L'IA et les jeux vidéos à travers une simulation du jeu Dofus DofusAISim

Titouan Knockaert, Gaspard Goupy, Gabriel Filip

Université Claude Bernard Lyon 1, France

Résumé Si vous avez déjà joué aux jeux vidéo, vous avez sûrement connu ce sentiment d'amertume après une défaite contre l'ordinateur. Aujourd'hui, l'intelligence artificielle est très exploitée dans les jeux vidéo pour contrôler des agents autonomes : c'est notamment le cas dans les tactical RPG. En effet, dans ce type de jeu tour par tour, des groupes s'affrontent en prenant des décisions tactiques pour gagner la partie. Le joueur est donc souvent confronté à une IA. Nous nous sommes alors demandé comment sont implémentées ces IA dans les tactical RPG. Ce rapport est une synthèse de notre projet sur une étude de l'intelligence artificielle dans les jeux vidéo, à travers une simulation du jeu Dofus. L'objectif de ce projet était d'implémenter plusieurs IA pour contrôler les agents du jeu, via différentes stratégies : arbre de décision, algorithme de Leader, Q-learning. Ainsi, nous avons pu comparer les algorithmes entre eux pour mesurer leur efficacité et trouver une solution adaptée pour résoudre notre problème. Nous en avons conclu que les arbres de décision sont un moyen efficace et rapide pour implémenter des IA, tout en restant généraux. Cependant, ils sont assez limités par leurs comportements prédéfinis et prévisibles ainsi que de par leur rigidité. Dans un second temps, nous avons constaté que l'apprentissage par renforcement est un procédé plus flexible, qui peut s'avérer très efficace pour implémenter une intelligence artificielle solide, mais nécessite plus de moyens.

Keywords: tactical RPG · IA · Q-learning

1 Introduction

Au fil des années, l'intelligence artificielle est devenue de plus en plus présente dans les jeux vidéo, à tel point qu'elle est désormais utilisée dans la grande majorité des jeux actuels, principalement pour la gestion des entités PNJ (Personnage non-joueur, NPC en anglais). C'est un gage de qualité qui participe pleinement à l'immersion et l'expérience de jeu des joueurs. Parmi les jeux les plus connus, on peut citer la licence GTA [1], proposant un monde vivant, très riche, contrôlé par des IA.

Cependant, l'intelligence artificielle peut se révéler d'autant plus intéressante dans certains types de jeux : c'est le cas des tactical RPG [2]. Majoritairement basés sur un gameplay tour par tour, le joueur doit prendre des décisions tactiques lors de ses combats afin de gagner. On pourrait qualifier ce type de jeu comme une version moderne des échecs. Ainsi, l'intelligence artificielle est le principal adversaire du joueur, et permet d'ajouter de la difficulté au gameplay. Il est donc légitime d'affirmer que l'IA définit la qualité du jeu. On pourrait alors se demander comment les intelligences artificielles sont-elles implémentées dans les tactical RPG, et c'est de quoi ce rapport de projet traitera.

Ce projet est réalisé dans le cadre de l'enseignement Projet pour l'orientation en Master [3] du M1 informatique de l'université Lyon 1 [4]. Il est encadré par Samir Aknine et aborde le sujet de l'intelligence artificielle dans les tactical RPG. Il porte sur la réalisation d'une simulation simplifiée du jeu Dofus en version 1.29, en se focalisant sur les combats uniquement, permettant à deux groupes de s'affronter. L'objectif principal est de mettre au point différentes intelligences artificielles pouvant contrôler les agents du jeu, afin de pouvoir les étudier et les comparer.

2 Simulation Dofus 1.29

2.1 Présentation de Dofus 1.29

Dofus [5] est un MMORPG français développé et édité par la société Ankama sorti en 2004, où les joueurs incarnent des personnages définis par leurs classes [6]. Le but du jeu est de monter les niveaux de son personnage ainsi que de l'améliorer via de l'équipement et des compétences. Dofus 1.29 [7] est une version emblématique du jeu sortie en 2009, récemment remis à neuf en 2019, pour fêter les 15 ans du jeu. Dofus contient un système de combat au tour par tour, sur un damier. Chaque classe dispose d'habilitées propres et a un rôle précis parmi les suivants : DPS, tank, healer (voir 7.2). Pour ce projet, 4 des 12 classes ont été utilisées :

1. Le IOP : classe DPS spécialisée dans les buffs et les dégâts au corps à corps.
2. Le CRA : classe DPS spécialisée dans les dégâts à distance, les sorts de recul et les debuffs.
3. L'ENIRIPSA : classe healer spécialisée dans les soins et les buffs.
4. Le SACRIEUR : classe tank spécialisée dans l'encaissement de dégâts et les buffs.

2.2 Présentation des combats

Les combats dans Dofus se déroulent sur une grille de combat (illustré Figure 1), au tour par tour, entre deux groupes, chaque agent jouant pendant le tour qui lui est propre, durant un temps limité. Les personnages disposent de points d'actions (PA) leur permettant de lancer des sorts et des points de mouvements (PM) leur permettant de se déplacer sur la grille. Chaque sort consomme un nombre de PA qui lui est propre et se déplacer d'une case coûte un PM. Le nombre maximum de PA et de PM peut être augmenté ou diminué via des sorts de buff et de debuff. Les PA et PM sont régénérés à la fin de chaque tour, tout point non utilisé pendant un tour est donc perdu, car non cumulable avec les points du tour suivant.

Les personnages peuvent causer des dégâts et des soins aux autres, via leurs sorts. Un personnage meurt quand sa vie atteint 0. Un combat se termine quand l'un des deux groupes n'a plus de personnages en vie.

Chaque personnage a des caractéristiques améliorées via ses points de compétences et son équipement. Certaines caractéristiques augmentent les dégâts occasionnés en fonction de leurs types : feu, terre, eau, air. Les sorts sont donc classés par ces types. *Nb : cette partie n'est pas implémentée dans la simulation.*

Figure 1. Grille de combat du jeu Dofus

2.3 Présentation de la simulation

La simulation réalisée pour ce projet constitue une version simplifiée des combats de Dofus 1.29. Les combats se déroulent sur une grille, illustré Figure 2. Les cases grises sont des cases où les personnages peuvent se déplacer,

les cases noires sont des obstacles qui bloquent leur vision. Le groupe d'un personnage est identifiable par la couleur de son cercle : bleu ou rouge. Les règles sont les mêmes que dans le jeu Dofus 1.29, à l'exception près que la simulation peut autoriser tous les personnages du groupe à jouer en même temps. Un groupe est composé de 6 personnages au maximum. Les personnages disposent des caractéristiques suivantes : vie, résistance aux dégâts, PA, PM et appartiennent à une classe. Les 4 classes les plus jouées sont implémentées, chacune disposant de 7 sorts uniques, qui correspondent aux sorts principaux de la classe dans le jeu Dofus.

Figure 2. Grille de combat de la simulation

Les sorts sont classables par leur type :
- sort de dégât : sort qui inflige des dégâts à la cible.
- sort de soin : sort qui rajoute de la vie à la cible.
- sort de buff : sort qui améliore des caractéristiques de la cible pendant un nombre prédéfini de tours.
- sort de debuff : sort qui dégrade des caractéristiques de la cible pendant un nombre prédéfini de tours.

Les sorts ont aussi d'autres caractéristiques :
- PA : nombre de points d'action nécessaire pour lancer le sort.
- scope : distance minimum et maximum où le sort peut être lancé depuis la case du lanceur (distance de Manhattan).
- line scope : certains sorts ne peuvent être lancés que linéairement (cases sur la même ligne horizontale ou verticale).
- cooldown : nombre de tours empêchant le sort d'être lancé après une utilisation.
- scope zone : définit la taille de la zone où le sort sera appliqué (distance de Manhattan).
- attract : attire la cible d'un nombre de cases vers le lanceur. Si un obstacle se trouve sur son chemin, la cible est bloquée et des dégâts supplémentaires sont appliqués.
- push : pousse la cible d'un nombre de cases. Si un obstacle se trouve sur son chemin, la cible est bloquée et des dégâts supplémentaires sont appliqués.
- approach : téléporte le lanceur sur une case voisine à celle de la cible.

Les dégâts sont calculés avec la formule :

$$D = damage - damage * resistance \tag{1}$$

Les dégâts contre les obstacles (via des push / attract) sont calculés avec la formule :

$$D = 0.5 * damage * \frac{nbCasesTot - nbCasesCurr}{nbCasesTot} \tag{2}$$

Note : nbCasesTot étant le nombre de cases total à pousser/attirer la cible ; nbCasesCurr le nombre de cases où la cible a déjà été poussée/attirée

3 Travail réalisé

3.1 Simulation

La simulation est réalisée avec la plateforme de développement Unity [8], sous le moteur de jeu Unity 2D. Elle est composée de scènes, chacune contenant un ensemble de *GameObjects*, à qui on attache des scripts C# et des

composants : ce sont eux qui régissent le gameplay. Par exemple, pour gérer l'interface utilisateur, un *GameObject* est ajouté, des composants UI lui sont attachés (Ex : composant de texte) ainsi qu'un script permettant de mettre à jour ces composants. Le moteur de jeu Unity exécute une grande boucle et lit toutes les données d'une scène d'un jeu, en passant par chaque script, afin d'exécuter la simulation. Pour plus d'informations sur le fonctionnement d'Unity, voir [9].

Le *GameObject* chargé du bon fonctionnement de la simulation est le *BattleManager*. Il permet, entre autres, d'initialiser la simulation, de lancer les combats, de gérer les tours, les inputs utilisateur, les fins de combats, et d'appliquer les décisions prises par les agents.

Tous les agents sur la grille sont des *GameObjects*. Ils ont plusieurs scripts attachés qui fonctionnent indépendamment et parallèlement, avec des buts précis : par exemple, le script *Character* stocke les données relatives au personnage. Ainsi, chaque agent possède un script *AI*, classe de base qui se dérive en classes spécialisées : *SimpleAI*, *IopAI*, *CraAI*, *SacriAI* et *EniAI*. Le script *AI* permet de contrôler l'agent associé : sa fonction d'update est chargée d'exécuter l'IA spécialisée de l'agent lorsque c'est à son tour de jouer. Une exécution de l'IA correspond à une décision de l'agent. Elle sera donc exécutée jusqu'à que celui-ci ait terminé son tour, c'est-à-dire quand il ne peut plus jouer, ou lorsque le temps de tour est écoulé.

Les modèles des classes sont réalisés avec Universal-LPC-Spritesheet-Character-Generator [10].

3.2 Algorithme de recherche de plus court chemin

Les IA ont besoin d'avoir accès à un algorithme de plus court chemin efficace et exact, un choix est donc à faire entre deux algorithmes de plus court chemin : Dijkstra ou A*.

A* est très efficace mais ne propose pas toujours le chemin le plus court, il est donc écarté pour privilégier Dijkstra. Dijkstra est exact mais a une complexité polynomiale $\theta((a+n)*log(n))$, avec n le nombre de cases sur le damier et a le nombre d'arcs. Les agents effectuant plusieurs appels par tour, cet algorithme va créer des soucis de fluidité lorsque le damier est grand car il est trop complexe. De plus Dijkstra calcule les plus courts chemins d'une case A à toutes les autres cases du plateau, alors qu'un agent cherche un chemin entre une case A à une case B. Pour optimiser l'algorithme, trois solutions sont envisageables :

1. Calculer Dijkstra pour chaque case du plateau en début de combat et vérifier la validité du chemin renvoyé.
2. Stopper le calcul de Dijkstra lorsque un chemin trouvé entre la case A et B est égal à la distance de Manhattan entre les deux cases.
3. Créer un objet sauvegardant le résultat d'un appel à Dijkstra et en affecter un par personnage.

La solution 1 est écartée car elle demande un temps de calcul trop long en début de combat et peut nuire à sa fluidité. La solution 2 peut apporter des améliorations tout comme ne rien changer au temps de calcul. La solution 3 est donc retenue, avec le fonctionnement suivant :

1. Un appel à l'objet se fait en utilisant la case cible.
2. L'objet vérifie s'il connaît déjà un chemin toujours valide jusqu'à la case cible.
3. Si le personnage est sur une case du chemin, le chemin privé des cases déjà traversées est renvoyé.
4. Sinon Dijkstra est recalculé.

Cette solution diminue énormément la complexité en temps de la simulation, certaines IA effectuant plus de 5 appels par prise de décision à Dijkstra, la complexité passe donc de 5 exécutions de Dijkstra à 5 vérifications de chemins : cette opération ayant une complexité $\theta(c)$, avec c le nombre de cases du chemin.

3.3 IA naïve

La première intelligence artificielle implémentée est qualifiée de "simple". En effet, elle est commune à toutes les classes et a des comportements naïfs. Elle sert de base de comparaison et de test avec les autres approches implémentées. Son comportement est défini Figure 3 avec un arbre de décision.
Cette IA prend des décisions basiques : elle ne sait utiliser que des sorts d'attaque, ne prend pas en compte ses buffs, les déplacements des agents ennemis et alliés, etc. En revanche, elle a l'avantage d'être polyvalente et de fonctionner de manière générale, peu importe la classe.

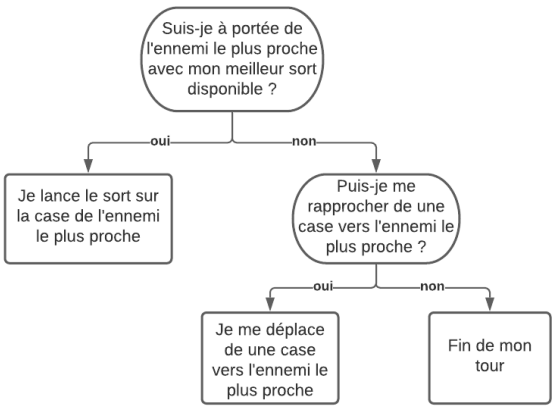

Figure 3. Arbre de décision - AI Simple

3.4 IA spécialisées

Les autres intelligences artificielles sont implémentées de manière spécialisée : elles sont propres à une classe précise. En effet, chaque classe a un gameplay différent, qui dépend majoritairement des sorts de la classe et de son type (healer, tank, dps). À savoir que le gameplay d'une classe n'est pas unique, il varie initialement selon le style de jeu du joueur, et d'autres critères non implémentés dans la simulation. Ici, les IA spécialisées cherchent à reproduire le gameplay type de la classe afin d'exploiter au mieux ses sorts, le but étant de pouvoir surpasser l'IA naïve. Quatre IA spécialisées sont implémentées pour les quatre classes de la simulation. Certains comportements, tels que tuer un ennemi, ou chercher à se mettre à portée d'un ennemi tuable, restent communs à toutes ces IA, car sont considérés comme de bonnes décisions de manière générale.

IOP Le IOP, DPS corps à corps, cherchent à toujours être buff, puis à infliger le maximum de dégâts aux ennemis les plus proches. Son arbre de décision est défini ici ou Figure 6.

CRA Le CRA, DPS à distance, essaye de rester en retrait afin que les ennemis ne puissent pas l'atteindre. Son arbre de décision est défini ici ou Figure 7. Un des plus gros atouts de cette IA est qu'elle combine l'utilisation de sorts de recul (qui font reculer la cible), et le fait de s'éloigner des ennemis si nécessaire. Elle est ainsi difficilement atteignable.

SACRIEUR Le SACRIEUR cherche à se rapprocher et attirer l'attention des ennemis pour encaisser un maximum de dégâts à la place de ses alliés. Pour cela, il utilise des sorts pour se téléporter sur ses ennemis, mais aussi pour les attirer vers lui. Son arbre de décision est défini ici ou Figure 8.

ENIRIPSA Le ENIRIPSA a pour but de rester derrière ses alliés, afin de ne pas prendre de dégâts et pouvoir prodiguer des soins. Son arbre de décision est défini ici ou Figure 9.

3.5 IA leader

L'intelligence artificielle du leader est implémentée en complément d'une IA naïve ou spécialisée. Il ne peut pas y avoir plus d'un seul leader actif dans un groupe. Il est désigné en début de combat, et à chaque fois qu'un leader actif meurt. Pour le désigner, un algorithme d'élection est appliqué, qui consiste à choisir un personnage healer en priorité, et avec le plus de vie courante. Le but du leader est de coordonner ses alliés via des ordres donnés. Il cherche à diviser le groupe en sous-groupes, pour que plusieurs alliés attaquent un même ennemi, et ainsi le tuer plus rapidement. Il demande aussi aux alliés avec peu de vie de battre en retrait et de se rapprocher d'un healer pour qu'ils se fassent soigner. Un ordre est composé de deux éléments : une liste de cibles (de taille 1 dans ce cas) et une case à atteindre. Le leader envoie les ordres à tous les alliés IA, qui vont exécuter l'arbre de décision Figure 4 avant le leur. Si une case est donnée, l'IA aura alors un ordre de déplacement. Si une liste de cibles est donnée, elle aura un ordre d'attaque, sinon elle fonctionnera de manière classique. À noter qu'un ordre de déplacement est prioritaire sur un ordre d'attaque.

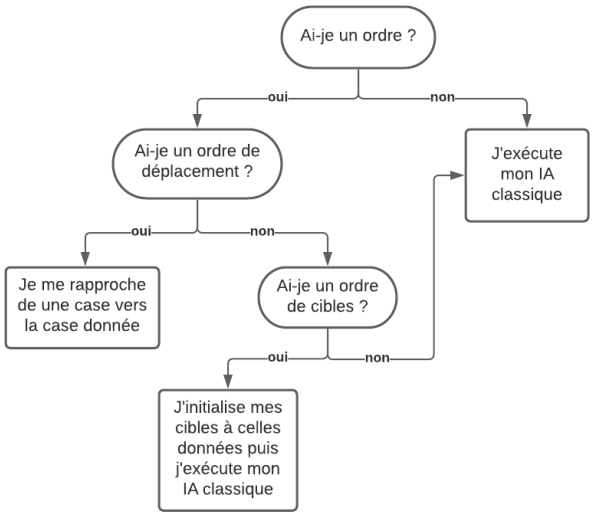

Figure 4. Arbre de décision - AI Leader

3.6 Q-learning

Les IA implémentées ont des comportements prédéfinis par leurs arbres de décision. Ces comportements sont considérés comme les plus adaptés dans une situation précise, et dépendent du gameplay ainsi que de la méta du jeu. Cependant, il est difficile de prévoir toutes les situations possibles. Et même si cela était le cas, il serait fastidieux de les intégrer à un arbre de décision, qui serait alors très complexe. C'est donc pourquoi les arbres de décision se veulent "généraux", afin de pouvoir s'adapter dans le plus de cas, mais causants ainsi des faiblesses dans l'IA. Néanmoins, en intelligence artificielle, il existe des méthodes permettant aux agents d'apprendre tout seul les meilleurs comportements à avoir : c'est le cas du Q-learning [11], une technique d'apprentissage par renforcement.

Fonctionnement du Q-learning Le Q-learning utilise un ensemble de couples (état, espérance) pour choisir le meilleur prochain coup possible. L'algorithme est capable de prévoir les différents états que le jeu peut prendre après avoir effectué une action, il choisit donc celle qui donnera l'état associé à l'espérance la plus haute. Pour créer cet ensemble de couples, l'algorithme passe par une première phase d'exploration où il choisit des actions au hasard et obtient des récompenses positives ou négatives en fonction des actions jouées, et de sa victoire ou non. Chaque action est donc associée à une récompense, de même pour la victoire. Tous les états traversés sont enregistrés dans un historique et l'espérance qui leur est associée est modifiée en fin de partie, grâce à la somme des récompenses acquises. Ainsi, les états menant à la victoire vont avoir des espérances plus hautes et ceux menant à la défaite vont avoir des espérances plus basses. Au fil des parties, l'algorithme diminue sa part d'exploration ϵ (pourcentage des actions jouées au hasard) et augmente sa part d'exploitation (pourcentage des actions jouées en choisissant l'état suivant avec la meilleure espérance), tout en continuant de modifier l'espérance associée aux états, suivant les récompenses obtenues. L'espérance des états est mise à jour toutes les X parties, avec la formule suivante :

$$v[state] = v[state] + \alpha * (\gamma * max(reward, v[nextState]) - v[state]) \tag{3}$$

v est un tableau d'espérances ; α est le taux d'apprentissage, il détermine à quel point la nouvelle information calculée surpassera l'ancienne ; γ est le taux d'actualisation, il détermine l'importance des récompenses futures.

Dans la simulation, un algorithme de Q-learning est implémenté dans un cadre basique, illustré Figure 5, à part des autres IA : un 1 vs 1 (2 agents) sur une grille de combat sans obstacle, et avec des actions limitées. Ces limitations sont volontairement définies afin de rendre l'implémentation plus simple. L'objectif est de découvrir le Q-learning à travers un cas pratique, et d'observer un apprentissage au fil des entraînements. Ici, les agents ont des actions prédéfinies : ne rien faire, réduire la distance, augmenter la distance, attaquer à longue distance, attaquer à courte distance, se soigner.

4 Expérimentations et discussion

Afin de mesurer l'efficacité des intelligences artificielles implémentées, plusieurs expérimentations sont mises en place. Le setup expérimental consiste à faire répéter 100 fois un combat entre deux groupes sur la grille du donjon 1. Afin de réduire les inégalités dues au premier groupe qui joue, le groupe 1 commence les 50 premiers combats, puis le groupe 2 les 50 suivants. Le temps d'expérimentation étant long et fastidieux, le procédé est automatisé : les combats s'enchaînent automatiquement et les résultats finaux sont sauvegardés dans un fichier. De plus, la vitesse de la simulation est augmentée.

4.1 IA naïve et spécialisée

Cette première expérience consiste à vérifier la supériorité d'une IA spécialisée face à une IA naïve.
Dans un premier temps, chaque classe s'affronte elle-même lors d'un 1vs1. L'agent du groupe 1 est contrôlé par une IA spécialisée, l'agent du groupe 2 par une IA naïve. Les résultats sont présentés Table 1.

Nb. wins	IOP	CRA	SACRIEUR	ENIRIPSA
IA naïve	0	1	0	0
IA spécialisée	100	99	100	100

Table 1. 1vs1 IA naïve et spécialisée pour la même classe

Les résultats sont unanimes : les IA spécialisées surpassent complètement l'IA naïve, en ce qui concerne leurs propres classes.

Dans un second temps, chaque classe va affronter une autre classe lors d'un 1vs1. L'agent du groupe 1 est une IA spécialisée, l'agent du groupe 2 est une IA naïve. Les résultats sont présentés Table 2, les cases représentant l'IA vainqueur du combat.

	IOP spé	CRA spé	SACRIEUR spé	ENIRIPSA spé
IOP naïf	spé	spé	spé	spé
CRA naïf	spé	spé	spé	?
SACRIEUR naïf	spé	spé	spé	naïf
ENIRIPSA naïf	spé	spé	spé	spé
Nb. wins IA spé	4/4	4/4	3/4	2/3

Table 2. 1vs1 IA naïve et spécialisée pour toutes les classes

Dans la quasi-totalité des cas, les IA spécialisées surpassent l'IA naïve, peu importe la classe et qui commence la partie. Ces résultats sont logiques et prévisibles, car l'IA naïve est très générale et ne prend pas en compte les particularités de la classe dans ses comportements. Il semble tout de même important de mentionner que l'IA spécialisée SACRIEUR perd contre l'IA naïve IOP dans le cas où le SACRIEUR commence à jouer, avec 16 victoires sur 50. À l'inverse, quand il joue en deuxième, il gagne avec 44 victoires sur 50. Dans ce cas, la victoire dépend grandement de qui est le premier agent qui joue, ainsi que de la disposition de la grille de combat. À noter que l'expérience entre l'IA spécialisée ENIRIPSA contre l'IA naïve CRA ne peut pas aboutir, car la plupart des combats ne se finissent pas. En effet, l'IA naïve CRA cherche à attaquer dès que possible, et ne sait pas prendre la décision de se déplacer pour utiliser un meilleur sort s'il est déjà à portée de l'ENIRIPSA. Il arrive donc que le CRA inflige des dégâts qui peuvent être à chaque fois compensés par les sorts de soin de l'ENIRIPSA.

4.2 IA spécialisée

Cette expérience consiste à comparer les différentes IA spécialisées entre elles. Chaque classe va affronter les autres classes lors d'un 1v1. Les résultats sont présentés Table 3, les valeurs des cases représentent le vainqueur avec le nombre de victoires de chacun, sur un total de 100.
En 1vs1, la meilleure classe est le CRA, suivie du IOP, puis du SACRIEUR et enfin de L'ENIRIPSA. Ces résultats sont normaux, premièrement car le CRA et le IOP sont des DPS, ce sont ceux qui infligent le plus

	IOP	CRA	SACRIEUR	ENIRIPSA
IOP	IOP (51,49)	CRA (83,17)	IOP (79,21)	IOP (100, 0)
CRA	CRA (83,17)	CRA (55,45)	CRA (66,34)	?
SACRIEUR	IOP (79,21)	CRA (66,34)	SACRIEUR (52,48)	SACRIEUR (86, 14)
ENIRIPSA	IOP (100,0)	?	SACRIEUR (86,14)	?

Table 3. 1vs1 IA spécialisées pour toutes les classes

de dégâts. L'IA spécialisée du CRA surperforme celle du IOP car elle arrive à le maintenir à distance, tout en attaquant. L'ENIRIPSA est dernier au classement car c'est un healer, il n'a pas de sorts d'attaque puissants. Il faut la aussi noter que 2 combats ne peuvent pas aboutir : CRA vs ENIRIPSA et ENIRIPSA vs ENIRIPSA, car dans certains cas, leurs IA ne peuvent pas prendre la décision d'avancer vers l'ennemi. Pour conclure sur cette expérience, certaines classes telles que le SACRIEUR ou l'ENIRIPSA n'ont pas beaucoup d'intérêt en 1vs1, et devraient être plus utiles dans une équipe.

Pour montrer l'utilité d'un healer dans un groupe, l'expérience suivante est réalisée : des combats entre deux groupes similaires, en remplaçant un agent par un ENIRIPSA. En pratique, il n'y a pas de grosses différences observées sur des petits groupes, le nombre de victoires de chacun est plus ou moins similaire. En revanche, pour de gros groupes, cela va dépendre aussi des classes des autres agents. Par exemple, pour un combat 4vs4, groupe 1 (2xIOP, 2xCRA), groupe 2 (1xIOP, 2xCRA, 1xENIRIPISA), le groupe 2 obtient 69 victoires contre 31. L'utilité du healer est donc prouvée. Pour un combat 4vs4, groupe 1 (4xCRA), groupe 2 (3xCRA, 1xENIRIPISA), le groupe 2 obtient 37 victoires contre 63. Ici, le groupe 1 occasionne trop de dégâts pour que le healer soit efficace.

Pour montrer l'utilité d'un tank dans un groupe, l'expérience suivante est réalisée : des combats entre deux groupes similaires, en remplaçant un agent par un SACRIEUR. Là encore, il semblerait que l'efficacité d'un tank dans le groupe dépende aussi des classes des autres agents et de leur nombre. Pour des petits groupes, un tank semble utile contre des ennemis allant au corps à corps (IOP), mais pas contre des ennemis restant à distance (CRA). Pour de gros groupes, l'ajout d'un tank est assez intéressant : par exemple, pour un combat 6vs6, groupe 1 (3xIOP, 2xCRA, 1xSACRIEUR), groupe 2 (4xIOP, 2xCRA), le groupe 1 obtient 77 victoires contre 23.

Pour conclure, grâce à cette expérience, les IA spécialisées ont prouvé leur efficacité quant aux rôles de leurs agents : DPS, healer, tank. Chacune de ces IA cherche à exploiter au mieux les capacités uniques de la classe jouée afin d'être la plus utile possible pour son groupe. Il est important de garder à l'esprit, qu'en pratique, l'utilité d'un tank et d'un healer, à l'inverse d'un DPS, dépend majoritairement de la composition des groupes (classes et nombre) : c'est ce qui constitue la méta du jeu. Ces IA ont tout de même certaines limites. Elles manquent d'aléatoire dans leurs décisions, ce qui rend leurs mouvements prévisibles. Certaines IA (CRA et ENIRIPSA) ne sont pas toujours capables de prendre des décisions faisant avancer le combat (et donc le bloque). Finalement, l'arbre de décision de l'IA doit rester un minimum général, pour réduire le couplage à la classe, et ne pas être trop complexe. Ainsi, il serait difficile d'obtenir une IA supérieure à un humain. De plus, c'est aussi à l'humain de définir son comportement, la rendant dépendante de la méta du jeu.

4.3 IA Leader

Cette expérience vise à mesurer l'efficacité de l'IA Leader. Pour cela, plusieurs parties sont réalisées avec les mêmes agents dans chaque groupe, le groupe 1 possédant une IA Leader. Une partie des résultats sont présentés Table 4.

	Nb. wins groupe 1	Nb wins groupe 2	Vainqueur
3xIOP	41	59	Groupe sans leader
2xIOP, 2xCRA	22	78	Groupe sans leader
2xIOP, 1xENIRIPSA	36	64	Groupe sans leader
1xIOP, 1xCRA, 1xENIRIPSA	55	45	Groupe avec leader
3xIOP, 1xCRA, 1xENIRIPSA	23	77	Groupe sans leader
1xIOP, 2xCRA, 2xSACRIEUR, 1xENIRIPSA	36	64	Groupe sans leader

Table 4. Tests de IA leader

Dans la majorité des cas, le groupe possédant une IA leader perd le combat. À partir de l'ensemble des tests réalisés sur l'IA leader, son pourcentage de victoire est estimé à 35% : cette IA n'est donc pas efficace. Cela

s'explique par des faiblesses dans son algorithme. Premièrement, un agent avec peu de vie aura comme ordre de se déplacer vers un healer, l'empêchant alors d'attaquer. Les dégâts non infligés par cet agent lors de son tour feront prendre l'avantage au groupe ennemi. Cette idée de repli pourrait fonctionner dans le cas où l'agent serait entièrement guéri par le healer, qui actuellement ne peut soigner que l'équivalent d'une ou de deux attaques par tour. Deuxièmement, l'algorithme d'assignation des cibles ne prend pas en compte les sorts de chaque allié. Ainsi, certains agents auront une cible qu'ils ne pourront pas atteindre avec leur meilleur sort disponible pour ce tour, alors qu'ils auraient pu en toucher une autre avec. Pour conclure, il semblerait que l'IA leader soit inefficace dans la simulation de par son manque de complexité mais aussi par rapport au contexte : elle serait plus utile dans un combat où les groupes seraient beaucoup plus nombreux, avec plus de vie, et des healers différents.

4.4 Q-learning

Cette dernière expérience se déroule dans un cadre particulier, l'IA Q-learning étant implémentée pour un 1vs1 uniquement, sur une grille sans obstacle. L'objectif principal est de pouvoir montrer qu'un apprentissage est fait au fur et à mesure des entraînements. Pour cela, il faut d'abord entraîner l'IA contre elle-même, afin de lui faire apprendre les comportements basiques à avoir pour gagner un combat. Ensuite, elle pourra s'entraîner contre une IA simple afin de s'améliorer. Les paramètres de l'algorithme seront les suivants pour toutes les expériences qui vont suivre : facteur d'apprentissage $\alpha = 0.001$, facteur d'actualisation $\gamma = 1$, les récompenses de toutes les actions sont fixées à 0, et celle de la victoire à 1.

Dans un premier temps, l'IA va faire un entraînement de 20,000 combats contre elle-même. Pour prouver qu'un apprentissage est fait, une méthode est de mesurer le temps moyen d'un combat, en vitesse accéléré, sur 10 combats. Le tableau 5 présente les résultats.

Combats	Temps moyen d'un combat
1-10	19.5s
100-110	10.4s
1000-1010	5.1s

Table 5. Entraînement IA Q-Learning vs IA Q-learning - Test sur la durée des combats

Pour conclure sur cet entraînement, le temps moyen d'un combat baisse au fur et à mesure des parties, ce qui montre que l'IA apprend le comportement à avoir pour gagner. À noter que cette mesure n'est pas totalement fiable : certains combats sont plus longs car les agents peuvent se soigner, et donc faire durer la partie.

Maintenant que l'IA devrait être plus intelligente, elle va pouvoir faire un entraînement de 500 combats contre une IA simple. Pour mesurer l'apprentissage, le taux de victoires va être calculé entre les combats 1-100, 101-250, 251-500. Le tableau 6 présente les résultats. Pour conclure sur cet entraînement, le taux de victoire

Combats	Taux de victoire IA-Qlearning
1-100	34%
101-250	61%
251-500	68%

Table 6. Entraînement IA Q-Learning vs IA simple - Test sur le taux de victoire

d'un combat augmente au fil des parties, ce qui montre que l'IA apprend le comportement à avoir pour contrer une IA simple. Il est important de garder à l'esprit que l'IA simple a un pattern prédéfini : elle aura toujours le même comportement dans une situation donnée. Ainsi, entraîner l'IA Q-learning contre l'IA simple revient à lui faire apprendre à jouer uniquement contre cette IA. Si le comportement de l'IA simple venait à changer, l'apprentissage ne serait plus forcément valide. De plus, l'apprentissage est assez aléatoire car il dépend en partie des actions jouées aléatoirement.

La dernière expérience sur l'IA Q-learning vise à mesurer son efficacité au fil des entraînements. Plusieurs sauvegardes de l'IA ont été faites et vont affronter une IA simple sur 100 combats, avec un taux d'exploration ϵ égal à 0.1, ce qui correspond à une probabilité de jouer au hasard de 10%. Les résultats sont présentés Table 7, et confirment bien l'efficacité de l'apprentissage fait par l'algorithme Q-learning.

	Nb. wins IA Q-learning	Nb wins IA Simple
0 entraînements	0	100
250 entraînements IA Q-learning	1	99
2,000 entraînements Q-learning	4	96
20,000 entraînements Q-learning	32	68
20,000 entraînements Q-learning et 100 entraînements IA simple	44	56
20,000 entraînements Q-learning et 250 entraînements IA simple	61	39
20,000 entraînements Q-learning et 500 entraînements IA simple	64	36

Table 7. Tests IA Q-learning vs IA simple

Limites de l'implémentation du Q-learning La limite la plus importante du Q-learning est le nombre d'états possibles dans le jeu. Dans la version simplifiée de la simulation pour le Q-learning, les états sont eux aussi simplifiés afin de pouvoir tous les explorer. Un état est représenté par la distance entre les deux agents (entre 1 et 19), la vie courante de l'agent et celle de l'agent ennemi (représentées entre -1 et 10). Ainsi, le nombre d'états total est inférieur à 2300. À titre de comparaison, si un état contenait toutes les informations à savoir : positions sur la grille (x,y) des agents, points de vie exacts des agents, il y aurait alors plus de 225 millions d'états différents pour une version déjà simplifiée, et où l'ajout d'un nouvel agent multiplierait par 15 000 le nombre d'états possibles. Cette simplification entraîne néanmoins des modifications : faire correspondre les actions aux états. Par exemple, plus de déplacement en x ou en y mais simplement la possibilité de s'approcher ou de s'éloigner de l'ennemi, ceci empêchant toutes les stratégies qui consistent à utiliser un obstacle comme protection, ce qui explique pourquoi la grille utilisée n'en contient pas. De plus, plus la représentation des états définit aussi l'intelligence de cette IA : plus les états sont complexes, plus l'IA devrait être efficace, mais plus l'apprentissage devrait être long. Il faut donc trouver un juste milieu.

Amélioration possibles Q-learning Le meilleur moyen d'éliminer le problème du nombre d'états est d'utiliser un réseau de neurones, utilisant un état en entrée et donnant une espérance en sortie. C'est l'algorithme Deep Q-learning, qui a notamment été utilisé dans les projets de la société Deepmind tels que Alphago.

5 Conclusion

Ce rapport est l'aboutissement du projet DofusAISim, réalisé dans le cadre de l'enseignement Projet pour l'orientation en Master du M1 informatique de l'université Lyon 1. L'objectif de ce projet était de réaliser une simulation des combats du jeu Dofus, afin de mettre aux points différentes intelligences artificielles pouvant contrôler les agents du jeu : IA simple, IA spécialisées, IA leader et IA Q-learning. Des expérimentations ont ensuite été réalisées pour mesurer l'efficacité des différentes IA. Les IA spécialisées ont largement surpassé l'IA simple, car elles permettent de jouer leurs classes en suivant la méta du jeu. Des différences de niveau logiques ont été observées entre les IA spécialisées, ce qui est dû au rôle de la classe. Cependant, elles ont pu prouver leur utilité dans un groupe, à condition que celui-ci soit suffisamment gros. Ces IA ont tout de même quelques limites : elles doivent rester générales pour mieux s'adapter à leur classe, ne pas être trop complexes, et ont des comportements prédéfinis par leur concepteur, donc prévisibles. L'IA du leader ne s'est pas avérée suffisamment efficace, premièrement dû à son manque de complexité, mais aussi au contexte de la simulation : elle aurait plus d'utilité dans des groupes avec beaucoup d'agents à gérer, et où les healers auraient un gameplay différent. Finalement, les tests sur l'IA Q-learning ont prouvé qu'un apprentissage a bien été réalisé au fil des entraînements, permettant à l'IA d'apprendre les comportements à avoir pour gagner. Elle a ainsi pu obtenir un taux de victoire maximum de 70% face à une IA simple. L'apprentissage par renforcement semble être une bonne méthode pour s'adapter à une méta de jeu qui évolue, mais nécessite du temps pour l'apprentissage.

Pour conclure, ce projet a abordé différentes manières d'implémenter des intelligences artificielles dans les tactical RPG, montrant l'efficacité des arbres de décision pour contrôler un agent, mais aussi ses limites. Il semblerait qu'utiliser des algorithmes d'apprentissage par renforcement, notamment via le Deep Q-learning, serait une manière plus flexible et efficace pour contrôler un agent du jeu. Ce sujet pourrait faire l'oeuvre d'un futur projet.

6 References

1. Grand Theft Auto https://fr.wikipedia.org/wiki/Grand_Theft_Auto
2. Tactical RPG https://fr.wikipedia.org/wiki/Tactical_RPG
3. Site de l'université Lyon 1 https://www.univ-lyon1.fr/
4. Page de l'UE POM https://perso.liris.cnrs.fr/sbrandel/wiki/doku.php?id=ens:pom
5. Présentation du jeu Dofus https://fr.wikipedia.org/wiki/Dofus
6. Classes du jeu Dofus https://www.dofus.com/fr/mmorpg/encyclopedie/classes
7. Dofus 1.29 https://www.dofus.com/fr/mmorpg/actualites/news/587778-dofus-1-29
8. Unity https://unity.com/fr
9. Fonctionnement de Unity https://unity3d.com/fr/learning-c-sharp-in-unity-for-beginners
10. Character Generator http://gaurav.munjal.us/Universal-LPC-Spritesheet-Character-Generator/#
11. Q-learning https://fr.wikipedia.org/wiki/Q-learning

7 Annexes

7.1 Projet

Gitlab du projet : ici

Clone Github : ici

Builds du projet : ici

7.2 Lexique

1. Gameplay : caractéristiques d'un jeu vidéo et la façon dont on y joue.
2. Classe : spécialité d'un personnage, lui donnant des caractéristiques particulières. (sorts, vie, etc.)
3. DPS : personnage en charge d'infliger un maximum de dégâts auprès du camp adverse.
4. Tank : personnage en charge d'encaisser un maximum de dégâts venant du camp adverse.
5. Healer : personnage en charge de soigner ses alliés.
6. Buff : sort améliorant les caractéristiques d'un personnage (résistance, vie, etc.)
7. Debuff : sort dégradant les caractéristiques d'un personnage (résistance, vie, etc.)
8. Lore : l'histoire d'un univers de fiction.
9. Meta : ensemble des stratégies et des méthodes qui ne sont pas explicitement définies, mais qui résultent de la seule expérience des joueurs qui émerge en dépit de ce que les développeurs avaient prévu.
10. MMORPG : jeu de rôle en ligne massivement multijoueur.

7.3 Images

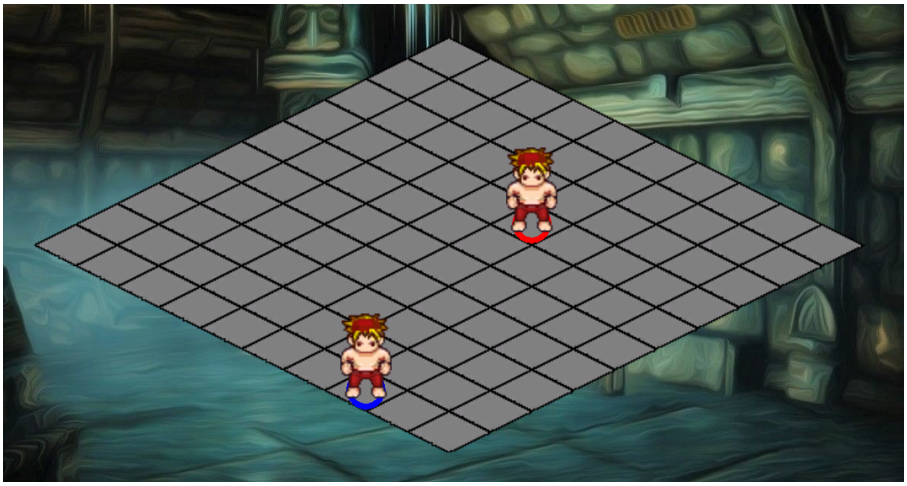

Figure 5. Grille de combat pour le 1vs1 avec Q-learning

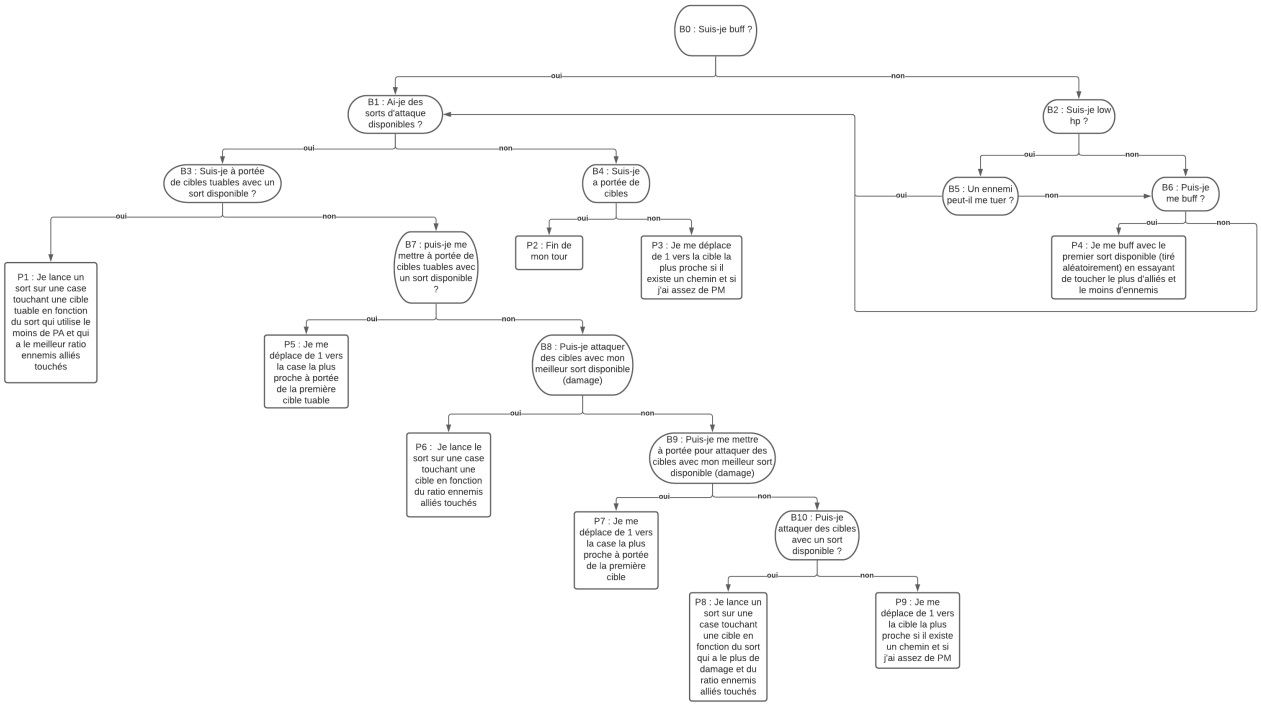

Figure 6. Arbre de décision - AI IOP

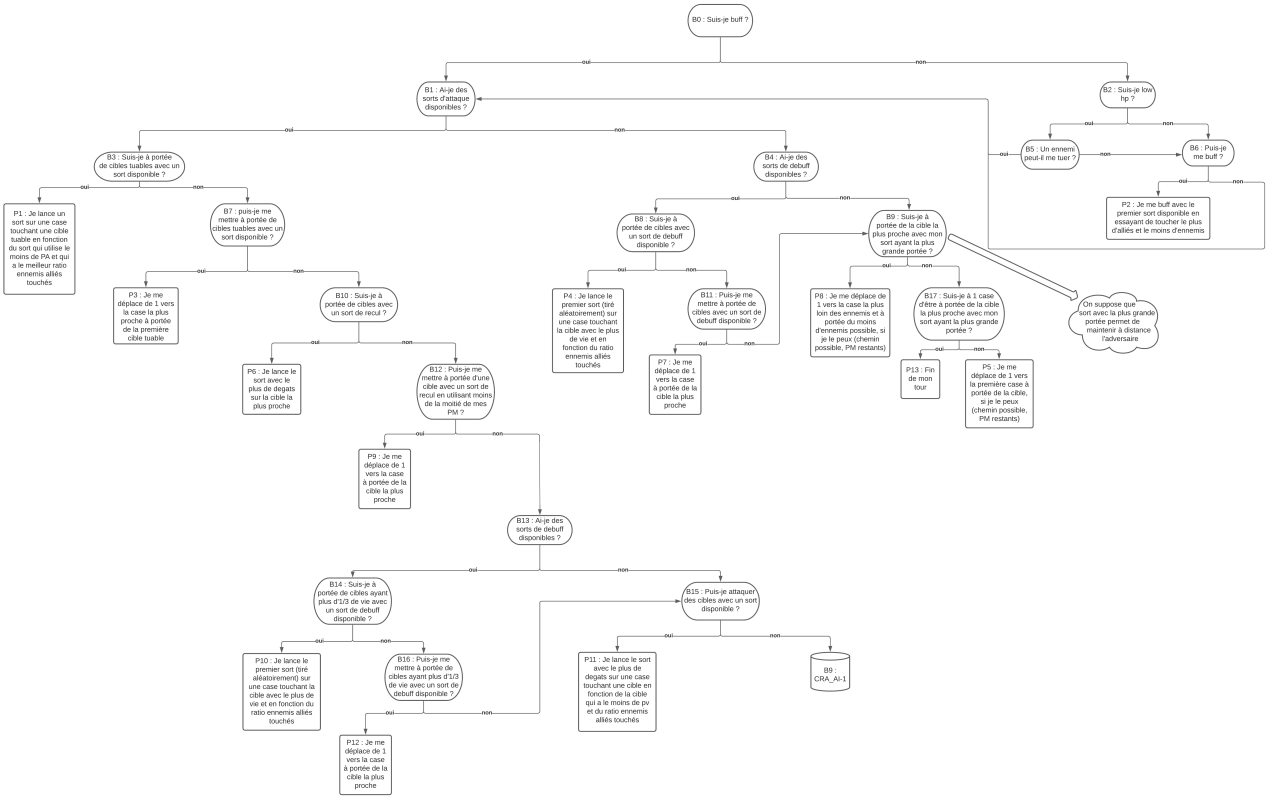

Figure 7. Arbre de décision - AI CRA

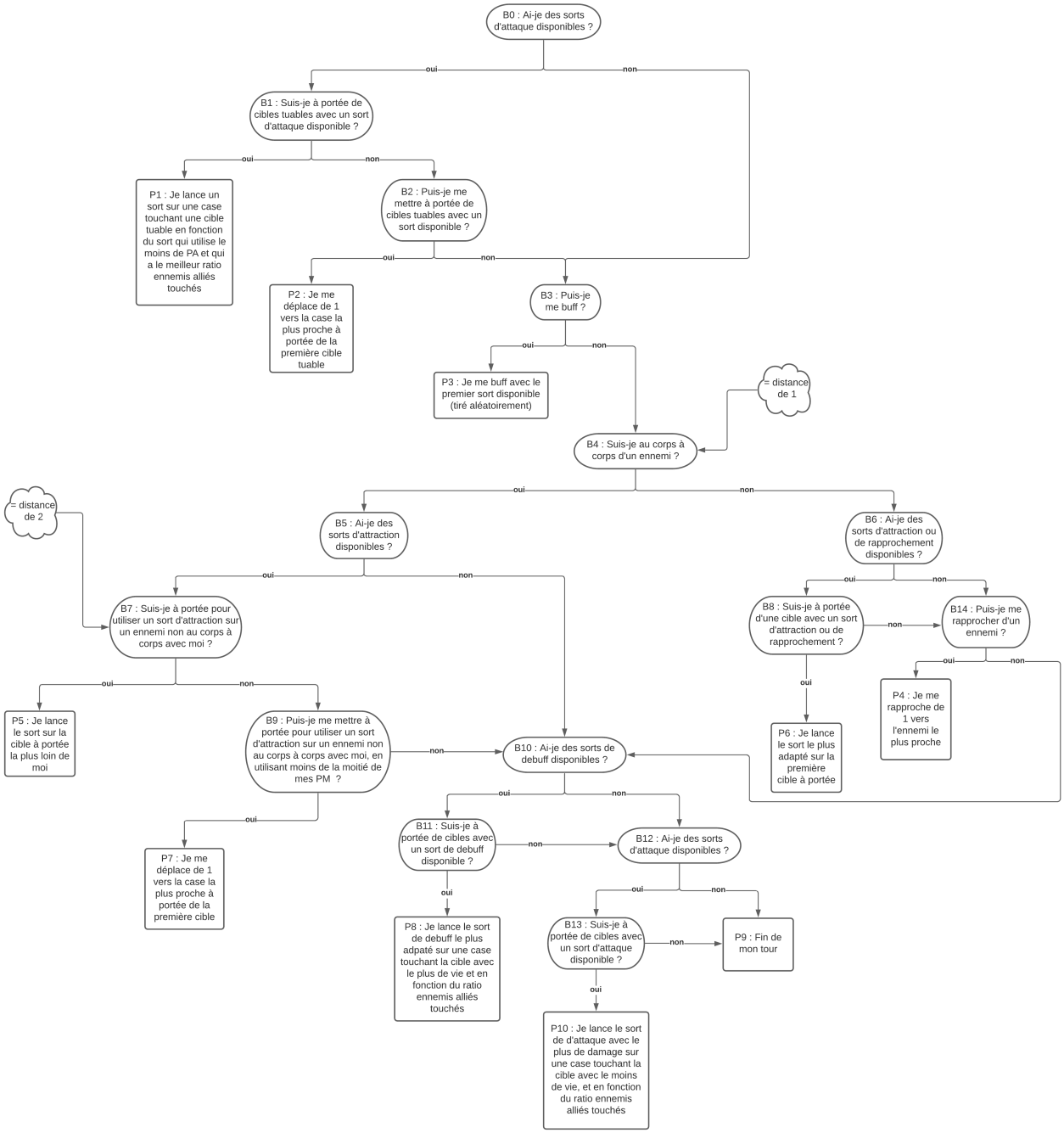

Figure 8. Arbre de décision - AI SACRIEUR

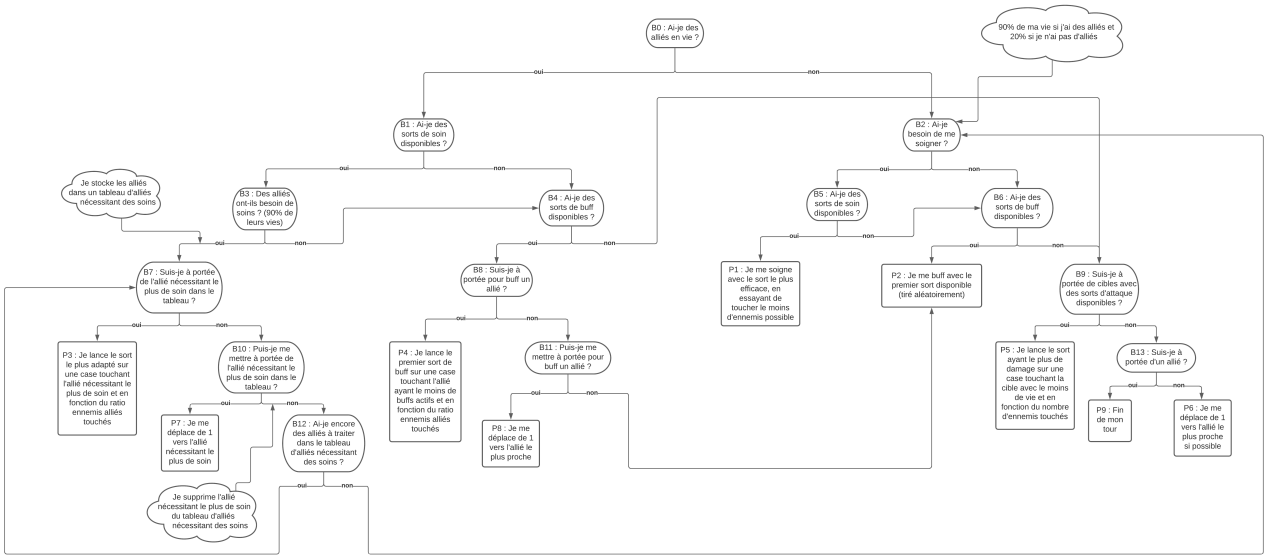

Figure 9. Arbre de décision - AI ENIRIPSA