donne une solution à des problèmes qui, autrement, resteraient insolubles.

Tout au long de la série, des ensembles de *contraintes* seront définis : elles agiront en barrières qui s'assurent que le modèle reste valide. Ces contraintes peuvent limiter la plage de valeur ou le temps des variables d'entrée ou n'importe quel autre aspect du modèle.

Pourquoi construire des modèles ?

Les joueurs de [Sid Meier's Civilization](#) se sont probablement faits la réflexion suivante :

« Un instant... Quelle est *la* bonne manière de démarrer une ville ? D'abord construire un monument, puis un grenier ? Ou bien le grenier d'abord ? Ou peut-être le temple en premier, puis le grenier ? Quelle est la *meilleure* décision ? Existe-t-il seulement une manière de répondre à cette question ? »

Des pensées similaires peuvent venir à l'idée d'un joueur de jeu de stratégie en temps réel. Équilibrer les paramètres des différentes unités dans ce type de jeu est particulièrement difficile. Il serait très intéressant d'avoir un système pour accélérer la résolution de ce problème d'équilibre, capable de répondre à des questions sur le système de combat sans devoir jouer une seule partie. Et s'il était possible de poser à ce système des questions comme « Combien de spadassins faut-il pour infliger une défaite à deux piquiers et trois archers ? » ou encore « Quelle est la combinaison la moins chère d'archers et de catapultes pour mettre à terre une tour de garde ennemie ? ».

En fait, c'est possible !

En modélisant ces problèmes de la bonne manière, des outils automatiques d'optimisation pourront chercher la meilleure réponse qui satisfasse à ces critères, *sans* devoir jouer des milliers de parties.

Voici un exemple d'un problème du même genre, un exemple qui sera réutilisé dans les futurs épisodes de cette série : un jeu nommé SuperTank. Le joueur contrôle un gigantesque tank, rempli de technologie de pointe, avec lequel il doit défaire d'autres supertanks. Avant chaque bataille, il peut sélectionner la combinaison exacte d'armes pour équiper son tank.



Le joueur possède cent crédits qu'il peut dépenser sur son armement. Son tank peut transporter au plus cinquante tonnes d'armes et dispose de trois « emplacements critiques » où il peut installer des armes spéciales à haute puissance.

Le jeu propose cinq types d'armes, le joueur peut en sélectionner autant qu'il veut de chaque type et même passer certains équipements.

	Dommages	Poids	Coût	Emplacements critiques
Mitrailleuse	2	1	5	0
Roquettes	8	3	12	0
Mégaroquettes	15	10	16	1
Laser	7	4	9	0
Ultralaser	20	16	18	1

Petit problème : comment équiper le tank pour causer le plus de dommages possible (en supposant qu'il s'agit de dommages causés à l'ennemi par seconde, peu importe la vitesse de tir) ? Par hypothèse, toutes les armes auront la même portée, le même arc de tir, la même précision, de telle sorte qu'elles sont toutes identiques à tout point de vue, à l'exception des données du tableau ci-dessus.

Vite ! Combien de mitrailleuses, de roquettes, de lasers devraient équiper le tank ? Quelle combinaison d'une ou plusieurs armes inflige le plus grand dommage sans dépasser le budget, le poids total et le nombre d'emplacements critiques disponibles ?

Essayez de résoudre ce problème à la main, voire avec une calculatrice.

Alors ?

Si vous avez essayé, vous avez pu remarquer que ce problème est particulièrement difficile, bien plus qu'attendu. Il est probablement possible de s'en sortir à coups d'équations alambiquées, mais ce n'est pas le domaine de prédilection des concepteurs de jeux vidéo.

D'ailleurs, comment évoluerait la solution pour d'autres paramètres ? Que se passerait-il si le tank pouvait porter dix tonnes supplémentaires (soit un total de soixante tonnes) ? Si le joueur avait cent dix ou nonante crédits à dépenser ? S'il avait quatre emplacements critiques ou seulement deux ?

Imaginez un instant un système qui pourrait calculer instantanément la combinaison d'armes qui cause les plus grands dommages pour n'importe quelles valeurs de ces trois paramètres (poids total, crédits disponibles, nombre d'emplacements critiques) : il suffirait d'introduire les données des armes, les trois caractéristiques du tank pour obtenir, directement, la meilleure manière de l'équiper. Ne serait-ce pas magique ?

Un tel système pourrait donner des réponses immédiatement à une série de questions fort utiles pour la conception du jeu.

- Comment évolue la charge optimale en modifiant les paramètres du tank ? Ceux des armes ?
- Quels dommages peut, au plus, infliger un tank pour des paramètres donnés ?
- Les quatre paramètres de charge arme sont-ils appropriés et équilibrés ?
- Y a-t-il des armes trop puissantes et utilisées trop fréquemment ? Si une arme est tellement puissante que la meilleure décision l'utilise *toujours*, alors l'embarquer n'est pas réellement un choix. Dans ce cas, il faudrait probablement retirer l'arme du jeu ou la rééquilibrer de telle sorte qu'il existe des cas où elle n'est *pas* utile.

Commented [TC1]: Puisque l'hypothèse est supposée, la subordonnée est vraie, ce qui indiquerait plutôt l'indicatif. Je préférerais cependant un avis plus autorisé.

- Y a-t-il des armes sous-utilisées, rarement ou jamais utilisées ? Comme précédemment, si une arme est tellement inutile que, peu importe le cas, la meilleure décision est de ne *pas* l'utiliser, cette décision n'a pas vraiment de sens non plus. Dans ce cas, il faudrait soit retirer l'arme du jeu, soit la rééquilibrer pour qu'il existe des situations où elle est utile.

Il s'agit de décisions très importantes sur la conception du jeu, toute personne impliquée devrait vouloir la réponse à ces interrogations : ces résultats sont extrêmement intéressants pour équilibrer SuperTank.

Quelques paragraphes ont suffi à décrire un problème remarquablement difficile à résoudre manuellement, mais les outils de base d'Excel peuvent le résoudre très facilement. D'ailleurs, la construction d'un modèle de décision pour ce problème sera l'objet d'un prochain article, où les questions précédentes trouveront leur réponse : cette construction prendra à peine quelques minutes, malgré la difficulté du problème ; avec un peu de travail, il devient possible de créer un outil puissant pour explorer l'espace de conception.

Feuille de route

Cette série illustrera quelques autres exemples plus compliqués et fournira également quelques feuilles de calcul de référence, afin de répéter les manipulations sans utiliser autre chose qu'Excel. Parmi ces exemples, notamment :

- un exemple de combat, relativement simple, pour un jeu de stratégie ;
- un modèle pour optimiser les coordonnées de plusieurs téléporteurs dans un jeu spatial massivement multijoueur, en prenant en compte tant les positions de chaque trou que de la répartition de la population ;
- un modèle pour déterminer le niveau de taxe dans une ville simplifiée, afin d'équilibrer le niveau de bonheur de la population par rapport aux taxes et les revenus, dans un jeu de type 4X comme *Sid Meier's Civilization* ;
- un modèle pour choisir la manière d'assigner des sorts et d'autres caractéristiques à des classes de personnages dans des jeux massivement multijoueurs ;
- un modèle d'optimisation pour déterminer l'ordre optimal des constructions sur une planète colonisée dans un jeu de stratégie de type 4X, comme le classique *Master of Orion* ;
- un exemple de choix de fonctionnalités pour un jeu en utilisant un modèle de décision pour effectuer le meilleur compromis.

En général, les exemples commenceront par chercher des stratégies optimales pour un joueur dans un sous-système spécifique du jeu, avant de progresser vers des modèles de décision qui aident à optimiser les paramètres pour tous les systèmes d'un jeu et les combinaisons de fonctionnalités.

Dans chaque cas, le problème sera décrit et modélisé dans Excel, avant une résolution avec le solveur inclus. La feuille de calcul sera aussi disponible pour chaque exemple, pour faciliter les tests et l'expérimentation sur chacun des modèles.

Cependant, la représentation sous-jacente, que ce soit une feuille de calcul ou du code dans un langage de programmation, n'a *aucune* espèce d'importance : le point important n'est pas de résoudre le problème avec le solveur d'Excel ou en C, C++, Java ou autre, mais bien la manière de modéliser la situation.

Pourquoi utiliser des modèles de décision ?

Certains lecteurs pourraient, à présent, se montrer incrédules : construire des modèles de décision semble requérir beaucoup de travail, pourquoi le faire quand des joueurs seront prêts à tester les différentes décisions sur des préversions du jeu ?

Tout d'abord, il est important de remarquer que **la modélisation des décisions ne s'applique pas à tous les problèmes**. Certains sont trop compliqués, trop difficiles à modéliser avec ces techniques ; d'autres aspects de la conception, comme l'esthétique ou l'amusement, sont difficiles voire impossibles à modéliser avec des nombres. La modélisation des décisions n'élimine *pas* d'emblée les phases de test externe ou interne.

Cela dit – et ce devrait être clair à la fin de la série –, la méthode de modélisation et d'optimisation des décisions donne aussi un ensemble d'outils unique et très puissant, qui peut, en tout ou en partie, résoudre bien des problèmes qui ne pourraient pas trouver de solution autrement. Ces outils peuvent fournir des réponses, une compréhension plus fine de toute sorte de question de conception, ce qui ne serait pas possible autrement. Comme pour tout outil, le praticien fait son choix, prend l'outil le plus approprié.

Dans de nombreux cas, les modèles de décision ne sont pas appropriés, trop encombrants pour être utiles. Cette série montrera néanmoins qu'ils se révèlent extrêmement pratiques dans d'autres cas : si les décisions au début du développement du jeu sont bien prises, en éliminant des problèmes avant même d'atteindre les phases de test, il est plus probable que le jeu complet sera solide, amusant et sans défaut.

Le programmeur fait face au même dilemme : son métier est difficile, mais il dispose d'un grand nombre d'outils pour trouver des défauts avant même que le code arrive aux tests. Notamment, ils ont des compilateurs qui crient en permanence dès la première faute de frappe ; les pratiques de programmation défensive montrent les défauts logiciels ; les revues de code identifient les défauts du code d'autrui et peuvent servir à éliminer de mauvaises habitudes de programmation ; les outils de profilage et d'analyse statique servent également à trouver les problèmes de performances et d'autres ennuis.

Les concepteurs n'ont pas ce genre d'outils. Leur travail est probablement aussi difficile, mais ils ne disposent pas de compilateur pour éviter les erreurs de syntaxe ; ils n'ont pas de profileur, d'outils de débogage ou d'analyse statique. Il est impossible de faire des revues de code, puisqu'il n'y a pas de « code » à proprement parler. Ils écrivent des spécifications, de la documentation sur les choix de conception... et c'est à peu près tout. Ils peuvent partager ces documents avec l'équipe et espérer avoir des retours de qualité, mais, en grande partie, il est nécessaire d'entrer dans le jeu pour voir si ces décisions tiennent la route.

Par conséquent, la conception de jeux est incroyablement risquée, elle prend beaucoup de temps et d'argent. Tout comme la programmation, cependant, l'erreur humaine est naturelle et fait partie intégrante du processus : il est *nécessaire* de disposer d'autant d'outils de bonne facture que possible pour protéger tant les concepteurs que leurs projets.

La route est longue avant d'avoir des outils de conception qui aideront l'exploration de l'espace de conception du même niveau que les compilateurs, débogueurs, profileurs, outils d'analyse statique, qui aident les programmeurs à explorer l'espace de développement. Cependant, de nouveaux outils apparaissent, pour la conception et la résolution dans quelques jeux, notamment [un vérificateur de jouabilité pour une variante de Cut the Rope, nommée Cut the Rope: Play Forever](#). Il faut aussi compter

sur le système de conception de jeu abstrait [Ludi](#), qui a notamment généré le jeu de plateau *Yavalath*, mais encore [l'assistant d'équilibrage automatique pour City Conquest](#).

La modélisation des décisions peut aider à franchir quelques étapes pour accroître et prolonger l'intelligence des concepteurs avec quelques outils automatiques. De toute façon, avec le choix d'avoir ces outils ou pas, pourquoi choisir de ne pas les avoir ?

Pas de feuille de calcul : des modèles !

Cette série d'articles s'oriente principalement vers les concepteurs – *tous* les concepteurs, qu'ils aient une expérience plus artistique, de programmation, narrative ou de jeu de plateau. Ils resteront donc relativement simples et respecteront ces quelques promesses :

- *pas de code*. Aucun article ne contiendra du code : tous les exemples utiliseront Excel et son solveur. Cependant, il est important de remarquer que cette série ne concerne *pas* l'utilisation de feuilles de calcul ou d'Excel, mais bien la modélisation et l'optimisation des décisions. Tout ce qui est présenté peut très bien être réalisé avec d'autres outils dans un langage de programmation, parfois même plus facilement ;
- *pas de mathématiques* – tout du moins, rien de compliqué. Cette série ne comportera que peu de mathématiques et, en tout cas, pas plus que les opérations arithmétiques de base : addition, soustraction, multiplication, division, peut-être à l'occasion une racine carrée. Les lettres grecques seront strictement interdits ;
- *pas de [feuille de calcul à quatre dimensions](#)*, deux suffiront largement.

Pour les concepteurs, cette série devrait donner tous les outils nécessaires pour créer des modèles de décision, sans devoir tenter d'écrire du code ou compter sur des programmeurs. Pour les programmeurs, elle devrait donner un guide relativement rapide pour la programmation de modèles de décision dans n'importe quel langage de programmation.

Ces articles ne sont prévus que comme des points de départ, des concepts à utiliser pour construire des modèles dans Excel, n'importe quel autre outil d'optimisation ou dans un solveur à écrire soi-même. Les feuilles de calcul sont un bon départ, mais ces modèles de décision seront plutôt prévus comme des tremplins pour des modèles plus sophistiqués et plus riches à intégrer dans l'architecture du jeu.

Avertissements

Avant d'entrer dans le vif du sujet, quelques avertissements semblent nécessaires. La modélisation et l'optimisation des décisions ne fournit d'aucune manière un système complet pour la conception d'un jeu et ne le prétend. Cette méthode se présente comme un outil, utile pour aborder certains aspects du processus de conception et, comme tout outil, a ses limites.

Notamment, voici les limites dont il importe d'être conscient :

- **ces outils sont faciles à mal utiliser**. Comme tout outil, les modèles de décisions peuvent être utilisés de manière inappropriée ou incorrecte ; un modèle de décision incomplet ou partiellement faux peut mener à des conclusions incorrectes. Tout comme du logiciel, plus le modèle est grand, plus il est probable qu'il soit incorrect. Il est aussi très facile de mal interpréter les résultats d'un modèle ou de construire un modèle incomplet qui ne prend pas en compte avec une précision suffisante le contexte des décisions à prendre ;
- **ces outils sont compliqués**, parfois. Certains problèmes de conception sont trop complexes pour être modélisés de la sorte. De nombreux problèmes ont trop de parties mouvantes ou

sont trop intégrés avec d'autres aspects du jeu pour être représentés de manière utile dans une feuille de calcul Excel indépendante. Dans ces cas, il faut décider de modéliser seulement une partie du système (ce qui peut donner un modèle invalide ou imprécis), de construire un modèle complet intégré dans le jeu lui-même (ce qui représente une grande quantité de travail) ou bien s'abstenir d'utiliser ces outils ;

- **tout ne peut pas être modélisé.** Les modèles de décision ne peuvent pas dire si une décision sera amusante pour les joueurs, si elle est esthétique, si elle s'intègre bien dans le contexte du jeu ou si elle présente au joueur une interface utilisable et accessible. Il n'existe bien souvent pas de manière de représenter ces préoccupations subjectives ou esthétiques dans un modèle. Cela signifie qu'il existe des limites aux cas d'utilisation des modèles de décision : ils sont bien plus utiles pour la conception des systèmes et l'optimisation des mécaniques de jeu et de ses dynamiques plutôt que de l'esthétique ;
- **ces outils ont leurs limites.** Toutes les solutions d'optimisation ont leurs limites, notamment le solveur d'Excel : il est possible de créer des modèles qui possèdent des solutions valides mais sont tellement complexes qu'aucun outil d'optimisation ne peut les trouver. Pour un nombre de variables suffisamment grand, le problème peut se révéler en dehors des capacités d'évaluation de toutes les combinaisons d'Excel : il doit alors utiliser d'autres techniques d'optimisation. Cette série montrera notamment des manières de simplifier l'expression des modèles pour que le solveur les appréhende plus facilement ; d'ailleurs, le développeur du solveur d'Excel, [Frontline](#), fournit un solveur bien plus puissant pour des problèmes plus grands, mais il reste possible de créer des modèles que ce solveur ne peut pas résoudre ;
- **ces outils ne garantissent pas l'optimalité.** À cause des limites inhérentes à la technologie, pour des modèles complexes, il est impossible de garantir que la solution donnée est *optimale*. En passant plus de temps à optimiser, la solution sera de meilleure qualité ; en relançant le modèle depuis zéro, il est aussi possible de dire que la solution est proche de l'optimalité à un niveau de confiance raisonnable.

Finalement, le point le plus important : **l'utilisateur doit s'assurer de modéliser les bons éléments.** Tous les problèmes ne sont pas importants au point de nécessiter ce genre d'efforts, il faut s'attribuer des priorités et éviter de trop se concentrer sur des défauts mineurs du modèle en ignorant d'autres, bien plus importants.

De manière générale, certaines choses doivent être vraies pour que la modélisation des décisions soit utile. La décision en question doit être encapsulable dans un modèle discret et faire correspondre la décision à une seule valeur. En d'autres mots, l'entrée doit être un ensemble fini de décisions possibles, le modèle fournit une seule sortie, de telle sorte que maximiser ou minimiser cette sortie donne la meilleure solution.

Dans ces cas où des intérêts subjectifs ne peuvent pas être insérés dans le modèle, comme des considérations d'ordre esthétique ou de jouabilité, il faudra soit les séparer clairement du modèle de décision, soit utiliser ces techniques comme une première phase, soit complètement abandonner l'approche.

Pour modéliser des décisions dans une feuille de calcul, il y a aussi une limite sur la complexité des modèles. Si le jeu effectue des actions très complexes, il n'est pas sûr qu'elles puissent être répliquées dans Excel. Il est important de garder à l'esprit, cependant, que ce n'est qu'une limite sur les modèles présentables à Excel, *pas* des modèles de décision eux-mêmes. Des solveurs spécifiques peuvent être implémentés dans le moteur de jeu, bien plus puissants que des feuilles de calcul : cette série a été écrite dans l'espoir d'inspirer des concepteurs à le faire.

D'un autre côté, toutes ces limitations ne sont pas suffisantes pour rendre la modélisation des décisions inutile. Même dans le cas où un problème est trop complexe, sa modélisation imparfaite peut donner un certain nombre de parties du système plus proches d'une configuration correcte, ce qui aide à trouver et à corriger un certain nombre de défauts dès le début du développement.

Même dans le cas où la modélisation ne permet pas de trouver la solution optimale à un problème, à cause de sa complexité ou de considérations humaines subjectives comme l'esthétique, l'espace de recherche est fortement réduit, la solution imparfaite élimine des zones de moindre intérêt, ce qui diminue la complexité du problème à résoudre.

Finalement, même pour ceux qui décideront de ne pas utiliser la modélisation des décisions, qui ne tenteront jamais d'optimiser sur une feuille de calcul ou de construire leurs propres solveurs, une compréhension de ces techniques peut toujours aider en changeant la manière de penser les décisions de conception d'un jeu.

Cette série est une exploration : elle observera bien des exemples de problèmes de conception de jeux et explorera des manières de les modéliser et de les optimiser qui offrent de puissants outils de conception. L'espoir est que mêmes les plus sceptiques, ceux qui se sentent le moins à l'aise avec l'optimisation explorent ces contrées par cette série, pour voir à quoi elle mène.

Conclusion

L'objectif final est de concevoir des jeux *correctement*.

La plupart des choix de conception sont subjectifs, sans « bonne » ou « mauvaise » réponse. Dans certains cas, cependant – bien plus nombreux que ce que l'on peut penser –, certains choix sont purement objectifs. Dans ces cas, le concepteur devrait vouloir connaître cette bonne réponse, au moins comprendre comment la définir et chercher cette solution, si elle existe.

La modélisation et l'optimisation des décisions sont des outils puissants qui vont dans cette direction, dans bien des cas. Ils devraient faire partie de la boîte à outils de chaque concepteur de jeux. Avec un peu de discipline, il devrait devenir clair que ces outils ont un potentiel peu exploité pour explorer la pièce noire de la conception de jeux plus vite et avec une sécurité accrue. Le reste de cette série en montrera différentes applications.

Remerciements

L'auteur souhaiterait remercier [Robert Zubek](#) et [Jurie Horneman](#) pour leurs retours sur cet article.

Cet article est une traduction autorisée de [Decision Modeling and Optimization in Game Design, Part 1: Introduction](#), écrit par Paul Tozour. Le traducteur souhaiterait remercier [Alexandre Laurent](#) pour ses relectures.