Génération de scénario : planification avec un opérateur défini par un modèle graphique

Rémi Lacaze-Labadie, Domitile Lourdeaux, Mohamed Sallak

Génération de scénario : planification avec un opérateur défini par un modèle graphique

R. Lacaze-Labadie[1] D. Lourdeaux[1] M. Sallak[1]

[1] Sorbonne universités, Université de technologie de Compiègne, CNRS,
Heudiasyc UMR 7253, 57 avenue de Landshut – 60203 COMPIEGNE Cedex, France

{remi.lacaze-labadie - domitile.lourdeaux - mohamed.sallak}@hds.utc.fr

Résumé

Utiliser la planification pour générer des scénarios présente de nombreux avantages tels que la variabilité, le contrôle et l'adaptation du scénario. Cependant, un scénario ne doit pas juste être une séquence d'actions valide mais doit également satisfaire la cohérence causale, c'est à dire que la séquence d'actions doit suivre une progression narrative logique et être explicable (i.e. l'apprenant doit comprendre tout ce qui se passe). Dans cet article, nous proposons un nouvel opérateur de planification permettant d'encapsuler des ensembles d'actions sous la forme de graphes. L'intérêt de cet opérateur et de contextualiser les actions et ainsi faciliter la cohérence causale des scénarios générés. Nous donnons un exemple de scénario généré et évaluons les performances de notre système sur un problème des IPCs.

Mots Clef

Planification, Scénarisation, Modèle graphique.

Abstract

Using planning techniques to generate scenarios offer many advantages such as the variability, the control and the adaptation of the scenarios to the learner. However, a scenario is not only a valid sequence of actions/events, it has to satisfy the causal coherence. The causal coherence refers to the notion that the sequence of events has to follow a logical progression of the narrative and the learner has to understand every event occurring in the scenario. In this article, we propose a new operator that allows to encapsulate sets of actions in the form of graphs. This operator allows to contextualize the actions and therefore to facilitate the causal coherence of the generated scenario. We provide an example of a scenario generated using this new operator and we evaluate the performance of our system on IPC problems.

Keywords

Planning, interactive narrative, graphical model.

1 Introduction

Dans cet article nous proposons l'utilisation de techniques de planification pour générer des scénarios dans le cadre de la formation en environnement virtuel (EV). Ce type de formation permet aux apprenants d'expérimenter, de s'entraîner et de voir l'impact de leurs décisions sans risque réel. Nous avons identifié plusieurs contraintes liées à la scénarisation dans ce contexte spécifique qu'est la formation. Premièrement, l'EV doit offrir une grande liberté d'action à l'apprenant [22], tout en maintenant un contrôle scénaristique. Ce contrôle permet à la fois de maintenir l'apprenant autour d'objectifs pédagogiques préalablement choisis par le formateur et également de s'adapter à l'apprenant en faisant varier certains critères de formation tels que la difficulté ou la criticité du scénario. Deuxièmement, le système de scénarisation doit être résilient, c'est à dire être capable de réagir aux actions de l'apprenant qui peut dévier du scénario initialement prévu (i.e "the boundary problem" [15]). Troisièmement, le système doit être capable de proposer un large panel de scénarios afin que l'apprenant puisse répéter la formation, c'est ce que nous appelons la variabilité du système. Enfin, le scénario doit suivre une progression logique, c'est à dire que chaque événement qui le compose doit être explicable.

Génération de scénarios. Pour répondre à ces contraintes, nous devons être capable de générer dynamiquement des scénarios plutôt que de les écrire manuellement. Cela permet de proposer à l'apprenant une grande variabilité de scénarios, alors que des scénarios scriptés seront forcément limités. Cela permet aussi de favoriser la liberté d'action de l'apprenant dans l'EV [20]. En effet, il est possible de générer un nouveau scénario si l'apprenant s'éloigne trop de celui initialement prévu, alors que dans le cas des scénarios scriptés il faudrait prévoir toutes les alternatives possibles, ce qui en pratique est impossible (i.e. "the authoring bottleneck" [21]). Enfin la génération de scénarios offre la possibilité de personnaliser ou d'adapter le scénario en modifiant certaines caractéristiques telles que sa difficulté. La planification s'avère être un bon moyen pour générer des scénarios car

elle formalise les aspects essentiels d'un scénario, à savoir les actions, la temporalité et la causalité [23]. Dans cet article, nous définissons un scénario comme une séquence d'événements qui décrit comment le monde évolue au cours du temps [20, 24, 12]. Par analogie, un plan est une séquence d'actions qui transforme le monde d'un état initial à un état final. Dans cette analogie, les actions au sens de la planification, sont les événements au sens du scénario, c'est pourquoi nous utiliserons ces deux termes de manière interchangeable. Les objectifs scénaristiques sont les buts du problème de planification et les critères de formation (e.g. la difficulté du scénario) sont des coûts attachés aux actions. Nous parlerons donc de coût d'un événement comme un ensemble de niveaux de critères que va engendrer l'apparition de l'événement. Par exemple, *déclencher une coupure de courant* pourra avoir un coût en difficulté de 1 et un coût en stress de 2. Ainsi nous pourrons planifier ce genre d'événements pour générer des scénarios plus ou moins difficiles, ou plus ou moins stressant.

Problématique. Contrairement à des problèmes de planification traditionnels (e.g. logistique, transport...) dans lesquels un plan valide suffit, un scénario doit également satisfaire la cohérence causale. Nous définissons la cohérence causale comme la progression logique des événements qui composent le scénario, dans le but de garantir que tous les événements soient explicables. Un événement est explicable si l'apprenant peut comprendre/expliquer pourquoi il est survenu, ce qui va souvent dépendre du contexte d'apparition de cet événement. En dehors de tout contexte, des événements tels que *faire sonner le téléphone* ou *déclencher une coupure de courant* peuvent ne pas être explicable ou ne pas avoir l'impact escompté. Pour satisfaire la cohérence du scénario, nous devons faire en sorte que ces événements n'arrivent pas par hasard. Si nous prenons l'exemple d'un scénario dans lequel l'apprenant doit s'occuper d'une victime, faire survenir une coupure de courant lorsque l'apprenant n'a aucunement besoin d'électricité n'aura pas le même impact que lorsque celui/celle-ci sera en train de réaliser une opération nécessitant du matériel électrique. L'apprenant peut ne pas comprendre les raisons de cette coupure de courant si cela a peu d'impact dans la réalisation de sa tâche. Il serait possible d'ajouter une précondition spécifiant que la coupure de courant doit se produire seulement si l'apprenant utilise un appareil électrique. Mais limiter cet événement avec une telle précondition ne paraît pas sensé. En effet, la coupure de courant devrait aussi pouvoir survenir dans d'autres circonstances (e.g. simuler un problème de luminosité). Nous avons ici un problème de cohérence causale, dans le sens où la coupure de courant ne sera pas toujours explicable et n'aura pas toujours l'impact escompté. *Rield* montre dans [20], que la cohérence causale ne peut se résoudre simplement en rajoutant des préconditions aux actions. En effet, un des avantages de la planification de scénarios est d'explorer toutes les possibilités en vue de créer des scénarios qui n'étaient même pas envisagés par le formateur. En rajoutant des préconditions pour contextualiser les actions, nous allons grandement limiter l'exploration. Le deuxième problème que nous avons identifié est lié à l'impact de ce type d'événement sur le scénario. Dans la planification classique, le coût d'une action ne dépend que de l'action et de ses variables (e.g. consommation de carburant qui dépend de la distance). A l'inverse, dans le cadre de la scénarisation, les niveaux de critères associés aux événements vont beaucoup varier en fonction du contexte. Par exemple, le coût en difficulté de la coupure de courant sera différent en fonction de ce qu'est en train de faire l'apprenant, chose qu'il n'est pas possible de prendre en compte dans la définition d'une action de planification classique.

Idée générale. Nous cherchons donc un moyen de contextualiser des actions afin de pouvoir évaluer leurs impacts sur le scénario et aussi garantir qu'elles seront planifiées dans un contexte explicable. Pour répondre à cette problématique, nous proposons d'englober des sous-ensembles d'actions dans des graphes, dont les nœuds sont soit des états du monde, soit des actions et dont les arcs sont des relations causales. L'intérêt d'utiliser des graphes d'actions est de pouvoir en extraire des séquences d'actions qui satisfont la cohérence causale. Chaque graphe d'action est défini dans un nouvel opérateur, appelé FRAG (pour fragment de scénario). La planification s'effectue dans l'espace d'actions (traditionnelles) et des FRAGs. Contrairement aux actions, appliquer un FRAG dans le plan consiste d'abord à extraire une séquence d'actions depuis le graphe avant de l'insérer dans le plan. En plus de faciliter la cohérence causale, ce nouvel opérateur permet aussi de définir des coûts contextuels. En effet, une même action peut apparaître dans plusieurs FRAGs (et aussi en tant qu'action classique), ce qui permet de lui associer un coût différent en fonction de son contexte. Par exemple l'événement qui déclenche une coupure de courant pourrait être modélisé dans deux FRAGs. Nous pouvons imaginer un FRAG qui connecte cet événement avec des actions où l'apprenant utilise du matériel électrique et un autre FRAG dans un contexte où l'apprenant réalise des tâches nécessitant une bonne luminosité. Le coût en difficulté de cet événement pourra alors être différent en fonction du contexte.

2 Contexte et travaux connexes

2.1 Cadre d'application

Nos travaux s'inscrivent dans le cadre du projet VICTEAMS (Virtual Characters for Team Training : Emotional, Adaptive, Motivated and Social). Ce projet vise à la création d'un environnement virtuel pour la formation des leaders d'équipe médicale à des compétences non-techniques (e.g. prise de décision, communication, leadership). Le contexte opérationnel est le sauvetage de victimes en temps de guerre ou après une attaque terroriste. Dans cet environnement, l'apprenant joue le rôle du leader médical et les personnages virtuels (PNJs) sont les membres

de l'équipe médicale. Dans la suite, nous utiliserons le terme "formateur" pour désigner la personne qui définit le contenu de la formation et donne les contraintes scénaristiques au planificateur. Nous utiliserons le terme critère de formation ou simplement critère pour désigner un attribut du scénario tel que la difficulté ou la criticité du scénario. Ces critères sont ensuite attachés aux actions et correspondent à l'impact de cette action sur l'apprenant (e.g. faire sonner le téléphone engendre un stress de 2). Notre système fonctionne comme avec des coûts classiques que l'on cherche à minimiser ou maximiser (e.g. le temps, la consommation en énergie). Dans ce projet, le rôle du planificateur est double. Il doit dans un premier temps générer/planifier un scénario qui atteint un but (e.g. sauver toutes les victimes) et satisfait les critères demandés par le formateur. Dans un second temps, le planificateur est en charge de suivre en direct l'exécution du scénario dans l'EV de façon à intervenir (re-planifier) lorsque l'apprenant s'éloigne du scénario initialement prévu. Pour cela, les scénarios ne sont pas composés exclusivement d'événements, mais également d'actions "attendues de l'apprenant" qui sont nécessaire pour le déroulement du scénario. Ce type d'actions fait partie intégrante de la planification, au même titre que les événements, et peut donc être inclu dans des FRAGs. Par exemple, avant l'événement *déclencher une coupure de courant*, on pourra avoir l'action *l'apprenant pose un garrot électronique* dans le plan. Pendant l'exécution du scénario, si une action de l'apprenant est attendue (parce que l'exécution est arrivée sur cette action) mais n'est plus possible (parce que l'action ne satisfait plus ses préconditions dans le monde courant), alors une re-planification est déclenchée.

2.2 Travaux connexes

Nous pouvons classifier les approches de génération de scénarios en deux catégories. Les approches s'appuyant sur la simulation (simulation-based narrative generation) et les approches délibératives (deliberative narrative generation). Dans les approches s'appuyant sur la simulation, le scénario progresse en fonction de l'évolution du monde et des actions prises par les personnages virtuels. C'est par exemple le cas de *Tale-Spin* [16]. On retrouve également dans cette catégorie la narration émergente [2, 1] ou encore les approches centrées sur les personnages [7]. Ces approches considèrent que la narration émerge des interactions entre les personnages virtuels qui évoluent souvent de manière autonome. Ces approches ne permettent pas d'exercer un contrôle externe quant au déroulement du scénario. Le formateur ne pourra donc pas spécifier de contraintes scénaristiques. Les approches délibératives sont celles dont le scénario est créé, souvent via un planificateur, comme une séquence d'actions satisfaisant certaines contraintes et paramètres préalablement choisis. A l'inverse des approches s'appuyant sur la simulation, la génération du scénario est centralisée dans le sens où elle dépend d'un unique auteur ayant le choix de la structure et des contraintes scénaristiques. C'est ensuite au système de trouver un scénario satisfaisant les contraintes demandées. Nous pouvons citer comme exemple le système *Universe* [14] qui utilise un planificateur hiérarchique pour générer des scénarios ou encore les travaux de *Porteous et al.* [18] qui utilisent une variation du planificateur *FF* [10].

Ces deux approches sont aux deux extrémités et la plupart des systèmes sont en réalité hybrides. D'un point de vue scénaristique, nous nous situons vers les approches délibératives qui permettent au formateur de contrôler le scénario. Le formateur joue le rôle de l'auteur en définissant ses objectifs scénaristiques (les buts à atteindre) et ses contraintes (les critères de formation). Toutefois, nous ne générons pas un scénario de A à Z comme on écrirait une histoire. Nous devons plutôt scénariser un monde simulé où l'apprenant est libre de ses actions et les personnages virtuels évoluent en partie de manière autonome. Il faudra alors être capable de re-planifier et adapter le scénario en fonction des divergences entre le scénario prévu et les actions de l'apprenant et des PNJs.

2.3 Limites de la planification

La planification répond aux besoins scénaristiques du projet VICTEAMS. En effet, être capable de planifier des scénarios favorise le contrôle du formateur en lui offrant la possibilité de faire varier des critères de formation (e.g. difficulté du scénario). De plus, il est possible de replanifier depuis n'importe quel état du monde si l'apprenant s'éloigne trop du scénario initialement prévu. Enfin, cela permet de pouvoir proposer une grande viabilité des scénarios sans avoir besoin de tous les imaginer à l'avance. Toutefois, nous avons identifié en introduction des problèmes quant à la cohérence causale des scénarios générés. Prenons un exemple de scénario dans lequel l'apprenant doit gérer l'évacuation de victimes. Dans le cas nominal, il/elle appellera un medevac une fois toutes les victimes examinées, le medevac arrivera et les victimes seront évacuées. Dans notre exemple un medevac est un véhicule aérien servant à évacuer les victimes. Nous souhaitons maintenant augmenter la difficulté de ce scénario. Pour ce faire, le planificateur dispose d'événements perturbateurs dont le coût en difficulté est supérieur à 0, par exemple l'arrivée d'une nouvelle victime. L'ajout d'une victime est une action qui n'a pas de précondition, ce qui fait qu'elle peut être planifiée à tout moment. Toutefois, cela ne sera pas toujours pertinent. Dans notre exemple, la victime peut par exemple être ajoutée avant ou après l'appel au medevac. D'un point de vu formation, il est intéressant de l'ajouter après que le medevac ait été appelé car l'apprenant devra gérer le fait qu'un medevac ne peut pas évacuer plus de victimes que le nombre annoncé lors de son appel. Si par contre la victime arrive avant l'appel au medevac, l'apprenant n'aura pas à gérer cette situation. Il est de plus difficile d'évaluer la difficulté réelle engendrée par cet événement sans en connaître le contexte. Nous proposons donc un moyen de contextualiser ce genre d'événements, à la fois

pour les placer dans des contextes qui permettent d'évaluer leurs impacts sur l'apprenant (en terme de difficulté par exemple) et également pour s'assurer que ces événements apparaissent dans un contexte de formation pertinent. Par exemple, faire arriver une nouvelle victime après que le medevac ait été appelé crée une situation pertinente, du point de vue des formateurs, à laquelle l'apprenant devra faire face. Dans les problèmes de planification traditionnels tels que les problèmes de logistique, la cohérence causal entre les actions et l'explicabilité du plan importe peu, ce qui est recherché c'est un plan valide qui minimise le coût total. Hors, comme le mentionne *Porteous et al.* dans [19], le besoin de générer un plan qui suit une progression logique et raconte une histoire rend la création des domaines de planification bien plus complexe. De plus, *Rield* montre que rajouter des préconditions pour contextualiser les événements limiterait les capacités du planificateur à explorer un grand nombre de possibilités. Nous pensons également qu'ajouter des préconditions et multiplier les actions de planification deviendraient vite maintenable. C'est pourquoi nous proposons un opérateur facilitant la création de ce type de domaine en permettant de modéliser des fragments de scénario via des graphes.

3 Proposition

Nous proposons un nouvel opérateur (appelé FRAG) qui regroupe un ensemble d'actions formant un fragment de scénario, c'est à dire la modélisation d'une situation particulière (e.g. l'arrivée d'une victime alors que l'évacuation a déjà été demandée). L'opérateur FRAG est composé de préconditions et d'une carte cognitive floue (FCM), qui est un graphe de causes/conséquences. Un FRAG est exécuté, puis converti en une séquence d'actions qui est ensuite insérée dans le plan. Ainsi, les actions connectées dans les graphes se retrouvent insérées les unes à la suite des autres, ce qui facilite l'apparition de séquences pertinentes dans le scénario généré. Les FRAGs permettent alors de contextualiser certaines actions, tel que l'ajout d'une victime, en les reliant avec d'autres actions, formant ainsi un contexte situationnel. Une même action peut-être placée dans plusieurs FRAGs et donc dans différents contextes. De plus, cette même action pourra avoir un coût différent pour chaque FRAG, ce qui permet de contextualiser son coût. Enfin, utiliser des graphes plutôt que des séquences d'actions statiques permet d'englober plusieurs alternatives au sein d'un même contexte et de créer des chemins dans le graphe qui engendrerons par exemple plus ou moins de difficulté. La séquence est générée dynamiquement en fonction de l'état courant du monde. Appliquer un FRAG revient donc à (1) initialiser son FCM depuis l'état courant, (2) exécuter le FCM, (3) générer une séquence d'actions depuis le résultat d'exécution du FCM et enfin (4) ajouter cette séquence d'actions dans le plan. Dans la partie suivante, nous donnons plus de détails sur ce qu'est un FCM et sur ces différentes étapes.

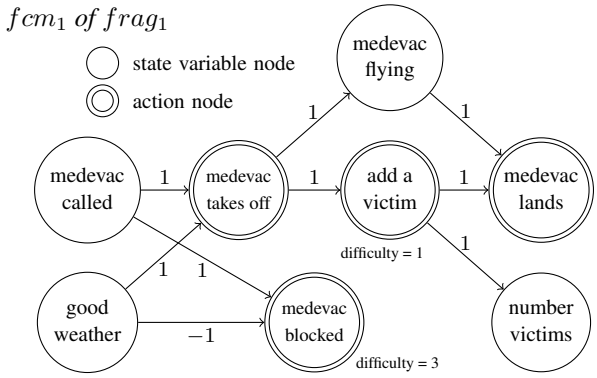

FIGURE 1 – Un exemple simplifié de FCM dans lequel deux alternatives sont possibles en fonction de la météo. L'une des alternatives étant plus difficile que l'autre, le planificateur pourra au préalable changer la météo (via une action classique) pour générer l'alternative correspondante à la difficulté demandée.

3.1 Cartes Cognitives Floues (FCM)

Les cartes cognitives floues, plus connues sous le nom anglais de *Fuzzy Cognitive Map* (FCM), sont des graphes orientés utilisés en théorie de la décision pour modéliser des systèmes complexes de connaissances d'experts. Elles ont été introduites par Kosko en 1986 [13], puis agrémentées de multiples extensions au cours du temps [17]. Un FCM s'appuie sur le principe de concepts (les nœuds du graphe) reliés entre eux par des relations de causes/conséquences (les arcs). Les arcs sont labellisés par un poids, souvent limités à l'intervalle [-1, 1], qui représente le degré d'influence du concept source (la cause) sur le concept destination (la conséquence). Un poids négatif signifie que le concept source refrène/empêche la réalisation du concept destination.

3.2 Exemple de FCM appliqué à la planification

Un exemple de FCM utilisé dans un FRAG est donné en figure 1. Chaque nœud du FCM correspond soit à une action (nœud action), soit à une variable d'état (nœud état). Un arc d'un nœud état vers un nœud action représente une précondition, un arc d'un nœud action vers un nœud état représente un effet et enfin un arc entre deux nœuds actions signifie que l'un doit se produire avant l'autre. Chaque nœud a une fonction d'activation qui régule sa valeur lors de l'exécution du FCM. Exécuter un FCM consiste à itérativement mettre à jour les nœuds du graphe jusqu'à ce que tous les nœuds aient convergé vers leurs valeurs finales. Un nœud a convergé si la différence de sa valeur entre deux itérations est inférieure à un seuil donné. Les nœuds actions sont régulés pour ne prendre que des valeurs de 0 (désactivé) ou 1 (activé). Idem pour les nœuds états associés à des propositions, où 0 signifie que la proposition ne doit pas être présente dans l'état du monde et 1 qu'elle doit l'être. Enfin les nœuds états associés à des variables numériques

peuvent prendre n'importe quelle valeur pourvu que cette valeur appartienne au domaine de la variable numérique. Dans le cadre des FRAGs, les poids des arcs sont limités à 1 sauf pour les arcs liant une proposition à une action qui peuvent aussi avoir un poids de -1. Un arc de -1 signifiant que l'action (nœud destination) peut être activée si la proposition (nœud source) est fausse.
L'exemple que nous proposons sur la figure 1 est un fragment de scénario dans lequel un medevac a été appelé par le leader médical afin d'évacuer les victimes. Par manque de place, les fonctions d'activations ne sont pas visibles sur la figure et les effets ne sont pas tous dessinés. L'idée de ce FCM est de modéliser une situation difficile dans un contexte d'évacuation de victimes. Une fois le medevac appelé par l'apprenant, le FCM propose deux possibilités en fonction de la météo. Si la météo est clémente, une nouvelle victime arrivera après que le medevac ait décollé. La difficulté résidera alors dans le fait qu'un medevac ne peut en théorie embarquer plus de victimes que le nombre établi lors de son appel (car le medevac doit préparer le matériel et le nombre de place en fonction des victimes prévues). Si par contre la météo est mauvaise, la difficulté sera que le medevac ne pourra pas décoller. Nous pouvons voir sur le FCM que dans ce cas, la difficulté estimée par les formateurs est supérieure à la première alternative. Le planificateur pourra prévoir un changement de météo (via une action de planification classique) avant l'appel de ce FRAG, et ainsi planifier l'alternative correspondant à la difficulté demandée. Dans cet exemple, nous pouvons voir que les actions d'ajouter une victime et d'empêcher le medevac de décoller sont contextualisées. De plus, leurs coûts correspondent à la difficulté engendrée dans ce contexte en particulier. Sans FRAG il serait par exemple difficile d'évaluer le réel impact de l'ajout d'une victime sans en connaître le contexte.

4 Formalisation

Dans cette partie, nous décrivons la notation utilisée et formalisons l'opérateur FRAG. Nous nous appuyons sur la notation STRIPS ainsi que sur le langage PDDL2.1 [9] pour les variables numériques.

4.1 Domaine et problème de la planification

Domaine. Un domaine de planification est un système transitoire $\Sigma = (S, A, FRAGS, \gamma, cost)$ où S est un ensemble fini d'états, A est l'ensemble des actions instanciées, FRAGS est l'ensemble des FRAGs instanciés, $\gamma : S \times A$ est la fonction de transition d'un état à un autre, et $cost : A \rightarrow \mathbb{R}_+$ est la fonction coût. Dans notre cas, le coût d'une action est une combinaison linéaire de critères. Un état du monde $s \in S$ est un ensemble de variables d'états $X = P \cup N$ où P est un ensemble de propositions (vrai) et N est un ensemble de variables numériques. Nous utilisons l'hypothèse du monde clos (CWA), ce qui signifie que toutes propositions non présentes dans un état sont considérées comme fausses. Une action A est un tuple $(name_a, pre_a, eff_a)$ où pre_a est l'ensemble des préconditions et eff_a l'ensemble des effets. Ces effets sont eux même subdivisés en un triple $(p^+_{eff}, p^-_{eff}, n_{eff})$ où $p^+_{eff} \subseteq P$ et $p^-_{eff} \subseteq P$ correspondent respectivement aux propositions à ajouter et à supprimer et n_{eff} est l'ensemble des effets numériques sous la forme (n, ass, exp) tel que $ass \in \{:=, +=, -=, *=, /=\}$ et exp représentent une expression. Plus de détails sur les préconditions et effets numériques sont donnés dans [11].

Problème. Un problème de planification est un triplet $P = (\Sigma, s_0, g)$, où Σ est le domaine précédemment défini, s_0 est l'état initial et g est le but du problème sous la forme d'un ensemble de variables d'états. Un état s satisfait g (noté $s \models g$) si toutes les propositions positives de g sont dans s, aucune proposition négative de g sont dans s et toutes les conditions numériques sont satisfaites dans s. Une solution du problème de planification est une séquence d'actions sous la forme $plan = \langle a_1, \ldots, a_n \rangle$ *t.q.* $a_i \in A$. Un scénario est une transformation du plan pour en faire un plan partiellement ordonné $pop = (A_{plan}, \prec_{plan})$ tel que A_{plan} est l'ensemble des actions du plan et $\prec_{plan}$ l'ensemble des contraintes de précédence sous la forme $a_i \prec a_j$ *t.q.* $a \in A$. Cette transformation consiste à ordonner les actions parallèles (qui doivent se produire simultanément) ainsi que les actions sans précondition (événements extérieurs).

Fonction d'évaluation. La fonction d'évaluation est utilisée pour sélectionner le nœud (n) le plus prometteur lors de la planification. Ici, nous ne parlons pas des nœuds d'un FCM mais des nœuds au sens de la planification, c'est à dire représentant l'état du monde. Nous définissons la fonction $f(n) = cost(\pi) + h(s)$ tel que $cost(\pi) = \sum_n^{i=1} cost(a_i)$ est le coût minimal pour atteindre l'état courant et $h : S \rightarrow \mathbb{R}$ est une heuristique indépendante du domaine [4], qui sert à estimer le coût restant pour atteindre un état but (e.g. h^{add} [5], h^{FF} [10]). Traditionnellement le nœud le plus prometteur est celui qui minimise f et nous garderons cette définition dans la partie algorithme pour rester générique. Toutefois, notre cadre d'application nous impose une légère transformation de cette fonction. Nous cherchons un scénario satisfaisant des critères demandés par le formateur. La fonction à minimiser est donc $g(n) = abs(cost\_attendu - f(n))$ de sorte que g est minimal (égale à 0) quand les critères estimés de la solution sont ceux demandés par le formateur.

4.2 L'opérateur FRAG

Un frag est un opérateur de planification défini sous la forme d'une paire (pre, fcm) tel que pre est l'ensemble des préconditions et $fcm \in FCM$, un FCM instancié. Un FCM instancié est un FCM dont les variables d'états associées aux nœuds sont instanciées par les constantes du problème.

Formalisation d'un FCM. Un $fcm \in FCM$ est formalisé comme une paire (C, W) tel que C est un vecteur de

nœuds (les concepts du FCM) de taille n et W est la matrice d'adjacence de taille $n \times n$ qui indique le poids associé à chaque paire de nœuds. On distingue deux types de nœud, à savoir les nœuds actions $C_A \subseteq C$ qui représentent une action exécutable dans l'EV et les nœuds états $C_X \subseteq C$ qui représentent un changement d'état. Parmi ces nœuds états, nous dénotons deux sous-ensembles qui nous servirons dans la partie algorithme : $C_N \subseteq C_X$ pour les nœuds états associés à des variables numériques et $C_P \subseteq C_X$ pour ceux associés à des propositions. Nous définissons donc $cx \in C_X$ et $ca \in C_A$ comme suit :

$$cx = (name, v, f, x) \text{ s.t. } x \in X$$
$$ca = (name, v, f, t, e = \{x_1, \ldots, x_n\}) \text{ s.t. } x_i \in X$$

Les deux types de nœuds ont : un nom, une valeur v et une fonction d'activation f. Chaque nœud état est associé à la variable d'état x qu'il modifie. Après exécution du FCM, une action est générée pour chaque nœud action *activé*, c'est à dire chaque nœud dont la valeur v est égal à 1. Les actions générées ont pour nom le nom du concept, n'ont pas de précondition et ont pour effet des changements d'états associés à la liste de variables d'états e (les détails de la génération des effets sont donnés dans la section algorithme). La variable t est simplement une variable qui contient le nombre d'itération avant que le nœud action ait convergé vers son activation (si il est activé). Cette variable permet de trier par ordre chronologique les nœuds activés de façon à ordonner la séquence d'actions et s'assurer que les causes arrivent avant les conséquences.

Initialisation et exécution d'un FCM. Initialiser un FCM depuis un état courant du monde consiste à initialiser tous ses nœuds actions à 0 (toutes les actions sont désactivées par défaut), puis à initialiser tous ses nœuds états depuis l'état courant (voir section 5). Exécuter un FCM consiste à calculer itérativement la valeur de chaque nœud jusqu'à ce qu'ils aient convergé. Un nœud a convergé quand sa valeur entre deux itérations est en dessous d'un seuil donné. Chaque nœud est mis à jour via sa fonction d'activation (FA) qui calcule la nouvelle valeur du nœud en fonction de la valeur de ses prédécesseurs à l'itération précédente selon l'équation suivante :

$$v_i(t) = f_i(\sum_{\substack{j=1 \\ j \neq i}}^{n} v_j(t-1) w_{ji}) \quad (1)$$

Il existe plusieurs types de fonctions d'activation tels que la fonction linéaire, sinusoïdale ou encore tangente hyperbolique. Le choix de la fonction se fait selon la régulation souhaitée du nœud (e.g. [-1,1], {0,1},...).

5 Algorithme

Notre planificateur utilise un algorithme à chaînage avant, dans lequel la recherche s'effectue en appliquant les actions et les FRAGs. La recherche est guidée par la fonction d'évaluation donnée à la section 4.1. Nous commençons par présenter la fonction $ExecuteFRAG$ qui est en charge d'exécuter un FRAG afin d'en générer une séquence d'actions à insérer dans le plan, puis nous présentons l'algorithme du planificateur.

5.1 Exécution d'un FRAG

Algorithm 1 Exécution d'un FRAG

```
1:  function EXECUTEFRAG(frag, s)
2:      for each c ∈ C_N do
3:          v_c ← x_c(s)
4:      for each c ∈ C_P do
5:          v_c ← 1 if x_c ∈ s else 0
6:      for each c ∈ C_A do
7:          v_c ← 0
8:      execute(fcm_frag)
9:      act_ca = sort({c ∈ C_A | v_c = 1}, key=t)
10:     for each c ∈ act_ca do
11:         // v_cx est la valeur du nœud lié à x (p ou n)
12:         eff ← {p+_eff | p ∈ P ∩ e_c, v_cp = 1} ∪
                  {p-_eff | p ∈ P ∩ e_c, v_cp = 0} ∪
                  {(n_eff, =, v_cn) | n ∈ N ∩ e_c}
13:         a ← (name_c, {}, eff)
14:         seq_a.add(a)
15:     return seq_a
```

La fonction $ExecuteFRAG$ donnée à l'algorithme 1 prend en entrée un FRAG instancié ($frag$) et un état du monde s. La notation fcm_{frag} désigne le FCM de $frag$ et C_N, C_P et C_A désigne les différents types de nœud du FCM (voir section 4). Cette fonction est décomposée en trois étapes : (1) l'initialisation du FCM, (2) l'exécution de ce FCM et enfin (3) la génération de la séquence d'actions en fonction du résultat d'exécution du FCM. L'initialisation du FCM se fait de la ligne 2 à 7 comme suit : tous les nœuds états associés à des variables numériques (C_N) sont initialisés avec la valeur numérique dans s de la variable d'état auquel ils sont associés, tous les nœuds états associés à des propositions (C_P) sont initialisés à 1 si la proposition existe dans s et 0 si elle n'existe pas, et enfin tous les nœuds actions sont inutilisés à 0 (non activé). A la ligne 8, le FCM est exécuté par une librairie externe (e.g. JFCM [8]). La dernière étape consiste à générer une séquence d'actions qui sera insérée dans le plan. Une action est générée pour chaque nœud action activé, c'est à dire qui à une valeur v_c à 1. L'ordre des actions est important, c'est pourquoi la liste des nœuds actions activés est dans un premier temps triée par ordre d'activation pendant l'exécution du FCM (ligne 9). Ensuite, de la ligne 10 à 13, une action est générée pour chaque nœud action activé. Une action générée a le même formalisme que les actions de l'ensemble A. Le nom de l'action correspond au nom du nœud, ces préconditions sont vides et ces effets sont générés depuis la liste de variables d'états e_c. Chaque proposition p dans e_c est ajoutée si la valeur du nœud état associé à p (v_{cp}) est

égale à 1 et retirée de la liste des effets si elle est égale à 0. Nous supposons ici que chaque variable d'état dans e_c a un nœud état qui lui est associé et que sa fonction d'activation produit une valeur de 0 ou 1 (autrement l'effet est ignoré). Chaque variable numérique n dans e_c est ajoutée en tant qu'effet numérique tel que la variable numérique doit être égale à la valeur du nœud état associé a n. Finalement, toutes ces actions sont renvoyées dans une liste.

5.2 Algorithme de planification

Algorithm 2 Algorithme de planification

```
1:  function PLAN(s_0, g, A, FRAGS)
2:      frontier ← {(⟨⟩, s_0)}, explored ← ∅
3:      while frontier ≠ ∅ do
4:          select n ← (π, s) ∈ frontier
5:          frontier.remove(n), explored.add(n)
6:          if s ⊨ g then
7:              return buildPlan(π)
8:          children ← {(π.a, γ(s, a)) | s ⊨ pre_a}
9:          F ← {frag ∈ FRAGS | s ⊨ pre_frag}
10:         for each frag ∈ F do
11:             seq_a ← EXECUTEFRAG(frag, s)
12:             s' ← s
13:             for each a ∈ seq_a do
14:                 s' ← γ(s', a)
15:                 children.add((π.a, s'))
16:         prune(frontier ∪ explored ∪ children)
17:         frontier ← frontier ∪ children
18:     return failure
```

Dans cette partie nous décrivons l'algorithme 2 qui planifie un scénario. A chaque itération, l'algorithme choisi le meilleur état à explorer puis il l'explore en appliquant toutes les actions possibles et en exécutant tous les FRAGs applicables (et en insérant leurs séquences d'actions résultantes).

Détails. On commence par définir $frontier$ comme l'ensemble des nœuds prêt à être explorés et $explored$ comme l'ensemble des nœuds déjà explorés. Un nœud $n \leftarrow (\pi, s)$ est une paire dont π est la politique courant et s l'état courant. Ligne 2, $frontier$ est initialisé avec le nœud initial composé d'une politique vide et de l'état initial s_0 et $explored$ est vide. L'algorithme s'exécute tant que $frontier$ n'est pas vide et qu'aucune solution n'a été trouvée. A chaque itération on sélectionne le nœud le plus prometteur (ligne 4) via $f(n)$. On met ensuite à jour $frontier$ et $explored$ (ligne 5) puis on explore ce nœud sauf s'il satisfait le but du problème g, auquel cas le plan est construit depuis π en remontant toutes les actions de l'état final vers l'état initial. Nous explorons s en appliquant toutes les actions applicables dans s (ligne 8) ainsi que les séquences d'actions générées par l'exécution de tous les FRAGs applicables (ligne 9 à 15). Ligne 16, l'étape d'élagage vise à se débarrasser de tous les nœuds non prometteurs, à savoir tous les nœuds dont l'état s a été atteint par un meilleur nœud. Enfin, $frontier$ est mis à jour avec les nouveaux nœuds atteints (ligne 17) puis ce processus est répété. Il est important de noter qu'un FRAG avec la même initialisation génère toujours la même séquence d'actions. Par conséquent, on réduit le nombre d'appel à $ExecuteFRAG$ en stockant dans un dictionnaire la séquence d'actions générée pour un FRAG et une initialisation donnée, ainsi nous pouvons la retourner directement plutôt que de faire appel plusieurs fois à $ExecuteFRAG$.

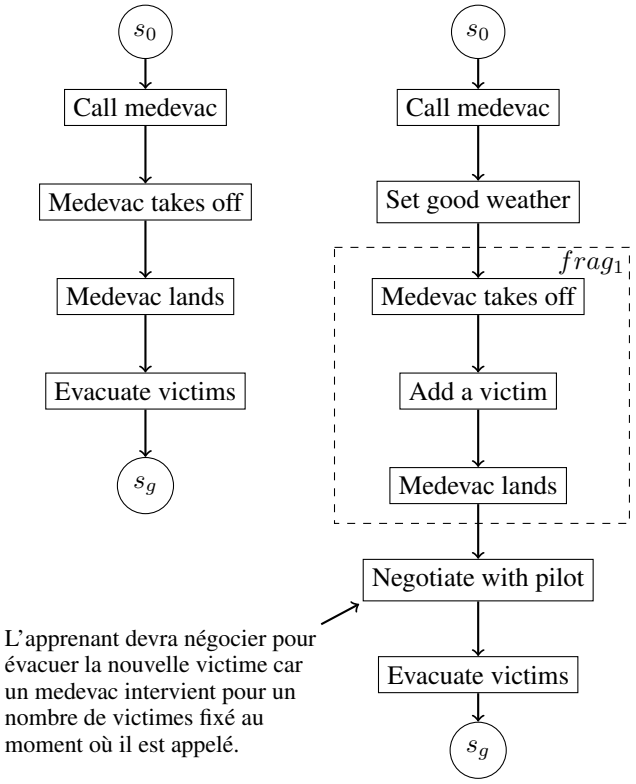

FIGURE 2 – A gauche un scénario de difficulté 0 et à droite un scénario de difficulté 1 qui contient une séquence d'actions provenant du FRAG de la figure 1.

6 Implémentation et exemple

Dans cette partie nous donnons des détails sur l'implémentation de notre planificateur, puis nous présentons un petit exemple de scénario généré. Notre algorithme peut-être facilement implémenté sur la base d'un planificateur faisant de la recherche en chaînage avant. Nous avons choisi $ActuPlan$, un planificateur numérique et temporel développé en Java et proposé par *Beaudry et al.* [3]. Les FCMs sont exécutés avec la librairie JFCM (Java Fuzzy Cognitive Maps) [8]. Et enfin nous utilisons l'heuristique h_{FF} [10].

6.1 Exemple de scénario

Dans l'exemple proposé à la figure 2 le scénario à gauche est le scénario de base (sans aucun critère demandé) dont le but est l'évacuation de victimes. Le scénario généré suppose que l'apprenant va appeler un medevac (medical eva-

cuation), que le medevac va décoller, atterrir sur la zone d'opération, puis les victimes pourront être évacuées. Dans un contexte de formation, nous souhaitons confronter l'apprenant à des situations plus ou moins difficile. Sur le scénario de droite nous avons demandé une difficulté de 1. Le planificateur va utiliser le FRAG défini à la figure 1 et choisir le chemin avec la difficulté de 1. Une météo clémente est d'abord planifiée pour satisfaire la précondition *good weather* du FCM. Une partie du scénario provient donc d'un FRAG qui permet d'ajouter une victime à un moment qui va réellement apporter la difficulté de 1 demandée. Nous pouvons voir un peu plus loin dans le scénario que l'apprenant va devoir négocier avec le pilote pour évacuer cette nouvelle victime avec les autres initialement prévues. Les formateurs ont modélisé que l'ajout d'une victime à ce moment impliquerait de devoir négocier avec le pilote, ce qui engendrerait une difficulté de 1 dans le scénario. L'ajout d'une victime ainsi que son coût en difficulté ont donc pu être contextualisés avec le FRAG de la figure 1.
La capacité de notre système à générer des scénarios cohérents est en cours d'évaluation. Nous prévoyons de générer plusieurs scénarios, avec et sans l'utilisation de FRAGs, et de les proposer à des formateurs pour évaluer leurs pertinences et leurs cohérences. D'un point de vue plus technique, nous évaluons dans la prochaine section les performances de notre planificateur dans le but de déterminer l'impact sur le temps de calcul qu'engendre l'exécution des FRAGs par rapport à la planification classique.

7 Résultats expérimentaux

Dans le but d'avoir un point de comparaison avec des problèmes de planification plus traditionnels, nous avons réalisé nos tests de performance sur des problèmes issus des compétitions internationales de planification (IPC). Nous pensons aussi que cela pourrait ouvrir des perspectives sur l'utilisation de notre opérateur à des problèmes autre que la scénarisation. Ces tests ont été réalisés sur une machine avec un CPU 2.7GHz Intel i7-6820HK et une limite de 4 GB de mémoire pour la JVM. Planifier avec des FRAGs engendre un surcoût car cela implique d'exécuter des FCMs et de générer des séquences d'actions. C'est pourquoi nous évaluons l'impact en terme de performance de leurs utilisations. Nous avons choisi le domaine ZenoTravel [1] qui consiste à embarquer et débarquer des passagers à différents endroits. Ce problème inclus la possibilité de voyager plus ou moins rapidement mais nous considérons dans nos tests une seule vitesse de vol. Les tests s'effectuent sur deux domaines contenant des FRAGs en plus du domaine de base qui lui n'en contient pas. Nous avons comparé le temps d'exécution de ces trois domaines sur des problèmes identiques. Dans un souci de comparaison, les deux domaines définis avec des FRAGs sont tout aussi permissifs que le domaine de base, à savoir que les nœuds actions des FCMs ne contiennent que des actions déjà présentes dans le domaine de base. Le domaine de base, que nous appellerons *no-frag*, a 4 actions qui sont *load*, *fly*, *unload* et *refuel*. Le second domaine appelé *lo-fly-unlo* contient un FRAG qui inclut les actions *load*, *fly* et *unload* et qui peut après exécution générer toutes les séquences possibles de ces 3 actions (en fonction de l'état du monde). Enfin, le troisième domaine appelé *refuel-fly* contient les deux actions *refuel* et *fly*. Les actions non présentes dans les FRAGs sont bien sûr gardées en tant qu'actions classiques.

1. http ://ipc02.icaps-conference.org/

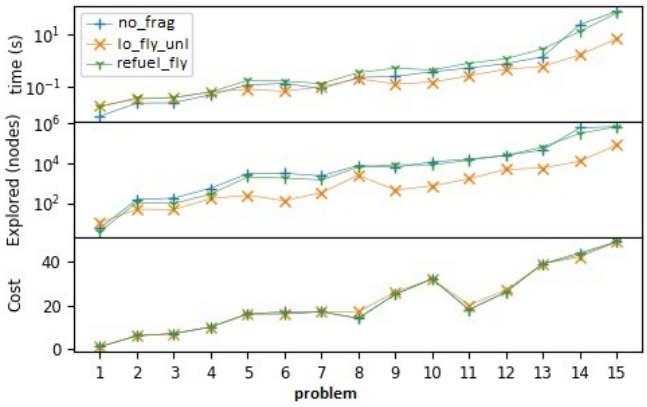

FIGURE 3 – Test de comparaison avec et sans utilisation de FRAGs.

7.1 Analyse des résultats obtenus

La figure 3 présente les résultats que nous avons obtenu sur les 15 premiers problèmes utilisés lors des compétitions internationales de planification (IPCs). Le graphe du bas montre que l'utilisation de FRAGs n'affecte pas le coût total des solutions. Pour se concentrer uniquement sur la comparaison avec et sans FRAGs, le coût de chaque action est de 1 donc le coût total correspond à la taille du plan. Le graphe du haut donne le temps d'exécution (en seconde) et le graphe du milieu donne le nombre de nœuds exploré pendant la recherche (les deux échelles sont logarithmiques). Commençons par comparer le domaine *refuel-fly* et le domaine *no-frag*. Le nombre de nœuds exploré est a peu près identique et le temps d'exécution est légèrement supérieur pour le domaine *refuel-fly*. Nous avons donc bien un temps additionnel d'exécution des FRAGs mais qui ne semble pas significatif. Cela s'explique par le fait que les FCMs s'exécutent assez vite et surtout que le résultat d'exécution est stocké dans un dictionnaire, ce qui réduit grandement le nombre d'exécution de FRAGs. A l'inverse nous pouvons voir que le domaine *lo-fly-unlo* explore moins de nœuds et s'exécute plus rapidement que le domaine *no-frag*. Ceci s'explique par le fait que les actions *load* puis *fly* ou *fly* puis *unload* apparaissent souvent consécutivement dans les solutions. En effet, un avion vole rarement sans au moins charger ou décharger une personne. Ainsi appliquer un FRAG générant la séquence d'actions *{load -> fly}* ou même *{load -> fly -> unload}* fait avancer la recherche vers l'état final plus rapidement. Cet effet n'apparaît pas sur le domaine *refuel-fly* car l'action *refuel*

apparaît finalement peu souvent dans les solutions et donc la séquence *{refuel -> fly}* est peu présente dans les solutions (probablement parce que le réservoir est suffisamment grand pour plusieurs vols). Les séquences d'actions générées se comportent en fait comme des *macro-actions* [6]. Pour résumer, l'exécution de FRAGs engendre effectivement un temps additionnel, qui reste cependant faible. De plus ce temps additionnel peut être compensé par une réduction du nombre d'états explorés si les séquences d'actions générées par les FRAGs apparaissent souvent consécutivement dans les solutions.

8 Conclusion et travaux futurs

Dans cet article, nous avons proposé un nouvel opérateur répondant au besoin de contextualiser des actions et leurs coûts associés. Nous avons vu que cette contextualisation est nécessaire pour générer des scénarios cohérents et qu'il ne suffit pas de rajouter des préconditions aux actions. C'est pourquoi, l'opérateur FRAG permet d'encapsuler un ensemble d'actions reliées entres elles par un graphe. Dans notre cas d'application, un FRAG est représentatif d'un fragment de scénario qui modélise une situation particulière. Par exemple une situation mettant l'apprenant en difficulté, ou une situation qui met l'apprenant face à un choix et mettra donc en jeu des compétences liées à la prise de décision. Ce fragment s'insère dans le scénario sous la forme d'une séquence d'actions. D'un point de vu plus théorique, il est possible d'utiliser les FRAGs pour connecter des actions qui apparaissent souvent dans les solutions et ainsi accélérer la planification. Pouvoir décrire ces situations sous la forme de modèles graphiques aux travers de FCMs permet également une meilleure intégration des connaissances experts. En effet, les FCMs sont des modèles connus pour leurs capacités à modéliser ce type de connaissances. Enfin, l'utilisation de graphes facilite la création des domaines de scénario par les experts, alors que des langages tels que PDDL sont moins abordable pour des non informaticiens.
Nos futurs travaux consistent entre autres à tester plus en profondeur la cohérence des scénarios générés. Nous souhaitons également tester la capacité des formateurs, non familier de la planification, à créer des scénarios à l'aide des FRAGs. Une des limitations de notre système est que les FRAGs génèrent des actions dont le coût ne varie pas en fonction des effets. Nous prévoyons donc de travailler sur la possibilité de définir des coûts variables.

9 Remerciements

Les auteurs remercient la Région Hauts-de-France et le FEDER 2014/2020 pour le cofinancement de ce travail. Celui-ci a été réalisé dans le cadre du projet VICTEAMS (ANR-14-CE24-0027), financé par l'ANR et la DGA. Il est labellisé par le Labex MS2T.

Références

[1] R. Aylett, S. Louchart, J. Dias, A. Paiva, and M. Vala. FearNot ! - An experiment in emergent narrative. In *Lecture Notes in Computer Science*, volume 3661 LNAI, pages 305–316. 2005.

[2] Ruth Aylett. Narrative in Virtual Environments - Towards Emergent Narrative. In *AAAI fall symposium on narrative intelligence*, pages 83–86, 1999.

[3] Eric Beaudry, Froduald Kabanza, and Francois Michaud. Using a Classical Forward Search to Solve Temporal Planning Problems under Uncertainty. Technical report, 2012.

[4] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2) :5–33, 2001.

[5] Blai Bonet, Gabor Loerincs, and Hector Geffner. A Robust and Fast Action Selection Mechanism for Planning. *Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence*, pages 714–719, 1997.

[6] Adi Botea, Markus Enzenberger, Martin Müller, and Jonathan Schaeffer. Macro-FF : Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, 24 :581–621, 2005.

[7] Marc Cavazza, Fred Charles, and Steven J. Mead. Character-Based Interactive Storytelling. *IEEE Intelligent Systems*, 17(4) :17–24, 2002.

[8] Dimitri De Franciscis. JFCM : A Java Library for FuzzyCognitive Maps. In Springer, editor, *Fuzzy Cognitive Maps for Applied Sciences and Engineering*, pages 199–220. 2014.

[9] Maria Fox and Derek Long. pddl2.1 : An Extension to pddl for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20 :61–124, 2003.

[10] Jörg Hoffmann. FF : The fast-forward planning system. *AI magazine*, 22 :57–62, 2001.

[11] Jörg Hoffmann. The metric-FF planning system : Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*, 20 :291–341, 2003.

[12] Florence Carre Ineris and Camilla Mörn Msb. Crisis and emergency situation Simulation considering cascading effects methodology. pages 1–28, 2017.

[13] Bart Kosko. Fuzzy cognitive maps. *International Journal of Man-Machine Studies*, 24(1) :65–75, 1986.

[14] Michael Lebowitz. Story-telling as planning and learning. *Poetics*, 14(1985) :483–502, 1985.

[15] Brian S Magerko. Player Modeling in the Interactive Drama Architecture. *Life Sciences*, pages 87–98, 2006.

[16] Jr Meehan. TALE-SPIN, An Interactive Program that Writes Stories. *Ijcai*, pages 91–98, 1977.

[17] Elpiniki I Papageorgiou and Jose L Salmeron. A review of fuzzy cognitive maps research during the last decade. *IEEE Transactions on Fuzzy Systems*, 21(1) :66–79, 2013.

[18] Julie Porteous and Marc Cavazza. Controlling narrative generation with planning trajectories : The role of constraints. *Lecture Notes in Computer Science*, 5915 LNCS :234–245, 2009.

[19] Julie Porteous, Alan Lindsay, Jonathon Read, Mark Truran, and Marc Cavazza. Automated Extension of Narrative Planning Domains with Antonymic Operators. *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)*, (Aamas) :4–8, 2015.

[20] MO Riedl. Narrative generation : balancing plot and character. *Journal of Artificial Intelligence Research*, 39 :217–268, 2004.

[21] Ulrike Spierling and Nicolas Szilas. Authoring issues beyond tools. *Lecture Notes in Computer Science*, 5915 LNCS :50–61, 2009.

[22] D Thue, V Bulitko, M Spetch, and T Romuanuik. Player agency and the relevance of decisions. In *Interactive Storytelling*, pages 210–150. 2010.

[23] R. Michael Young. Notes on the use of plan structures in the creation of interactive plot. *Narrative Intelligence - Papers from the 1999 AAAI Fall Symposium - TR FS-99-01*, (Walton 1990) :164–167, 1999.

[24] Alexander Zook, Stephen Lee-urban, Mark O Riedl, Heather K Holden, Robert A Sottilare, and Keith W Brawner. Automated Scenario Generation : Toward Tailored And Optimized Military Training In Virtual Environments. *Foundations of Digital Games'12*, page 9, 2012.