Titre: Title:	Paramétrisation d'une méthode de production pilotée par la demande avec un algorithme d'apprentissage par renforcement
Auteur: Author:	Louis Duhem
Date:	2022
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Duhem, L. (2022). Paramétrisation d'une méthode de production pilotée par la demande avec un algorithme d'apprentissage par renforcement [Master's thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/10549/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Paramétrisation d'une méthode de production pilotée par la demande avec un algorithme d'apprentissage par renforcement

LOUIS DUHEM
Département de mathématiques et de génie industriel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie industriel

Septembre 2022

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Paramétrisation d'une méthode de production pilotée par la demande avec un algorithme d'apprentissage par renforcement

présenté par **Louis DUHEM**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Pierre BAPTISTE, président
Maha BEN ALI, membre et directrice de recherche
Guillaume MARTIN, membre et codirecteur de recherche
Jean-Marc FRAYRET, membre

REMERCIEMENTS

J'aimerais remercier ma directrice de recherche Maha Ben Ali ainsi que mon codirecteur Guillaume Martin pour leur accompagnement dans ce projet. Ils ont su m'orienter et m'aider à mener ce projet à terme, et ce malgré les fuseaux horaires nous ayant séparés.

RÉSUMÉ

Avec l'augmentation des exigences clients et une demande plus diversifiée, le domaine industriel fait face à de nouveaux enjeux en termes de production et satisfaction client. Il s'agit d'optimiser les niveaux de stock tout en réduisant les temps de livraison. Néanmoins, les anciennes méthodes de production ne sont plus suffisantes pour répondre à ces nouveaux objectifs. Les industriels cherchent donc à innover et s'intéressent à de nouvelles méthodes basées sur de nouvelles règles de production. Le *Demand Driven Material Requirements Planning* (ou DDMRP) est un méthode de production pilotée par la demande qui s'inscrit dans cette nouvelle aire d'innovation industrielle. Basée sur des règles simples, elle tend à réduire les niveaux de stock et les temps de livraison en s'appuyant sur la demande réelle. Cependant, cette méthode est encore assez récente. Par conséquent, sa documentation est encore assez limitée. En particulier, sa paramétrisation lui fait défaut : il peut paraître difficile, dans le cas d'une implémentation du DDMRP dans un cas pratique, de paramétrer correctement l'algorithme.

Le sujet de cette étude est la paramétrisation de la méthode DDMRP à l'aide d'un algorithme d'apprentissage par renforcement. Ce projet vise à apporter des ajustements dynamiques à des seuils et horizons de détection de pics afin d'assister et améliorer la paramétrisation de la méthode de production. Un agent d'apprentissage par renforcement est implémenté dans un environnement géré en DDMRP. Cet agent vise également à augmenter les performances du DDMRP. Il est notamment question d'optimiser les stocks et améliorer la satisfaction client. L'agent, dérivé et adapté d'un algorithme de *Branching Deep Q-Learning*, est intégré à une modélisation d'atelier géré en DDMRP. Cet atelier est représenté à l'aide d'une simulation à événements discrets et est repris d'un modèle déjà existant.

Le projet est entièrement réalisé avec le langage Python sur l'éditeur de code VS Code. Nous utilisons les librairies classiques de Python, et intégrons en particulier la librairie Keras du module Tensorflow afin de créer l'agent. L'agent développé est testé sur des environnements d'essai avant d'être intégré à la simulation en DDMRP. Il est intégré à un modèle sans agent, qui correspond au modèle de Ptak et Smith. Une fois l'agent intégré et validé, nous effectuons d'abord des expériences en lien avec les paramètres de l'apprentissage par renforcement. Ces expériences nous permettent de construire un agent efficace dans un environnement DDMRP. Ensuite, nous cherchons à identifier les cas métiers dans lesquels l'apprentissage par renforcement est le plus efficace. Cela nous permet de démarquer les profils de demande les plus adaptés à l'utilisation de l'apprentissage par renforcement.

Les résultats sont principalement analysés à l'aide des indicateurs de l'apprentissage par renforcement, relatant de la performance de l'agent. Ces indicateurs sont également comparés à ceux du modèle sans agent, qui sert de base à notre étude. Les résultats de simulation ont montré qu'il est possible d'intégrer à un agent à une simulation d'un atelier géré en DDMRP. L'agent est non seulement capable d'apprendre dans un tel environnement, mais il est aussi apte à dépasser les performances d'un modèle sans agent. De plus, les résultats montrent que l'agent construit et étudié dans ce travail est efficace dans des environnements à fréquence de pics faible. Si la fréquence de pics augmente, un atelier avec une charge de travail élevée peut nuire à sa performance.

ABSTRACT

Confronted with more demanding customers and a more diversified demand, the industrial field faces new stakes regarding production and customer satisfaction. The stakes are to optimize the stocks while reducing delivery time. Nevertheless, the former manufacturing methods can't reach these new goals. The industrials attempt to innovate and find new methods with simple production rules. The *Demand Driven Material Requirements Planning* (DDMRP) is a demand-driven production method which belongs to this new industrial innovation. Based on simple and consistent rules, this method attempts to limit the stocks levels and reduce delivery time by using real demand. However, this method is still new. Consequently, its literature is limited. Particularly, its parametrization defaults : it seems difficult to correctly set up the algorithm, especially in a real world case study.

The subject of this study is the parametrization of the DDMRP method with a reinforcement learning algorithm. This project attemps to bring dynamic adjustments to order spike thresholds and horizon spike thresholds. It is done to assist and improve the parametrization of the production method. A reinforcement learning agent is implemented in a DDMRP-driven environment. This agent also aims to increase the DDMRP performance. It tries to optimize the stocks and improve customer satisfaction. The agent, which is inspired et adapted form an *Branching Deep Q-Learning* algorithm, is integrated to a workshop model run by DDMRP. This workshop is modeled by a discret event simulation. It is adapted from a model which already exists.

The project is entirely done with the Python language with the code editor VS Code. We use standard Python libraries, and we add the Keras library of the Tensorflow module to make the agent. The developed agent is tested on "dummy" environments before we use it in the DDMRP model. It is integrated to an agent-free model, which matches withe the Ptak and Smith's model. Once the agent is integrated and validated, we run experiments regarding the reinforcement learning parameters. These experiments allow us to create an efficient agent in a DDMRP-driven environnement. Then, we aim to identify the real world case studies which give us the best reinforcement learning results. This lets us identify the most adapted demand streams for the reinforcement learning.

The results are mostly analysed using reinforcement learning indicators, relating the agent's performance. These indicators are also compared to the results of an agent-free model, which we use as a baseline. The project shows it is possible to integrate an agent to a simulation with a DDMRP-driven workshop. The agent can learn in this environement, and it can also

be better than a model-free agent. The built and studied agent in this study is efficient in environements with low spikes frequency. If the spikes frequency increases, a workshop with a high workload can jeopardize its results.

TABLE DES MATIÈRES

REMERCIEMENTS iii

RÉSUMÉ iv

ABSTRACT vi

TABLE DES MATIÈRES viii

LISTE DES TABLEAUX xi

LISTE DES FIGURES xii

LISTE DES SIGLES ET ABRÉVIATIONS xiv

LISTE DES ANNEXES xv

CHAPITRE 1 INTRODUCTION 1
1.1 Mise en contexte 1
1.2 Objectifs 2
1.3 Structure du mémoire 2

CHAPITRE 2 REVUE DE LITTÉRATURE 3
2.1 DDMRP 3
2.1.1 Une nouvelle méthode de production 3
2.1.2 Fonctionnement du DDMRP 5
2.1.3 Ajustements dynamiques 7
2.1.4 Le DDMRP dans la littérature 10
2.2 Apprentissage par renforcement 12
2.2.1 Définitions 12
2.2.2 Concepts de base 13
2.2.3 Exemples d'application 14
2.2.4 Apprentissage par renforcement et DDMRP 16
2.3 Réseaux de neurones 16
2.3.1 Définition 16
2.3.2 Exemples d'application 17
2.4 Conclusion 18

CHAPITRE 3 CAS D'ÉTUDE ET APPROCHE MÉTHODOLOGIQUE 19
3.1 Cas d'étude 19
3.1.1 L'atelier de production de type hybride 19
3.1.2 Les classes du modèle 20
3.1.3 Intégration de l'agent 21
3.2 Méthodologie générale 23
3.3 Conclusion 25

CHAPITRE 4 MODÉLISATION ET IMPLÉMENTATION 26
4.1 Choix inhérents au modèle 26
4.1.1 Espace d'action et espace d'état 26
4.1.2 Fonctions récompense 29
4.1.3 Choix de l'algorithme d'apprentissage par renforcement 31
4.2 Description fonctionnelle du modèle 33
4.2.1 Modèle de l'atelier de production géré en DDMRP 33
4.2.2 Apprentissage par renforcement 36
4.3 Implémentation du modèle 38
4.3.1 Modèle conceptuel 38
4.3.2 Adaptation du modèle DDMRP pour le rendre compatible avec une composante d'intelligence artificielle 39
4.3.3 Création et validation de l'agent d'apprentissage par renforcement . . 41
4.4 Intégration de l'agent au modèle 43
4.5 Vérification et validation du modèle 44
4.5.1 Indicateurs clés de performance 44
4.5.2 Vérification du modèle 46
4.5.3 Validation et optimisation du modèle 46
4.6 Conclusion 50

CHAPITRE 5 EXPÉRIMENTATION 51
5.1 Plan d'expériences 51
5.1.1 Comparaison de différentes façons de paramétrer le système 52
5.1.2 Analyse de l'effet de facteurs externes (charge de travail et fréquence des pics de demande) 53
5.1.3 Paramétrage de l'expérimentation 54
5.2 Expérience 1 : variation des OST et des OSH 54
5.3 Expérience 2 : comparaison de différentes fonctions récompense 58
5.4 Expérience 3 : variation de la fréquence des pics et charge de travail 66

5.5 Conclusion 70

CHAPITRE 6 CONCLUSION ET RECOMMANDATIONS 71
6.1 Synthèse des travaux 71
6.2 Limites de la solution proposée 71
6.3 Perspectives de recherche 72

RÉFÉRENCES 73

ANNEXES 77

LISTE DES TABLEAUX

Tableau 4.1	Liste des paramètres d'un buffer	27
Tableau 4.2	Les trois différents agents	30
Tableau 5.1	Récompenses accumulées moyennes par épisode associé à l'expérience 2	59
Tableau 5.2	Pourcentage de meilleure récompense accumulée (PMRA) par épisode associé à l'expérience 2	62
Tableau 5.3	Taux d'apprentissage associé à l'expérience 2 par fonction récompense et par valeur initiale d'OST	63
Tableau 5.4	Comparaison des récompenses accumulées moyennes de l'agent et de la Baseline associées à l'expérience 3	67
Tableau A.1	Revue de littérature DDMRP (partie 1/5)	77
Tableau A.2	Revue de littérature DDMRP (partie 2/5)	78
Tableau A.3	Revue de littérature DDMRP (partie 3/5)	79
Tableau A.4	Revue de littérature DDMRP (partie 4/5)	80
Tableau A.5	Revue de littérature DDMRP (partie 5/5)	81
Tableau B.1	Revue de littérature apprentissage par renforcement (partie 1/6)	82
Tableau B.2	Revue de littérature apprentissage par renforcement (partie 2/6)	83
Tableau B.3	Revue de littérature apprentissage par renforcement (partie 3/6)	84
Tableau B.4	Revue de littérature apprentissage par renforcement (partie 4/6)	85
Tableau B.5	Revue de littérature apprentissage par renforcement (partie 5/6)	86
Tableau B.6	Revue de littérature apprentissage par renforcement (partie 6/6)	87
Tableau E.1	RAM par épisode des scénarios de l'expérience 1 (partie 1/3)	94
Tableau E.2	RAM par épisode des scénarios de l'expérience 1 (partie 2/3)	95
Tableau E.3	RAM par épisode des scénarios de l'expérience 1 (partie 3/3)	96
Tableau F.1	RAM par épisode des scénarios de l'expérience 3 (partie 1/3)	97
Tableau F.2	RAM par épisode des scénarios de l'expérience 3 (partie 2/3)	98
Tableau F.3	RAM par épisode des scénarios de l'expérience 3 (partie 3/3)	99

LISTE DES FIGURES

Figure 2.1 L'effet coup de fouet 4
Figure 2.2 Le flux poussé en production 5
Figure 2.3 Le flux tiré en production 6
Figure 2.4 Le DDMRP : entre flux poussé et flux tiré 6
Figure 2.5 Exemple d'affichage de buffer pour un produit fini de notre simulation 9
Figure 2.6 Exemple d'un seuil (OST) et d'un horizon (OSH) de détection de pic 10
Figure 2.7 Un cycle d'apprentissage par renforcement 13
Figure 2.8 Un exemple de réseau de neurones 17
Figure 3.1 Modèle DDMRP d'un atelier de production géré de manière hybride . 19
Figure 3.2 Diagramme de classes simplifié du modèle 20
Figure 3.3 UML simplifié du modèle avec intégration de l'agent 22
Figure 3.4 Méthodologie générale 24
Figure 4.1 Calcul de la demande qualifiée 28
Figure 4.2 Schéma simplifié du fonctionnement du réseau de neurones 29
Figure 4.3 Architecture du BDQ inspirée de Tavakoli et al. (2018) 32
Figure 4.4 Profils de demande à 3 produits 34
Figure 4.5 Modèle conceptuel de l'apprentissage dans la simulation 39
Figure 4.6 Liaison de composition double entre l'agent et l'environnement 40
Figure 4.7 Liaison de composition entre l'agent et le médiateur 41
Figure 4.8 Mise en avant des liens permettant à l'agent de communiquer avec le système 44
Figure 4.9 Récompenses accumulées moyennes d'une phase d'évaluation pour un modèle avec agent (courbe bleue) et un modèle sans agent (courbe noire) 49
Figure 5.1 Plan d'expériences 52
Figure 5.2 Évolution des récompenses accumulées moyennes par épisode pour les scénarios à OST et OSH variables 55
Figure 5.3 Évolution des récompenses accumulées moyennes par épisode pour les scénarios à OST variables et OSH fixes 56
Figure 5.4 Évolution des récompenses accumulées moyennes par épisode pour les scénarios à OST fixes et OSH variables 56
Figure 5.5 Évolution des récompenses accumulées moyennes (RAM)par épisode pour différentes fonctions récompenses 61

Figure 5.6 Niveaux de Buffer et EFN d'un scénario sans agent qui utilise R_1 pour fonction récompense 64
Figure 5.7 Niveaux de Buffer et EFN d'un scénario avec agent qui utilise R_1 pour fonction récompense 64
Figure 5.8 Taux de service par épisode d'un scénario avec et sans agent ayant R_2 pour fonction récompense 65
Figure 5.9 Nombre de pics détectés pour le dernier épisode du scénario à une fréquence de pics 5 jours 68
Figure 5.10 Nombre de pics détectés pour le dernier épisode du scénario à une fréquence de pics 20 jours 69
Figure C.1 UML du modèle 88

LISTE DES SIGLES ET ABRÉVIATIONS

MRP	Material Requirements Planning
DDMRP	Demand Driven Material Requirements Planning
OS	Objectif Spécifique
RL	Reinforcement Learning
NFE	Net Flow Equation
EFN	Equation du flux net
OST	Order Spike Threshold
OSH	Order Spike Horizon
SED	Simulation à événements discrets
ADU	Average Daily Usage
DLT	Decoupled Lead Time
LTF	Lead Time Factor
VF	Variability Factor
MOQ	Minimum Order Quantity
TOR	Top Of Red
TOY	Top Of Yellow
TOG	Top Of Green
OTD	On Time Delivery
KPI	Key Performance Indicator
RAM	Récompense Accumulée Moyenne
PMRA	Pourcentage de Meilleure Récompense Accumulée
MRA	Meilleure Récompense Accumulée
TA	Taux d'Apprentissage
TCA	Taux de Convergence de l'Algorithme
TE	Temps d'Exécution
BDQ	Branching Dueling Q-Network

LISTE DES ANNEXES

Annexe A	Revue de littérature DDMRP	77
Annexe B	Revue de littérature apprentissage par renforcement	82
Annexe C	Diagramme de classes du modèle	88
Annexe D	Eléments de code	89
Annexe E	Résultats expérience 1	94
Annexe F	Résultats expérience 3	97

CHAPITRE 1 INTRODUCTION

1.1 Mise en contexte

Les industriels sont aujourd'hui confrontés à une nouvelle norme [1] en termes d'exigences de production. Cela rend parfois inadaptées des méthodes de production conventionnelles telles que le *Material Requirements Planning*, plus communément appelé MRP, une méthode de gestion de production pilotée par les prévisions de vente [2]. Cette "nouvelle norme" se caractérise par une tolérance du client bien plus faible aux longs délais de livraison, une plus grande variété de produits, et également des demandes accrues en termes de sécurité client et de protection environnementale [1].

Ces enjeux ont laissé place au développement de nouvelles méthodes de production qualifiées de *demand driven* ou "pilotées par la demande". Cette méthodologie "pilotée par la demande" est particulièrement intéressante dans un contexte de "marchés volatiles, avec une demande changeante, [...] avec des produits présentant une grande variété" [3]. Ces exigences ont amené les industriels à remettre en question leurs méthodes de production conventionnelles pour se tourner vers des alternatives.

Pour répondre à ces nouveaux besoins, le *Demand Driven Material Requirements Planning* ou DDMRP est apparu. Comme le MRP, le DDMRP est une méthode de gestion de production. Sa finalité est de réduire les stocks et limiter les temps de livraison. Le DDMRP se distingue du MRP comme étant une méthode de gestion de production davantage pilotée par la demande, avec une mise en avant des flux d'informations et de matières. Le DDMRP utilise des points de découplage appelés *buffers* placés tout au long de la chaîne de production et de la nomenclature des produits [1]. L'évaluation du niveau de stock et la projection sur les *buffers* vont permettre l'émission d'ordres d'approvisionnement ou de fabrication.

Un système DDMRP nécessite cependant une paramétrisation pour que la méthode soit efficace. Ces paramètres peuvent être difficiles à choisir par le gestionnaire de production, puisqu'ils ne sont parfois pas explicitement définis [1]. Dans ce projet, nous proposons d'utiliser l'intelligence artificielle afin d'ajuster et optimiser la paramétrisation du DDMRP.

1.2 Objectifs

Des travaux sur le DDMRP ont déjà été effectués par le laboratoire de recherche Poly-DDMRP en collaboration avec l'école des Mines d'Albi, notamment sur la modélisation et la simulation de systèmes de production dans un environnement DDMRP [4]. Ce mémoire est une extension des travaux précédents. Comme énoncé plus haut, la méthode DDMRP repose sur plusieurs paramètres. Néanmoins, personne ne propose une façon adéquate de piloter ces paramètres. L'objectif général de ce projet est de développer un outil capable d'ajuster des paramètres spécifiques du DDMRP et permettant d'améliorer les performances de la méthode. Les objectifs spécifiques (OS) de ce mémoire sont :

- **OS1** : Développer un algorithme permettant un apprentissage dans un environnement de gestion de stock ;
- **OS2** : Intégrer l'algorithme d'apprentissage à l'environnement DDMRP ;
- **OS3** : Valider le pilotage du DDMRP grâce à l'apprentissage automatique à l'aide d'indicateurs pertinents.

Nous référons aux objectifs spécifiques le long de ce mémoire avec les abréviations OS1, OS2 et OS3.

1.3 Structure du mémoire

Ce mémoire est composé de six chapitres incluant ce chapitre introductif. Le chapitre 2 présente un état de l'art de la méthode DDMRP ainsi que des concepts préliminaires sur les outils d'apprentissage utilisés. Ensuite, le chapitre 3 expose notre cas d'études ainsi que la méthodologie utilisée afin d'aborder notre problème. Le chapitre 4 détaille l'implémentation de notre modèle ainsi que les difficultés rencontrées. Le chapitre 5 est la présentation et la mise en place du plan d'expériences. Il est accompagné d'une analyse des résultats. Le dernier chapitre est la conclusion de ce travail.

CHAPITRE 2 REVUE DE LITTÉRATURE

La revue de littérature a pour finalité d'étudier l'état des connaissances actuelles concernant le problème identifié. Elle présente un aperçu de l'utilisation actuelle du DDMRP et de l'apprentissage par renforcement. Ce chapitre est divisé en trois sections. La première vise à présenter la méthode DDMRP, en insistant notamment sur les éléments qui seront optimisés dans cette étude. Cette première partie permet également au lecteur de se familiariser avec les notions de base du DDMRP. La deuxième section traite de l'apprentissage par renforcement, elle introduit la théorie puis présente des applications actuelles dans le monde industriel. Enfin la dernière section aborde la notion de réseaux de neurones, une composante de notre algorithme d'apprentissage par renforcement.

2.1 DDMRP

La méthode DDMRP a été créée dans les années 2010 aux États-Unis par Carol Ptak et Chad Smith [1] comme une alternative au MRP. Le DDMRP est défini comme une "méthode pour modéliser, planifier et gérer les chaînes d'approvisionnement, en protégeant le flux d'informations et de matières" [5]. La plupart des informations présentées sont tirées du livre de Ptak et Smith [1].

2.1.1 Une nouvelle méthode de production

Le DDMRP est une méthode de production innovante et récente qui vise à répondre aux besoins que présentent la nouvelle norme. L'environnement de production est de plus en plus complexe, à cause notamment d'une exigence client plus élevée pour les délais de livraison. L'essor d'Internet permet désormais à un client d'obtenir un produit rapidement à un prix compétitif [1]. Les anciennes méthodes de production n'ont pas été conçues pour répondre à de tels enjeux.

De plus, la variété des produits d'une entreprise a drastiquement grandi afin de répondre à une demande client plus variée et plus large. Les chaînes d'approvisionnement, quant à elles, se sont vues changées par la production internationale et l'externalisation [1]. Pour faire face à cette complexité, Ptak et Smith proposent de ne pas répondre à la complexité par la complexité [1]. Tout comme le MRP était basé sur des règles simples, une nouvelle méthode simple peut résoudre le problème de la complexité.

Le MRP est souvent la cause de la nervosité des systèmes de production dans lesquels il est utilisé [1]. Cela signifie qu'une petite variation à un endroit précis du système peut se répercuter et avoir des conséquences sur l'ensemble du système. Cette variation de demande se couple à de mauvais échanges d'informations et à des politiques de paramétrage propres à chaque maillon de la chaîne. Elle remonte ensuite du client au fournisseur et dérègle les stocks de matières premières et de produits finis. Ce dérèglement peut se traduire par des stocks trop élevés ou des stocks trop faibles [6]. Ce phénomène est appelé "effet coup de fouet" (voir la Figure 2.1).

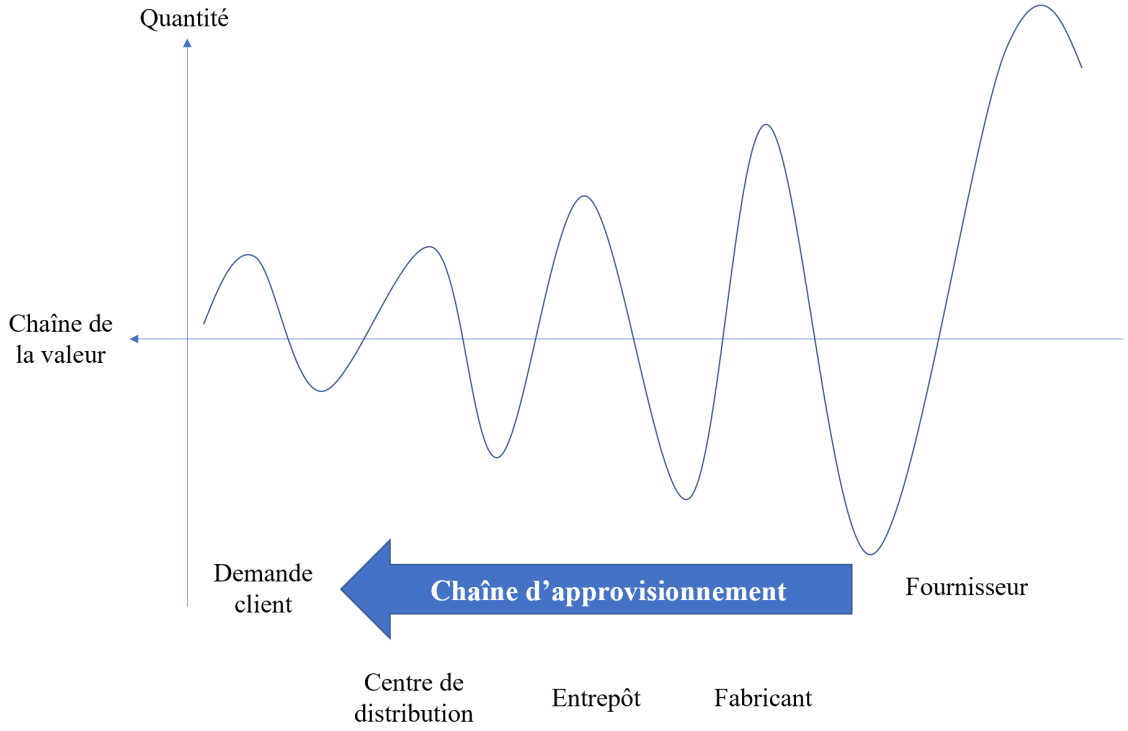


FIGURE 2.1 L'effet coup de fouet

Le phénomène d'effet coup de fouet est évitable en réorganisant et en retravaillant les informations importantes : les flux, en particulier les flux de matériel et d'informations [1]. Ces deux notions sont traitées comme des flux car ce sont des entités en mouvement dans notre système, parfois même en grande quantité. Le DDMRP vise à mieux protéger ces flux et à leur donner plus d'importance [1].

La méthode DDMRP est mise en oeuvre pour réduire les délais de livraison. Cela est rendu

possible par une prise de décision basée sur la demande réelle plutôt que sur des prévisions et la mise en place de certaines fonctionnalités comme les points de découplage. Un point de découplage est un "endroit dans la chaîne logistique où la demande réelle pénètre afin de différencier ce qui dans la chaîne est directement tiré par le client final de ce qui est poussé par l'amont" [7].

Les principaux objectifs du DDMRP sont de s'adapter à la "nouvelle norme", en évitant de présenter des stocks trop importants et des temps de livraison trop longs. Il est désormais possible de présenter son fonctionnement.

2.1.2 Fonctionnement du DDMRP

Ptak et Smith définissent le DDMRP comme une **méthode d'exécution et de planification formelle à plusieurs échelons, qui tend à mettre en avant les flux d'informations pertinentes à travers l'établissement et la gestion de points de découplage stratégiquement placés** [1].

Le DDMRP reprend des fonctionnalités pertinentes de la méthode MRP. Le fonctionnement de cette dernière repose principalement sur le concept de flux poussé : la production est planifiée en fonction des prévisions de vente [1]. Les stocks de matières premières et de produits finis sont remplis de façon à satisfaire ces prévisions. Néanmoins, une telle méthode est souvent la cause de stocks trop élevés (voir la Figure 2.2). A cela s'oppose le flux tiré, avec cette méthode la production est basée sur la demande réelle et les commandes fermes [1]. Le flux tiré permet de conserver des stocks bas, mais les délais de livraison sont plus longs (voir la Figure 2.3).

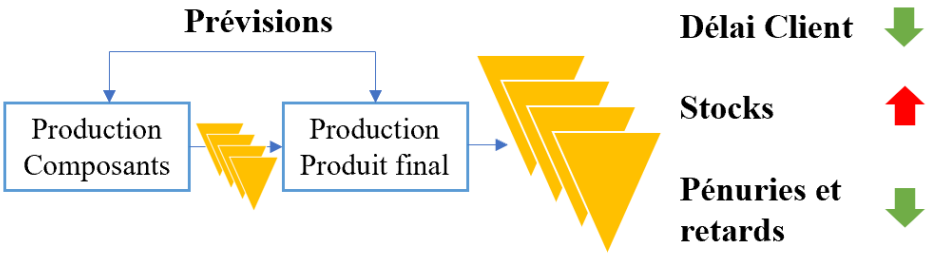


FIGURE 2.2 Le flux poussé en production

La méthode DDMRP tend à concilier le flux poussé et le flux tiré : certaines références sont fabriquées en flux tiré, tandis que les autres sont gérées en flux poussé [1]. Ces références fabriquées en flux tiré sont stockées dans des points de découplage appelés *buffers*. Ils sont

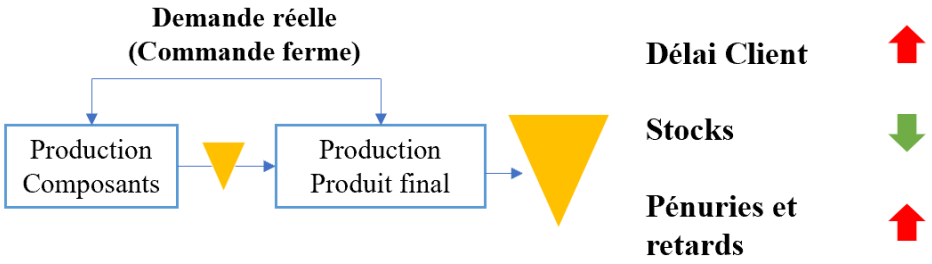


FIGURE 2.3 Le flux tiré en production

placés dans la nomenclature des produits finis et sont donc distribués le long de la chaîne logistique. Les buffers contrôlent les ordres de fabrication ou les ordres d'approvisionnement. Ces *buffers* sont symbolisés par un trapèze divisé en quatre zones visibles à la Figure 2.4. Cette solution permet de limiter la dépendance qui règne à l'intérieur de la chaîne de production. Rappelons que l'un des enjeux du DDMRP est de limiter cette dépendance qui donne naissance à l'effet coup de fouet. Les points de découplage permettent également de limiter les temps de livraison en tirant partie de la méthode du flux poussé.

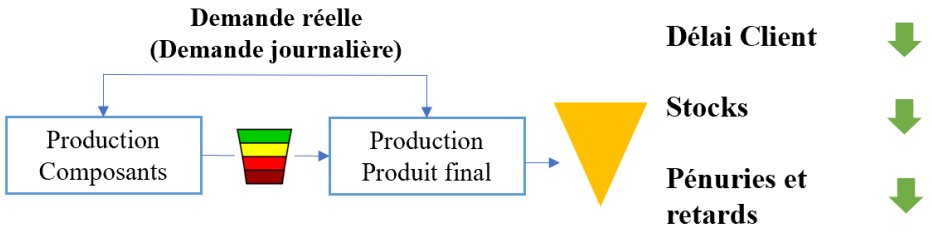


FIGURE 2.4 Le DDMRP : entre flux poussé et flux tiré

Revenons maintenant sur l'aspect fonctionnel du DDMRP, en présentant la partie "signaux" de la définition de Ptak et Smith. Nous allons plus précisément expliquer quelles formes vont prendre ces signaux en détaillant la notion de point de découplage ou *buffer*. Un *buffer* représente un point d'indépendance de la ligne de production. C'est ce qui limite principalement l'effet coup de fouet. Ils sont placés tout au long de la ligne de production et représentent des informations sur les stocks. Son fonctionnement est davantage précisé dans la prochaine section. Nous retenons que les *buffers* servent comme réserve dont le niveau de stock est maintenu. Si les stocks ne sont pas suffisants, le gestionnaire de production reçoit un "signal" simple à interpréter.

Les éléments fonctionnels du DDMRP ont été présentés, mais comment instaurer la méthode DDMRP dans une chaîne de production ? Ce processus se réalise en cinq étapes [1] :

1. *Strategic Inventory Positioning* : déterminer où seront placés les points de découplage ou *buffers* ;
2. *Buffer Profiles and Levels* : définir les niveaux de protection de ces points ;
3. *Dynamic adjustments* : apporter des ajustements en fonction de l'influence des niveaux de protection sur les changements du marché, les événements ;
4. *Demand Driven Planning* : génération des ordres de commande ;
5. *Visible and collaborative execution* : processus par lequel le DDMRP gère les ordres d'approvisionnement ouverts.

Plus de détails sur ces étapes sont disponibles dans le livre de Ptak et Smith [1]. Nous allons donc nous attarder sur la partie que nous tendons à améliorer dans le cadre de ce projet, soient les ajustements dynamiques.

2.1.3 Ajustements dynamiques

Les ajustements dynamiques constituent les modifications temporaires qu'un gestionnaire de production peut apporter à un système de production. La fréquence de ces ajustements doit être choisie par le gestionnaire de production. Souvent ces ajustements se font tous les 5 jours ou bien chaque semaine [1]. Le DDMRP est un système de gestion avec un ensemble de règles qui évolue dans un environnement dynamique, sensible à la variabilité de la demande client. On rend certains éléments du DDMRP également dynamiques. Afin d'élaborer cette notion, on revient à la définition du *buffer*. Les *buffers* peuvent être illustrés par des niveaux de stocks définis que l'on veut maintenir. Ils ont trois buts : absorber le choc (celui de l'effet coup de fouet), compresser les temps de livraison et générer les ordres de commande [1]. Afin de remplir ces fonctions, les *buffers* présentent quatre zones, qui sont calculées avec quatre équations. Ces zones présentent des tailles différentes, et permettent de définir si on doit commander la référence gérée par le *buffer*.

Les équations dépendent de différents facteurs propres à notre environnement de production et calculables en tout temps. Ces facteurs sont les suivants :

— *Average Daily Usage* ou ADU : la commande moyenne journalière, c'est la moyenne des demandes journalières sur une fenêtre de temps définie ;
— *Decouple Lead Time* ou DLT : "le plus long temps cumulé entre le buffer et les références précédents" [4] ;

— *Lead Time Factor* ou LTF : calculé à partir de l'ADU et du DLT, ce facteur permet de considérer les incertitudes des temps d'acheminement des produits. Il est adapté en fonction des temps d'acheminement de l'atelier ;
— *Variability Factor* ou VF : il permet de prendre en compte les incertitudes sur la demande ;
— *Minimum Order Quantity* ou MOQ : c'est la taille minimale d'une commande.

Ces facteurs nous permettent de calculer les tailles de zone des buffers, présentées par les équations 2.1, 2.2, 2.3 et 2.4

$$Zone\ verte = Max(ADU * DLT * LTF, MOQ) \tag{2.1}$$

$$Zone\ jaune = ADU * DLT \tag{2.2}$$

$$Zone\ rouge\ base = ADU * DLT * LTF \tag{2.3}$$

$$Zone\ rouge\ sécuritaire = ADU * DLT * LTF * VF \tag{2.4}$$

On veut suivre l'évolution du stock dans nos quatre zones de *buffer*, néanmoins les stocks bruts ne suffisent pas. En plus des stocks bruts (ou *on-hand*), il faut considérer les commandes en attente (ou *qualified sales order demand*) et les ravitaillements de stock qui sont prévus (ou *on-order*). Cela permet d'avoir une idée plus réaliste du stock, car on prend en compte des changements de stock imminentes. Pour cela, on utilise la *net flow equation* ou NFE, qui nous permet de calculer la *Net flow position* ou NFP [1]. Son expression est définie à l'équation 2.5 :

Net Flow Position (NFP) = on-hand + on-order - qualified sales order demand (2.5)

soit **Equation du flux net (EFN) = Stock en main + Stock en cours de production - demande qualifiée**

Cette équation remplace la décision sur stock physique par une décision sur un équivalent à la position du stock. Elle permet de ne pas prendre de décisions qui pourraient créer un effet coup de fouet. On peut ainsi suivre l'évolution de notre EFN dans notre buffer pour chacun de nos produits [1]. La Figure 2.5 représente un exemple d'évolution d'EFN dans un

buffer. La position de l'EFN dans les différentes zones du buffer donne des indications sur la gestion du stock. Par exemple, cela permet d'indiquer le moment où il faut remplir les stocks : lorsque l'EFN est inférieure à la valeur haute de la zone jaune, on génère un ordre de commande afin de refaire passer l'EFN à la valeur haute de la zone verte [1].

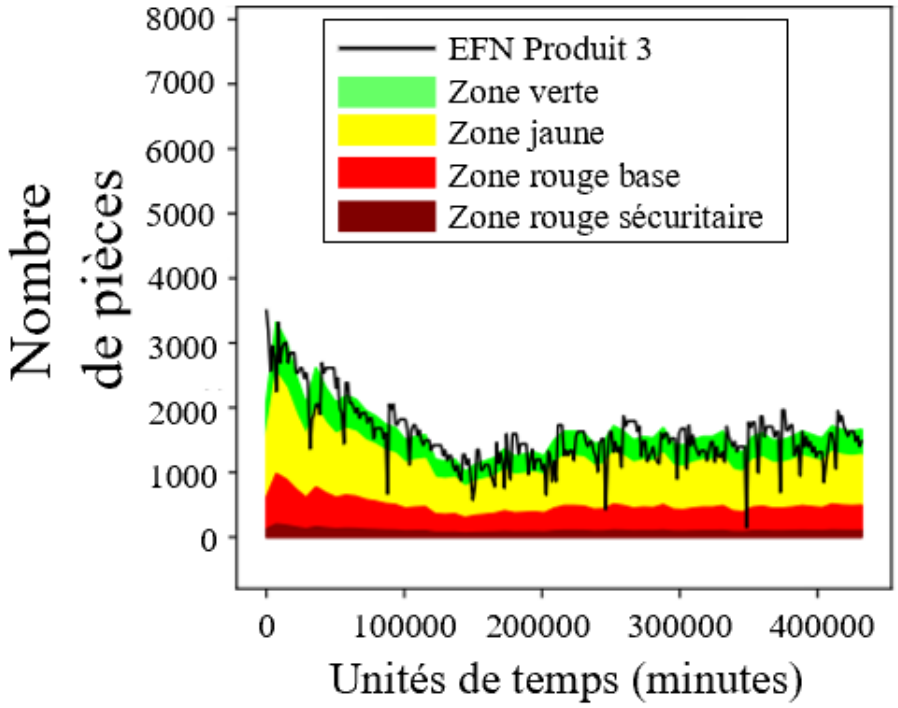


FIGURE 2.5 Exemple d'affichage de buffer pour un produit fini de notre simulation

Dans l'équation de l'EFN, la demande qualifiée est la demande du jour à laquelle on ajoute la demande des pics dans l'horizon de détection [8]. Le calcul de la demande qualifiée dans la Figure 2.6 proposée par Ptak et Smith requiert la fixation d'un seuil de détection de pic de demande (ou *Order Spike Threshold*, OST) et d'un horizon de détection de pic de demande (ou *Order Spike Horizon*, OSH). L'OST sert à détecter des pics de demande anormaux qui pourraient venir menacer l'intégrité du *buffer*. L'OSH définit la fenêtre de temps sur laquelle on va chercher à détecter les pics de demande [1]. La Figure 2.6 illustre un exemple d'OST et d'OSH sur une période de 8 jours. Dans la littérature, aucune solution n'est proposée pour les ajuster.

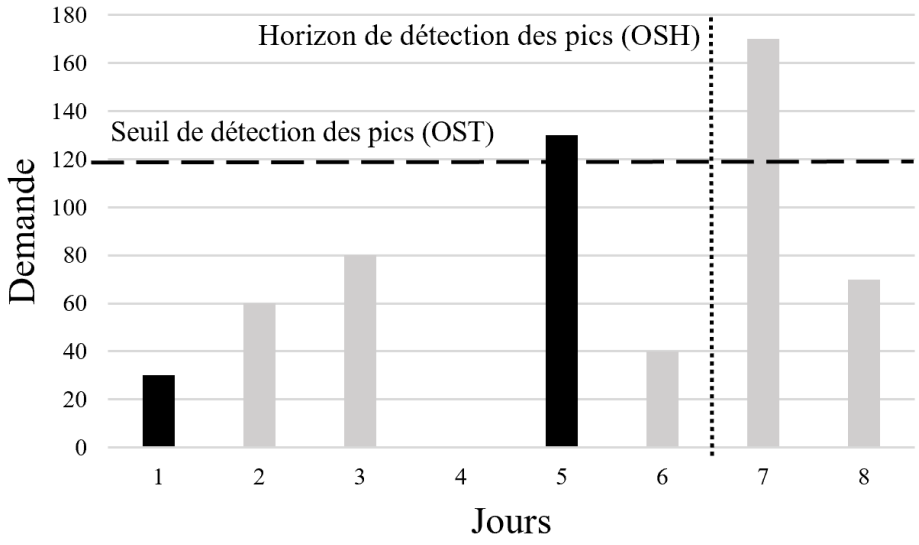


FIGURE 2.6 Exemple d'un seuil (OST) et d'un horizon (OSH) de détection de pic

2.1.4 Le DDMRP dans la littérature

Étant une méthode de production assez récente, les recherches scientifiques portant sur le DDMRP sont peu nombreuses. Azzamouri et al. [3] ont dressé une revue de littérature systématique afin de dresser l'évolution du DDMRP dans la littérature. Cet article montre que le DDMRP n'est pas encore assez étudié dans la littérature et que la méthode n'est pas encore bien validée. Les cas traités sont majoritairement pédagogiques et ne constituent pas des cas industriels. En particulier, la paramétrisation de cette méthode de production est largement délaissée. Les articles cités dans ce rapport sont répertoriés en **Annexe A**.

Les recherches montrent bien que le DDMRP est applicable en industrie, et n'est pas limité à un domaine industriel [3]. On peut le retrouver par exemple dans l'industrie automobile [9] ou la fabrication d'encre [10]. Bahu et al. montrent, à travers 30 études de cas, que le DDMRP est adapté dans de nombreux domaines industriels [11] [3]. Néanmoins, on dénote que les résultats des implémentations de DDMRP sont rarement publiés [3]. De plus, le DDMRP semble surtout utilisé dans des industries à processus discrets, et semble délaissé par les industries à processus continus (gaz, ciment, mines...) [3].

Les performances du DDMRP ont été souvent démontrées et comparées à celles du MRP [3]. Miclo et al. ont publié une série d'articles portant à démontrer les intérêts du DDMRP face au MRP de façon quantitative [12]. D'autres auteurs se sont intéressés à exposer une comparaison entre le DDMRP et le MRP en milieu industriel, notamment Kortabarria et al. [13], Shofa et Widyarto [9] ou Ihme et Stratton [10]. Ces recherches sont effectuées sur des analyses

quantitatives (niveaux d'inventaire, temps de mise en oeuvre...) et/ou qualitatives (réalisation d'entretiens). Les résultats s'appuient également sur les données des entreprises, et des outils déjà mis en place tels que les ERP [10]. Les résultats quantitatifs de l'implémentation du DDMRP sont : réduction de l'inventaire, des retards et des temps de mise en oeuvre. Les résultats qualitatifs sont une meilleure visibilité de la chaîne d'approvisionnement et la réduction des instabilités dans le système.

Les études portées sur le DDMRP se différencient également dans la façon de le simuler et le modéliser [3]. Certains ont opté pour des outils simples, tels que Excel [10] [9]. La plupart des auteurs proposent une implémentation par simulation à événements discrets [4] [12] [13] [14].

La littérature du DDMRP comporte plusieurs lacunes. La première est la question de son implémentation. Les auteurs ont pu montrer qu'il était possible de l'implémenter en industrie, mais la question du "Comment ?" est bien souvent laissée de côté [3]. De fait, Orue et al. proposent une revue de littérature afin d'établir une "standardisation du processus d'implémentation du DDMRP" [15]. L'article montre une fois de plus l'absence de résultats publiés du DDMRP en industrie. Martin et al. proposent une "cartographie de processus explicite pour le modèle Demand Driven Adaptive Enterprise" [16]. Le but est d'établir une cartographie interprétable et exploitable dans différents types d'industrie.

Peu de littérature du DDMRP porte sur les paramètres et les ajustements dynamiques. Ptak et Smith laissent peu de directives quant aux paramètres et aux ajustement dynamiques [1]. Ils proposent plusieurs façons de les calculer, mais ne favorisent pas une méthode par rapport à une autre. Pourtant, dès 2015, Miclo expose que la modification des paramètres du DDMRP influe sur les résultats de celui-ci [12]. Pourtant, les recherches portées sur les paramètres et les ajustements dynamiques sont rares [3]. Dessevre et al. proposent un modèle d'ajustement dynamique au niveau des temps de mise en oeuvre [17]. Cette étude a abouti à une réduction des stocks et à un meilleur niveau de service. Les pistes de recherche concernant l'ajustement des paramètres du DDMRP sont donc nombreuses.

Parmi les paramètres du DDMRP, les OST et OSH agissent sur la détection de pics. Ils sont importants pour le calcul de la demande qualifiée et donc l'EFN. Ptak et Smith proposent différentes méthodes de calcul, sans en favoriser une par rapport à une autre [1]. A notre connaissance, seuls Damand et al. ont étudié la paramétrisation des OST et OSH [18]. Néanmoins, ils utilisent un algorithme génétique et précisent qu'il serait pertinent d'envisager l'emploi d'une entité d'apprentissage. Notre étude vise à combler ce manque de la littérature. Nous proposons une méthode DDMRP dans laquelle les OST et OSH vont s'ajuster de façon autonome à l'aide d'une entité d'apprentissage. Pour cela, nous allons utiliser l'apprentissage par renforcement.

2.2 Apprentissage par renforcement

2.2.1 Définitions

L'apprentissage par renforcement (*reinforcement learning* ou RL) est un type d'algorithme d'intelligence artificielle qui permet d'apprendre à travers "l'essai et l'erreur" [19]. C'est un type d'algorithme de l'apprentissage machine, dont le but est de prendre des actions et observer les conséquences de ces actions. Concrètement, une entité appelée un agent prend des décisions dans un environnement afin d'atteindre un but précis. Des récompenses négatives ou positives seront attribuées à ces décisions. Au fur et à mesure de l'exécution de l'algorithme, l'agent "apprend" et tend vers de meilleures décisions. Son objectif est de maximiser les récompenses et atteindre le but de façon optimale. Ce but est parfois représenté par un état terminal, si l'environnement arrive à cet état, l'algorithme s'arrête. On distingue ainsi les états terminaux et les états non terminaux. Si l'algorithme ne parvient pas à arriver à un état terminal ou s'il n'y a pas d'état terminal, on peut lui imposer une contrainte temporelle.

Dans notre cas, nous nous intéressons au *Deep reinforcement learning*, ou apprentissage par renforcement profond. Cette approche de l'apprentissage par renforcement utilise un modèle de réseau de neurones [19] pour réaliser une approximation de la fonction récompense. Ces modèles sont décrits plus en détails dans la section **4.2.2**.

L'apprentissage par renforcement est caractérisé par un vocabulaire qui lui est bien spécifique, dont voici les notions les plus importantes [19] :

- **L'environnement** est tout ce qui est extérieur à l'agent. Cet environnement est caractérisé par un état, qui prend la forme d'un ensemble de valeurs ;
- **L'agent** est l'entité qui prend les décisions et qui agit sur l'environnement ;
- **Une observation** est une caractéristique de l'état de l'environnement que l'agent peut observer. Il est parfois intéressant de ne pas rendre tout l'environnement observable par l'agent ;
- **Une action** est une modification de l'état de l'environnement sur un pas de temps. L'action est choisie par l'agent selon une politique de décision ;
- **Une récompense** est un score (un réel) associé à un changement d'état. Elle dépend de la valeur de l'état renvoyée par l'environnement et interprétée par l'agent ;
- **Une politique** est une fonction qui prescrit une action à faire pour un état non terminal. Elle couvre tous les états non terminaux.

Les principales notions ayant été présentées, nous pouvons introduire les concepts de base.

2.2.2 Concepts de base

Construire un algorithme d'apprentissage par renforcement présuppose de modéliser un environnement en créant deux fonctions : la fonction transition et la fonction récompense. La fonction transition assure un changement d'état suite à une action précise choisie par l'agent. La fonction récompense retourne la récompense d'une action suite à un changement d'état [19]. Autrement dit elle attribue une valeur numérique à un état. Cette fonction définit la tâche de l'environnement ainsi que son but (par exemple maximiser un taux de service). Une fois ces deux fonctions définies, on a un modèle de l'environnement.

Les algorithmes d'apprentissage par renforcement reposent sur la répétition de cycles. Un cycle représente le passage dans l'algorithme d'un état à un nouvel état. Le tuple composé de l'état, l'action, la récompense et le nouvel état est appelé l'expérience (voir la Figure 2.7) [19].

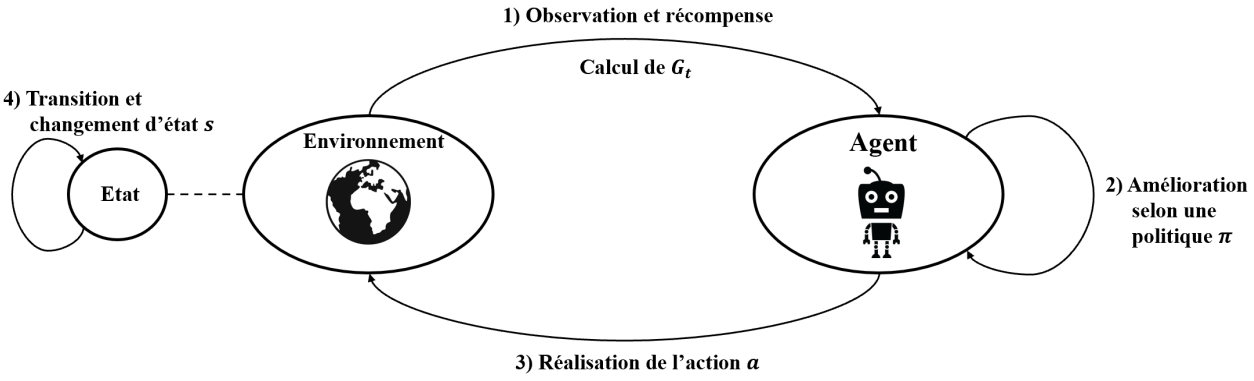


FIGURE 2.7 Un cycle d'apprentissage par renforcement

Le cycle est composé des étapes suivantes :

1. Observation et récompense : l'agent observe l'environnement à travers un espace d'états et lui attribue une valeur numérique à travers une récompense ;
2. Amélioration de la politique π : l'agent améliore la politique afin qu'elle recommande de meilleures actions ;
3. Réalisation de l'action a : l'agent sélectionne une action a dans l'espace d'états et la préconise à l'environnement ;
4. L'environnement réalise l'action a et effectue son changement d'état s.

Un cycle entier est appelé un pas de temps. Une tâche est terminée lorsqu'un état terminal est atteint ou lorsqu'on a atteint un nombre maximal de pas de temps. Une séquence de pas

de temps déterminé par un début et une fin de tâche est appelée un épisode.

A la fonction transition et la fonction récompense s'ajoute la fonction action-valeur, ou Q-fonction. Elle est basée sur la fonction récompense. Cette fonction calcule le retour (i.e. la somme des récompenses collectées dans un seul épisode) attendu si l'agent, considérant une politique π, prend une action a à l'état s. Nous notons G_t le gain, la somme des retours au temps t, c'est-à-dire la somme des récompenses du pas de temps $t+1$ au pas de temps final T ($G_t = R_{t+1} + R_{t+2} + ... + R_T$, avec R_t la récompense attribuée au pas de temps t). La Q-fonction permet de calculer le gain espéré d'une action a prise à un état s, autrement dit la somme des récompenses futures espérée lorsque l'action est prise dans un état particulier. La définition mathématique de la Q-fonction est l'équation (2.6) [19] :

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \tag{2.6}$$

Il est important de comprendre l'importance de cette fonction puisqu'elle est primordiale au fonctionnement du RL. La maximiser revient à trouver l'action qui donne la plus grande somme de futures récompenses espérée, c'est-à-dire l'action qui est la plus bénéfique à l'environnement. Dans la pratique, on ne calcule pas sa valeur exacte, on utilise généralement un approximateur [19]. Dans notre cas, l'approximateur est un modèle de réseau de neurones (voir plus de détails dans la section **2.3**).

L'apprentissage par renforcement est un type d'algorithme qui utilise des tâches "concrètes, bien définies et simples" [19]. En contrepartie, l'apprentissage par renforcement nécessite beaucoup d'expériences, et il est bien souvent difficile de comprendre et d'interpréter les récompenses obtenues. Le défi avec cet algorithme est "l'attribution du crédit temporel" [19]. Il faut pouvoir identifier quels états et quelles actions doivent être récompensées et après combien de pas de temps. Il existe plusieurs types d'algorithmes d'apprentissage par renforcement et donc plusieurs façons de procéder. Nous allons l'illustrer avec des exemples d'application.

2.2.3 Exemples d'application

Le domaine industriel présente déjà plusieurs applications de l'apprentissage par renforcement. Dans leur papier, Xanthopoulos et al. font un survol de plusieurs applications de l'apprentissage par renforcement dans un contexte industriel [20]. Nous avons répertorié quelques exemples en **Annexe B**.

L'apprentissage par renforcement peut être utilisé pour l'optimisation discrète, dont fait partie la planification de production. L'algorithme a pour tâche de limiter les coûts [21] en étant soumis à certaines contraintes de production [22]. Dans plusieurs études telles que celles de Wang et al., les performances de l'algorithme sont comparées avec des heuristiques de planification de la production traditionnelles [23]. D'autres études explorent d'autres applications : Tsai et al. nous présentent par exemple un bras articulé contrôlé par un agent de RL dans une ligne de production de chaussures [24]. L'apprentissage par renforcement ne se limite donc pas à l'optimisation de la planification de la production. Néanmoins, il faut bien adapter la définition du problème et des espaces tels que les états et les actions.

Certains auteurs cherchent à mettre en avant des enjeux de l'apprentissage par renforcement. Lai et al. exposent par exemple des méthodes qui permettent de limiter le surapprentissage. Le surapprentissage apparaît lorsque l'agent est trop adapté aux données d'entraînement, il n'arrive alors plus à prédire et s'adapter à de nouvelles données [25].

Stockheim et al. évoquent le problème de l'équilibre exploitation-exploration, l'une des plus grosses difficultés de l'apprentissage par renforcement [21]. Concrètement, il s'agit d'un compromis que l'agent doit faire lorsqu'il doit décider s'il va explorer, c'est-à-dire essayer une action au hasard et voir si celle-ci est bénéfique, ou exploiter, utiliser les expériences passées afin de déterminer la meilleure action à prendre dans un état bien précis [19]. D'une façon assez similaire Dittrich et Fohlmeister apportent une attention particulière à l'étude de ce compromis [26].

En outre, Zhou et al. montrent qu'il est possible d'adapter l'apprentissage par renforcement pour qu'il soit utilisé sur des bases de données de grande taille [27]. Le fléau de la dimension est un enjeu important pour l'industrie si la méthode d'apprentissage par renforcement vient à se développer et se répandre.

Tout comme nous l'avons déjà précisé, l'apprentissage par renforcement peut prendre différentes formes. Kemmer et al. comparent plusieurs algorithmes de RL, et tirent des conclusions quant à l'utilisation de chaque méthode [28]. D'une façon assez similaire, Neves et al. incitent les futures recherches à se pencher sur d'autres algorithmes de RL, afin d'étoffer l'étude et insister sur le fait qu'il existe beaucoup de façons différentes de procéder [29].

Enfin, nous évoquons l'importance de la paramétrisation. **Un algorithme d'apprentissage par renforcement mal paramétré peut présenter des performances catastrophiques** [19]. Xanthopoulos et al. insistent particulièrement sur ce point et montrent que les paramètres ont un impact important dans les résultats [20]. Un système géré en apprentissage par renforcement peut présenter de moins bons résultats que des heuristiques traditionnelles si celui-ci est mal paramétré. C'est l'un des enjeux de notre recherche.

2.2.4 Apprentissage par renforcement et DDMRP

A notre connaissance, un seul article a utilisé l'apprentissage par renforcement pour l'ajustement d'un modèle opératoire piloté en DDMRP. Cuartas Murillo et al. proposent un "algorithme hybride basé sur l'apprentissage par renforcement et la méthodologie DDMRP pour la gestion de l'inventaire" [30]. Leur algorithme d'apprentissage par renforcement agit directement sur les niveaux d'inventaire et sur les temps de mise en oeuvre. L'article expose la comparaison de trois agents différents, mais n'agit pas sur les paramètres du DDMRP comme nous proposons de le faire. De plus, nous proposons d'utiliser un agent d'apprentissage par renforcement utilisant un réseau de neurones.

2.3 Réseaux de neurones

Les modèles de réseaux de neurones vont nous permettre d'approximer la Q-fonction de l'équation (2.6).

2.3.1 Définition

Schmidhuber définit dans son rapport technique un réseau de neurones comme un : « [Ensemble] de processeurs appelés des neurones, chacun produisant une séquence d'activations à valeurs réelles. Les neurones d'entrée sont activés [...] à travers l'environnement. Certains neurones peuvent influencer l'environnement en déclenchant des actions » [31]. Le but d'un réseau de neurones est "d'apprendre", c'est-à-dire trouver les poids des neurones qui permettent de simuler un "comportement désiré" [31].

Un réseau de neurones (voir la Figure 2.8) est d'abord composé d'une couche d'entrée, dans laquelle on injecte des paramètres d'entrée. Ces paramètres d'entrée passent dans les couches cachées, qui créent des compositions de fonctions non linéaires. On active un neurone ou non avec une fonction d'activation [31]. Enfin, nous arrivons à la couche de sortie, c'est-à-dire la couche qui retourne les sorties du réseau de neurones [31].

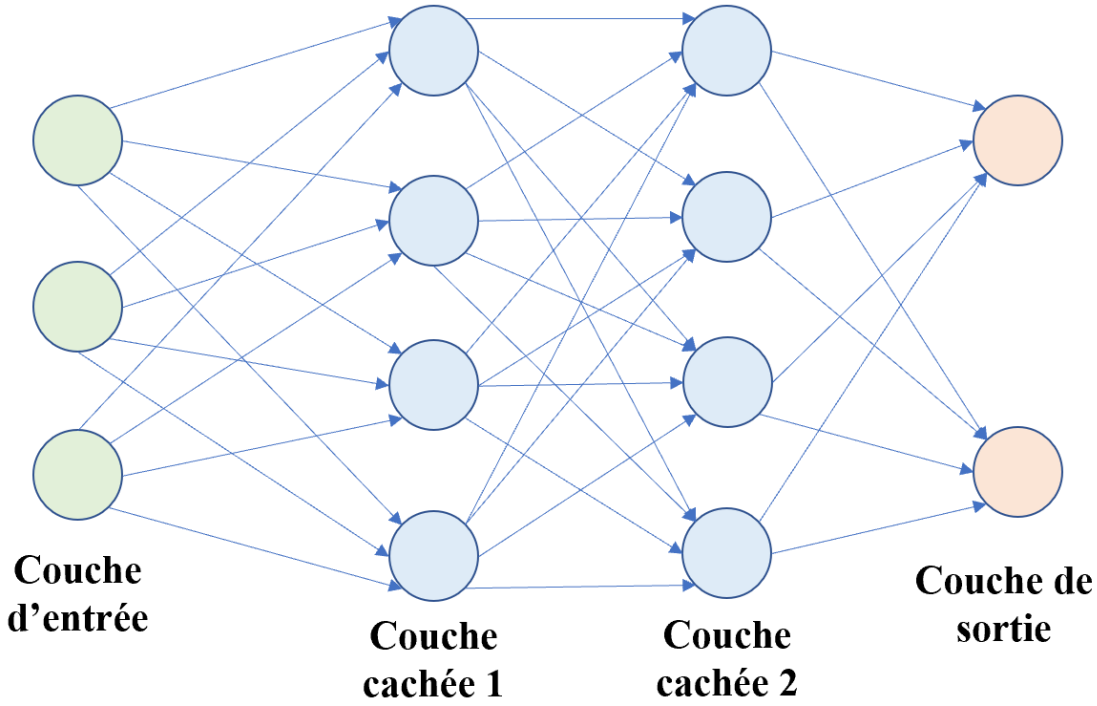


FIGURE 2.8 Un exemple de réseau de neurones

2.3.2 Exemples d'application

Des applications de réseaux de neurones en rapport avec l'apprentissage par renforcement. Quand on utilise un réseau de neurones pour approximer la Q-fonction dans un algorithme de *Q-learning*, on parle du *deep Q-learning*. Morales (2020) présente le fonctionnement d'un tel algorithme : les paramètres d'entrée sont des valeurs pertinentes permettant au réseau de neurones de calculer en sortie les Q-valeurs pour chaque action possible [19]. Dittrich et Fohlmeister (2020) utilisent cette logique afin d'intégrer des réseaux de neurones dans un algorithme de *Q-learning* utilisé pour un système multi-agent coopératif [26]. On peut également utiliser les réseaux de neurones afin de créer un système de décision partagé et ainsi obtenir des degrés de liberté indépendants pour des actions différentes (par exemple dans le déplacement d'un robot, comme nous le montrent Tavakoli et al. [32]). Enfin, il est important de considérer les avantages et les inconvénients des réseaux de neurones. Bien souvent, ces réseaux sont simples à mettre en place, et ils fonctionnent assez bien, néanmoins ils peuvent baisser en performance sur des systèmes de grande dimension, c'est le problème du fléau de la dimension [29].

2.4 Conclusion

Dans ce chapitre, nous avons présenté les éléments de base associés au DDMRP et à l'apprentissage par renforcement. Nous avons également exposé plusieurs exemples d'application de ces concepts.

La revue de littérature expose la diversité de paramètres du DDMRP. Bien que caractérisée par les ajustements dynamiques, il semble que certains paramètres soient délaissés quant à leur pilotage. En outre, certains éléments de paramétrisation ne présentent pas une unique méthode de calcul reconnue et ayant fait ses preuves. Ces différentes recommandations de calcul peuvent être déroutantes pour un gestionnaire de production. Bien qu'il soit possible de calculer des OST d'au moins trois façons différentes, il est légitime de se demander si l'une des méthodes est meilleure que l'autre. De même, bien qu'il soit conseillé de fixer un OSH à au moins un DLT, on peut se demander quelle est la limite de ce facteur. Dans ce travail,nous proposons de définir une méthode unique de pilotage des OST et OSH, en utilisant l'apprentissage par renforcement. La littérature nous a montré que cette méthode d'apprentissage machine a fait ses preuves dans la gestion de production. La question de recherche ayant été affinée, il est désormais possible de présenter le cas d'étude et l'approche méthodologique de notre projet.

CHAPITRE 3 CAS D'ÉTUDE ET APPROCHE MÉTHODOLOGIQUE

L'objectif de ce travail est d'intégrer une composante d'intelligence artificielle à un environnement géré en DDMRP afin d'en optimiser la paramétrisation. Ce chapitre présente le cas d'étude et permet au lecteur de mieux appréhender le fonctionnement du système étudié. Le lecteur va prendre connaissance de l'environnement, les processus clés du modèle, ainsi que les acteurs principaux et leurs interactions pertinentes. Dans un second temps, l'approche méthodologique est exposée.

3.1 Cas d'étude

3.1.1 L'atelier de production de type hybride

Dans ce projet, nous traitons le cas d'un système de production complexe piloté par *DDMRP*. Il s'agit d'un atelier de type hybride, où un ensemble de p produits sont fabriqués par n machines parallèles à partir de m matières premières. Nous nous basons sur le modèle développé par [4] illustré par la Figure 3.1. L'atelier est "hybride" car il "fait intervenir à la fois un flux de production tiré par la demande et un flux poussé d'ordres de production" [4]. Un tel modèle est utilisé car il couvre un grand nombre de cas d'applications réels [4] et qu'il permet de reprendre un modèle déjà existant.

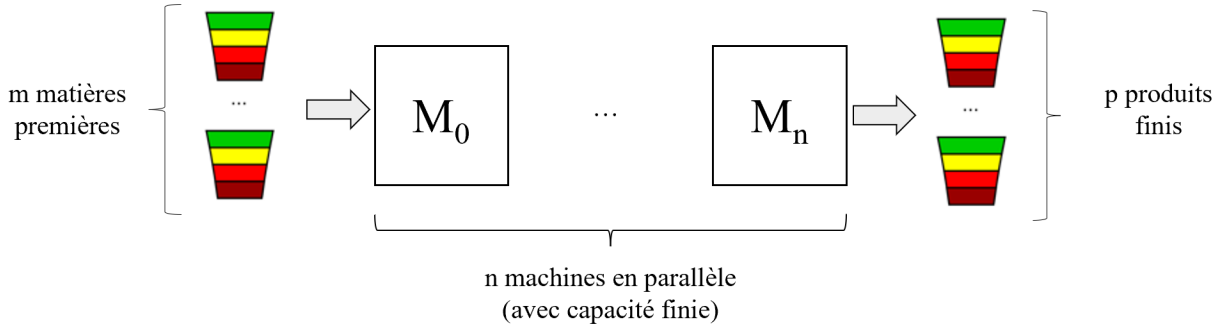


FIGURE 3.1 Modèle DDMRP d'un atelier de production géré de manière hybride

Les stocks de m matières premières et p produits finis sont contrôlés par des buffers. Le modèle est composé de plusieurs classes. Les classes sont définies comme des composantes du modèle interagissant les unes avec les autres.

3.1.2 Les classes du modèle

Nous proposons une représentation simplifiée des classes du modèle dans la Figure 3.2 à travers un diagramme de classes avec les conventions traditionnelles d'UML. Nous y précisons les attributs et opérateurs principaux des différentes classes.

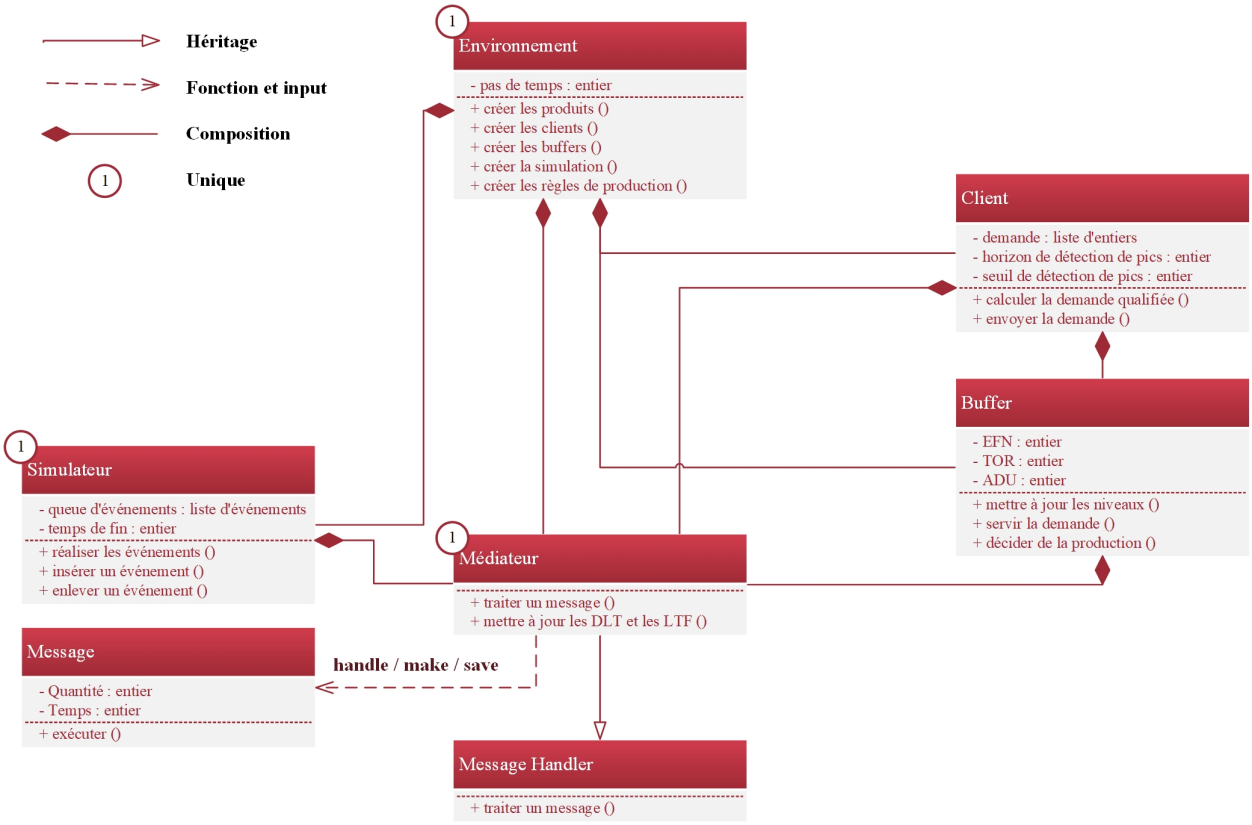


FIGURE 3.2 Diagramme de classes simplifié du modèle

Les classes interagissent entre elles à travers des liens. Le lien de composition (flèche rouge continue à losange plein sur la Figure 3.2) symbolise qu'une classe est dépendante d'une autre. Si une classe **A** présente un lien de composition vers la classe **B**, nous proposons d'utiliser la terminologie suivante pour le reste de l'étude : **A** est composé de **B**. L'environnement est par exemple composé d'un médiateur, autrement dit on peut accéder au médiateur depuis l'environnement. A noter qu'une relation entre classes peut être réciproque, deux classes peuvent dépendre l'une de l'autre et être chacune accessibles depuis l'autre. Un lien d'héritage (flèche rouge continue à triangle sur la Figure 3.2) est utilisé lorsqu'une classe hérite de tout ce qu'il y a dans une autre classe, cela signifie qu'une classe possède tous les attributs et les méthodes de la classe dont elle hérite. Enfin le lien de fonction (flèche rouge discontinue sur la Figure 3.2) et entrée représente l'existence d'une fonction dans une classe qui a comme

paramètre une autre classe. La mention "unique" est utilisée lorsque la simulation ne peut compter qu'un seul élément de ce type, cet élément est appelé singleton.

Ce diagramme est volontairement simplifié afin de ne pas être surchargé, en réalité le modèle est bien plus complexe, un diagramme complet est rendu disponible en **Annexe C**.

Une expérience de simulation est associée à une instance de la classe **Environnement**, qui sert en fait d'interface avec les autres opérations. A cette classe **Environnement** sont associées des classes avec différentes fonctionnalités [4] :

- les classes éléments de l'atelier (**Client** et **Buffer**), leur dépendance est traduite par un lien de composition ;
- les classes qui assurent le bon fonctionnement de la simulation (**Simulateur**), leur dépendance est traduite par un lien de composition ;
- la classe **Médiateur**, qui permet une "communication bidirectionnelle entre tous les objets" [4], leur dépendance est traduite par un lien de composition.

La classe **Médiateur** sert à assurer les échanges entre les objets sans avoir à les relier. C'est pour ça que les classes **Simulateur**, **Client** et **Buffer** sont reliées au médiateur par un lien de composition. A noter que la classe **Client** est composée de la classe **Buffer** car chaque client est associé à un buffer de produit fini.

La classe **Message Handler** permet à une entité d'envoyer et recevoir des messages d'autres entités. Le lien d'héritage entre la classe **Médiateur** et la classe **Message Handler** signifie que que la classe **Médiateur** hérite de tous les attributs et méthodes de la classe **Message Handler**. C'est grâce à cela que la classe **Médiateur** gère la communication bidirectionnelle entre les entités de la simulation.

Enfin, le lien de fonction entre la classe **Médiateur** et la classe **Message** symbolise qu'il existe trois fonctions (*handle*, *make* et *save*) dans la classe **Médiateur** ayant pour entrée un objet de type **Message**. Ce lien permet au **Médiateur** de gérer la distribution et la réception des messages.

3.1.3 Intégration de l'agent

L'agent est une entité qui modifie l'environnement en lui faisant effectuer des actions. Dans notre étude, les actions sont des modifications de paramètres. Ces actions sont par la suite évaluées par l'agent, grâce à une fonction récompense. L'agent peut apprendre de son environnement et observe les conséquences des actions qui ont été prises. Cela lui permet de proposer à son environnement une meilleure paramétrisation.

La question à se poser est la suivante : Où peut-on intégrer l'agent dans ce modèle de

classes ? (OS2) Autrement dit, il faut se demander quels sont les liens entre l'agent et le reste des classes. Nous proposons un diagramme de classes avec agent dans la Figure 3.3.

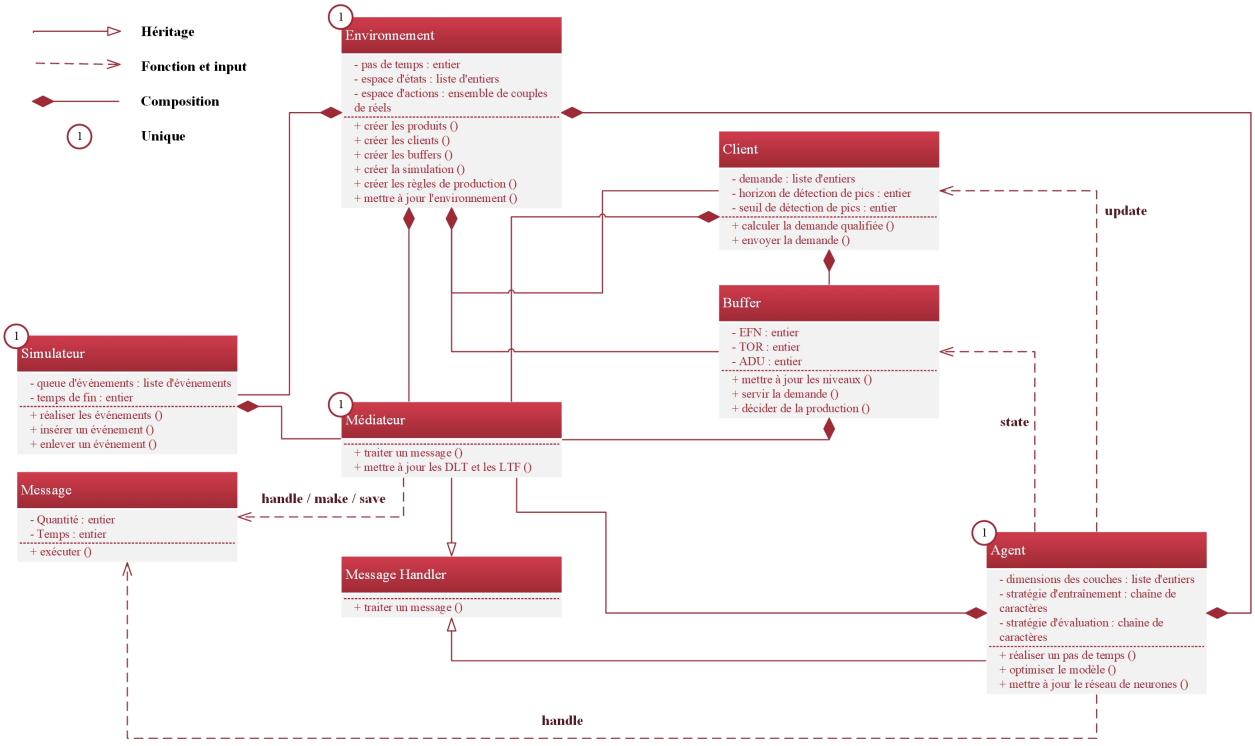


FIGURE 3.3 UML simplifié du modèle avec intégration de l'agent

Dans ce diagramme, toutes les liaisons de l'agent sont représentées, même si le diagramme est toujours simplifié. Ce sont avec ces classes que l'agent a le plus d'interactions. Cette répartition a été choisie afin de répondre au cahier des charges de l'agent, par exemple pouvoir accéder aux paramètres de l'environnement et les modifier, ou bien même accéder à l'état d'un buffer.

L'**Agent** interagit avec les autres classes, il est composé du **Médiateur** pour assurer la communication avec elles. Afin de rendre la modification de paramètre faisable, il faut pouvoir accéder à l'agent depuis l'environnement, autrement dit l'**Environnement** doit être composé de l'**Agent**. Cette composition est en fait double, puisque l'agent doit accéder aux **Buffers** et aux **Clients**. Il peut le faire en passant par l'**Environnement**. Les liens sont davantage développés et expliqués dans la partie **4.3** et **4.4**.

3.2 Méthodologie générale

Dans cette section, nous présentons la méthodologie utilisée pour résoudre le problème posé. Le sujet du projet est l'intégration d'un algorithme d'apprentissage par renforcement à une simulation dans un environnement géré en DDMRP afin d'y optimiser la paramétrisation.

La méthodologie est résumée dans la Figure 3.4 et est détaillée dans les chapitres suivants. La section 4.1 porte sur le choix du modèle, nous y décidons notamment des éléments de la paramétrisation à optimiser. Nous nous positionnons également sur l'algorithme d'apprentissage par renforcement. La section 4.2 est une description fonctionnelle du modèle déjà existant. Cela permet d'apporter des précisions quant au fonctionnement du modèle déjà utilisé et met en avant les éléments que nous modifier. La section 4.3 porte sur l'implémentation du modèle. D'une part le modèle DDMRP développé par [4] est adapté à l'intégration d'un agent, d'autre part nous créons un agent adapté à notre cas d'étude (OS1). Il nous est alors possible d'intégrer l'agent au modèle dans la section 4.4 (OS2). La résultante de cette intégration est vérifiée dans la section 4.5, où nous effectuons le calibrage (OS3). Nous nous aidons d'un environnement test et nous modifions la demande et les paramètres de notre algorithme.

Notre travail présente plusieurs agents que nous allons comparer afin d'évaluer leurs performances. C'est pourquoi, dans la réalisation des phases, nous revenons régulièrement à la phase 4.4 après la phase 4.5. Le retour à la phase 4.4 après la phase 4.5 consiste à de simples ajustements de l'intégration du modèle, par exemple l'optimisation des paramètres d'apprentissage par renforcement. La méthodologie est de créer un agent (OS1), l'intégrer (OS2), le vérifier, le valider (OS3), et en créer un nouveau en répétant ces étapes. Le chapitre 5 traite de l'expérimentation, qui est réalisée à travers trois expériences. L'expérimentation nous permet d'apporter une analyse des résultats dans la partie 5.5. Enfin nous concluons le projet en apportant des recommandations et en évoquant les limites.

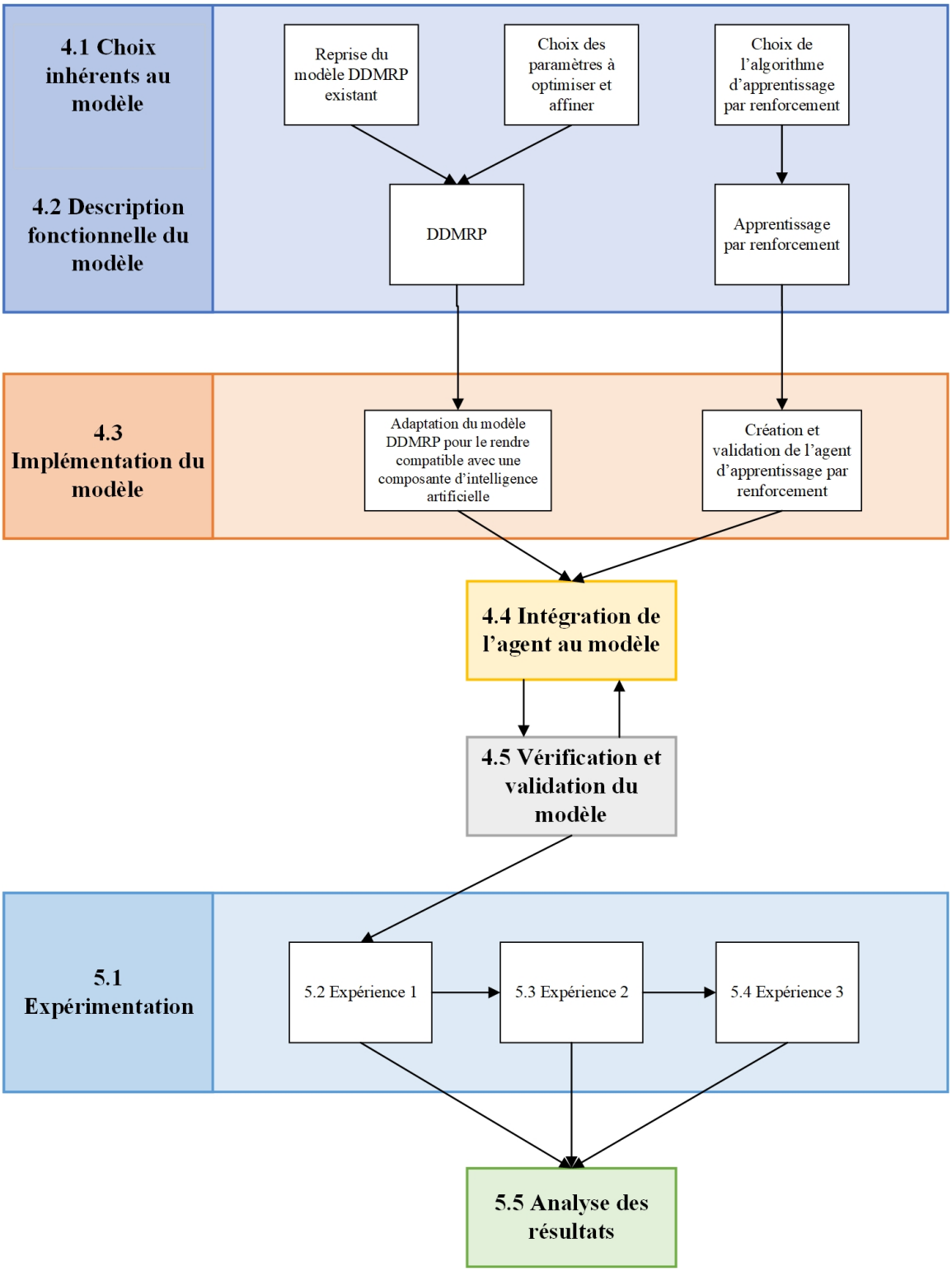


FIGURE 3.4 Méthodologie générale

3.3 Conclusion

Dans un premier temps, nous avons exposé le cas d'étude en présentant l'étendue des travaux déjà réalisés et sur lesquels nous allons nous appuyer. Dans un second temps, nous avons établi une méthodologie divisée en plusieurs étapes. Cette méthodologie va nous permettre de résoudre le problème de façon rigoureuse. Il est désormais possible de passer à la modélisation et à l'implémentation de la simulation.

CHAPITRE 4 MODÉLISATION ET IMPLÉMENTATION

Dans ce chapitre, nous présentons d'abord les choix inhérents au modèle, notamment en rapport avec l'apprentissage par renforcement. Ensuite, nous proposons une description fonctionnelle du modèle, qui permet d'appréhender la structure de la simulation. Cela nous permet d'expliquer l'implémentation du modèle, et donc l'intégration de l'agent au modèle. Enfin, nous vérifions et nous validons le modèle.

4.1 Choix inhérents au modèle

Avant de présenter la description fonctionnelle du modèle, nous exposons les choix et hypothèses inhérents à l'intégration d'un agent. Nous rappelons également que notre étude porte sur la paramétrisation du système, il est donc nécessaire de justifier la décision des paramètres à optimiser.

4.1.1 Espace d'action et espace d'état

Nous résumons dans le Tableau 4.1 les paramètres d'un buffer, et nous précisons si nous les pilotons par agent ou pas. Autrement dit, nous définissons l'espace d'actions. Cette décision de pilotage est motivée par l'état de la littérature autour du DDMRP.

Les OST et les OSH, de par leur faible présence dans la littérature, et n'ayant aucune méthode de calcul explicitement recommandée [1], constituent de bonnes cibles pour le pilotage par agent. De plus, les auteurs précisent qu'il est possible de calculer ces paramètres avec des données historiques, ce qui est tout à fait adapté à l'utilisation d'un algorithme d'apprentissage machine [1].

Pour rappel, l'OST ou "seuil de détection de pic", sert à limiter l'effet d'un pic de commande qui pourrait venir menacer l'intégrité du buffer [1]. L'OSH ou "horizon de détection de pic" sert à savoir si nous devons considérer ou non un pic de commande dans la demande qualifiée en fonction de la date de la commande. Nous rappelons que les commandes d'approvisionnement se font selon l'équation du flux net, dont l'expression est donnée à l'équation (4.1) [1]. Dans cette équation, la demande qualifiée correspond à la demande du jour à laquelle on additionne la demande des pics détectés dans l'horizon de détection [8].

$$EFN = Stock\ en\ main + Stock\ en\ cours\ de\ production - demande\ qualifiée \tag{4.1}$$

Tableau 4.1 Liste des paramètres d'un buffer

Paramètre	Rôle	Piloté par agent
ADU	Relater la moyenne des demandes journalières sur une fenêtre de temps	Non
DLT	Calculer le temps cumulé entre le buffer et les références précédentes	Non
LTF	Considérer les incertitudes sur les temps d'acheminement	Non
VF	Considérer les incertitudes sur la demande	Non
MOQ	Définir la taille minimale d'une commande	Non
OST	Définir un seuil pour lequel une commande est considérée comme un pic	Oui
OSH	Définir un horizon pour lequel les pics de commande sont détectés	Oui

Dans la Figure 4.1, la demande qualifiée correspond à la somme des demandes du jour 1 (le jour actuel) et celle du jour 5 (les commandes considérées dans le calcul de la demande qualifiée sont affichées en noir dans la figure), dans la mesure où le jour 5 est dans l'horizon de détection et que sa demande dépasse le seuil de détection. Le jour 7 n'est pas compté dans la demande qualifiée, bien que sa demande dépasse le seuil de détection, car il n'est pas dans l'horizon de détection. Dans notre exemple, la demande qualifiée est égale à 30 (jour 1) + 130 (jour 5) = 160 pièces.

Ptak et Smith [1] proposent les méthodes suivantes afin de calculer l'OST :

1. A 50% du TOR ;
2. A la base de la zone rouge (valeur haute de la zone rouge sécuritaire) ;
3. Avec l'ADU ou *Average Daily Usage*, avec $OST = 3 * ADU$.

Quant à l'OSH, il est recommandé de lui donner une valeur d'au moins un DLT [1]. Il est donc possible de faire varier sa valeur à $1.25 * DLT$ ou même $1.5 * DLT$. Toutefois, l'OST et l'OSH peuvent tous les deux être déterminés avec des données historiques [1]. Utiliser un historique de données en rapport avec ces valeurs consiste en une façon innovante de les calculer.

L'espace d'action interagit donc avec les OST et les OSH. La possibilité d'un espace d'action

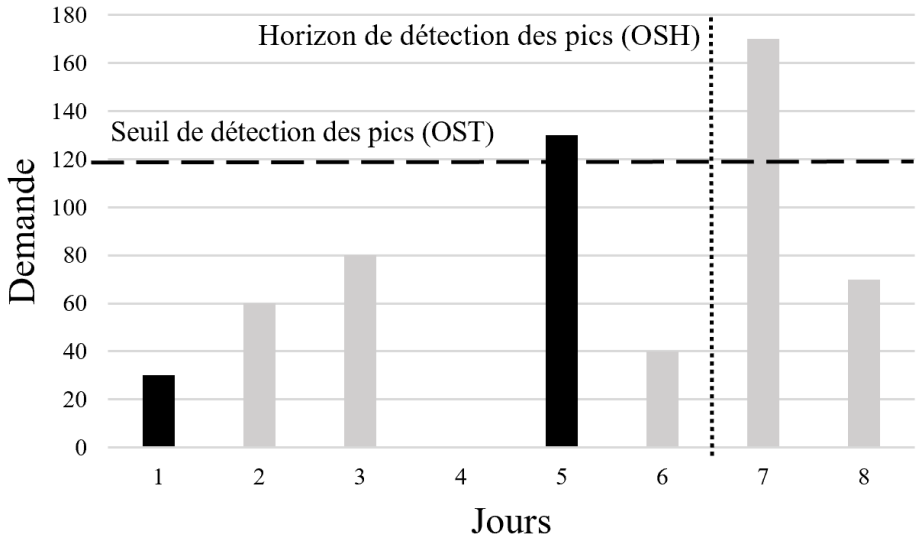


FIGURE 4.1 Calcul de la demande qualifiée

continu a été écartée, car cela impliquait trop de complexité. Afin de contourner le problème, nous discrétisons l'espace d'action, en utilisant un facteur d'OST et un facteur d'OSH. Ces facteurs peuvent prendre un nombre de valeurs finis dans l'intervalle $[0, 1]$. Une action est donc un couple de la forme $(\alpha_{OST}, \alpha_{OSH})$. Si les facteurs peuvent prendre n valeurs finies chacun, l'espace d'actions a pour dimension n^2.

Les facteurs représentent les variations des OST et des OSH, voir les équations (4.2) et (4.3), nous considérons que :

$$OST = \alpha_{OST} * TOR, \tag{4.2}$$

$$OSH = DLT * (1 + \alpha_{OSH}), \tag{4.3}$$

où TOR (*top of red*) est le niveau supérieur de la zone rouge du buffer, DLT (*decoupled lead time*) est le délai de découplage. Si nos facteurs peuvent prendre $n = 4$ valeurs chacun, alors $\alpha_{OST} \in \{0.25; 0.5; 0.75; 1.0\}$ et $\alpha_{OSH} \in \{0.25; 0.5; 0.75; 1.0\}$. Ces équations sont déterminées en fonction des recommandations de Ptak et Smith, afin d'avoir $OST < TOR$ et $OSH > DLT$ [1].

Quant à l'espace d'état, nous proposons d'utiliser un espace multi-état qui pourra être constitué par exemple des NFP ou des ADU de nos différents buffers. La modification de l'espace d'état est un élément important de l'optimisation du modèle, ce qui correspond à la modifi-

cation des paramètres d'apprentissage par renforcement dans le but de rendre l'apprentissage plus efficace.

Comme on peut le voir dans la Figure 4.2, le réseau de neurones prend en entrée les valeurs de notre espace d'états. Ces valeurs d'entrée vont parcourir les couches cachées, et vont ressortir au niveau de la couche de sortie, qui nous donne les Q-valeurs des actions possibles. Les Q-valeurs correspondent aux valeurs de Q-fonction (voir équation 4.5) de nos différentes actions possibles, et donc de nos différents couples $(\alpha_{OST}, \alpha_{OSH})$. Avec l'exemple $n = 4$, nous considérons 16 actions possibles.

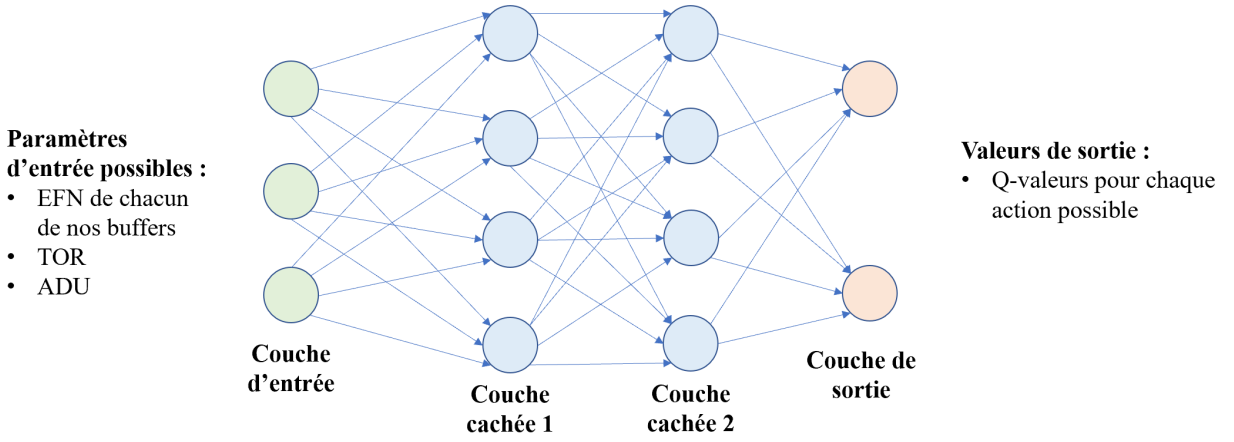


FIGURE 4.2 Schéma simplifié du fonctionnement du réseau de neurones

Un choix doit également être fait quant à la récurrence des actions de décision, elles peuvent être par exemple quotidiennes ou hebdomadaires. Nous envisageons dans un premier temps des interactions de l'agent hebdomadaires et des mises a jours du réseau de neurones bimensuelles (autrement dit une fois toutes les deux semaines). Nous voulons laisser au système le temps d'adopter la décision et qu'elle ait un impact significatif. Ces valeurs sont réétudiées au moment de l'optimisation des paramètres du RL.

Une fois les paramètres à piloter identifiés, nous nous intéressons par la suite à l'algorithme d'apprentissage par renforcement.

4.1.2 Fonctions récompense

Nous rappelons que l'objectif de l'apprentissage par renforcement est d'adopter une stratégie (ou politique) qui maximise la fonction récompense. La fonction récompense est une fonction qui est construite dans le but d'atteindre un objectif particulier. Pour notre atelier de production géré en DDMRP, nous considérons les trois objectifs suivants :

— Optimiser les stocks ;
— Maximiser la satisfaction client ;
— Optimiser les stocks et maximiser la satisfaction client.

Ces objectifs sont associés à trois agents distincts. Ces agents ont chacun une fonction récompense différente, nous les présentons dans le Tableau 4.2. Dans ce tableau, l'équation du flux net (EFN) est introduite à l'équation 4.1. **TOG**, **TOY** et **TOR** sont respectivement les niveaux supérieurs des zones vertes, jaunes et rouges de nos buffers. *On-time Delivery* (**OTD**) est, pour un échantillon de demandes, le ratio des demandes livrées complètes et à temps sur le total des demandes reçues [4] au cours de cet échantillon. Nous rappelons que les fonctions récompense interviennent dans le calcul de la Q-fonction (voir équation 4.5), dans la mesure où cette fonction calcule la somme des futures récompenses espérée d'une action prise à un état donné et à un pas de temps t.

Tableau 4.2 Les trois différents agents

Agent	Objectif	Fonction récompense
1	Optimiser les stocks	$R_1 = \begin{cases} -(1+\frac{EFN-TOG}{EFN}) & \text{si } TOG < EFN \\ -1 & \text{si } TOY < EFN \leq TOG \\ 3 & \text{si } TOR < EFN \leq TOY \\ 0 & \text{si } 0 \leq EFN \leq TOR \\ -1 & \text{si } EFN < 0 \end{cases}$
2	Maximiser la satisfaction client	$R_2 = OTD$
3	Optimiser les stocks et maximiser la satisfaction client	$R_3 = R_1 + OTD$

La fonction récompense du premier agent est une fonction inspirée de l'article de Cuartas Murillo et Aguilar [30] sur le développement d'un algorithme hybride d'apprentissage par renforcement et de DDMRP utilisé pour la gestion de stock. Cette fonction vise à évaluer la position de l'EFN dans les buffers afin d'optimiser les niveaux de stock. Nous récompensons une EFN située dans la zone jaune. Nous donnons une récompense neutre à une EFN située dans la zone rouge. Nous pénalisons une EFN située dans la zone verte. Nous pénalisons une EFN négative, cela peut arriver en cas de pénurie. Enfin nous pénalisons de façon pondérée quand l'EFN est au dessus de la limite supérieure verte, plus l'EFN est éloignée de cette limite supérieure, plus la pénalité est grande. La version de Cuartas Murillo et Aguilar [30] ne considérait pas la possibilité d'être au dessus de la zone verte ou en pénurie. Nous avons ajouté ces deux éventualités afin d'avoir une fonction récompense couvrant tous les états possibles.

Le deuxième agent adopte une logique différente : nous utilisons un indicateur proposé par Martin qui vise à améliorer la "qualité du service" [4]. Nous définissons l'OTD avec l'équation 4.4, en considérant un échantillon de demandes reçues sur une fenêtre de temps définie. Cette fonction récompense tend à maximiser la satisfaction client mais ne considère pas les niveaux de stock.

$$OTD = \frac{Nombre\ de\ demandes\ livrées\ et\ complètes}{Nombre\ de\ demandes\ reçues} \tag{4.4}$$

Le troisième et dernier agent est un agent hybride des deux premiers. C'est la somme des deux premières fonctions récompense. Nous espérons, avec cet agent, optimiser à la fois les stocks et la satisfaction client.

4.1.3 Choix de l'algorithme d'apprentissage par renforcement

L'apprentissage par renforcement a une grande diversité d'algorithmes qui s'appliquent à différents types de problèmes. Nous soulignons donc l'importance de bien définir le problème afin de choisir un algorithme cohérent à sa résolution.

Nous nous sommes rapidement tournés vers un algorithme d'apprentissage par renforcement utilisant un réseau de neurones. Ce choix a été fait en considérant la littérature développée de ce type d'algorithme. De plus, il a fallu considérer les enjeux de dimension. Les problèmes d'apprentissage automatique amènent souvent à traiter des espaces de grande dimension, notamment des espaces d'état de petite taille qui prennent des valeurs dans des ensembles continus et potentiellement infinis. Dans de nombreux cas, on veut utiliser une fonction d'approximation afin de calculer la Q-fonction, dont nous rappelons la définition à l'équation 4.5, avec $G_t = R_{t+1} + R_{t+2} + ... + R_T$, R_t étant la récompense obtenue au pas de temps t. En effet, il est impossible de calculer la valeur exacte à chaque itération sans avoir une complexité élevée. La solution que nous avons retenue est d'utiliser un réseau de neurones, cet outil de l'intelligence artificielle a l'avantage de pouvoir découvrir des "relations sous-jacentes" entre les Q-valeurs. L'agent peut ainsi apprendre de ces relations entre Q-valeurs et les exploiter [19]. Dans notre cas, notre donnée principale est la demande. C'est une donnée à valeurs réelles.

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \tag{4.5}$$

Nous utilisons un algorithme de *Branching Dueling Q-Network* (BDQ). Le BDQ vise à régler des problèmes ayant des espaces d'actions à plusieurs dimensions [32]. L'idée est de trai-

ter "chaque dimension d'action avec un degré d'indépendance" [32]. Cela permet de réduire considérablement les sorties du réseau de neurones. L'architecture du réseau de neurones se divise entre les différentes dimensions d'action pour représenter la fonction Q-valeur, tout en conservant un "module de décision partagé" afin de conserver une trace de l'état d'entrée [32]. L'architecture BDQ a été reconnue comme efficace dans des environnement ayant des actions similaires, et permet une exécution plus rapide dans des espaces d'actions de grande taille [32].

Une représentation du BDQ est proposée à la Figure 4.3. Dans cette figure, les trapèzes représentent les couches cachées du réseau de neurones avec leur taille associée. Ces couches cachées peuvent être communes, associées à l'état ou associées aux actions. Cette terminologie est utilisée afin de les différencier car elles peuvent être paramétrées de façons différentes. La "Valeur de l'état" représente la composante partagée entre toutes les actions, tandis que la partie action-avantage, propre à chaque action, est représentée à travers les cases d'"Avantages dimension" et les "Q-valeurs". Nous avons distingué les différentes dimensions en les colorant de différentes façons (bleu, orange et vert). Cette représentation est reprise de Tavakoli et al. [32] et a été modifiée afin de s'adapter à notre problème. L'implémentation en Python du réseau de neurones est disponible en **Annexe D** avec les éléments de code.

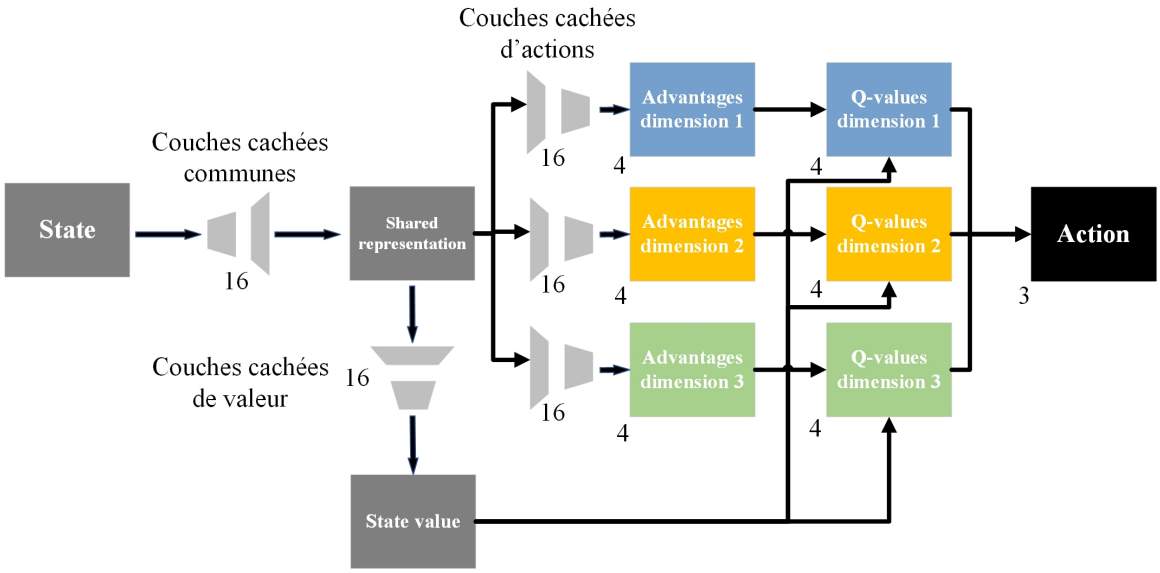


FIGURE 4.3 Architecture du BDQ inspirée de Tavakoli et al. (2018)

En plus de cette architecture de réseau de neurones, nous reprenons l'idée d'un tampon de relecture priorisé de Tavakoli et al. [32]. Ce tampon de relecture permet de prioriser les expériences permettant un bon apprentissage, plutôt que de choisir les expériences de façon uniforme.

Nous utilisons le langage Python sur l'éditeur de code VS Code. Python nous permet une prise en main facile accompagnée d'une grande liberté. Cela n'aurait pas été permis avec un logiciel de simulation tel que Arena ou Anylogic. La libraire retenue est Keras, du module Tensorflow. Keras est une API (*Application Programming Interface*) d'apprentissage profond écrite en Python. Keras a été développé afin de permettre une expérimentation rapide, "être capable d'aller d'une idée aussi vite que possible est la clé pour faire de la bonne recherche" [33]. Keras présente les avantages d'être un module simple, flexible et puissant.

La justification des choix et hypothèses ayant été présentés, nous pouvons passer à la description fonctionnelle du modèle.

4.2 Description fonctionnelle du modèle

Une brève description du modèle et de son fonctionnement ont été présentés à travers le section **3.1**. Néanmoins, il est nécessaire d'approfondir la description fonctionnelle du modèle afin de faciliter la compréhension lors de l'intégration de l'agent au modèle. Pour cela, nous exposons certains aspects fonctionnels et pratiques du modèle DDMRP, avant de revenir sur l'apprentissage par renforcement.

4.2.1 Modèle de l'atelier de production géré en DDMRP

Pour simuler l'atelier de production introduit dans la section **3.1.1**, le choix a été fait d'utiliser une simulation à événements discrets (SED). Ce type de simulation repose sur un moteur d'exécution des événements [4]. Elle comporte également une file d'attente qui va stocker des événements caractérisés par une date d'exécution et un ensemble de règles de priorité. A chaque fois qu'un événement est réalisé, on prend le prochain événement par date d'exécution. La SED présente l'avantage d'être très présente dans la littérature pour la modélisation du DDMRP [4, 12, 13]. De plus, les algorithmes à événements discrets présentent de récentes utilisations de l'apprentissage par renforcement profond [32].

L'atelier est paramétré de façon à avoir au moins un produit et une machine. De plus, nous considérons qu'il y a un buffer par produit fini et un buffer par matière première.

La génération de la demande est inspirée du modèle de Dessevre [8]. Un profil de demande est caractérisé par le temps interarrivée de deux commandes. Chaque produit présente des profils de demande différents car ils sont générés de façon aléatoire. Néanmoins le paramètre d'interarrivée de deux commandes est le même pour tous les produits d'une même réplication. Les temps d'arrivée entre deux commandes sont distribués selon une loi exponentielle de paramètre 2 jours. La taille d'une commande est tirée uniformément à $\pm 20\%$ de la taille

moyenne du profil de demande. La taille moyenne du profil de demande est également tirée de façon uniforme entre 50 et 200 produits. A ces commandes "régulières" s'ajoutent des pics de commande. Les pics de commandes sont distribués selon une loi exponentielle de moyenne 5, 10 ou 20 jours. La taille des pics de commande varie entre 2, 3, 4 ou 5 fois la taille d'une commande régulière, ce facteur de taille est également tiré de façon uniforme. Un exemple de profils de demande est illustré à la Figure 4.4 pour une simulation à 3 produits et pour une fréquence de pics de commande tous les 20 jours.

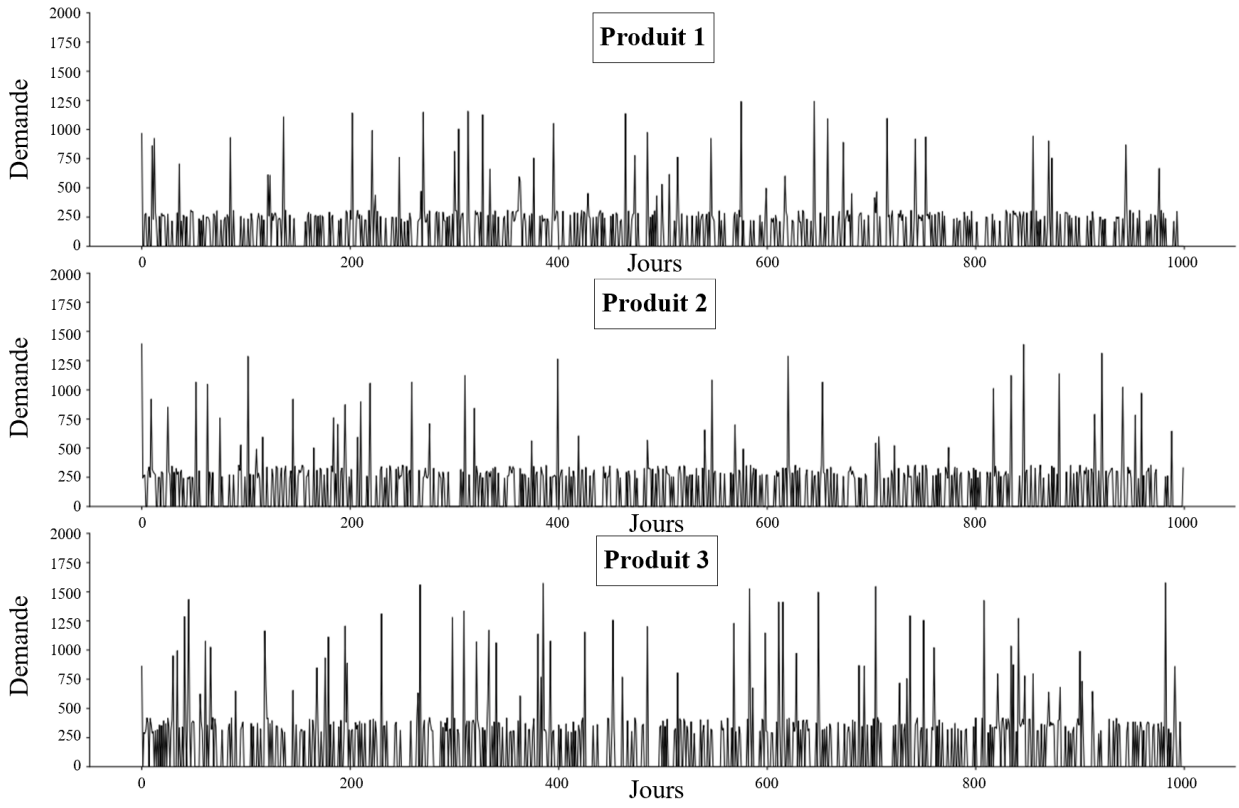


FIGURE 4.4 Profils de demande à 3 produits

Le modèle développé par Martin [4] permet de faire varier la charge de travail de l'atelier parmi 3 valeurs : 50%, 80% et 95%. Les temps d'opération et de mise en route des machines ont été calculés afin de correspondre avec ces charges de travail. Faire varier la charge de travail revient en fait à modifier le nombre de machines par ligne de production. Nous utiliserons ces paramètres afin d'étudier l'effet de la charge de travail sur la performance de l'apprentissage par renforcement.

La section **3.1.2** présente une simulation divisée en plusieurs classes. Les classes de types "élément" (**produit**, **machine**, **buffer**...) occupent un rôle classique : le client demande des produits au système, chaque produit est caractérisé par une nomenclature, les produits sont

fabriqués par une ou plusieurs machines. Nous précisons que chaque machine est associée à une file d'attente et un serveur qui gère un ordre à la fois. Enfin, chaque buffer est un buffer physique au sens du DDMRP [1, 4].

Le simulateur a pour objectif d'exécuter et distribuer les événements, il est associé à une file d'attente et un serveur qui va exécuter les événements de la file d'attente. Comme il a été indiqué précédemment, un événement est caractérisé par une date d'exécution et une instruction. Finalement, nous précisons que tous les événements sont traités selon un ordre, dans le cas où deux événements devraient être traités à la même date [4]. Voici le détail par ordre croissant de priorité :

1. servir les commandes en attente ;
2. mettre à jour l'*Average Daily Usage* ou ADU (Commande moyenne journalière) ;
3. mettre à jour le *Decoupled Lead Time* ou DLT ;
4. mettre à jour le *Lead Time Factor* ou LTF ;
5. mettre à jour les tailles des zones ;
6. décider des quantités à produire.

L'algorithme du modèle DDMRP sans agent est présenté ci-dessous.

Entrer les paramètres du modèle DDMRP (les politiques, le nombre de produits, de clients, la demande, etc.) ;
Créer l'environnement du modèle DDMRP, qui va contenir le problème, les valeurs des paramètres et les politiques ;
pour *chaque réplication* **faire**
    Créer un problème ;
    Initialiser l'environnement ;
    Créer les éléments du modèle DDMRP (les produits, les buffers...) ;
    Exécuter les événements du modèle ;
    Évaluer le modèle en calculant les indicateurs ;
fin
Afficher les indicateurs

Algorithme 1 : Structure du modèle DDMRP

4.2.2 Apprentissage par renforcement

La section précédente présente un modèle de simulation à événements discrets. Notre objectif est donc d'intégrer un algorithme d'apprentissage par renforcement au modèle de simulation. Pour ce faire, nous intégrons d'abord deux événements propres à l'apprentissage par renforcement :

— le **pas de temps** correspond à la réalisation d'un cycle présenté à la Figure 2.7. Pour rappel, un pas de temps est composé d'un état, une action, une récompense et un nouvel état. Concrètement, on fait avancer la simulation en modifiant nos paramètres ;
— l'**entraînement du réseau de neurones** est une phase d'amélioration de l'approximateur de la Q-fonction. Cet événement permet d'avoir un agent mieux entraîné qui prendra supposément de meilleures décisions.

Ces événements viennent s'intégrer aux événements déjà existants et sont traités selon une fréquence de réalisation. En effet, chaque événement est réalisé et est créé à des intervalles de temps réguliers. Par conséquent, il nous faut décider de la fréquence de réalisation de nos deux nouveaux événements.

Avant de présenter la structure de l'algorithme, nous proposons les éléments de vocabulaire suivants :

1. Un **scénario** est une combinaison de variables (politiques d'ADU, DLT, demande...) qui simule une situation réelle ;
2. Une **réplication** est la répétition d'un scénario. À chaque nouvelle réplication, l'agent est réinitialisé. Les réplications permettent de considérer les effets de l'aléatoire ;
3. Un **épisode** est un ensemble de pas de temps, caractérisé par le début et la fin d'une tâche.

Nous rappelons qu'un état correspond à un ensemble de valeurs relatant d'une observation de l'environnement. Une action vient modifier l'environnement. Une récompense traduit l'effet d'un changement d'état, selon un objectif à définir. La Q-fonction définie à l'équation 2.6 vise à estimer les conséquences d'une action. Le but de notre algorithme est de construire une politique maximisant nos récompenses [21].

Un tampon de relecture est utilisé dans l'algorithme du BDQ. Ce tampon de relecture sert à stocker des échantillons dans une mémoire de relecture. Ils sont utilisés pour calculer de nouvelles cibles et optimiser le réseau à l'aide de petits échantillons. Cet échantillonnage permet d'améliorer la diversité du processus d'entraînement ainsi que ses performances. Nous utilisons des petits échantillons de taille 16.

L'algorithme du BDQ est présenté ci-dessous à l'Algorithme 2. Il est inspiré de celui présenté par Hasselt [34].

1 **Initialiser** le réseau primaire Q_θ, le réseau cible Q'_θ et le tampon de relecture ;
2 **pour** *chaque épisode* **faire**
3 | **pour** *chaque pas de temps* **faire**
4 | | **Observer** l'état s_t et **sélectionner** une action a_t en utilisant une politique dérivée de q ;
5 | | **Réaliser** l'action a_t et **observer** le nouvel état s_{t+1} et la récompense r_{t+1} ;
6 | | **Stocker** (s_t, a_t, r_t, s_{t+1}) dans le tampon de relecture ;
7 | **fin**
8 | **pour** *chaque étape de mise à jour* **faire**
9 | | **échantillonner** du tampon de relecture $e_t = (s_t, a_t, r_t, s_{t+1})$;
10 | | **Attribuer** à $q^*(s_t, a_t)$ la valeur $r_t + \gamma q_\theta(s_{t+1}, argmax_{a'} q_{\theta'}(s_{t+1}, a'))$;
11 | | **Effectuer** la descente du gradient sur $(q^*(s_t, a_t) - q_\theta(s_t, a_t))^2$;
12 | | **Mettre à jour** les poids du réseau cible ;
13 | **fin**
14 **fin**

Algorithme 2 : Algorithme du *BDQ*

Nous précisons le rôle des lignes suivantes de l'Algorithme 2 :

- ligne 1 : initialiser un réseau de neurones consiste à initialiser les poids de ses noeuds de façon arbitraire ;
- ligne 3 : une étape de pas de temps correspond à la sélection d'une action et la mise à jour de l'espace d'états ;
- ligne 4 : une politique est une fonction qui prescrit une action à faire en fonction d'un état quelconque de l'environnement. La sélection d'action se fait à l'aide d'une stratégie de sélection ;
- ligne 6 : le tampon de relecture stocke les expériences dans un historique de données ;
- ligne 8 : mettre à jour le réseau de neurones consiste à mettre à jour les poids du réseau cible afin qu'il soit plus adapté au calcul de la Q-fonction ;
- ligne 9 : échantillonner permet de limiter les effets d'un grand espace d'états ou grand historique de données ;
- ligne 10 : il s'agit de calculer la Q-valeur de la l'action choisie. Elle est calculée à l'aide du réseau primaire et du réseau cible, à partir d'une relation de récurrence et d'une séparation en deux composantes de la Q-fonction ;
- ligne 11 : la valeur calculée est la fonction perte, elle évalue les performances de prédiction du réseau de neurones. Effectuer une descente de gradient dessus permet de l'optimiser et trouver ses zones d'intérêt.

Similairement à Morales, nous avons introduit le facteur γ [19] (ligne 10 de l'Algorithme 2).

Il permet d'ajouter une dimension temporelle à l'apprentissage par renforcement : c'est la résolution du problème de crédit temporel. La récompense obtenue à une certaine action peut dépendre des actions ayant été prises précédemment. Il faut donc pouvoir pondérer le poids d'une action sur une récompense en fonction du moment où elle a été prise : c'est le rôle de γ. Une action prise 10 pas de temps plus tôt aura un poids moins important qu'une action prise 1 ou 2 pas de temps avant. Nous fixons γ à 0.99.

Nous rappelons qu'un algorithme d'apprentissage par renforcement doit régler la question du compromis exploration-exploitation. L'exploration permet d'essayer des actions de façon aléatoire et d'évaluer leur impact avec la fonction récompense, tandis que l'exploitation tend à prendre les meilleurs chemins et prendre les meilleurs actions étant donné un état. Un algorithme d'apprentissage par renforcement doit pouvoir équilibrer ces deux notions [19]. La phase de "Sélection d'action" comprend donc en fait une décision en rapport avec l'exploration et l'exploitation. La stratégie qui ne comprend que de l'exploitation est appelée stratégie "gloutonne" (*greedy strategy*). Elle est souvent utilisée pour la phase d'évaluation. Pour l'entraînement, on favorisera une stratégie faisant intervenir un facteur ϵ. ϵ peut être fixe ou variable, son rôle est d'effectuer un pourcentage d'actions en exploration et un autre pourcentage en exploitation. On effectue un tirage uniforme sur $[0, 1]$, si le tirage est plus grand que ϵ, on exploite, sinon on explore [19]. Pour la phase d'entraînement, nous avons choisi une stratégie gloutonne à ϵ exponentiellement décroissant : à chaque fois qu'on sélectionne une action, on décroît ϵ de façon exponentielle. Nous initialisons ϵ à 0.9, et nous posons sa valeur minimale à 0.1. Cela permet à la phase d'entraînement d'effectuer beaucoup d'exploration au début, et de progressivement favoriser l'exploitation. Les codes Python des stratégies gloutonne et à ϵ exponentiellement décroissant sont disponibles en **Annexe D** avec les éléments de code.

Nous avons présenté plus en détails le modèle DDMRP ainsi que les éléments essentiels de notre algorithme d'apprentissage par renforcement (OS1). Nous pouvons maintenant présenter les modifications apportées au modèle.

4.3 Implémentation du modèle

4.3.1 Modèle conceptuel

Le modèle conceptuel est présenté à la Figure 4.5. Il reprend tous les éléments présentés précédemment et est composé des étapes d'apprentissage suivantes :

1. observation par l'agent des EFN de chaque produit et de la demande ;

2. attribution d'une récompense à l'aide d'une des fonctions récompense définies à la section **4.1.2** ;
3. amélioration de la politique, autrement dit mise à jour du réseau de neurones unitaire ;
4. sélection d'une action à l'aide de l'espace d'actions défini à la section **4.1.1** ;
5. changement d'état de l'environnement, c'est-à-dire mise à jour des OST et des OSH.

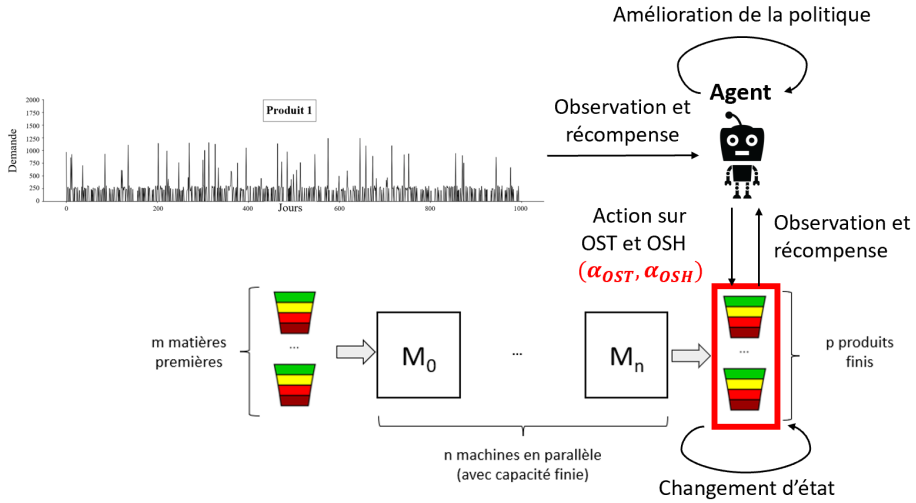


FIGURE 4.5 Modèle conceptuel de l'apprentissage dans la simulation

4.3.2 Adaptation du modèle DDMRP pour le rendre compatible avec une composante d'intelligence artificielle

L'agent est l'élément le plus significatif à intégrer au modèle, voir la Figure 3.3. Cet agent, en plus d'être unique dans la simulation, doit présenter des liens avec les classes avec lesquelles il interagit. Par exemple, l'agent doit pouvoir accéder à l'espace d'état et faire une observation, ou bien même envoyer une requête d'action.

En intégrant l'agent, il ne faut pas créer de cycle. Si un cycle est créé, certaines instances de classes vont être conservées dans la mémoire. Il faut s'assurer que la mémoire puisse être nettoyée sans conserver des références à des objets qui ne devraient pas exister. En évitant les cycles, on s'assure que rien n'est lié de manière cyclique et que la mémoire est nettoyée correctement en parcourant l'arbre des dépendances. Une double liaison de composition est établie entre l'agent et l'environnement. Ainsi l'environnement peut accéder à l'agent, et l'agent peut accéder aux buffers et aux clients en passant pas l'environnement, voir la Figure 4.6.

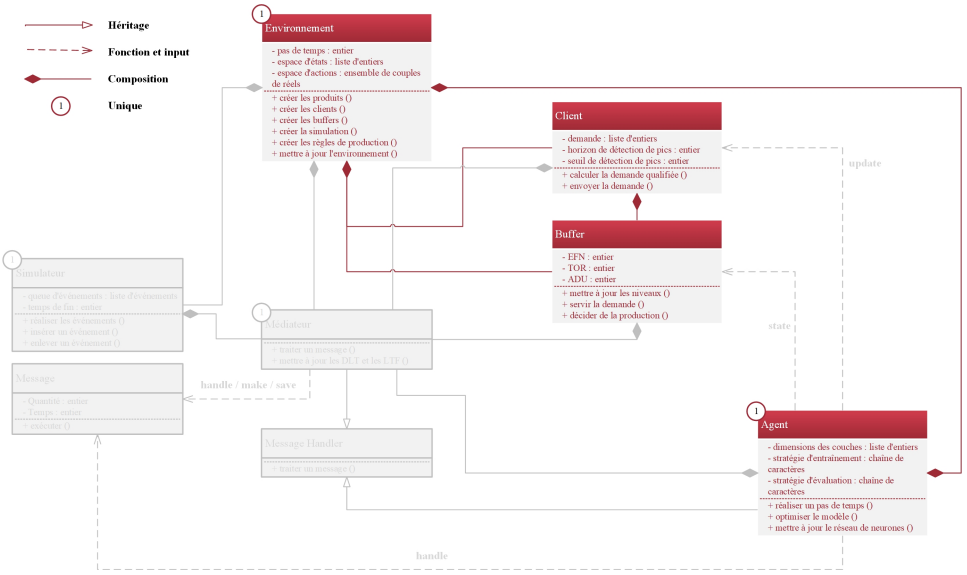


FIGURE 4.6 Liaison de composition double entre l'agent et l'environnement

L'agent doit être également capable d'envoyer et recevoir des messages, cette fonctionnalité est développée plus tard. L'agent est relié au médiateur afin de pouvoir communiquer avec le système, voir la Figure 4.7.

La section **4.2.2** introduit deux nouveaux événements à notre simulation à événements discrets, qui sont les **pas de temps** et l'**entraînement du réseau de neurones**. La notion de **pas de temps** n'est pas vraiment adaptée à la SED, car elle se confond avec les pas de temps de la simulation. Par conséquent, nous remplaçons les **pas de temps** par les **interactions de l'agent**. Une **interaction de l'agent** correspond à une modification de l'espace d'état et un ajustement des paramètres. Afin d'intégrer ces événements à la simulation, nous devons leur attribuer une priorité, comme pour les événements présentés dans la section **4.1.1**. Nous proposons donc la liste de priorités suivante :

1. servir les commandes en attente ;
2. réaliser une interaction de l'agent ;
3. mettre à jour l'*Average Daily Usage* ou ADU (Commande moyenne journalière) ;
4. mettre à jour le *Decoupled Lead Time* ou DLT ;
5. mettre à jour le *Lead Time Factor* ou LTF ;
6. mettre à jour les tailles des zones ;
7. décider des quantités à produire ;
8. mettre à jour le réseau de neurones.

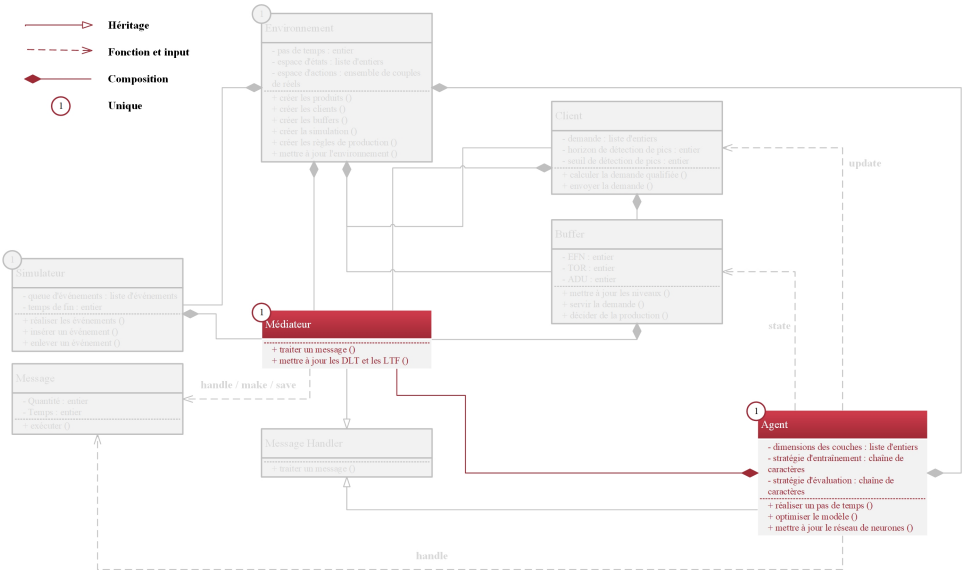


FIGURE 4.7 Liaison de composition entre l'agent et le médiateur

Nous avons décidé de prioriser l'interaction par rapport à la mise à jour des zones et des paramètres afin de mettre à jour au plus tôt l'espace d'état et les paramètres à ajuster. De plus nous effectuons la mise à jour du réseau de neurones en dernier car c'est l'action la moins prioritaire.

Enfin, nous modifions la structure de l'algorithme afin d'y intégrer la notion d'épisodes pour l'apprentissage par renforcement. Le nouvel algorithme de la modélisation avec agent est présenté à l'Algorithme 3 (OS2).

Dans l'Algorithme 3, l'agent et sa capacité à apprendre sont évalués à chaque réplication (ligne 4) de chaque scénario (ligne 2) car il est initialisée à chaque nouvelle réplication (ligne 5). Des réplications sont utilisées afin de limiter les effets des éléments aléatoires de la simulation. Un épisode représenté ensuite une simulation de 1000 jours, stoppée par une contrainte de temps. Pendant l'exécution, les événements de la SED sont réalisés en considérant également les événements propres à l'agent (ligne 10). Nous précisons également que l'agent est évalué à travers différentes statistiques portées sur les récompenses (ligne 11), le score final (ligne 16) et le temps d'exécution (ligne 17).

4.3.3 Création et validation de l'agent d'apprentissage par renforcement

Afin de confirmer le bon fonctionnement de la classe Agent, nous utilisons un environnement Test. Il nous permet de vérifier si le score est bien maximisé et si notre agent arrive à bien appliquer l'apprentissage par renforcement. La validation de l'agent à l'aide de l'environnement

```
1  Entrer les paramètres de la simulation DDMRP (les politiques, le nombre de
     produits, de clients, la demande, etc.) pour chaque scénario ;
2  pour chaque scénario faire
3  |  Créer l'environnement de la simulation DDRMP, qui va contenir le problème, les
   |    valeurs des paramètres et les politiques ;
4  |  pour chaque réplication faire
5  |  |  Créer et initialiser l'agent ;
6  |  |  pour chaque épisode faire
7  |  |  |  Créer un problème associé au scénario ;
8  |  |  |  Initialiser l'environnement ;
9  |  |  |  Créer les éléments de la simulation (les produits, les clients, les buffers...)
   |  |  |    ;
10 |  |  |  Exécuter les événements de la simulation (y compris l'entraînement du
   |  |  |    modèle) ;
11 |  |  |  Evaluer le modèle en calculant la moyenne et la variance des
   |  |  |    récompenses ;
12 |  |  |  si on a atteint une condition d'arrêt alors
13 |  |  |  |  Sortir de la boucle pour ;
14 |  |  |  fin
15 |  |  fin
16 |  |  Calculer le score d'évaluation final ;
17 |  |  Afficher condition d'arrêt, score final, temps d'exécution ;
18 |  fin
19 fin
20 Afficher les indicateurs
```

Algorithme 3 : Algorithme principal du modèle DDMRP en apprentissage par renforcement

test n'est que partielle. En effet, si nous arrivons à créer un agent qui maximise une fonction récompense pour un environnement test, cela ne veut pas dire qu'il le fera pour un autre environnement. Nous devrons prêter attention à la paramétrisation de notre agent, par exemple son espace d'état ou même sa fonction récompense. Ces caractéristiques de l'agent ont un effet important sur sa performance, puisqu'elles influent directement sur son apprentissage.

Nous avons désormais un environnement DDMRP apte à intégrer un agent. De plus, nous avons un agent fonctionnel (OS1). Cela nous permet de passer à l'intégration de l'agent au modèle (OS2).

4.4 Intégration de l'agent au modèle

La phase d'intégration de l'agent au modèle correspond à l'ajout du code d'apprentissage par renforcement à la simulation DDMRP. Il est question de faire communiquer l'agent avec le modèle et enclencher la paramétrisation de l'agent.

Tout comme le reste du modèle, l'agent peut envoyer et recevoir des messages, notamment à travers la classe **Message Handler**. Les classes et les liens dédiés à la communication de l'agent avec le système sont exposés dans la Figure 4.8. L'agent doit pouvoir traiter un message qu'il reçoit, qui vient directement de la liste d'événements. Ces messages sont uniquement les événements d'**interaction** et de **mise à jour du réseau de neurones**. Cela est rendu possible par l'opérateur **handle** de l'agent, qui appelle d'autres fonctions et réalise certaines actions lorsqu'elle reçoit des messages spécifiques.

La première étape a été de faire communiquer l'agent avec un seul buffer. Ensuite, nous sommes passés à un système avec p buffers. Nous précisons également qu'une optimisation des paramètres du RL ont été effectués à cette étape. Cette optimisation s'est faite de façon rétrospective avec la phase de vérification et de validation, ce qui nous a amené notamment à modifier les fonctions récompenses et l'espace d'actions afin de les rendre plus adaptés à l'apprentissage. Cette méthode ne garantit pas l'optimalité des solutions, mais permet tout de même d'obtenir un apprentissage consistant. Nous revenons sur ce point dans la section **4.5.3**.

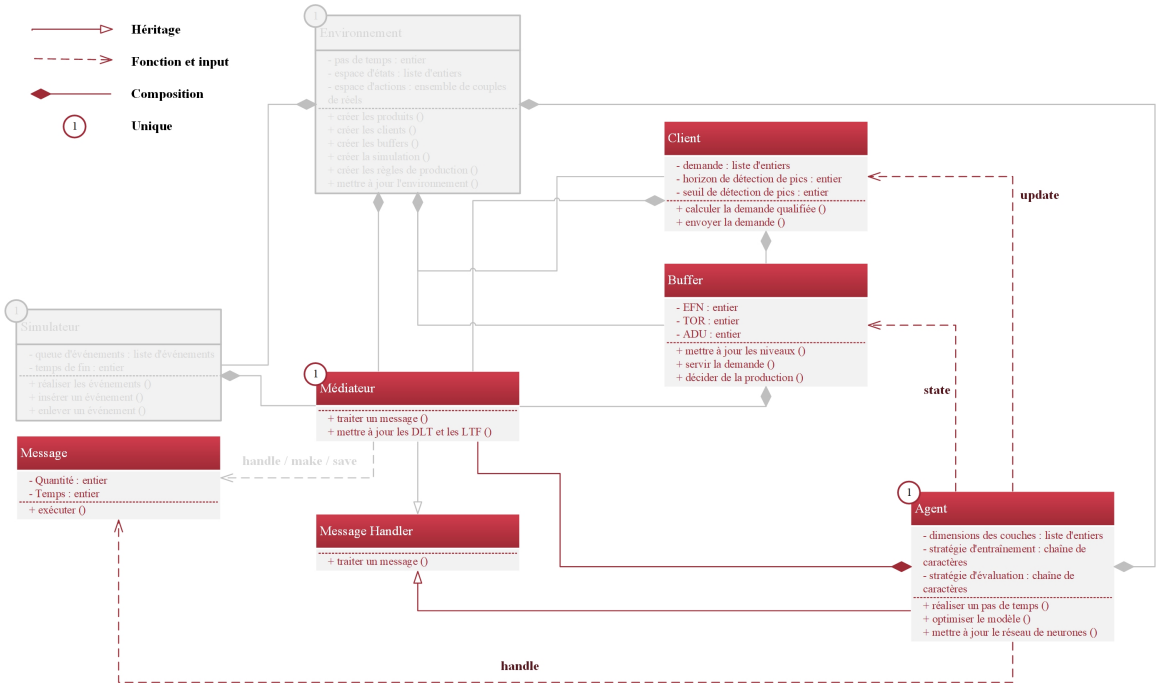


FIGURE 4.8 Mise en avant des liens permettant à l'agent de communiquer avec le système

4.5 Vérification et validation du modèle

La vérification du modèle permet de s'assurer que notre algorithme 3 est correct, autrement dit que l'algorithme produit le résultat attendu. La validation du modèle vise à montrer que nos résultats sont fiables et que le modèle proposé est plus performant qu'un modèle DDMRP sans agent (OS3). Avant ces deux étapes, nous proposons de présenter les indicateurs clés de performance qui sont utilisés pour ces deux étapes.

4.5.1 Indicateurs clés de performance

Nous distinguons les KPI (*Key Performance Indicator* ou Indicateur clé de performance) en deux catégories : les KPI métier et les KPI d'apprentissage par renforcement. Les KPI métier quantifient l'efficacité de l'atelier géré en DDMRP. Ils incluent des informations sur la satisfaction client, le niveau des stocks et l'évolution des ordres de fabrication dans le système. Les KPI d'apprentissage par renforcement évaluent la performance de l'algorithme d'apprentissage par renforcement. Nous y considérons les récompenses moyennes des épisodes et la vitesse de convergence.

Les KPI métier sont les suivants :

1. Les **EFN**. L'évolution des EFN dans les buffers sur un épisode nous donne une idée

assez claire de l'état des stocks de notre modèle.

2. Le **taux de service moyen**. Il s'agit de l'OTD généralisé sur l'ensemble d'un épisode. Autrement dit le ratio entre les demandes livrées et complètes et le nombre total de demandes reçues, inspiré de Martin [4].
3. Le nombre de **pics détectés**. C'est le nombre de pics de commandes détectés en un épisode par le seuil de détection de pics.

Ces indicateurs ont l'avantage de refléter le fonctionnement général de l'atelier sans entrer trop dans les détails, tout en évaluant la satisfaction client. Ils nous permettent également d'expliquer les phénomènes en lien avec la détection des pics.

Les indicateurs d'apprentissage par renforcement sont inspirés de l'article de Cuartas Murillo et Aguilar [30]. Ces indicateurs permettent d'aborder des éléments de complexité, correction et performance de l'algorithme. Les KPI d'apprentissage par renforcement sont les suivants :

1. La **récompense accumulée moyenne** :

$$RAM = \frac{r_1 + r_2 + ... + r_t}{N} \tag{4.6}$$

où N est le nombre d'épisodes et r_t est la récompense à l'épisode t. Cet indicateur est primordial dans la vérification et la validation du modèle, c'est l'indicateur principal d'un algorithme d'apprentissage par renforcement. Nous utilisons une moyenne mobile afin d'évaluer l'apprentissage, par exemple la moyenne des 30 derniers épisodes. Cela permet de mettre davantage en avant le phénomène d'apprentissage et la croissance des récompenses en fonction des épisodes.

2. Le **pourcentage de meilleure récompense accumulée** :

$$PMRA_t = \frac{max(r_1 + r_2 + ... + r_t)}{MRA} \tag{4.7}$$

avec t l'indice d'épisode, r_t la récompense à l'épisode t, $MRA = max(r_1 + r_2 + ... + r_N)$, N étant le nombre d'épisodes total et MRA la meilleure récompense accumulée. Cet indicateur représente "la proportion de meilleure récompense accumulée atteinte" [30]. Elle nous permet de savoir le nombre d'épisodes nécessaire pour arriver à la meilleure récompense accumulée.

3. Le **taux de convergence de l'algorithme** ou TCA. Il nous permet d'évaluer la vitesse de convergence de l'algorithme, que l'on peut obtenir en comparant la RAM et le nombre d'épisodes.

4. Le **taux d'apprentissage** ou TA :

$$TA = RAM_{100} - min(RAM) \tag{4.8}$$

la différence entre la valeur de la RAM à l'épisode 100 et la valeur minimale de la RAM. Cet indicateur nous permet d'évaluer l'efficience de l'apprentissage de l'agent.

Les indicateurs clés de performance ayant été introduits, nous pouvons exposer la méthodologie de vérification du modèle.

4.5.2 Vérification du modèle

La phase de vérification permet de s'assurer que le modèle réalisé fonctionne et que la correction de l'algorithme est bonne. La vérification s'est effectuée à travers ces différentes étapes :

— Débogage du code : il faut d'abord s'assurer que les commandes utilisées sont compatibles avec les conventions Python, et que le programme se termine ;
— Vérification du code par les tiers : une vérification régulière par des tiers a été effectuée afin de s'assurer du bon avancement du code ;
— Utilisation d'un débugger : l'outil de débogage de Python a été utilisé afin de s'assurer de la bonne évolution des variables clés, par exemple l'espace d'état ou la récompense accumulée ;
— Utilisation d'affichages de variables et sorties graphiques : nous avons pu stocker les données de la simulation dans un tableur Excel afin de voir l'évolution des variables entre chaque épisode. De plus, nous nous sommes appuyés sur l'affichage des NFE, les OST, les OSH et les indicateurs clés de performance.

Nous rappelons qu'une vérification préalable de l'agent d'apprentissage par renforcement avait été réalisée à l'étape **4.3** avec un environnement test. La vérification du modèle sert à vérifier le comportement de l'agent dans un autre environnement, avec un espace d'état différent. De plus, il faut vérifier que l'agent a été correctement intégré à la SED.

Les étapes de vérification ont dans un premier temps été effectuées sur un problème avec un seul client et donc un seul produit fini et un seul buffer. Une fois la vérification effectuée sur ce type de problème, nous avons pu élargir le problème à trois buffers et trois produits finis.

4.5.3 Validation et optimisation du modèle

La validation du modèle est l'étape qui nous permet de montrer que le modèle avec agent est plus performant que le modèle sans apprentissage par renforcement (OS3). Utiliser une

approche classique de vérification et validation de simulation, telle que celle proposée par Banks [35], consisterait à faire correspondre les données de simulation à des données réelles. Or nous ne pouvons pas faire correspondre notre simulation à des données réelles puisque nous avons construit un modèle théorique.

La méthodologie proposée est de comparer notre modèle d'apprentissage par renforcement avec le modèle sans intelligence artificielle de Martin [4]. Le modèle est validé si nous parvenons à obtenir des résultats similaires ou meilleurs (qui seront évalués par l'étude de certains KPI). Cette méthodologie est composée des quatre étapes suivantes :

1. Intégrer un agent d'apprentissage par renforcement au modèle DDMRP déjà créé en conservant des données de sortie valides ;
2. Utiliser des données d'entrée et des hypothèses cohérentes ;
3. Optimiser les paramètres d'apprentissage par renforcement afin d'obtenir des résultats optimaux ;
4. Compiler les résultats du modèle avec agent et les comparer avec ceux obtenus pour un modèle sans agent.

Nous avons déjà réalisé les étapes 1 et 2 à travers les sous-sections du chapitre **4**. Il nous reste donc à optimiser le modèle et comparer les résultats.

L'étape d'optimisation est assez particulière car il est possible de modifier de nombreux paramètres. Nous proposons d'abord de distinguer trois types de paramètres :

1. les paramètres de l'atelier : ce sont les paramètres de l'atelier et du modèle DDMRP, par exemple le nombre de clients ou le temps de simulation ;
2. les paramètres inhérents à l'algorithme d'apprentissage par renforcement : ce sont les paramètres qui modifient le comportement de l'agent, par exemple l'espace d'état, l'espace d'action, le facteur γ, la fréquence des interactions, etc. ;
3. la fonction récompense : elle fait partie des paramètres de l'apprentissage par renforcement, mais son effet est assez important pour lui consacrer une étude approfondie.

Pour effectuer l'optimisation, nous avons principalement utilisé les indicateurs d'apprentissage par renforcement. Nous nous intéressons davantage à modifier l'espace d'état, la fonction récompense ou l'espace d'action, plutôt que travailler sur le nombre de clients, le temps de simulation ou les profils de demande. Il a été également question d'optimiser le réseau de neurones afin de considérer les compromis associés. Les paramètres métier du modèle ont été très rarement modifiés afin d'étudier en profondeur l'agent et son paramétrage. Ainsi, les

profils de demande, la structure de l'atelier et les délais de livraison des fournisseurs ont été très peu changés, car il est plus pertinent de créer un agent adapté à un type de problème spécifique. Néanmoins, ils ont été légèrement modifiés afin d'identifier des configurations qui pourraient être plus bénéfiques à l'apprentissage.

L'optimisation a été faite par essai-erreur, en essayant de trouver les limites de nos différents paramètres. La méthodologie d'optimisation consiste à évaluer les effets des paramètres sur la récompense accumulée moyenne définie à l'équation 4.6. Pour cela, les valeurs initiales ont été fixées en s'inspirant de la littérature du RL [19]. Les paramètres ont ensuite été modifiés un par un, afin de leur attribuer des valeurs extrêmes. Cela a permis d'évaluer l'impact des paramètres sur la simulation et en particulier la récompense accumulée moyenne. Ensuite, l'optimisation s'est précisée en s'approchant de valeurs jugées optimales, dans la mesure où elles assuraient un apprentissage satisfaisant. Nous précisons qu'une approche avec réplications a été adoptée dans la mesure où le modèle présente des éléments d'aléatoire. Il n'est pas possible de garantir l'optimalité de la qualité de l'apprentissage, néanmoins l'apprentissage a été jugé comme suffisant avec les paramètres définis après l'optimisation.

Dans le modèle, la RAM est utilisée de façon mobile et non globale, en utilisant la moyenne sur les 30 derniers épisodes. Ce choix a été fait afin de mettre en avant la croissance de la fonction et de ne pas obtenir une fonction trop lisse. Cet indicateur a l'avantage de représenter de façon très simple la performance de l'algorithme, nous nous attendons par exemple à obtenir une RAM croissante en fonction des épisodes. La Figure 4.9 représente un exemple de comparaison entre deux modèles (un sans agent et un avec) à l'aide de la récompense accumulée moyenne d'une phase d'évaluation. L'étape d'évaluation est réalisée à politique fixe. Cette politique a été améliorée pendant la phase d'entraînement. La fonction RAM du modèle sans agent est calculable dans la mesure où les fonctions récompense utilisent seulement des niveaux de buffer et des taux de service. Il suffit en fait de ne considérer que l'étape d'observation de l'espace d'états et d'attribution de la fonction récompense présents dans un cycle d'apprentissage par renforcement. Les autres étapes ne sont pas considérés car elles amènent à une modification des OST et des OSH, ce qui n'est pas souhaité pour un modèle sans agent. La fonction RAM du modèle sans agent est sans surprise, quasi-constante. Tandis que la fonction RAM du modèle avec agent est bien croissante en fonction des épisodes.

A travers les différents tests effectués, voici les observations ayant pu être effectuées sur les paramètres inhérents à l'apprentissage par renforcement :

— une trop grande variation de l'OSH n'est pas pertinente, nous nous intéressons donc à un OSH majoré par $1.5 * DLT$ dans l'espace d'actions.
— un épisode doit couvrir une demande d'environ 1000 jours. Il ne faut pas une durée trop

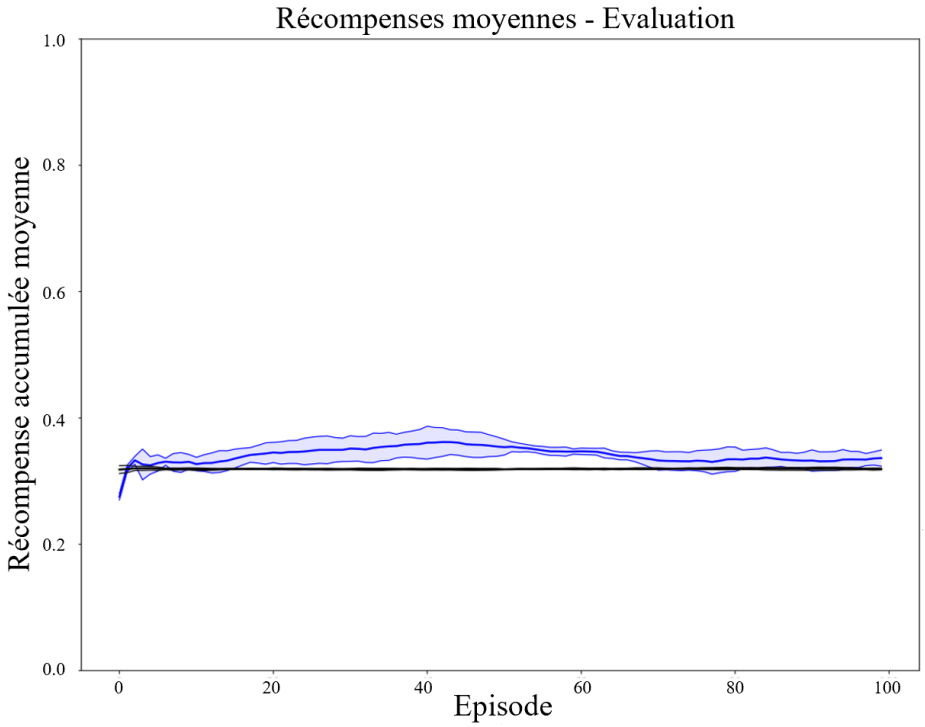


FIGURE 4.9 Récompenses accumulées moyennes d'une phase d'évaluation pour un modèle avec agent (courbe bleue) et un modèle sans agent (courbe noire)

courte afin de permettre à l'agent d'apprendre, mais il faut aussi limiter la complexité et ne pas avoir des épisodes trop longs. Nous travaillons avec des épisodes de 1000 jours.

— un espace d'état à plusieurs états est plus performant qu'un espace d'état à un seul état.

— une fréquence d'interaction supérieure à 10 jours ouvrés est conseillée, afin que celle-ci soit supérieure à l'ordre de grandeur des OSH (autour de 10 jours ouvrés). Nous travaillons avec une fréquence d'interaction de 25 jours.

— une fréquence de mise à jour du réseau de neurones de l'ordre de grandeur de 500 jours ouvrés est conseillée. Cela correspond à une mise à jour du réseau de neurones toutes les 20 interactions, et à deux mises à jour du réseau par épisode.

— un compromis doit être réalisé entre la taille du réseau de neurones et la fréquence d'interaction.

— la variation du facteur γ a un impact négligeable sur la simulation.

4.6 Conclusion

Cette section présente les différentes étapes de la réalisation du modèle, en passant par les choix effectués, la création de l'agent (OS1) et l'intégration de l'agent au modèle (OS2). Après avoir défini des indicateurs clés de performance, nous avons pu valider le modèle et même l'améliorer. Il est maintenant possible de passer à l'expérimentation.

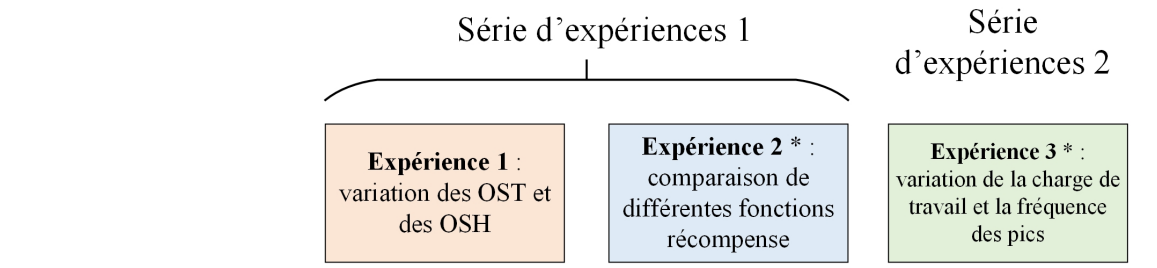
CHAPITRE 5 EXPÉRIMENTATION

Ce chapitre expose la phase d'expérimentation. Après avoir décrit le plan d'expériences, nous présentons les résultats de trois expériences. Ces résultats sont accompagnés d'une discussion générale et de recommandations.

5.1 Plan d'expériences

Le plan d'expériences vise à explorer deux aspects dans le contexte de notre recherche : les limites de validité pour l'utilisation de l'apprentissage par renforcement et sa capacité à faire face à des variations de la demande. A chaque simulation avec agent sera associée une simulation sans agent appelée **Baseline**, c'est une simulation où les OST et les OSH sont fixes. Elle s'oppose à une simulation avec agent où les OST et OSH sont pilotés. La Baseline nous sert à évaluer la performance de l'algorithme de RL. Nous n'utilisons la Baseline que dans la deuxième et la troisième expérience.

Dans toutes les expériences, les OST initiaux sont calculés avec l'une des trois méthodes suggérées par Ptak et Smith [1] : 50% du TOR (notée 50%), base de la zone rouge (notée RS) ou avec l'ADU (notée ADU). De même, les OSH initiaux sont calculés avec trois formules lorsqu'ils sont variables : $1.0 \times DLT$, $1.25 \times DLT$ ou $1.5 \times DLT$. Le plan d'expériences est résumé à la Figure 5.1. Il présente en tout 63 scénarios, nous numérotons donc ces scénarios de 1 à 63. Chaque scénario a été répliqué 3 fois afin de limiter les effets de l'aléatoire.



OST Variable	**Oui / Non**	Oui	Oui
OSH Variable	**Oui / Non**	Non	Non
Fonction récompense	R2	**R1 / R2 / R3**	R2
Valeur initiale de OST	**50% / RS / ADU**	**50% / RS / ADU**	**50% / RS / ADU**
Valeur initiale de OSH	**1.0 / 1.25 / 1.5 * DLT**	1.0 * DLT	1.0 * DLT
Fréquence des pics	20	10	**5 /10 / 20**
Charge de travail	80%	80%	**50% / 80% / 95%**
Nombre de scénarios	27	9	27

* : Expérience où une comparaison avec la Baseline est faite

FIGURE 5.1 Plan d'expériences

5.1.1 Comparaison de différentes façons de paramétrer le système

La première série d'expériences aborde les paramètres propres à l'apprentissage par renforcement : l'espace d'actions et les fonctions récompenses. Ce sont les paramètres qui nous permettent d'assurer un bon apprentissage. Ce sont également les paramètres qui nous permettent d'obtenir un agent qui prend de meilleures décisions. Cette première série d'expériences s'inscrit dans une logique d'optimisation des paramètres de l'apprentissage par renforcement, cette notion a déjà été mise en avant dans la partie **4.5.3**. L'objectif général de ces expériences est de créer un agent consistant et optimal dans un contexte de gestion de production, afin de fixer des paramètres optimaux et pouvoir étudier l'effet de facteurs externes sur la simulation. Cet objectif général est atteint en étudiant l'impact de l'espace d'actions (équations 4.2 et 4.3) et la fonction récompense (Tableau 4.2) sur la simulation.

Cette série d'expériences tend également à mettre en avant l'étape d'optimisation des paramètres de RL. Naturellement, les performances de l'agent sont sujettes de l'environnement externe. Les expériences mettent en avant ce phénomène en démontrant les potentielles conséquences d'un agent mal paramétré et potentiellement d'un problème mal défini. Nous insistons sur l'optimisation de ces paramètres car elle constitue un résultat majeur de ce travail. Les expériences viennent brièvement évaluer la sensibilité du modèle d'apprentissage aux

paramètres de RL, mais n'aboutissent pas à proposer une méthodologie de paramétrisation.

L'expérience 1, présentée à la section **5.2**, étudie les effets de la variation des OST et des OSH. Elle comprend trois types de configurations : pour 9 scénarios où les OST et les OSH sont variables, pour 9 scénarios où les OST sont variables et les OSH sont fixes, enfin pour 9 scénarios où les OST sont fixes et les OSH sont variables. L'objectif de cette expérience est de vérifier s'il est pertinent d'ajuster les OST et les OSH du modèle DDMRP avec apprentissage par renforcement. L'idée de départ est de faire varier ces deux paramètres, néanmoins il se peut que l'un des deux ne soit pas adapté à l'utilisation d'un agent. En isolant les deux paramètres, nous pouvons montrer leur impact sur le système, et nous pouvons décider s'il est pertinent de les ajuster dynamiquement.

L'expérience 2, présentée à la section **5.3**, aborde la notion d'objectif dans l'apprentissage par renforcement. Rappelons que, pour tout algorithme d'apprentissage par renforcement, nous devons définir un objectif à atteindre ou maximiser. La fonction récompense est définie en fonction de cet objectif. Nous avons déjà défini trois fonctions récompense dans la section **4.4.2**, cette expérience consiste à comparer les résultats de ces fonctions récompense. Cette expérience permet notamment de privilégier une fonction récompense pour l'expérience 3.

5.1.2 Analyse de l'effet de facteurs externes (charge de travail et fréquence des pics de demande)

La deuxième série d'expériences nous permet d'étudier l'effet des paramètres d'entrée sur le système qui est une série temporelle de demande. Il paraît cohérent de s'interroger sur les effets de cette demande sur la simulation.

L'expérience 3, présentée à la section **5.4**, est orientée métier et analyse l'effet de facteurs externes soient la fréquence des pics de demande et la charge de travail. Nous allons modifier le profil de demande à travers la fréquence des pics et le ratio entre la charge et la capacité. Cette expérience nous permet d'évaluer les effets des paramètres d'entrée sur la simulation. Elle va également nous permettre de déterminer dans quel environnement l'apprentissage par renforcement est le plus efficace. Cette étape est réalisée dans l'optique d'une analyse de la sensibilité du modèle d'apprentissage aux conditions externes. Notons tout de même que, pour deux scénarios ayant la même charge de travail et la même fréquence de pics, les profils de demande sont les mêmes.

5.1.3 Paramétrage de l'expérimentation

Les expérimentations ont été menées sur le logiciel Visual Studio Code. Chaque scénario est répliqué 3 fois afin d'incorporer l'aspect aléatoire dans les temps d'opération et les temps de réglage des machines. Ce nombre de scénarios a été déterminé par essai-erreur et s'est montré suffisant pour comparer les scénarios simulés, c'est-à-dire obtenir des intervalles de confiance à 95% assez étroits. Si les intervalles de confiance d'un scénario se chevauchent après 3 réplications, nous simulons à nouveau ce scénario avec davantage de réplications, jusqu'à obtenir une distinction. Nous précisons également qu'il a été défini afin de considérer des contraintes de temps d'exécution. Notons que pour un même scénario, d'une réplication à une autre, les profils de demande sont les mêmes.

Une réplication est composée de 100 épisodes. Un épisode correspond à une simulation de 1000 jours. Nous travaillons sur 3 produits passant par des machines distinctes avec des temps d'opération et de réglage différents. L'espace d'état considéré correspond à la demande moyenne mobile des 30 derniers jours et les niveaux d'EFN de chaque produit.

Pour les 9 premiers scénarios de l'expérience 1, le réseau de neurones est composé de 64 couches cachées communes, 32 couches cachées d'actions et 32 couches cachées de valeur (voir la Figure 4.3). Pour le reste des scénarios, le réseau de neurones est composé de 16 couches communes, 16 couches d'actions et 16 couches de valeur. La taille du réseau de neurones change pour s'adapter à des espaces d'action de tailles différentes. Ces paramètres ont été fixés et choisis suite à plusieurs essais-erreurs et plusieurs simulations pendant la phase d'optimisation, à travers différentes combinaisons de couches cachées. Ils permettent de trouver des compromis, tel celui de la taille du réseau de neurones et la fréquence des interactions. Une structure uniquement avec des couches communes a été considérée mais a présenté de moins bon résultats qu'avec des couches d'actions et des couches de valeur.

Les résultats sont présentés sous forme de tableau, comportant les récompenses accumulées moyennes (RAM définies à l'équation 4.6) par épisode et d'autres KPI tels que le taux d'apprentissage (équation 4.8) et le taux de service.

5.2 Expérience 1 : variation des OST et des OSH

L'expérience 1 analyse l'effet de la variation des OST et OSH sur la performance du modèle DDMRP. Comme première étape de l'expérimentation, nous tentons de valider l'hypothèse d'un apprentissage possible avec les OST et des OSH. Il ne s'agit pas de montrer qu'on peut mieux faire que la Baseline, mais de s'assurer qu'on arrive à apprendre, c'est-à-dire qu'on obtient une fonction RAM croissante. Pour cette étape, nous considérons juste la récompense

R_2 (maximiser le niveau de service), avec une fréquence de pics tous les 20 jours et une charge de travail de 80%. Les résultats sommaires sont présentés sous la forme de tableaux et de graphiques, tandis que les résultats détaillés sont fournis en **Annexe E**.

La Figure 5.2 représente la fonction RAM par épisode des scénarios à OST et OSH variables. Nous observons des fonctions RAM croissantes pour tous les scénarios.

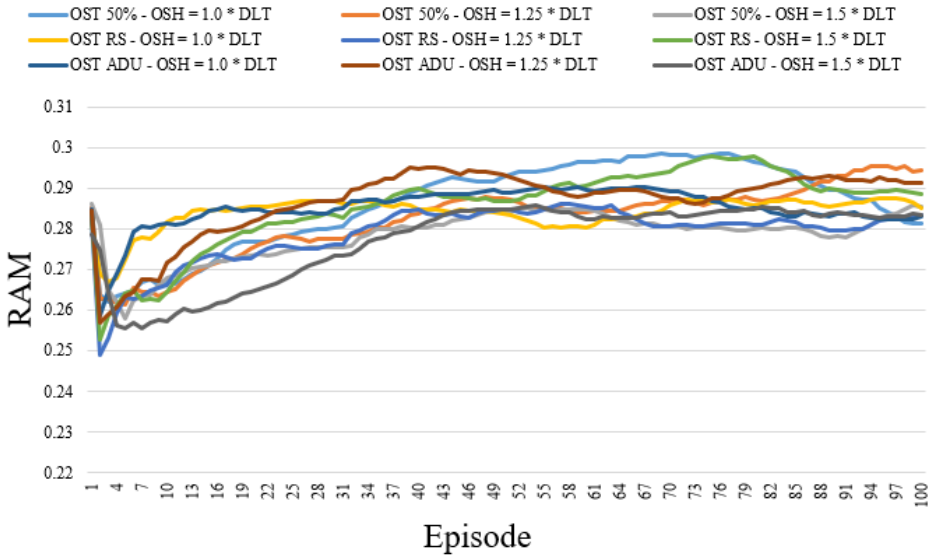


FIGURE 5.2 Évolution des récompenses accumulées moyennes par épisode pour les scénarios à OST et OSH variables

La Figure 5.3 représente la fonction RAM par épisode des scénarios à OST variables et OSH fixes. Nous observons des fonctions RAM croissantes pour tous les scénarios.

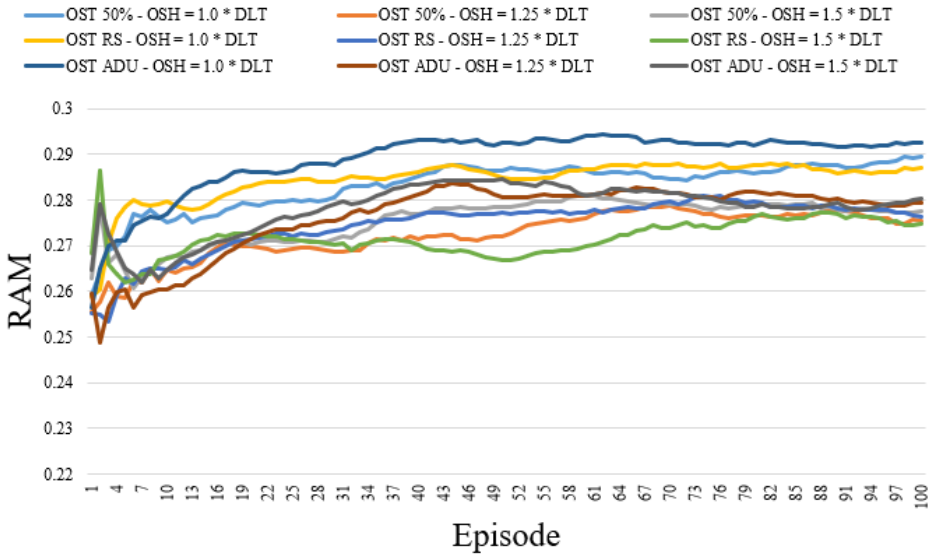


FIGURE 5.3 Évolution des récompenses accumulées moyennes par épisode pour les scénarios à OST variables et OSH fixes

Enfin, la Figure 5.4 représente la fonction RAM par épisode des scénarios à OST fixes et OSH variables. Les courbes des fonctions RAM sont quasiment constantes pour ces scénarios.

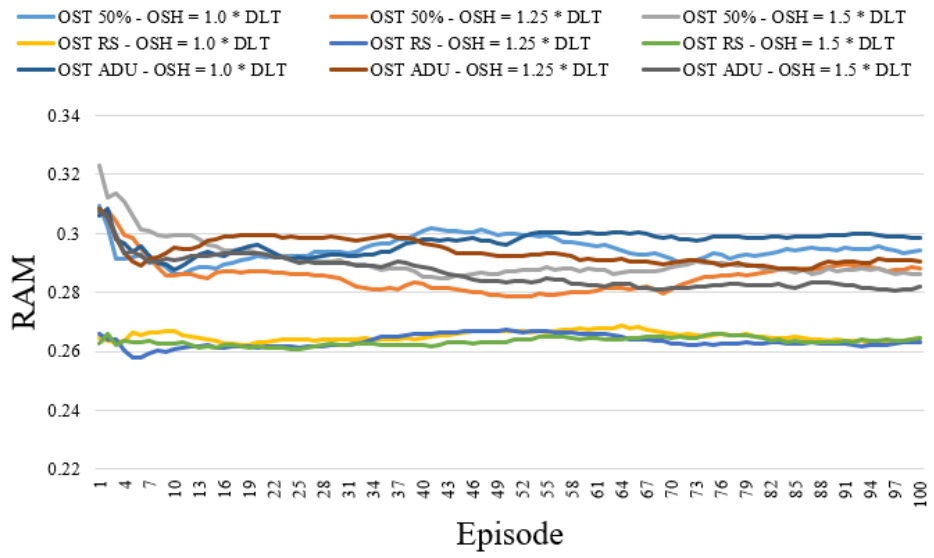


FIGURE 5.4 Évolution des récompenses accumulées moyennes par épisode pour les scénarios à OST fixes et OSH variables

Notons que pour cette série d'expériences, nous ne tentons pas de comparer à la Baseline. Ainsi, pour la Figure 5.2, la Figure 5.3 et la Figure 5.4, nous n'avons pas affiché les intervalles de confiance et les courbes des Baselines. Les résultats montrent que, pour des cas d'OST et OSH variables ou même OST variables et OSH fixes, l'apprentissage par renforcement est fonctionnel, car la fonction RAM est croissante. Néanmoins, quand les OST sont fixes et les OSH variables, l'apprentissage n'est pas prononcé. On peut conclure que la variation des OSH seulement n'a pas d'impact sur le modèle d'apprentissage par renforcement et ne permet pas l'apprentissage. **Il est possible de faire de l'apprentissage par renforcement avec les OST**.

La Figure 5.2 et la Figure 5.3 ont des profils très similaires. La raison est la suivante : s'il est possible d'apprendre avec des OST variables et OSH fixes, alors il est possible d'apprendre avec des OST variables et des OSH variables. Pour les scénarios d'OST et OSH variables, l'agent apprend avec le pilotage des OST, mais est pénalisé par le pilotage des OSH. Dans ce cas, nous avons un espace à 16 actions, puisque $\alpha_{OST} \in \{0.25; 0.5; 0.75; 1.0\}$ (défini à 4.2) et $\alpha_{OSH} \in \{0.0; 0.15; 0.35; 0.5\}$ (défini à 4.3), ce qui donne 16 couples de $(\alpha_{OST}, \alpha_{OSH})$. Nous constatons que l'effet de la variation des OSH est négligeable vu que les actions visant à varier les OSH ont des effets similaires sur le système (voir Figure 5.4), ce qui nuit à l'apprentissage. Ce résultat est légèrement visible à l'aide des temps de convergence. Nous remarquons que les scénarios avec OSH fixes convergent plus rapidement (grossièrement entre 30 et 40 épisodes) que les scénarios avec OST et OSH variables (au moins 50 épisodes).

Le phénomène d'absence d'apprentissage avec les OSH peut s'expliquer de plusieurs façons. Premièrement, les variations d'OSH ont sûrement trop peu d'impact sur le système. Ils ont donc un effet négligeable sur la fonction récompense, les espaces d'actions, etc. Cela implique qu'il n'est sûrement pas nécessaire de les faire varier et qu'ils ne peuvent pas améliorer l'apprentissage par renforcement. Dans ce cas, il serait mieux d'utiliser un OSH égal à $1.0 \times DLT$ tel que proposé par Ptak et Smith [1].

Deuxièmement, les OSH ont certainement besoin d'une fréquence d'interaction et de mise à jour du réseau de neurones différentes de celles utilisées pour les OST. Nous pourrions alors considérer d'introduire dans le système deux réseaux de neurones différents, avec deux agents différents : un agissant sur les OST et un autre sur les OSH. Cela nécessiterait d'étudier en profondeur un agent dédié aux OSH et trouver une fréquence d'interaction qui lui serait adapté si cela est possible.

Enfin, l'apprentissage par renforcement est davantage fonctionnel avec des espaces d'actions réduits. Cela permet d'avoir une meilleure exploration et de ne pas délaisser d'action. Un espace à 4 actions a plus de chances de réussir qu'un espace à 16 actions, car l'espace à 16

actions aura des chances de délaisser certaines actions. Il faut également considérer que nous réalisons environ 50 interactions par épisode et donc seulement 50 actions par épisode. Un faible nombre d'interactions comme celui-ci n'est pas cohérent avec un espace à 16 actions. Dans un environnement aussi complexe que celui du DDMRP, nous souhaitons favoriser un espace d'actions simple et de petite taille. Cela permet également de limiter la taille du réseau de neurones et donc de perdre en complexité. Un espace d'actions à 4 actions est donc sûrement préférable à un espace à 16 actions. Pour envisager un espace d'actions plus grand, nous suggérons d'augmenter la longueur de la série de demande, mais cela augmente considérablement la complexité.

Pour conclure, cette première expérience permet de valider que l'ajustement des OST via l'apprentissage par renforcement améliore la performance du système DDMRP, mais pas l'ajustement des OSH. Pour le reste de l'étude, nous faisons varier uniquement les OST tout en considérant des OSH fixes et égaux à $1.0 \times DLT$.

5.3 Expérience 2 : comparaison de différentes fonctions récompense

La deuxième expérience porte sur la fonction récompense qui est un point central de l'algorithme d'apprentissage par renforcement, dans la mesure où nous voulons maximiser les retours de cette fonction récompense. L'optimisation des stocks et la maximisation de la satisfaction client sont deux finalités du DDMRP, il paraît donc cohérent de les considérer comme des objectifs de notre étude. Dans la section **4.4.2**, nous nous sommes fixés trois objectifs différents :

- Optimiser les stocks (R_1), mesurés par les valeurs des EFN ;
- Maximiser la satisfaction client (R_2), mesurés par les taux de service locaux ;
- Optimiser les stocks et maximiser la satisfaction client (R_3) en conciliant les deux fonctions précédentes.

Tout comme l'expérience 1, nous utilisons principalement des indicateurs propres à l'apprentissage par renforcement afin de choisir la fonction récompense permettant le meilleur apprentissage. De plus, nous comparons les résultats des scénarios à la Baseline, en utilisant la récompense accumulée moyenne (RAM) du dernier épisode et celle de la Baseline.

Dans le Tableau 5.1, nous affichons les récompenses accumulées moyennes respectivement après épisode 1, épisode 25, épisode 50, épisode 75 et épisode 100, et ceci pour les différents scénarios. Nous affichons également celles obtenues avec la Baseline.

Tableau 5.1 Récompenses accumulées moyennes par épisode associé à l'expérience 2

Scénario	Fonction récompense	Initialisation OST	Épisode 1	Épisode 25	Épisode 50	Épisode 75	Épisode 100	Baseline
28	R1	50%	0.310 ± 0.001	0.298 ± 0.014	0.308 ± 0.015	0.302 ± 0.017	0.303 ± 0.011	0.305 ± 0.001
29	R1	RS	0.315 ± 0.006	0.296 ± 0.006	0.302 ± 0.019	0.297 ± 0.010	0.298 ± 0.009	0.278 ± 0.001
30	R1	ADU	0.314 ± 0.006	0.295 ± 0.012	0.303 ± 0.007	0.304 ± 0.006	0.304 ± 0.011	0.290 ± 0.001
31	R2	50%	0.259 ± .003	0.280 ± 0.005	0.286 ± 0.007	0.286 ± 0.006	0.290 ± 0.007	0.303 ± 0.001
32	R2	RS	0.259 ± 0.002	0.284 ± 0.011	0.285 ± 0.014	0.287 ± 0.011	0.287 ± 0.004	0.273 ± 0.001
33	R2	ADU	0.257 ± 0.003	0.286 ± 0.010	0.293 ± 0.005	0.292 ± 0.006	0.293 ± 0.009	0.295 ± 0.002
34	R3	50%	0.286 ± 0.001	0.283 ± 0.013	0.294 ± 0.014	0.292 ± 0.008	0.289 ± 0.006	0.292 ± 0.001
35	R3	RS	0.291 ± 0.005	0.279 ± 0.003	0.286 ± 0.017	0.286 ± 0.010	0.289 ± 0.008	0.271 ± 0.001
36	R3	ADU	0.290 ± 0.002	0.289 ± 0.007	0.299 ± 0.006	0.291 ± 0.009	0.290 ± 0.007	0.285 ± 0.001

La Figure 5.5 représente les courbes des récompenses accumulées moyennes par épisode pour les scénarios des fonctions récompenses R_1 et R_2. Les courbes des Baselines ne sont pas affichées afin d'améliorer la lisibilité. De même, nous n'avons pas affiché les courbes de R_3 afin de mettre en évidence la différence entre R_1 et R_2. En effet, la figure montre des profils d'apprentissage différents pour R_1 et R_2. L'apprentissage de R_1 a tendance à démarrer à une valeur de RAM élevée, plus élevée que celle de la Baseline, tandis que l'apprentissage de R_2 présente une courbe croissante du début à la fin. L'agent R_3 reprend naturellement les caractéristiques des deux fonctions récompense. Nous mettons ces observations en avant avec le Tableau 5.2. Le pourcentage de meilleure récompense accumulée (PMRA) permet d'identifier le nombre d'épisodes nécessaires pour atteindre la meilleure récompense accumulée (MRA). Pour R_1 et R_3, la MRA est atteinte rapidement (dès le premier épisode pour R_1, quasiment avant le cinquantième épisode pour R_3), ce qui n'est pas le cas pour R_2, qui met au moins 50 voire 75 épisodes pour atteindre sa MRA. La MRA atteinte assez tôt pour R_1 et R_3 dénote un problème dans la croissance de la fonction RAM, puisqu'on s'attend plutôt à obtenir cette MRA vers la fin de l'apprentissage [30]. Cela démontre que R_1 et R_3 ne sont pas sujettes à un apprentissage satisfaisant.

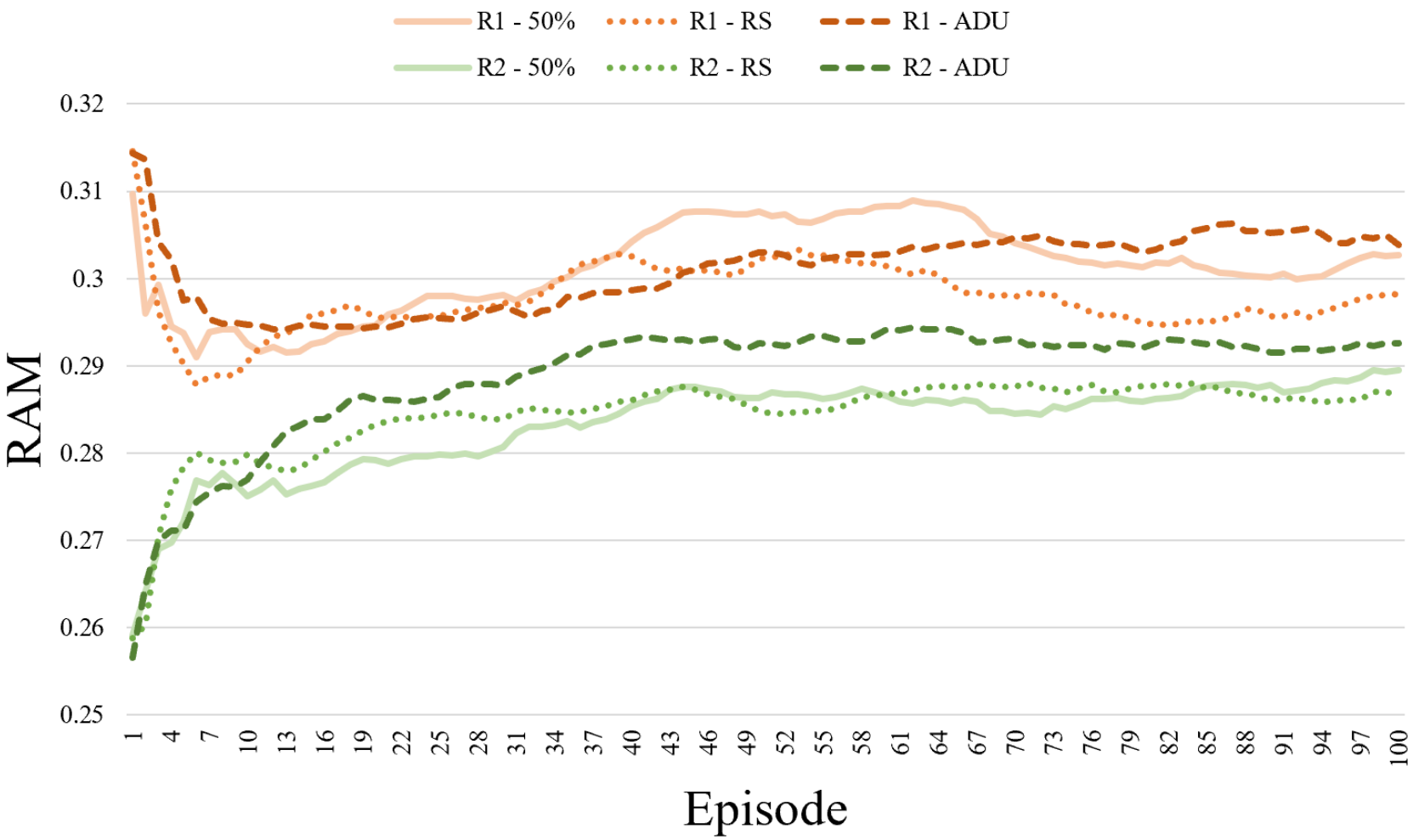


FIGURE 5.5 Évolution des récompenses accumulées moyennes (RAM)par épisode pour différentes fonctions récompenses

Bien que les valeurs de RAM de R_1 semblent plus élevées que R_2 et R_3, cela ne signifie pas que R_1 est meilleur. En effet, il n'est pas pertinent de comparer les valeurs numériques des RAM de deux fonctions récompenses différentes, il faut plutôt se référer à la Baseline. En effectuant cette comparaison à l'aide du Tableau 5.1, nous remarquons que R_1 a tendance à donner de meilleurs résultats que la Baseline, ce qui n'est pas toujours le cas pour R_2. Nous nuançons ce résultat dans la mesure où la fonction R_1 a tendance à donner un départ avantageux à l'agent d'apprentissage. La Figure 5.5 illustre bien ce phénomène de départ avantageux pour R_1. Dès le premier épisode, la valeur de la récompense accumulée moyenne est plus élevée que les récompenses des autres épisodes. De même, dès le premier épisode, les récompenses accumulées moyennes sont plus élevées que la Baseline (ceci est observable avec le Tableau 5.1). Le début de l'apprentissage de R_1 n'est pas nécessairement une anomalie : au début de l'apprentissage, les poids du réseau de neurones sont aléatoires, ce qui peut créer dans certains cas un départ avantageux pour la fonction R_1. Cet avantage s'estompe par la suite, et l'apprentissage redescend rapidement en reprenant rapidement sa croissance.

Tableau 5.2 Pourcentage de meilleure récompense accumulée (PMRA) par épisode associé à l'expérience 2

Scénario	Fonction récompense	Initialisation OST	Épisode 1	Épisode 25	Épisode 50	Épisode 75	Épisode 100
28	R1	50%	1.0	1.0	1.0	1.0	1.0
29	R1	RS	1.0	1.0	1.0	1.0	1.0
30	R1	ADU	1.0	1.0	1.0	1.0	1.0
31	R2	50%	0.895	0.967	0.993	0.993	1.0
32	R2	RS	0.898	0.988	0.999	0.999	1.0
33	R2	ADU	0.871	0.973	0.996	1.0	1.0
34	R3	50%	0.974	0.974	0.999	1.0	1.0.
35	R3	RS	1.0	1.0	1.0	1.0	1.0
36	R3	ADU	0.962	0.962	1.0	1.0.	1.0

Le Tableau 5.3 rassemble les taux d'apprentissage (défini à l'équation 4.8) des trois différentes fonctions récompenses avec trois différentes façons pour initialiser les OST. Nous rappelons que le taux d'apprentissage défini à l'équation 4.8 mesure la différence entre la RAM au dernier épisode et sa valeur minimale sur l'ensemble des épisodes. Il permet d'évaluer l'efficience de l'apprentissage de l'agent. A proprement parler cet indicateur traduit simplement des observations graphiques en observations numériques. Cet indicateur n'indique pas forcément la performance d'un agent dans un environnement changeant.

Le désavantage de la fonction R_1 est son taux d'apprentissage trop peu prononcé (il ne dépasse pas les 0.012). Certes, R_1 permet d'obtenir de meilleurs résultats que la Baseline

Tableau 5.3 Taux d'apprentissage associé à l'expérience 2 par fonction récompense et par valeur initiale d'OST

	R1	R2	R3
50%	0.012	0.030	0.014
RS	0.010	0.028	0.022
ADU	0.010	0.036	0.021

dans les derniers épisodes, mais le faible taux d'apprentissage est problématique car cela dénote un environnement qui n'est pas propice à l'apprentissage. De plus, l'agent R_1 ne démontre pas une nette convergence. Cette faible performance de R_1 peut s'expliquer par sa fonction récompense, définie au Tableau 4.2. Ceci insinue que la variation des OST a peu d'effet sur les niveaux de stocks.

D'après le Tableau 5.3, l'agent R_2 fournit un taux d'apprentissage très satisfaisant (au plus bas, son taux d'apprentissage est à 0.028), en plus des valeurs satisfaisantes de PMRA de R_2, qui témoignent d'une croissance continue (voir le Tableau 5.1). C'est un résultat espéré pour un agent d'apprentissage par renforcement. R_2 fournit également un taux de convergence de 50 épisodes, qui est un taux jugé raisonnable [19]. Néanmoins, les résultats de R_2 par rapport à la Baseline semblent mitigés (on n'est pas systématiquement au-dessus de la Baseline à la fin de l'entraînement). Ce phénomène s'explique par des paramètres d'atelier trop peu propices à l'apprentissage par renforcement, nous reviendrons sur ce point avec la troisième expérience.

Finalement, la fonction R_3, étant une combinaison des deux fonctions récompenses, reprend les défauts de R_1. Dans le Tableau 5.3, le taux d'apprentissage de R_3 est davantage prononcé que celui de R_1, mais moins important que celui de R_2. Au mieux son taux d'apprentissage atteint 0.022, puisqu'il est pénalisé par R_1. R_3 met environ 70 épisodes à converger, qui un taux assez haut mais que nous ne jugeons pas aberrant [19]. L'agent R_3 fournit cependant des résultats satisfaisants par rapport à la valeur de la Baseline.

Les résultats présentés ci-dessus tendent à montrer que l'apprentissage par renforcement est davantage adapté à la satisfaction client qu'à l'optimisation des stocks. Nous illustrons ce propos grâce à la Figure 5.7, la Figure 5.6 et la Figure 5.8.

La Figure 5.6 représente les zones de buffer et les valeurs d'EFN du produit 1 pour le dernier épisode d'un scénario sans agent qui utilise R_1 pour fonction récompense. La Figure 5.7 représente les mêmes informations pour le dernier épisode du scénario avec agent. Enfin, la Figure 5.8 représente les taux de service pour un scénario avec et sans agent qui utilise R_2 comme fonction récompense.

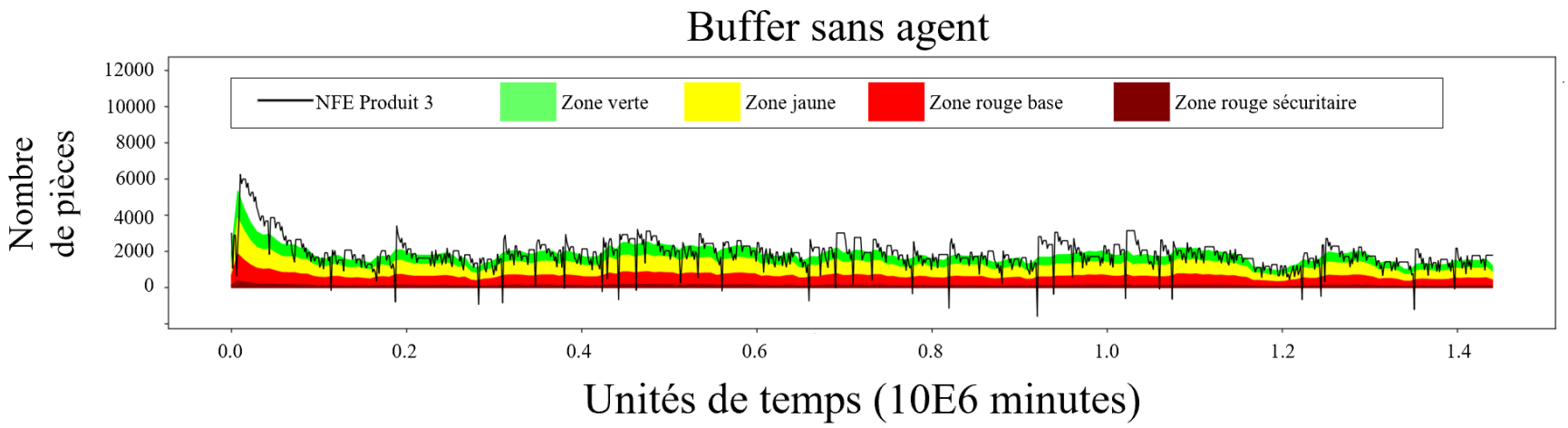


FIGURE 5.6 Niveaux de Buffer et EFN d'un scénario sans agent qui utilise R_1 pour fonction récompense

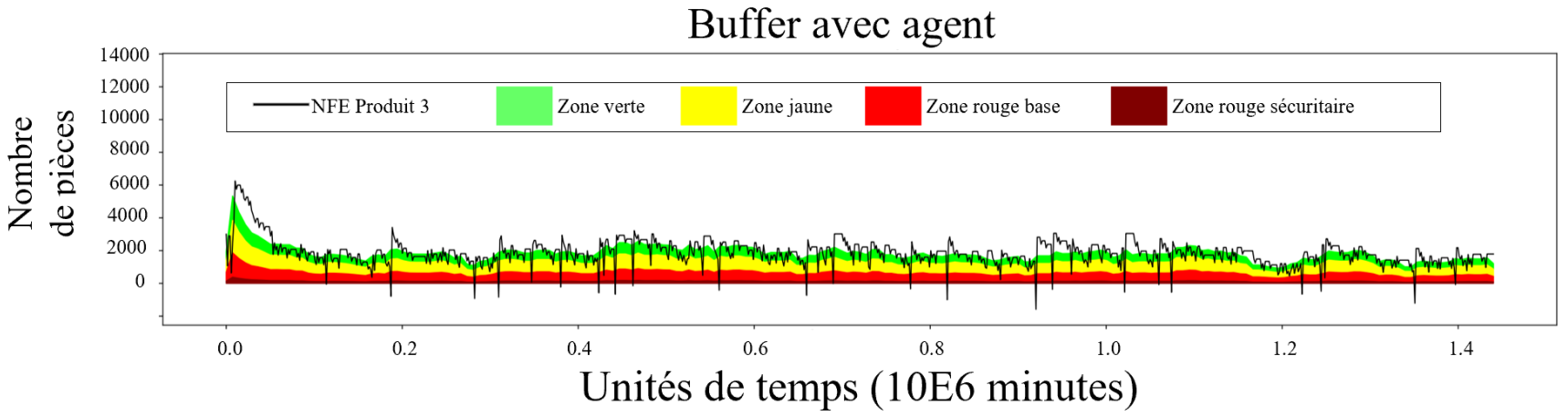


FIGURE 5.7 Niveaux de Buffer et EFN d'un scénario avec agent qui utilise R_1 pour fonction récompense

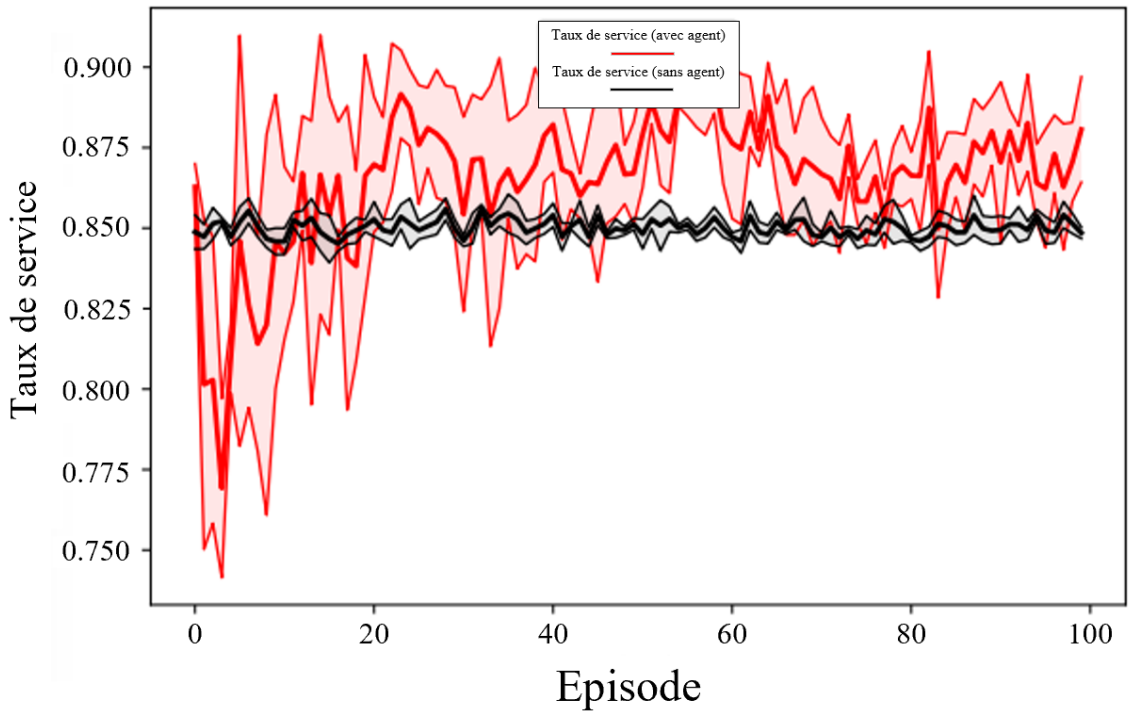


FIGURE 5.8 Taux de service par épisode d'un scénario avec et sans agent ayant R_2 pour fonction récompense

La Figure 5.6 et la Figure 5.7 ont le désavantage d'être peu différenciables. Les niveaux d'EFN et de zones du Buffer, présentés sous forme de séries temporelles de grande taille (1000 jours), ne montrent pas une différence significative entre les résultats avec ou sans agent. Par contre, nous observons dans la Figure 5.8 une nette différence entre les courbes avec et sans agent. Les deux séries temporelles, d'une longueur de 100 épisodes, sont visuellement différenciables grâce à leurs intervalles de confiance. Le phénomène d'apprentissage est nettement visible sur la courbe du taux de service avec agent, dans la mesure où elle est globalement croissante et vient dépasser la courbe du taux de service sans agent. La fonction R_2 a donc un effet observable visuellement sur la satisfaction client, ce qui confirme que la satisfaction client est adapté à l'utilisation de l'apprentissage par renforcement.

R_1 est un agent qui pourrait fonctionner mais demanderait sûrement beaucoup d'étude et beaucoup de reparamétrisation. Les valeurs de récompense ont été améliorées pendant le calibrage, mais elles requièrent davantage d'attention. R_3 quant à lui reprend en trop grande partie les défauts d'apprentissage de R_1. R_2 a l'avantage de fournir des résultats consistants et cohérents en termes d'apprentissage par renforcement, ce qui insinue une meilleure satisfaction des clients en ajustant les OST. En termes de gestion de production, cela signifie que l'apprentissage par renforcement permet d'améliorer la performance du système en termes

de satisfaction client, mais moins au niveau de l'optimisation des stocks. Nous choisissons de poursuivre l'étude en utilisant seulement R_2.

L'agent R_2 retenu est capable de nous donner des résultats consistants en termes d'apprentissage. Pour la suite, nous tentons d'ajuster dynamiquement les OST, tout en gardant les OSH fixes et égaux à $1.0 \times DLT$, et en utilisant R_2 comme fonction récompense. Nous précisons que ces décisions ont été prises suite à des observations empiriques, et qu'il n'est pas possible de justifier qu'ils sont optimaux. Quant à l'ordre choisi, nous indiquons qu'il a été détermine dans une optique de limiter le nombre d'expériences. En effet, le choix de limiter l'espace d'actions aux OST permet d'éliminer un grand nombre de scénarios associés aux OSH.

Nous avertissons également que la qualité de l'apprentissage est corrélé à l'environnement externe de l'atelier de production. Ainsi, dans un cas d'application réel, il s'agira de s'adapter et modifier les paramètres de RL en conséquence. Il ne paraît pas envisageable de proposer une méthode rigoureuse de paramétrisation, nous suggérons donc de procéder par essai-erreur en testant les limites des paramètres. Cette première série d'expériences met en avant cet aspect en montrant qu'un agent mal paramétré peut ne pas fournir un bon apprentissage. Ces notions viennent répondre à des enjeux de définition et identification du problème d'apprentissage par renforcement, qui doit pouvoir s'adapter à son environnement.

Nous pouvons maintenant challenger le modèle amélioré grâce au RL face à différents contextes, et nous intéresser à la comparaison des résultats de l'apprentissage par renforcement par rapport à ceux de la Baseline.

5.4 Expérience 3 : variation de la fréquence des pics et charge de travail

L'objectif de cette expérience est de vérifier si face à différents contextes, ajuster les OST est mieux que d'utiliser les méthodes de calcul de Ptak et Smith. En effet, nous considérons 3 fréquences des pics de demande et 3 niveaux de charge de travail. Si l'ajustement des OST n'est pas bénéfique, alors nous proposons d'identifier les contextes dans lesquels il vaut mieux éviter l'ajustement des OST La première comparaison se fait sur le niveau de la RAM. Nous comparons la valeur de la RAM du dernier épisode avec celui de la Baseline et traduisons ces résultats avec deux niveaux d'observation :

1. L'agent donne de meilleurs résultats que la Baseline et les intervalles de confiance ne se chevauchent pas (vert dans le Tableau 5.4) ;
2. L'agent donne de moins bons résultats que la Baseline et les intervalles de confiance ne se chevauchent pas (rouge dans le Tableau 5.4).

Les résultats de la performance de l'agent par rapport à la Baseline sont résumés dans le Tableau 5.4. Ce tableau met en perspective les effets de conditions environnementales (fréquence des pics et charge de travail) et ceux de paramètres de gestion (initialisation des OST). Les résultats détaillés sont présentés en **Annexe F**. Pour ces scénarios, nous avons veillé à ce que les intervalles de confiance pour l'épisode 100 et la Baseline ne se chevauchent pas (sinon un nombre de réplications plus haut serait requis). Cela nous permet d'exposer une observation binaire en ce qui concerne la performance de l'agent par rapport à la Baseline. Tout d'abord, nous remarquons que l'agent est la plupart du temps meilleur que la Baseline (22 scénarios sur 27). Si un agent performe moins bien que la Baseline, alors cet agent n'est pas adapté pour être utilisé dans ce cas. Par cas nous entendons une combinaison de nos trois degrés de liberté : calcul OST, fréquence des pics et charge de travail.

TABLEAU 5.4 Comparaison des récompenses accumulées moyennes de l'agent et de la Baseline associées à l'expérience 3

Fréquence des pics	Charge de travail	Meilleur que Baseline ? (OST 50%)	Meilleur que Baseline ? (OST RS)	Meilleur que Baseline ? (OST ADU)
5	50%	Oui	Oui	Oui
5	80%	Oui	Oui	Oui
5	95%	Oui	Oui	Oui
10	50%	Oui	Oui	Oui
10	80%	Non	Oui	Oui
10	95%	Oui	Oui	Oui
20	50%	Oui	Oui	Oui
20	80%	Non	Oui	Non
20	95%	Non	Non	Oui

Ces premiers constats révèlent qu'**il est possible de construire un agent permettant d'ajuster les OST et donner de meilleures performances que d'utiliser les formules de Ptak et Smith et ceci dans différents contextes**. Si les conditions changent, le choix d'utiliser l'agent reste le bon dans la majorité des cas, bien que touts les contextes ne nous ont pas montré un agent meilleur qu'une Baseline. Ce résultat n'est pas surprenant car nous travaillons avec un agent ayant des paramètres de RL égaux pour chaque scénario. Or, il se peut que chaque problème ait besoin d'une paramétrisation d'apprentissage bien spécifique. Nous avons observé ce résultat lors de la validation du modèle : chaque paramétrisation de l'agent donne un résultat unique et plus ou moins adapté à notre scénario.

Ainsi, bien que cette expérience ne présente pas de façon approfondie la sensibilité du modèle d'apprentissage aux conditions externes, nous précisions que ces facteurs ont bien un impact sur la qualité de l'apprentissage. Nous avons utilisé dans la troisième expérience un

agent unique et qui semble fonctionnel pour plusieurs scénarios. Néanmoins, il paraît très peu probable de créer un agent qui fonctionne pour absolument tous les contextes. Il a été construit et défini à l'aide des résultats de la première et la deuxième expérience, et a été jugé optimal pour notre étude, mais ses performances sont à nuancer, il est sûrement possible de trouver un meilleur agent pour chaque contexte présenté. Dans la pratique, nous suggérons donc d'identifier un profil de demande bien particulier, et lui affecter un agent adapté. La paramétrisation de l'agent est essentiel à son apprentissage, comme nous l'avons montré avec les deux premières expériences.

Nous souhaitons désormais identifier les cas qui sont davantage problématiques et qui peuvent nuire aux performances de l'agent. Tout comme nous pouvons observer sur le Tableau 5.4, les scénarios avec des fréquences de pics de 5 et 10 sont peu problématiques. Les agents dans ces scénarios sont tous meilleurs ou au moins égaux à la Baseline, à une exception près (scénario avec charge de travail à 80% et initialisation d'OST à 50%). Nous notons donc qu'un environnement présentant une faible fréquence de pics de demande est propice à l'utilisation d'un agent d'apprentissage par renforcement. Nous tentons d'expliquer ce phénomène avec la Figure 5.9 et la Figure 5.10 affichant le nombre de pics détectés pour deux scénarios qui se différencient par leur fréquence de pics (l'un est à 5 et l'autre à 20 jours).

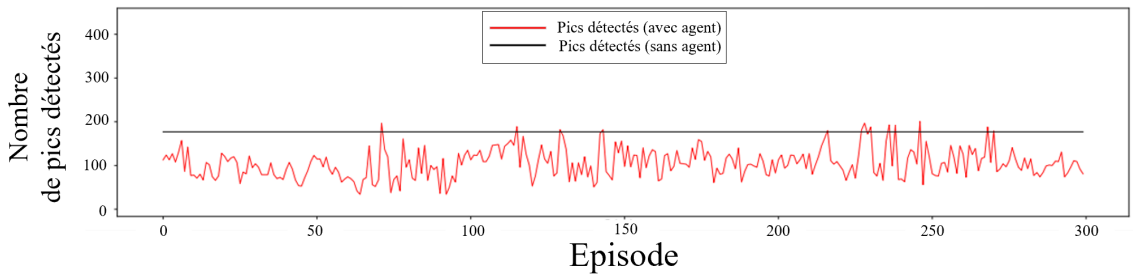


FIGURE 5.9 Nombre de pics détectés pour le dernier épisode du scénario à une fréquence de pics 5 jours

La différence entre les pics détectés par l'agent et la Baseline est radicalement différente pour les deux scénarios. Le scénario à une fréquence de 5 jours est assez proche du nombre de pics détectés par la Baseline, tandis que le scénario à une fréquence de 20 jours ne l'est pas du tout. Il s'agit d'une faiblesse de l'agent :dans un environnement à haute fréquence de pics, il ne détecte pas assez de pics.

Le Tableau 5.4 montre que l'apprentissage par renforcement a plus de difficultés avec des charges de travail élevées : tous les scénarios problématiques présentent une charge de travail à plus de 80%. Nous recommandons donc d'utiliser l'apprentissage par renforcement dans des ateliers ne dépassant pas 80% de charge de production si la fréquence de pics est élevée.

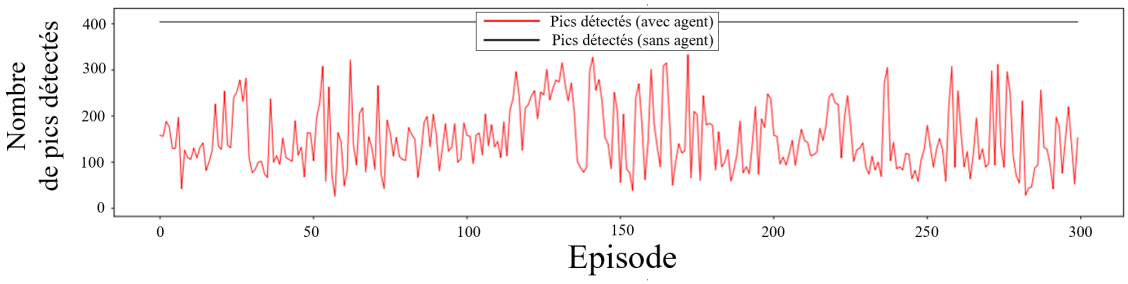


FIGURE 5.10 Nombre de pics détectés pour le dernier épisode du scénario à une fréquence de pics 20 jours

Enfin, le Tableau 5.4 met en évidence un problème d'ajustement avec un OST initialisé à 50% du *Top Of Red* (TOR). Sur 9 scénarios ayant un OST initialisé à 50% du TOR, 3 présentent un agent moins efficace que la Baseline. Ptak et Smith précisent que l'initialisation de l'OST à 50% du TOR est celle qui conserve le mieux le taux de service [1]. Par conséquent, un agent utilisant le taux de service comme référence de récompense a logiquement plus de difficultés à dépasser la Baseline dans le cas d'une initialisation de l'OST à 50% du TOR.

5.5 Conclusion

Les expériences menées nous ont apporté plusieurs résultats, le premier étant l'absence d'apprentissage par variation des OSH. La deuxième expérience a montré qu'il était possible d'apprendre avec différentes fonctions récompenses. Nous avons choisi de maximiser la satisfaction client avec la fonction R_2 car elle donnait les meilleures résultats d'apprentissage. La troisième expérience nous a orienté sur des enjeux métier. Nous avons développé une méthode de pilotage d'OST capable de mieux performer que les formules proposées par Ptak et Smith. En outre, nous avons pu identifier les contextes les moins propices à l'utilisation d'un agent d'apprentissage par renforcement.

CHAPITRE 6 CONCLUSION ET RECOMMANDATIONS

6.1 Synthèse des travaux

Ce projet répond à l'un des enjeux grandissant du DDMRP et encore trop peu développé dans sa littérature : le paramétrage. L'objectif est de proposer une méthode d'ajustement dynamique s'appliquant aux paramètres du DDMRP et permettant d'améliorer les performances de la méthode. Nous avons étudié les bienfaits d'ajuster, grâce à l'apprentissage par renforcement, les valeurs des OST et des OSH de différents produits dans un atelier de production géré en DDMRP.

Cette étude nous a apporté différents résultats. Tout d'abord, nous avons modélisé un agent capable d'apprendre dans un environnement géré en DDMRP (OS1). Cet agent, adapté d'un algorithme de *Dueling Q-learning* a été construit à l'aide d'un réseau de neurones, pour ensuite être intégré à la simulation à événements discrets développée par Martin [4] (OS2). Cuartas et al. [30] avaient démontré la possibilité d'utiliser l'apprentissage par renforcement en agissant sur les stocks. Nous avons démontré que l'apprentissage par renforcement peut intervenir sur les paramètres du DDMRP. L'ajustement des OST avec l'apprentissage par renforcement s'est montré bénéfique au système DDMRP, ce qui n'est pas le cas pour l'ajustement des OSH. Avec une politique d'OSH fixe, nous avons montré que l'apprentissage par renforcement est plus fonctionnel en utilisant la satisfaction client que l'optimisation des stocks. Enfin, l'agent a démontré de meilleures performances qu'une modélisation à OST et OSH fixes, et ce pour différents contextes de demande et production.

6.2 Limites de la solution proposée

La solution proposée présente des limites. L'ajustement des OST s'est montré plus bénéfique au système que l'ajustement des OSH, néanmoins cela ne signifie pas que l'ajustement des OSH est totalement inadapté à l'apprentissage par renforcement, car il peut nécessiter des paramètres de RL différents. Ensuite, utiliser une fonction récompense ignorant la réduction des niveaux de stocks nous fait prendre le risque d'augmenter les quantités de produits stockés. En outre, l'agent utilisé n'est certainement pas adapté à tous les problèmes et tous les ateliers gérés en DDMRP, en particulier en ce qui concerne les profils de demande. Enfin, l'approche expérimentale a le désavantage de présenter peu de configurations. Le projet ayant été limité par des contraintes de temps et d'exécution, il semble que certains aspects pertinents n'aient pas été abordés pour montrer que la solution recommandée est la bonne, par exemple

l'impact d'autres conditions environnementales telles que l'effet de saisonnalité ou la variation des temps d'interarrivée des pics de demande.

6.3 Perspectives de recherche

Le projet ne manque pas de perspectives d'améliorations. De nombreuses pistes n'ont pas pu être explorées pendant la réalisation de celui-ci.

La première serait d'affiner l'étude de l'ajustement des OSH. Cette perspective consiste à paramétrer un agent de façon à ce qu'il soit adapté à l'ajustement des OSH. Ainsi, on peut imaginer l'utilisation d'un deuxième agent, propre aux OSH, entraîné en parallèle du premier. La principale difficulté est de trouver une fréquence d'interaction de l'agent propice à un apprentissage consistant.

La deuxième perspective est l'amélioration de la fonction récompense. Dans la pratique, le troisième agent est sûrement celui qui est le plus adapté. Avec le DDMRP, nous souhaitons en effet à la fois optimiser les stocks et maximiser la satisfaction client. Une fonction récompense qui délaisse les stocks pourrait faire balancer ce compromis. Ce résultat n'a pas été clairement observable dans nos résultats, mais il faut y prêter attention. Afin d'améliorer R_3, nous suggérons d'abord de retravailler R_1. De plus, il serait possible de pondérer la moyenne entre R_1 et R_2, par exemple en donnant plus d'importance à R_2. Cela permettrait de conserver l'idée de compromis entre satisfaction client et optimisation des stocks.

Enfin, la dernière idée d'amélioration porte sur l'étude d'autres contextes. Le profil de demande étudié reste assez spécifique, et ne répond peut-être pas à tous les enjeux de l'industrie. Il s'agirait de distinguer des profils de demande et pourquoi pas des industries qui sont plus adaptés à l'apprentissage par renforcement que d'autres. Cette étude peut s'accompagner d'une modélisation de différentes agents associés à différents contextes.

RÉFÉRENCES

[1] C. Ptak et C. Smith, *Demand Driven Material Requirements Planning (DDMRP)*. South Norwalk, Connecticut, USA : Industrial Press, Inc., 2016.

[2] M. Pillet *et al.*, *Gestion de production : les fondamentaux et bonnes pratiques*, 5e éd. Paris, France : Editions d'Organisation, Groupe Eyrolles, 2011.

[3] A. Azzamouri *et al.*, "Demand driven material requirements planning (DDMRP) : A systematic review and classification," *Journal of Industrial Engineering and Management*, vol. 14(3), p. 439–456, 2021. [En ligne]. Disponible : http://www.jiem.org/index.php/jiem/article/view/3331

[4] G. Martin, "Contrôle dynamique du Demand Driven Sales and Operations Planning," thèse de doctorat, Université de Toulouse, Toulouse, France, 2020. [En ligne]. Disponible : http://www.theses.fr/2020EMAC0010

[5] Trencadis, "Webinaire DDMRP," 3 juin 2020. [En ligne]. Disponible : https://www.youtube.com/watch?v=NJizs_e0Jso

[6] Blue Lean Consulting. (2016) L'effet "coup de fouet". [En ligne]. Disponible : https://www.bluelean.fr/blog/production/l-effet-coup-de-fouet.html

[7] S. Ruel et K. Evrard Samuel, "La gestion de l'incertitude dans une chaîne logistique globale : Une étude de cas dans le secteur informatique," *Management et Avenir*, vol. 1, n°. 51, p. 78–98, janv. 2012. [En ligne]. Disponible : https://www.cairn.info/revue-management-et-avenir-2012-1-page-78.htm?ref=doi

[8] G. Dessevre et M. Ben Ali, "Modélisation et simulation d'un module d'ajustement de la capacité d'un système DDMRP," communication présentée à 13ème Conférence Francophone de Modélisation, Optimisation et Simulation- MOSIM'20, Agadir, Maroc, 12-14 Novembre 2020. [En ligne]. Disponible : https://hal.archives-ouvertes.fr/hal-03178098/

[9] M. J. Shofa et W. O. Widyarto, "Effective production control in an automotive industry : MRP vs. demand-driven MRP," dans *AIP Conference Proceedings*, 2017. [En ligne]. Disponible : https://aip.scitation.org/doi/abs/10.1063/1.4985449

[10] M. Ihme et R. Stratton, "Evaluating Demand Driven MRP : a case based simulated study," communication présentée à International Conference of the European Operations Management Association, Neuchatel, Switzerland, Juin 2015. [En ligne]. Disponible : http://irep.ntu.ac.uk/id/eprint/26668

[11] B. Bahu, L. Bironneau et V. Hovelaque, “Compréhension du DDMRP et de son adoption : premiers éléments empiriques,” *Logistique et Management*, vol. 27(1), p. 20–32, 2019.

[12] R. Miclo *et al.*, “MRP vs. demand-driven MRP : Towards an objective comparison,” communication présentée à 2015 International Conference on Industrial Engineering and Systems Management (IESM), Seville, Spain, 21-23 Octobre 2015. [En ligne]. Disponible : https://ieeexplore.ieee.org/document/7380288

[13] A. Kortabaria *et al.*, “Material managament without forecasting : From MRP to demand driven MRP,” *Journal of Industrial Engineering and Management*, vol. 11(4), p. 632–650, 2018. [En ligne]. Disponible : http://www.jiem.org/index.php/jiem/article/view/2654

[14] J. Jiang et S.-C. Rim, “Strategic WIP inventory positioning for make-to-order production with stochastic processing times,” *Mathematical Problems in Engineering*, 2017. [En ligne]. Disponible : https://www.hindawi.com/journals/mpe/2017/8635979/

[15] A. Orue, A. Lizarralde et A. Kortabarria, “Demand driven MRP - the need to standardise an implementation process,” *International Journal of Production Management and Engineering*, vol. 8(2), p. 65–73, 2020. [En ligne]. Disponible : https://polipapers.upv.es/index.php/IJPME/article/view/12737

[16] G. Martin *et al.*, “Vers une cartographie de processus explicite pour le modèle Demand Driven Adaptive Enterprise,” communication présentée à 12ème Conférence internationale de Modélisation, Optimisation et SIMulation - MOSIM’18, Toulouse, France, Juin 2018. [En ligne]. Disponible : https://hal.archives-ouvertes.fr/hal-01852605

[17] G. Dessevre *et al.*, “Decoupled Lead Time in finite capacity flowshop : a feedback loop approach,” communication présentée à 2019 International Conference on Industrial Engineering and Systems Management (IESM), Shangai, China, 25-27 Septembre 2019. [En ligne]. Disponible : https://ieeexplore.ieee.org/document/8948198

[18] D. Damand, Y. Lahrichi et M. Barth, “Parametrisation of demand-driven material requirements planning : a multi-objective genetic algorithm,” *International Journal of Production Research*, 2022.

[19] M. Morales, *Grokking Deep Reinforcement Learning.* Shelter Island, NY, USA : Manning Publications Co., 2020.

[20] A. S. Xanthopolous *et al.*, “Reinforcement learning-based and parametric production-maintenance control policies for a deteriorating manufacturing system,” *IEEE Access*, vol. 6, p. 576–588, févr. 2018. [En ligne]. Disponible : https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8114172

[21] T. Stockheim, M. Schwind et W. Koenig, “A reinforcement learning approach for supply chain management,” communication présentée à 1st European Workshop on Multi-Agent Systems (EUMAS), Oxford, UK, Déc. 2003. [En ligne]. Disponible : https://www.researchgate.net/publication/228523960_A_reinforcement_learning_approach_for_supply_chain_management

[22] H. Cao, H. Xi et S. F. Smith, “A reinforcement learning approach to production planning in the fabrication/fulfillment manufacturing process,” communication présentée à Proceedings of the 2003 Winter Simulation Conference, New Orleans, LA, USA, 7-10 Déc. 2003, p. 1417–1423. [En ligne]. Disponible : https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1261584

[23] J. Wang *et al.*, “Real-time decision support with reinforcement learning for dynamic flowshop scheduling,” communication présentée à Smart SysTech 2017 ; European Conference on Smart Objects, Systems and Technologies, Munich, Germany, 20-21 Juin 2017. [En ligne]. Disponible : https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8084561

[24] Y.-T. Tsai *et al.*, “Utilization of a reinforcement learning algorithm for the accurate alignment of a robotic arm in a complete soft fabric shoe tongues automation process,” *Journal of Manufacturing Systems*, vol. 56, p. 501–513, juill. 2020. [En ligne]. Disponible : https://www.sciencedirect.com/science/article/pii/S0278612520301114

[25] Y.-H. Lai *et al.*, “Cognitive optimal-setting control of aiot industiral applications with deep reinforcement learning,” *IEEE Transactions on industrial informatics*, vol. 17, n°. 3, p. 2116–2123, mars 2021. [En ligne]. Disponible : https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9072609

[26] M.-A. Dittrich et S. Fohlmeister, “Cooperative multi-agent system for production control using reinforcement learning,” *CIRP Annals*, vol. 69, n°. 1, p. 389–392, 2020. [En ligne]. Disponible : https://www.sciencedirect.com/science/article/pii/S0007850620300263

[27] T. Zhou *et al.*, “Multi-agent reinforcement learning for online scheduling in smart factories,” *Robotics and computer-integrated Manufacturing*, vol. 72, 2021. [En ligne]. Disponible : https://www.sciencedirect.com/science/article/pii/S0736584521000855

[28] L. Kemmer *et al.*, “Reinforcement learning for supply chain optimization,” communication présentée à European Workshop on Reinforcement Learning 14, Lille, France, Octobre 2018.

[29] M. Neves, M. Vieira et P. Neto, “A study on a q-learning algorithm application to a manufacturing assembly problem,” *Journal of Manufacturing Systems*, vol. 59, p.

426–440, avr. 2021. [En ligne]. Disponible : https://www.sciencedirect.com/science/article/pii/S0278612521000509

[30] C. A. Cuartas Murillo et J. L. Aguilar, “Hybrid algorithm based on reinforcement learning and DDMRP methodology for inventory management,” Universidad EAFIT, Rapport technique, 2021. [En ligne]. Disponible : http://hdl.handle.net/10784/30241

[31] J. Schmidhuber, “Deep learning in neural networks : An overwiew,” The Swiss AI Lab IDSIA, Rapport technique, 2014. [En ligne]. Disponible : https://doi.org/10.48550/arXiv.1404.7828

[32] A. Tavakoli, F. Pardo et P. Kormushev, “Action branching architectures for deep reinforcement learning,” communication présentée à Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, 2-7 février 2018. [En ligne]. Disponible : https://ojs.aaai.org/index.php/AAAI/article/view/11798

[33] About Keras. [En ligne]. Disponible : https://keras.io/about/

[34] H. van Hasselt, A. Guez et D. Silver, “Deep reinforcement learning with double Q-Learning,” dans *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, Février 2016. [En ligne]. Disponible : https://arxiv.org/abs/1509.06461

[35] J. Banks, *Handbook of simulation : principles, methodology, advances, applications, and practice*. John Wiley & Sons, 1998.

[36] D. F. Bozutti et K. Esposto, “Sales and Operations Planning : a comparison between the demand-driven and traditional approaches,” *International Journal of Production Management and Engineering*, vol. 7(1), p. 23–28, 2018. [En ligne]. Disponible : https://polipapers.upv.es/index.php/IJPME/article/view/9469

ANNEXE A REVUE DE LITTÉRATURE DDMRP

Tableau A.1 Revue de littérature DDMRP (partie 1/5)

Source	**Auteur et année**	**Problématique**	**Méthodologie**	**Résultats**	**Apports**
Demand Driven Material Requirements Planning (DDMRP) : A Systematic Review and Classification [3]	Azzamouri et al., 2021	Evolution du DDMRP et des recherches faites dessus, notamment les résultats apportés et les pistes de recherche déjà explorés ou non.	Revue de littérature par recherche d'articles publiés entre 2011 et 2020 sur le DDMRP. Comparaison du DDMRP avec d'autres méthodes de production.	Le DDMRP est assez peu étudié dans la littérature. On n'a pas encore assez de résultats, la recherche est encore nécessaire pour valider la méthode, par exemple l'étude des paramètres.	Proposition de recherche pour rendre le DDMRP plus viable et scientifiquement approuvé.

Tableau A.2 Revue de littérature DDMRP (partie 2/5)

Source	**Auteur et année**	**Problématique**	**Méthodologie**	**Résultats**	**Apports**
Material Management without Forecasting : From MRP to Demand Driven MRP [13]	Kortabarria et al., 2018	Étude des performances d'une entreprise après la conversion MRP vers DDMRP.	Analyse quantitative et qualitative des données d'une entreprise suite à un changement de méthode de production. Rassemblement des données d'entretiens semi-construits, de documents et d'enregistrements d'archive. Comparaison des résultats avant et après DDMRP.	Augmentation de la visibilité dans la chaîne d'approvisionnement et réduction de l'inventaire.	Implémentation physique du DDMRP dans une entreprise. Preuves quantitatives de l'intérêt de cette méthode.
Effective Production Control in an Automotive Industry : MRP vs. Demand-Driven MRP [9]	Shofa and Widyarto, 2017	Prouver l'intérêt du DDMRP par rapport au MRP dans un milieu industriel.	Comparaison de l'inventaire effectif de la méthode MRP et la méthode DDMRP avec les données d'une entreprise automobile en Indonésie.	Le DDMRP donne de meilleurs résultats en termes de délai de mise en oeuvre et niveau d'inventaire.	Comparaison du DDMRP par rapport au MRP dans une entreprise avec des données concrètes.

Tableau A.3 Revue de littérature DDMRP (partie 3/5)

Source	Auteur et année	Problématique	Méthodologie	Résultats	Apports
Evaluating Demand Driven MRP : a case based simulated study [10]	Ihme and Stratton, 2015	Étude d'une implémentation de DDMRP dans une entreprise de fabrication d'encre.	Simulation d'une entreprise gérée en DDMRP à l'aide des données de le la compagnie et du système ERP existant.	Possibilité pour l'entreprise d'améliorer ses performances avec la méthode DDMRP, notamment les retards, les niveaux d'inventaire et les instabilités du système.	Preuve à travers la simulation du potentiel du DDMRP et sa capacité à limiter les instabilités du système.
Decoupled Lead Time in finite capacity flowshop : a feedback loop approach [15]	Orue et al., 2020	Étude du processus d'implémentation du DDMRP dans une entreprise.	Revue de littérature systématique pour analyser la standardisation du processus d'intégration du DDMRP en entreprise.	Résultats positifs et encourageants du DDMRP mais encore trop peu présent en industrie.	Exposition d'une standardisation d'implémentation. Mise en avant des lacunes de la littérature quant à l'implémentation du DDMRP en entreprise.

Tableau A.4 Revue de littérature DDMRP (partie 4/5)

Source	Auteur et année	Problématique	Méthodologie	Résultats	Apports
Strategic WIP Inventory Positioning for Make-to-Order Production with Stochastic Processing Times [14]	Jiang and Rim, 2017	Création d'un modèle mathématique pour placer et quantifier les en cours.	Définir les stations de production qui doivent garder du stock en inventaire pour les produits en cours de production. Étude réalisée dans un environnement avec des demandes aléatoires et stochastiques.	Réduction des temps de mise en oeuvre.	Utilisation d'un algorithme génétique pour simuler le DDMRP.
MRP vs Demand-Driven MRP : Towards an Objective Comparison [12]	Miclo et al., 2015	Premières recherches sur l'intérêt d'utiliser le DDMRP par rapport au MRP.	Démonstration quantitative des avantages du DDMRP par rapport au MRP à travers une simulation à événements discrets.	Résultats contrastés pour le DDMRP, bonne réaction face à la variabilité mais parfois les niveaux d'inventaire sont trop faibles. La modification des paramètres influe sur les résultats et constitue un enjeu du DDMRP.	Première partie d'une série d'articles sur la comparaison MRP/DDMRP, première quantification et comparaison des deux méthodes.

Tableau A.5 Revue de littérature DDMRP (partie 5/5)

Source	Auteur et année	Problématique	Méthodologie	Résultats	Apports
Vers une cartographie de processus explicite pour le modèle Demand Driven Adaptive Enterprise [16]	Martin et al., 2018	Cartographie du processus appliqué à la méthodologie Demand Driven Adaptive Enterprise afin de s'adapter aux enjeux de l'industrie.	Établissement d'une méthode Demand Driven Adaptive Enterprise à l'aide des publications existantes et de conseils d'experts.	Cartographie adaptable et interprétable aux différents cas où elle est appliquée. Possibilité de recherche et d'approfondissement.	Mise en place d'une méthode adaptée à différents niveaux.
Sales and Operations Planning : a comparison between the demand-driven and traditional approaches [36]	Bozutti and Espôsto, 2018	Étude du rôle du S&OP (Sales and Operation Planning) dans la gestion de la chaîne de production et l'intérêt de le rendre davantage piloté par la demande.	Revue de littérature conduite sur des articles de S&OP pour comparer l'approche traditionnelle et l'approche pilotée par la demande.	Le S&OP est développé et étudié dans le sens de la production pilotée par la demande, mais plus d'étude devraient être menées afin d'analyser les performances dans l'industrie.	Comparaison d'une nouvelle façon de produire à une approche traditionnelle. Proposition de recherche et d'amélioration pour la littérature du S&OP piloté par la demande.

ANNEXE B REVUE DE LITTÉRATURE APPRENTISSAGE PAR RENFORCEMENT

TABLEAU B.1 Revue de littérature apprentissage par renforcement (partie 1/6)

Source	**Auteur et année**	**Problématique**	**Méthodologie**	**Résultats**	**Apports**
A reinforcement learning approach for supply chain management [21]	Stockheim et al., 2003	Optimisation de la planification d'une chaîne de production décentralisée par un agent d'apprentissage par renforcement.	Un agent d'apprentissage par renforcement va considérer les tâches à effectuer dans une chaîne de production afin de réaliser une commande. L'agent va ainsi calculer un planning optimisé qui prend compte des coûts de fabrication et des pénalités de retard.	Les performances du système sont meilleures par rapport à une heuristique d'apprentissage simple. On dénote tout de même des performances variées pour différents paramètres de notre agent, notamment au niveau de l'équilibre exploitation-exploration.	Comparaison de l'apprentissage par renforcement à des méthodes d'apprentissage standards. Variabilité des résultats de l'algorithme et de l'agent en fonction des paramètres d'entrée.

Tableau B.2 Revue de littérature apprentissage par renforcement (partie 2/6)

Source	Auteur et année	Problématique	Méthodologie	Résultats	Apports
A reinforcement learning approach to production planning in the fabrication/fulfilment manufacturing process [22]	Cao et al., 2003	Résolution d'un problème de planification de la production en « multi-période » avec l'apprentissage par renforcement.	Modélisation du problème par un processus de décision Markovien (MDP), et développement d'un agent d'apprentissage par renforcement afin de réduire les coûts des actions à prendre, sous certaines contraintes.	Validation de l'efficacité de la méthode et construction d'un plan de production optimal. Réduction des temps de livraison pour les différents produits.	Utilisation d'une simulation de Monte Carlo doublée d'apprentissage par renforcement. Schéma d'apprentissage en deux phases.
Cognitive Optimal-Setting Control of AIoT Industrial Applications with Deep Reinforcement Learning [25]	Lai et al., 2021	Limiter les effets de l'overfitting (ou surapprentissage) dans les applications industrielles de l'apprentissage par renforcement.	Utilisation d'un algorithme d'apprentissage par avantage amélioré, une méthode qui permet d'évaluer « l'avantage » de chaque action, c'est-à-dire sa plus-value par rapport à d'autres actions.	Diminution du surapprentissage et amélioration de l'algorithme d'apprentissage par avantage. Le modèle présente de meilleurs résultats sur des ensembles d'entraînement.	Diminution du surapprentissage. Apprentissage par avantage.

Tableau B.3 Revue de littérature apprentissage par renforcement (partie 3/6)

Source	**Auteur et année**	**Problématique**	**Méthodologie**	**Résultats**	**Apports**
Cooperative multi-agent system for production control using reinforcement learning [26]	Dittrich and Fohlmeister, 2020	Utilisation d'un agent d'apprentissage par renforcement pour un système multi-agent coopératif.	Considération d'indicateurs clés par un algorithme de deep q-learning afin d'améliorer les performances du système et réduire les coûts et les temps de livraison.	Après plusieurs itérations, l'agent présente de meilleurs résultats qu'une planification aléatoire, on réussit à réduire le temps de cycle moyen des produits, et ainsi les temps de livraison.	Étude du compromis exploration-exploitation. Introduction de deep q-learning.
Multi-agent reinforcement learning for online scheduling in smart factories [27]	Zhou et al., 2021	Introduction d'un agent d'apprentissage par renforcement dans une usine intelligente afin d'optimiser une planification sur des bases de données de grande taille.	Comparaison de plusieurs métaheuristiques de planification dans une usine intelligente. Conception d'algorithme d'aide à la décision exécutés sur des systèmes distribués. L'apprentissage se fait au niveau des céduleurs, mais également avec les autres céduleurs.	Amélioration des prises de décision et des performances de la planification. Capacité des céduleurs à apprendre de grandes bases de données et à anticiper des événements imprévus.	Introduction de réseaux de neurones. Création de céduleurs interdépendants.

Tableau B.4 Revue de littérature apprentissage par renforcement (partie 4/6)

Source	Auteur et année	Problématique	Méthodologie	Résultats	Apports
Real-time decision support with reinforcement learning for dynamic flow-shop scheduling [23]	Wang et al., 2017	Améliorer la planification d'un atelier dynamique avec l'apprentissage par renforcement.	Proposition d'un algorithme d'apprentissage par renforcement et comparaison des performances d'un modèle régi par des règles heuristiques. Discussion sur l'algorithme d'apprentissage machine.	Meilleures performances de l'apprentissage par renforcement dans la plupart des ensembles d'entraînement, notamment lorsqu'il y a beaucoup de commandes. Étude des avantages et des inconvénients de l'apprentissage par renforcement.	Comparaison d'une méthode heuristique et d'un algorithme d'apprentissage machine. Discussion quant à l'efficacité et la pertinence d'utiliser le RL.
Reinforcement learning-based and Parametric Production-maintenance control policies for a deteriorating Manufacturing System [20]	Xanthopoulos et al., 2017	Maintenir un taux de service élevé et des stocks bas en utilisant l'apprentissage par renforcement dans un environnement où les produits sont soumis à des défauts de détérioration provenant des machines.	Paramétrisation du problème et mise en forme de l'environnement sous un modèle stochastique. Ajout d'une composante d'apprentissage par renforcement sur les politiques de contrôle de la production.	Meilleures performances pour la composante comprenant l'apprentissage par renforcement, nuancées par le fait que les paramètres ont un impact important sur les résultats.	Démonstration de la pertinence d'utiliser l'apprentissage par renforcement dans un contexte industriel. Importance de la paramétrisation de l'algorithme.

Tableau B.5 Revue de littérature apprentissage par renforcement (partie 5/6)

Source	Auteur et année	Problématique	Méthodologie	Résultats	Apports
Reinforcement learning for supply chain optimization [28]	Kemmer et al., 2018	Étude de la performance de deux agents d'apprentissage par renforcement dans une usine de production simulée par un processus de décision Markovien (MDP).	Optimisation de la productivité d'une entreprise de beurre constituée d'une usine et plusieurs entrepôts. L'agent agit notamment sur la quantité de beurre à produire et où le stocker.	Les coûts observés sont moins élevés pour les agents de RL sur un scénario simple. Pour des environnements plus complexes, un seul agent se démarque. Les deux agents montrent une bonne réaction face à la variabilité de l'environnement.	Comparaison d'algorithmes d'apprentissage par renforcement. Simulation d'un système de production par MDP.
A study on a Q-Learning algorithm application to a manufacturing assembly problem [29]	Neves et al., 2021	Étude de l'utilisation d'un algorithme d'apprentissage par renforcement dans un processus d'assemblage.	Formulation du problème comme un MDP puis développement d'un algorithme de Q-learning et d'un environnement DQN pour construire un jouet.	Assemblage optimal 98,3% du temps, bonne réaction de l'algorithme face à la variabilité, mise en garde face au fléau de la dimension.	Limites et avantages du deep Q-learning, incitation à explorer d'autres algorithmes.

Tableau B.6 Revue de littérature apprentissage par renforcement (partie 6/6)

Source	**Auteur et année**	**Problématique**	**Méthodologie**	**Résultats**	**Apports**
Utilization of a reinforcement learning algorithm for the accurate alignment of a robotic arm in a complete soft fabric shoe tongues automation process [24]	Tsai et al., 2020	Application d'un algorithme d'apprentissage par renforcement pour les mouvements d'un bras articulé dans une ligne de production de chaussure.	Définition d'un problème en MDP, ainsi que d'un espace d'actions à six actions. Utilisation d'un Deep learning network (DNN) afin de déterminer les actions en fonction d'images reçues par le robot.	Très bonnes performances de l'algorithme, notamment en comparaison avec des résultats obtenus pour des travailleurs humains, néanmoins cela nécessite une certaine période d'apprentissage.	Importance de la phase d'apprentissage, application à un environnement tridimensionnel, étude des performances.
Action branching Architectures for Deep Reinforcement Learning [32]	Tavakoli et al., 2018	Développement d'un algorithme à actions discrètes pour un système d'action à plusieurs dimensions.	Utilisation d'une architecture neuronale afin d'obtenir un système de décision partagé avec plusieurs libertés d'action.	Obtention d'indépendance entre les différentes actions, meilleures performances qu'un algorithme traditionnel de *Deep deterministic policy gradient.*	Système de décision partagé. Architecture neuronal.

ANNEXE C DIAGRAMME DE CLASSES DU MODÈLE

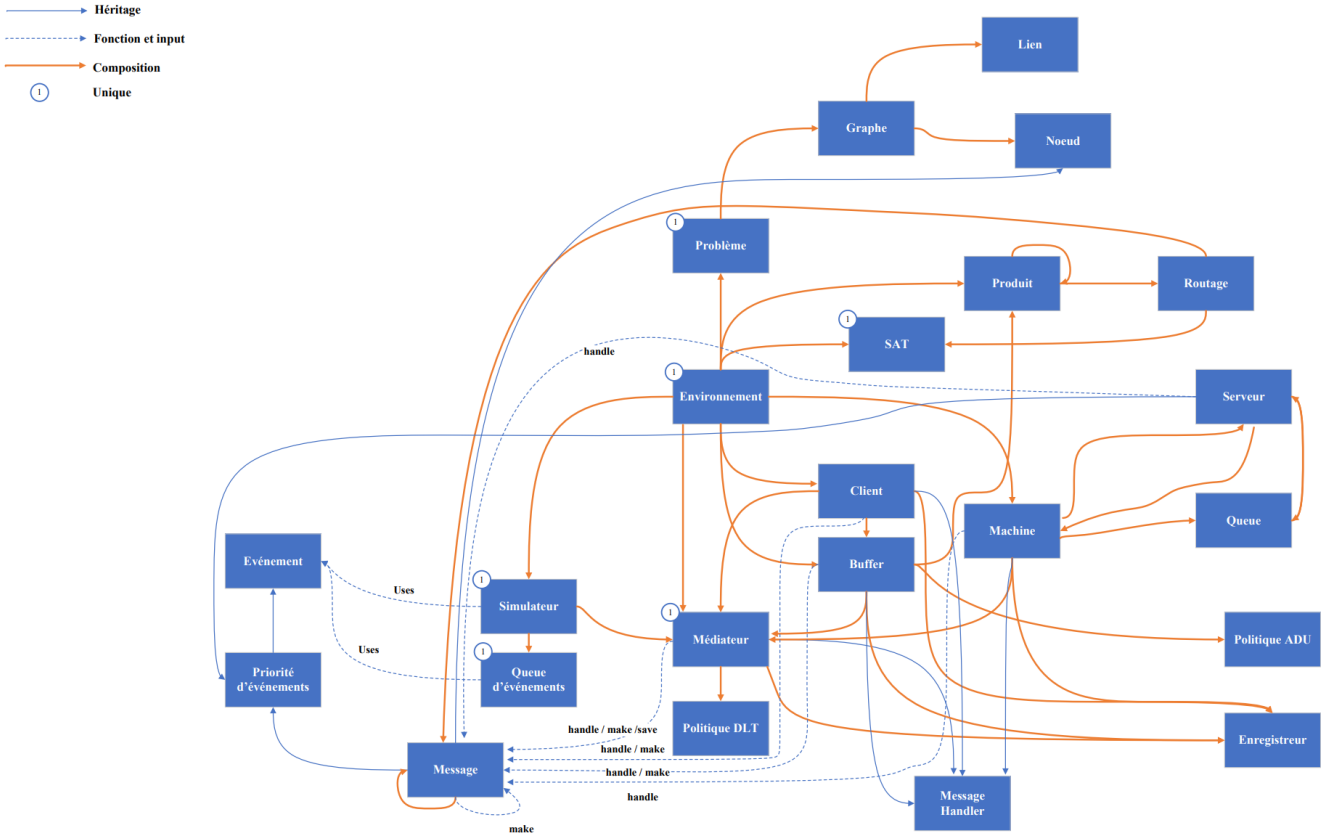


FIGURE C.1 UML du modèle

ANNEXE D ELÉMENTS DE CODE

```python
# GREEDY STRATEGY

import numpy as np
import tensorflow as tf

class GreedyStrategy():
    def __init__(self):
        self.exploratory_action=False

    def select_action(self,model,state):
        state_tensor = tf.convert_to_tensor(state)
        state_tensor = tf.expand_dims(state_tensor, 0)
        q_values = model(state_tensor, training=False)
        actions=[]
        for i in range(len(state)):
            actions.append(np.argmax(q_values[0][i]))
        return(actions)
```

```python
# EPSILON GREEDY EXPONENTIAL STRATEGY

import numpy as np
import tensorflow as tf

class EGreedyExpStrategy():
    def __init__(self, init_epsilon=1.0, min_epsilon=0.1,
       decay_steps=20000):
        self.epsilon = init_epsilon
        self.init_epsilon = init_epsilon
        self.decay_steps = decay_steps
        self.min_epsilon = min_epsilon
        self.epsilons = 0.01 / np.logspace(-2, 0, decay_steps, \
           endpoint=False) - 0.01
        self.epsilons = self.epsilons * (init_epsilon - \
```

```
16              min_epsilon) + min_epsilon
17          self.t = 0
18          self.exploratory_action_taken = None
19
20      def _epsilon_update(self):
21          self.epsilon = self.min_epsilon if self.t >= \
22              self.decay_steps else self.epsilons[self.t]
23          self.t += 1
24          return self.epsilon
25
26      def select_action(self, model, state):
27          self.exploratory_action_taken = False
28          state_tensor = tf.convert_to_tensor(state)
29          state_tensor = tf.expand_dims(state_tensor, 0)
30          # reminder q_value shape is
31          # batch_size*action_streams*actions_per_stream
32          q_values = model(state_tensor, training=False)
33          action_streams = q_values.shape[1]
34          # retrieve the number of actions to choose
35          actions=[]
36          if np.random.rand() > self.epsilon:
37              for i in range(action_streams):
38                  exploration = False
39                  action=np.argmax(q_values[0][i])
40                  actions.append(action)
41          else:
42              for i in range(action_streams):
43                  exploration = True
44                  # get a random action between 0
45                  # and actions_per_stream
46                  action=np.random.randint(0, q_values.shape[2])
47                  actions.append(action)
48          self._epsilon_update()
49          self.exploratory_action_taken = exploration
50          return actions
```

```
# NEURAL NETWORK FONCTION

import tensorflow as tf

def branching_network(input_dims, num_action_branches,
 actions_per_branch, hidden_commons=[512, 256],
 hidden_values=[128], hidden_actions=[128]):
    """
    N : number of action streams
    n : actions per branch
    b : batch size
    m : input columns
    """
    # input layers
    input_layer = tf.keras.Input(input_dims, name='input_layer')
    # initializer
    initializer = tf.keras.initializers.LecunNormal()
    # flattening 3D array into 2D
    out = input_layer
    # normalization
    out = tf.keras.layers.Normalization()(input_layer)
    # expected output size is b*(N*n)
    out = tf.keras.layers.Flatten()(out)

    # making the common network
    # expected output b*h_dim[-1] (default is b*256)
    for i, hidden_dim in enumerate(hidden_commons):
        out = tf.keras.layers.Dense(
            hidden_dim, activation='selu',
            name=f'hidden_common_{i}',
            kernel_initializer = initializer)(out)

    # making the action streams
    # expected output are the action advantage scores
    # size b*N*n
```

```
    action_scores = []
    for action_stream in range(num_action_branches):
        # individual action stream
        action_out = out
        for j, hidden in enumerate(hidden_actions):
            if hidden != 0:
                action_out = tf.keras.layers.Dense(
              hidden, activation="selu",
              name=f"action_hidden_{j}_stream_{action_stream}",
              kernel_initializer = initializer)(action_out)
        action_scores.append(
            tf.keras.layers.Dense(
                actions_per_branch, activation=None,
                name=f"action_bin_{action_stream}"
            )(action_out)
        )
    # stacking all actions scores into a single tensor
    total_action_scores = tf.stack(action_scores, axis=1,
       name="stack_advantages")
    # following paper, removing average
    # advantage from each score
    # average should be taken on the action dimension (n)
    mean_action_score = tf.reduce_mean(total_action_scores,
       axis=2, keepdims=True)
    # adjusted action score should be of size b*N*n
    adjusted_action_score = tf.keras.layers.Subtract()(
       [total_action_scores, mean_action_score])

    # make the value branch
    # expected out size b*1
    state_out = out
    for i, hidden in enumerate(hidden_values):
        if hidden != 0:
            state_out = tf.keras.layers.Dense(
                hidden, activation='selu',
               name=f'state_hidden_{i}',
```

```
72                  kernel_initializer = initializer
73              )(state_out)
74      state_score = tf.keras.layers.Dense(
75          1, activation=None, name="state_score"
76      )(state_out)
77
78      # compute q values
79      # expected shape b*N*n
80      q_values = tf.keras.layers.Add()([state_score,
81          adjusted_action_score])
82
83      # return model
84      model = tf.keras.Model(input_layer, outputs=q_values)
85
86      return model
```

ANNEXE E RÉSULTATS EXPÉRIENCE 1

Ces tableaux rassemblent des RAM par épisode de nos scénarios, ainsi que la valeur de la RAM du scénario exécuté sans agent (Baseline).

TABLEAU E.1 RAM par épisode des scénarios de l'expérience 1 (partie 1/3)

Scénario	Initialisation OST	Initialisation OSH	Épisode 3	Épisode 25	Épisode 50	Épisode 75	Épisode 100	Baseline
1	50%	1.0 * DLT	0.262 ± 0.021	0.279 ± 0.015	0.293 ± 0.019	0.298 ± 0.019	0.281 ± 0.011	0.304 ± 0.001
2	50%	1.25 * DLT	0.262 ± 0.021	0.278 ± 0.012	0.287 ± 0.012	0.287 ± 0.008	0.294 ± 0.009	0.266 ± 0.001
3	50%	1.5 * DLT	0.265 ± 0.033	0.274 ± 0.019	0.284 ± 0.010	0.280 ± 0.016	0.286 ± 0.017	0.279 ± 0.001
4	RS	1.0 * DLT	0.267 ± 0.013	0.286 ± 0.010	0.284 ± 0.005	0.286 ± 0.015	0.285 ± 0.008	0.273 ± 0.002
5	RS	1.25 * DLT	0.253 ± 0.023	0.276 ± 0.012	0.285 ± 0.004	0.281 ± 0.011	0.284 ± 0.010	0.258 ± 0.001
6	RS	1.5 * DLT	0.259 ± 0.006	0.282 ± 0.003	0.287 ± 0.021	0.298 ± 0.019	0.289 ± 0.008	0.266± 0.001
7	ADU	1.0 * DLT	0.265 ± 0.028	0.284 ± 0.019	0.289 ± 0.014	0.287 ± 0.010	0.283 ± 0.003	0.293 ± 0.003
8	ADU	1.25 * DLT	0.259 ± 0.032	0.285 ± 0.013	0.293 ± 0.027	0.287 ± 0.014	0.291 ± 0.004	0.276 ± 0.001
9	ADU	1.5 * DLT	0.264 ± 0.025	0.269 ± 0.020	0.285 ± 0.004	0.284 ± 0.006	0.283 ± 0.009	0.280 ± 0.001

Tableau E.2 RAM par épisode des scénarios de l'expérience 1 (partie 2/3)

Scénario	Initialisation OST	Initialisation OSH	Épisode 1	Épisode 25	Épisode 50	Épisode 75	Épisode 100	Baseline
10	50%	1.0 * DLT	0.259 ± .003	0.280 ± 0.005	0.286 ± 0.007	0.286 ± 0.006	0.290 ± 0.007	0.303 ± 0.001
11	50%	1.25 * DLT	0.256 ± 0.015	0.269 ± 0.012	0.272 ± 0.014	0.277 ± 0.002	0.275 ± 0.003	0.267 ± 0.001
12	50%	1.5 * DLT	0.263 ± 0.002	0.271 ± 0.007	0.279 ± 0.008	0.278 ± 0.007	0.278 ± 0.004	0.279 ± 0.001
13	RS	1.0 * DLT	0.259 ± 0.002	0.284 ± 0.011	0.285 ± 0.014	0.287 ± 0.011	0.287 ± 0.004	0.273 ± 0.001
14	RS	1.25 * DLT	0.255 ± 0.016	0.272 ± 0.004	0.277 ± 0.003	0.281 ± 0.002	0.276 ± 0.006	0.257 ± 0.001
15	RS	1.5 * DLT	0.268 ± 0.006	0.271 ± 0.012	0.267 ± 0.012	0.274 ± 0.003	0.275 ± 0.005	0.267 ± 0.001
16	ADU	1.0 * DLT	0.257 ± 0.003	0.286 ± 0.010	0.293 ± 0.005	0.292 ± 0.006	0.293 ± 0.009	0.295 ± 0.002
17	ADU	1.25 * DLT	0.259 ± 0.009	0.274 ± 0.006	0.281 ± 0.004	0.281 ± 0.007	0.279 ± 0.001	0.275 ± 0.001
18	ADU	1.5 * DLT	0.265 ± 0.004	0.276 ± 0.005	0.285 ± 0.007	0.280 ± 0.010	0.280 ± 0.002	0.281 ± 0.002

Tableau E.3 RAM par épisode des scénarios de l'expérience 1 (partie 3/3)

Scénario	Initialisation OST	Initialisation OSH	Épisode 3	Épisode 25	Épisode 50	Épisode 75	Épisode 100	Baseline
19	50%	1.0 * DLT	0.291 ± 0.006	0.292 ± 0.012	0.300 ± 0.018	0.293 ± 0.007	0.294 ± 0.009	0.303 ± 0.001
20	50%	1.25 * DLT	0.304 ± 0.008	0.286 ± 0.005	0.279 ± 0.017	0.285 ± 0.008	0.288 ± 0.015	0.267 ± 0.001
21	50%	1.5 * DLT	0.314 ± 0.004	0.291 ± 0.017	0.287 ± 0.020	0.290 ± 0.012	0.286 ± 0.009	0.280 ± 0.001
22	RS	1.0 * DLT	0.263 ± 0.003	0.264 ± 0.002	0.267 ± 0.005	0.266 ± 0.006	0.264 ± 0.009	0.273 ± 0.002
23	RS	1.25 * DLT	0.264 ± 0.003	0.261 ± 0.007	0.267 ± 0.001	0.262 ± 0.005	0.263 ± 0.004	0.256 ± 0.001
24	RS	1.5 * DLT	0.262 ± 0.006	0.261 ± 0.005	0.263 ± 0.002	0.266 ± 0.008	0.264 ± 0.001	0.264 ± 0.002
25	ADU	1.0 * DLT	0.298 ± 0.004	0.292 ± 0.006	0.296 ± 0.013	0.299 ± 0.019	0.298 ± 0.005	0.295 ± 0.001
26	ADU	1.25 * DLT	0.299 ± 0.002	0.298 ± 0.017	0.292 ± 0.019	0.290 ± 0.010	0.291 ± 0.004	0.275 ± 0.001
27	ADU	1.5 * DLT	0.299 ± 0.003	0.290 ± 0.007	0.284 ± 0.009	0.282 ± 0.007	0.282 ± 0.007	0.281 ± 0.002

ANNEXE F RÉSULTATS EXPÉRIENCE 3

Ces tableaux rassemblent des RAM par épisode de nos scénarios, ainsi que la valeur de la RAM du scénario exécuté sans agent (Baseline).

Tableau F.1 RAM par épisode des scénarios de l'expérience 3 (partie 1/3)

Scénario	Fréquence des pics	Initialisation OST	Charge de travail	Épisode 1	Épisode 25	Épisode 50	Épisode 75	Épisode 100	Baseline
37	5	50%	50%	0.664 ± 0.005	0.706 ± 0.010	0.723 ± 0.011	0.723 ± 0.010	0.718 ± 0.010	0.704 ± 0.001
38	5	50%	80%	0.275 ± 0.005	0.347 ± 0.021	0.355 ± 0.015	0.331 ± 0.014	0.336 ± 0.013	0.319 ± 0.001
39	5	50%	95%	0.158 ± 0.007	0.176 ± 0.004	0.179 ± 0.011	0.182 ± 0.015	0.185 ± 0.006	0.176 ± 0.001
40	5	RS	50%	0.666 ± 0.005	0.694 ± 0.022	0.712 ± 0.019	0.717 ± 0.008	0.714 ± 0.009	0.658 ± 0.001
41	5	RS	80%	0.278 ± 0.006	0.325 ± 0.017	0.329 ± 0.018	0.332 ± 0.025	0.334 ± 0.009	0.294 ± 0.001
42	5	RS	95%	0.156 ± 0.010	0.183 ± 0.009	0.191 ± 0.004	0.184 ± 0.006	0.185 ± 0.001	0.166 ± 0.002
43	5	ADU	50%	0.664 ± 0.006	0.706 ± 0.007	0.710 ± 0.019	0.710 ± 0.024	0.708 ± 0.005	0.684 ± 0.001
44	5	ADU	80%	0.276 ± 0.007	0.340 ± 0.016	0.344 ± 0.016	0.340 ± 0.019	0.347 ± 0.024	0.316 ± 0.001
45	5	ADU	95%	0.154 ± 0.009	0.176 ± 0.005	0.170 ± 0.008	0.175 ± 0.006	0.178 ± 0.008	0.168 ± 0.001

Tableau F.2 RAM par épisode des scénarios de l'expérience 3 (partie 2/3)

Scénario	Fréquence des pics	Initialisation OST	Charge de travail	Épisode 1	Épisode 25	Épisode 50	Épisode 75	Épisode 100	Baseline
46	10	50%	50%	0.691 ± 0.003	0.710 ± 0.024	0.722 ± 0.014	0.715 ± 0.010	0.728 ± 0.005	0.709 ± 0.001
47	10	50%	80%	0.340 ± 0.001	0.339 ± 0.003	0.338 ± 0.004	0.338 ± 0.004	0.337 ± 0.003	0.342 ± 0.001
48	10	50%	95%	0.152 ± 0.002	0.169 ± 0.003	0.164 ± 0.004	0.165 ± 0.005	0.167 ± 0.003	0.159 ± 0.001
49	10	RS	50%	0.691 ± 0.003	0.716 ± 0.020	0.749 ± 0.032	0.759 ± 0.037	0.756 ± 0.034	0.701 ± 0.001
50	10	RS	80%	0.341 ± 0.002	0.348 ± 0.006	0.352 ± 0.009	0.349 ± 0.008	0.347 ± 0.007	0.333 ± 0.001
51	10	RS	95%	0.151 ± 0.002	0.169 ± 0.004	0.171 ± 0.005	0.169 ± 0.003	0.168 ± 0.003	0.158 ± 0.001
52	10	ADU	50%	0.691 ± 0.002	0.723 ± 0.019	0.719 ± 0.025	0.732 ± 0.052	0.726 ± 0.027	0.692 ± 0.001
53	10	ADU	80%	0.337 ± 0.003	0.347 ± 0.010	0.347 ± 0.022	0.347 ± 0.008	0.342 ± 0.014	0.320 ± 0.001
54	10	ADU	95%	0.151 ± 0.007	0.164 ± 0.016	0.164 ± 0.019	0.161 ± 0.020	0.166 ± 0.006	0.158 ± 0.001

Tableau F.3 RAM par épisode des scénarios de l'expérience 3 (partie 3/3)

Scénario	Fréquence des pics	Initialisation OST	Charge de travail	Épisode 1	Épisode 25	Épisode 50	Épisode 75	Épisode 100	Baseline
55	20	50%	50%	0.614 ± 0.001	0.628 ± 0.004	0.635 ± 0.005	0.635 ± 0.003	0.635 ± 0.003	0.630 ± 0.001
56	20	50%	80%	0.265 ± 0.002	0.274 ± 0.003	0.278 ± 0.003	0.277 ± 0.002	0.277 ± 0.001	0.279 ± 0.0001
57	20	50%	95%	0.139 ± 0.011	0.134 ± 0.005	0.141 ± 0.006	0.139 ± 0.010	0.143 ± 0.004	0.150 ± 0.001
58	20	RS	50%	0.614 ± 0.002	0.631 ± 0.021	0.633 ± 0.021	0.634 ± 0.011	0.628 ± 0.008	0.606 ± 0.001
59	20	RS	80%	0.268 ± 0.006	0.271 ± 0.012	0.267 ± 0.012	0.274 ± 0.003	0.275 ± 0.005	0.267 ± 0.002
60	20	RS	95%	0.140 ± 0.013	0.142 ± 0.014	0.139 ± 0.015	0.139 ± 0.010	0.143 ± 0.004	0.150 ± 0.001
61	20	ADU	50%	0.612 ± 0.006	0.623 ± 0.016	0.625 ± 0.013	0.631 ± 0.011	0.636 ± 0.016	0.574 ± 0.002
62	20	ADU	80%	0.267 ± 0.002	0.275 ± 0.002	0.280 ± 0.002	0.278 ± 0.002	0.278 ± 0.001	0.281 ± 0.001
63	20	ADU	95%	0.136 ± 0.001	0.142 ± 0.007	0.146 ± 0.008	0.144 ± 0.003	0.145 ± 0.002	0.131 ± 0.001