UQAC
Université du Québec
à Chicoutimi

ANALYSE DE L'IMPACT DE L'ARCHITECTURE ENTITÉ-COMPOSANT-SYSTÈME SUR LES PERFORMANCES ET LA CONSOMMATION ÉNERGÉTIQUE DANS LES SERVEURS DE JEUX MULTI-JOUEURS.

PAR LORI LOU

MÉMOIRE PRÉSENTÉ À L'UNIVERSITÉ DU QUÉBEC À CHICOUTIMI EN VUE DE L'OBTENTION DU GRADE DE MAÎTRISE ÈS SCIENCES (M. SC.) EN INFORMATIQUE

QUÉBEC, CANADA

RÉSUMÉ

L'essor des jeux vidéo multijoueurs en ligne et l'accroissement du nombre de centres de données associés soulèvent des enjeux majeurs en matière de consommation énergétique et de performance des serveurs. Face à cette problématique, ce mémoire étudie l'impact de l'architecture Entité-Composant-Système, fondée sur la programmation orientée donnée, par rapport à l'architecture traditionnelle en Programmation Orientée Objet. La question de recherche posée est la suivante : Est-ce que l'implémentation d'un Entité-Composant-Système dans la programmation d'un serveur de jeu multijoueurs en ligne permettrait de réduire le coût en performance des processeurs de serveurs afin d'en réduire leur consommation énergétique ? Pour y répondre, une expérimentation comparative a été conduite à travers la création de serveurs utilisant chacune de ces deux architectures, et des mesures précises de consommation et d'efficience ont été réalisées. En optimisant l'usage du cache CPU par une meilleure localité des données et une plus grande vectorisation, les résultats montrent que l'ECS permet de diminuer l'empreinte énergétique tout en augmentant la capacité de traitement des serveurs. Cette recherche souligne ainsi l'intérêt de l'architecture Entité-Composant-Système comme alternative plus durable et performante pour les serveurs de jeux vidéo en ligne.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ABRÉVIATIONS

ECSA Entité-Composant-Système Architecture
ECS Entité-Composant-Système
IPS Images par secondes
POO Programmation Orientée Objet
P2P Pair-à-Pair
RAM Random Access Memory
COD Conception Orientée Données
POD Programmation Orientée Données
SDT Structures De Tableaux
SIMD Single Instruction, Multiple Data

REMERCIEMENTS

Je souhaite exprimer ma profonde gratitude à l'ensemble du corps professoral de l'UQAC pour leur générosité dans le partage de leurs connaissances, leur bienveillance, leur pédagogie et leur soutien constant, qui m'ont permis de mener à bien la rédaction de ce mémoire. Je tiens également à remercier amicalement Lucas Guichard pour son accompagnement tout au long de mon parcours universitaire et sa contribution, directe ou indirecte, à l'aboutissement de ce projet de recherche.

Merci à tous ceux qui ont su me soutenir et croire en moi, que ce soit les inconnus curieux au détour d'une discussion ou les proches intéressés, je vous suis tous reconnaissant pour la pierre que vous avez su ajouter à l'édifice de ce mémoire.

AVANT-PROPOS

Ce mémoire s'inscrit dans une démarche continue d'apprentissage et de développement d'expertise dans le domaine du jeu vidéo, une passion qui m'anime depuis mon plus jeune âge. Ce travail représente pour moi une occasion d'élargir mes compétences afin de contribuer pleinement aux projets de création vidéoludique qui me tiennent à cœur, en y apportant des solutions techniques cohérentes et bénéfiques pour tous.

L'architecture Entité-Composant-Système (ECS) incarne des valeurs qui me sont chères, en permettant d'imaginer des jeux toujours plus ambitieux tout en préservant une qualité d'expérience optimale pour les joueurs.

La question de la consommation énergétique associée à cette architecture a toujours suscité en moi une réflexion particulière. Ses qualités m'ont conduit à envisager l'ECS comme une alternative énergétique à la programmation orientée objet. C'est de ces interrogations et de cet intérêt qu'est née l'idée de ce mémoire.

INTRODUCTION

1 CONTEXTE

Depuis l'émergence et la montée en popularité des jeux vidéo multijoueurs en ligne avec une estimation de 1,1 milliard de joueurs en ligne en 2020 [1], l'industrie vidéoludique produit des jeux nécessitant de plus en plus de ressources, que ce soit des ressources humaines avec des projets impliquant des centaines de personnes ou encore des ressources matérielles afin de pouvoir simuler les derniers jeux qui continuent d'innover pour rendre leurs expériences toujours plus complètes et spectaculaires. Qu'il s'agisse de jeux de tir compétitifs orchestrant la synchronisation de milliers de joueurs au sein de multiples instances de jeu, ou de jeux massivement multijoueurs requérant l'intégration de milliers d'individus sur de vastes cartes, l'ensemble de ces applications repose sur l'utilisation de serveurs dédiés à la transmission des données et à la gestion des interactions entre les joueurs. En vue de la quantité astronomique du nombre de joueurs à travers le monde, les serveurs de jeux sont répartis tout autour du globe au travers de plus de 5000 centres de données [2]. Afin que tous ces joueurs puissent rejoindre une partie de jeu, il faut obligatoirement qu'un serveur soit disponible en tout temps afin de pouvoir héberger une partie. Rien que pour le jeu Counter-Strike 2, voir Figure 1, en vue d'une moyenne de 10 joueurs par parties, cela totalise environ 120 000 parties devant être hébergées pour 1 200 000 joueurs actifs en moyenne [3], et cela pour seulement un seul jeu populaire parmi des centaines.

---

1. https://www.statista.com/topics/1551/online-gaming/, Saisi le 9 juillet 2025
2. https://fr.statista.com/infographie/24147/pays-avec-le-plus-de-data-centers-centres-de-donnees/, Saisi le 9 juillet 2025
3. https://steamdb.info/app/730/charts/, Saisi le 9 juillet 2025

FIGURE 1 : Image tirée du jeu Counter-Strike 2. (2023)

2 PROBLÈME

Le problème actuel réside alors dans le fait que les serveurs de jeux vidéo engendrent une consommation significative de bande passante et d'énergie. Le domaine du jeu vidéo multijoueur se révèle être un vecteur de pollution important, tant au niveau de la construction des centres de données que dans leur gestion des ressources serveurs [1]. Les centres de données, dans leur globalité, ont engendré un grand nombre de recherches dans le cadre de leur consommation énergétique afin de réduire au maximum leur consommation, d'un point de vue économique et environnemental. La plupart des recherches visent à améliorer le matériel informatique comme les alimentations par exemple [2]. Cependant, dans le cadre du jeu vidéo, un autre aspect au-delà du matériel est très important : le logiciel. En effet, puisqu'un serveur de jeu doit pouvoir héberger une partie et simuler les interactions des différents joueurs, ceux-ci doivent exécuter le jeu et pour un logiciel aussi coûteux en performances, sa consommation énergétique est donc importante.

3 QUESTION DE RECHERCHE

Est-ce que l'implémentation d'un Entité-Composant-Système (ECS) dans la programmation d'un serveur de jeu multijoueurs en ligne permettrait de réduire le coût en performance des processeurs de serveurs afin d'en réduire leur consommation énergétique ? Le présent projet de recherche vise alors à explorer l'utilisation de l'ECS, une alternative au paradigme de programmation prédominant : la POO, dans l'objectif d'analyser son impact sur les performances et la consommation énergétiques des jeux hébergés sur serveurs dédiés. Là où un serveur faisait fonctionner une seule partie en POO, on espère pouvoir en faire fonctionner au moins deux en utilisant l'ECS, ce qui réduirait considérablement leur impact énergétique.

4 STRUCTURE DU DOCUMENT

Dans le chapitre suivant intitulé « prérequis » nous allons présenter les concepts principaux sur lesquels la recherche se repose. Ce premier chapitre permet d'apporter une base de connaissance suffisante à la bonne compréhension de la recherche, il définit le jeu vidéo, sa composante multijoueurs en ligne, l'informatique verte et les architectures de programmation orientée objet, orientée donnée et l'ECS. Les prérequis apportent aussi une compréhension du fonctionnement du cache processeur et de l'efficience de l'architecture ECS sur son utilisation. Le second chapitre est dédié à l'état de l'art sur les ECS, les différents cadriciels disponibles et utilisés dans l'industrie, les patrons les plus couramment utilisés et ce qui a déjà été étudié dans le cadre de l'optimisation des performances des logiciels via cette architecture. Le chapitre trois repose sur la méthodologie employée pour programmer un serveur de jeu avec une architecture ECS et Programmation Orientée Objet (POO) comparables, comment mesurer la différence de coût énergétique entre les deux modèles et comment détecter les failles de performances sur ce serveur de jeu. Le quatrième chapitre présente les analyses

des résultats obtenus suite à l'application de la méthodologie en comparant les performances énergétiques et logiques entre les différentes architectures via les résultats des bancs de test. Le cinquième chapitre présente une discussion globale du travail effectué dans le cadre de cette recherche avec un développement détaillé sur ses apports, ses limitations et différentes ouvertures d'approfondissement du sujet. Le document conclue sur un résumé de ces cinq chapitres incluant une discussion brève à propos des résultats obtenus ainsi qu'un résumé global des apports de ce travail, ses limitations et diverses directions de travaux futurs.

CHAPITRE I

PRÉREQUIS

1.1 LE JEU VIDÉO

Le jeu vidéo est un logiciel de divertissement permettant à un utilisateur d'interagir avec un environnement numérique qui réagit en fonction de règles et de conditions spécifiques à chaque jeu. Les jeux vidéo sont accessibles sur diverses plateformes telles que les consoles de jeu, les ordinateurs personnels, les smartphones et les tablettes.

Les jeux vidéo se distinguent par leur capacité à offrir des expériences immersives et diversifiées, allant des jeux de rôle aux simulations, en passant par les jeux de tir à la première personne, les jeux de stratégie en temps réel et les jeux de sport. Cette diversité de genres permet d'atteindre un public varié et de répondre à une multitude de préférences et d'attentes.

1.1.1 CARACTÉRISTIQUES PRINCIPALES

INTERACTIVITÉ

L'interactivité est une composante essentielle dans la définition d'un jeu vidéo, caractérisée par la capacité des joueurs à influencer et à interagir avec le monde virtuel et les éléments du jeu en temps réel [4]. Cette interaction dynamique distingue les jeux vidéo des formes de médias passives comme les films ou les livres. Selon Salen et Zimmerman [3], l'interactivité est le processus par lequel les actions des joueurs affectent directement les réponses du système de jeu, créant ainsi une boucle de rétroaction continue entre le joueur et

4. Temps réel : qui décrit une réponse fiable aux entrées dans un intervalle de temps donné

le jeu. Cela implique non seulement des commandes directes telles que le mouvement et les actions du personnage, mais aussi des décisions stratégiques et des choix narratifs qui peuvent altérer l'évolution du jeu. Juul [4] souligne que l'interactivité est ce qui permet aux joueurs de s'engager de manière active et immersive, en donnant un sens de contrôle et d'influence sur l'univers du jeu. Ce niveau d'engagement est crucial pour l'expérience utilisateur, rendant les jeux vidéo particulièrement immersifs et captivants [5]. Ainsi, l'interactivité ne se limite pas à la simple manipulation d'objets virtuels, mais englobe un spectre plus large d'engagement et de participation active, qui est fondamental pour l'immersion et le plaisir du joueur.

RÈGLES ET OBJECTIFS

Les règles et les objectifs constituent des éléments fondamentaux dans la définition des jeux vidéo, structurant l'expérience de jeu et orientant le comportement des joueurs. Les règles sont les contraintes et les directives prédéfinies qui régissent les interactions possibles dans le jeu, définissant ce que les joueurs peuvent et ne peuvent pas faire. Selon Salen et Zimmerman [3], les règles créent un cadre de référence qui rend le jeu compréhensible et jouable, en fournissant une structure au sein de laquelle les joueurs peuvent opérer et prendre des décisions stratégiques. Les objectifs, quant à eux, fournissent une direction et un but aux joueurs, motivant leur engagement et leurs actions dans le jeu. Juul [4] soutient que les objectifs sont essentiels pour donner un sens de progression et d'accomplissement, en offrant des défis à surmonter et des récompenses à obtenir. Les objectifs peuvent varier de simples tâches à accomplir à des missions complexes, mais leur présence est cruciale pour maintenir l'intérêt et la motivation des joueurs [5]. Ensemble, les règles et les objectifs créent une expérience de jeu cohérente et engageante, en fournissant des mécanismes clairs pour l'interaction et en établissant des buts à atteindre, ce qui est essentiel pour le plaisir et la satisfaction des joueurs.

GRAPHISMES ET AUDIO

Les graphismes et l'audio jouent un rôle crucial dans la définition et l'expérience des jeux vidéo, contribuant de manière significative à l'immersion et à l'engagement des joueurs. Les graphismes, qui incluent les visuels et les animations, sont responsables de la représentation visuelle du monde du jeu, des personnages et des objets. Ils créent l'atmosphère et l'esthétique du jeu, influençant fortement l'expérience sensorielle et émotionnelle du joueur [6]. Des graphismes de haute qualité peuvent rendre un jeu visuellement attrayant et immersif, tandis qu'une direction artistique cohérente peut renforcer le thème et l'identité du jeu [5]. L'audio, comprenant la musique, les effets sonores et les dialogues, est également essentiel pour l'immersion. La musique de fond peut établir le ton et l'ambiance, les effets sonores peuvent fournir un retour d'information important sur les actions du joueur et les événements du jeu, et les dialogues peuvent enrichir la narration et le développement des personnages [7]. Ensemble, les graphismes et l'audio forment un environnement multimédia qui peut transformer l'expérience de jeu en une expérience riche et captivante, en stimulant les sens et en plongeant les joueurs dans l'univers du jeu.

NARRATION

La narration dans le cadre des jeux vidéo se réfère à l'utilisation de structures narratives pour créer des histoires immersives et engageantes au sein de l'expérience de jeu. Contrairement à d'autres formes de médias où la narration est généralement linéaire et passive, la narration dans les jeux vidéo est souvent interactive et peut varier en fonction des actions et des décisions des joueurs [8]. Les éléments narratifs dans les jeux vidéo peuvent inclure des dialogues, des séquences cinématiques, des journaux de bord et des quêtes, tous conçus pour immerger les joueurs dans l'histoire et les inciter à progresser dans le jeu [9]. Selon Murray

[10], la capacité des jeux vidéo à offrir des récits interactifs permet aux joueurs de participer activement à l'évolution de l'histoire, ce qui renforce leur engagement émotionnel et leur connexion avec le monde du jeu. Cette forme unique de narration, parfois appelée « narration émergente », se distingue par sa capacité à adapter l'histoire en fonction des choix et des actions des joueurs, créant ainsi des expériences personnalisées et dynamiques. En somme, la narration dans les jeux vidéo est une composante essentielle qui enrichit l'expérience de jeu en ajoutant profondeur, contexte et motivation, tout en permettant une interactivité qui transcende les limites des médias traditionnels.

1.1.2 HISTORIQUE ET ÉVOLUTION

Le début du jeu vidéo remonte aux années 1950 et 1960, avec les premières expériences en informatique interactive. L'un des premiers exemples est « Tennis for Two, » créé par William Higinbotham en 1958, souvent considéré comme l'un des premiers jeux vidéo interactifs [11]. Cependant, c'est avec la commercialisation de « Pong » par Atari en 1972 que les jeux vidéo ont commencé à captiver un large public, marquant le début de l'industrie vidéoludique moderne [12]. Dans les décennies suivantes, les jeux vidéo ont connu une évolution rapide avec l'introduction des consoles de salon et des jeux d'arcade dans les années 1970 et 1980. Les avancées technologiques ont permis le développement de graphismes plus sophistiqués, de *gameplay* plus complexes et de narratives plus riches. Les années 1990 ont vu l'émergence des jeux en trois dimensions et des jeux en ligne, ouvrant la voie à des expériences de jeu multijoueurs et immersives [13]. Aujourd'hui, les jeux vidéo sont une forme de média dominante, intégrant des technologies avancées telles que la réalité virtuelle et augmentée, et continuent d'évoluer avec les progrès technologiques et les changements dans les préférences des joueurs. Cette progression rapide témoigne de la capacité des jeux vidéo à se réinventer et à s'adapter, tout en restant une forme d'art et de divertissement influente et populaire.

1.1.3 IMPACT ET RÉCEPTION

Les jeux vidéo ont un impact culturel, social et économique considérable. Ils sont utilisés non seulement pour le divertissement, mais aussi pour l'éducation, la formation professionnelle, et même la thérapie. L'industrie du jeu vidéo est une des industries de divertissement les plus lucratives, dépassant souvent les revenus combinés des industries du cinéma et de la musique [5].

1.2 LE JEU VIDÉO MULTIJOUEURS

Le jeu vidéo étant par définition très diversifié dans sa conception et les expériences qu'il offre au joueur, certains proposent aux utilisateurs de jouer à plusieurs dans le même environnement numérique. Les jeux proposant cette option sont des jeux « multijoueurs ».

1.2.1 CARACTÉRISTIQUES PRINCIPALES

Que ce soit de manière synchrone ou asynchrone, les différents joueurs de la même partie s'échangent constamment des informations pour mettre à jour leur environnement numérique en fonction des interactions de chaque utilisateur.

1.2.2 LE MULTIJOUEURS LOCAL

Dans les débuts du jeu vidéo, le multijoueurs a toujours été utilisé sous la forme d'un jeu qui regroupe plusieurs joueurs sur la même caméra de rendu graphique (voir Figure 1.1). Les contrôleurs [6] de chaque joueur sont connectés au même système physique, souvent une console de salon ou un ordinateur, et permettent d'interagir et de manipuler l'environnement

5. https://fr.statista.com/infographie/22382/chiffre-affaires-mondial-industrie-du-divertissement-jeux-video-cinema-musique-enregistree/, Saisi le 9 juillet 2025

6. manette de jeu, télécommande, capteur, etc.

numérique du jeu. Chaque joueur a, la plupart du temps, un contrôleur qui permet de faire bouger son propre personnage comme dans les jeux de combats sur bornes d'arcade ou dans les jeux de salon.

FIGURE 1.1 : Multijoueurs local 4 joueurs sur la même caméra. (Royal Vermin, 2025)

Le multijoueurs a aussi souvent été exploité sous la forme de l'écran scindé (Voir Figure 1.2). Cela consiste à séparer l'affichage du jeu en plusieurs parties, une pour chaque joueur, comme dans les jeux de courses par exemple. Dans ces deux cas, les joueurs se doivent d'être présents en même temps devant le même écran afin de profiter de cette expérience, bien que de nos jours, il soit possible de jouer à ce genre de jeu en ligne via le streaming d'écran.

FIGURE 1.2 : Multijoueurs local 2 joueurs en écran scindé. (After School, 2023)

1.2.3 LE MULTIJOUEURS EN LIGNE

Aujourd'hui le jeu multijoueur est un standard, quasiment tous les plus gros titres de l'industrie incluent la possibilité de jouer en multijoueur si ce n'est directement leur expérience principale. Cependant, au lieu de proposer un multijoueur local comme vu au-dessus, le standard actuel est d'offrir une expérience en ligne. Un jeu multijoueur en ligne propose la même expérience que le local excepté le fait que chaque utilisateur joue sur sa propre machine et son propre jeu au travers d'un réseau.

Cette expérience en ligne implique un transfert de données entre chaque utilisateur qu'on appellera désormais « clients ». Dans le processus de programmation d'un jeu vidéo multijoueur en ligne, tout l'environnement du jeu n'est évidemment pas constamment envoyé à travers le réseau. Les variables, propriétés et fonctions qui doivent être envoyées sur le réseau lorsqu'elles sont modifiées ou exécutées sont dites répliquées. Souvent la position, la vie et les attaques vont être répliquées sur le réseau afin que tous les autres clients puissent en être

informés. Tout ce qui concerne et impacte uniquement un seul client, comme par exemple des effets visuels cosmétiques, des animations, des effets sonores que seul lui est censé voir ne sont pas répliqués et sont dits « local ».

1.2.4 PRINCIPALES ARCHITECTURES RÉSEAUX

Dans le domaine du jeu vidéo en ligne, il y a trois principales architectures réseaux permettant la communication entre chaque client : serveur dédié, serveur d'écoute et pair-à-pair. Dans le cadre de cette recherche, nous allons nous concentrer uniquement sur les serveurs dédiés de par leur omniprésence dans le secteur et de leur impact écologique pour tenter de le réduire. S'il s'avère toutefois que les résultats de cette recherche sont concluants dans le cadre du serveur dédié, l'architecture ECS pourrait également venir s'appliquer aux autres architectures réseau afin de pouvoir offrir des expériences plus fluides et économiques aux utilisateurs.

SERVEUR DÉDIÉ

Le serveur dédié est, comme son nom l'indique, une architecture réseau qui consiste à dédier un serveur à l'hébergement d'une instance de jeu. Ce que l'on appelle « instance de jeu » est simplement un jeu qui est en train d'être exécuté. Chaque client y est constamment connecté pour échanger des données. Dans cette architecture, aucun client n'est en liaison directe les uns avec les autres, toutes les données transitent uniquement via le serveur, voir figure 1.3.

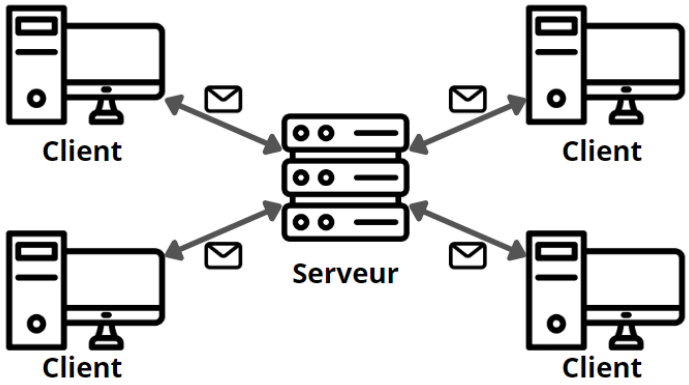

FIGURE 1.3 : Représentation de l'architecture serveur dédié.

Contrairement aux clients, le serveur de jeu ne joue pas mais s'occupe de recevoir, traiter, simuler et propager les informations répliquées que les clients lui envoient. Si un joueur souhaite effectuer une action qui impacte la perception du jeu par les autres clients, alors cette action sera simulée par le serveur sur son instance de jeu. Cela permet au serveur, après la simulation, de renvoyer l'état du monde à tous les clients afin que tous les joueurs aient une vision de leur environnement virtuel similaire.

Cette architecture est la plus populaire dans l'industrie du jeu vidéo car elle permet à l'instance principale du jeu, le serveur, d'avoir une complète autorité sur ce que les clients envoient comme information. Cet avantage permet d'homogénéiser la perception du monde auprès de tous les clients car ils ne voient que ce que le serveur leur montre, le second bénéfice de cet avantage est d'éviter, dans une certaine mesure, la triche liée à l'envoi de données : rien ne pourrait empêcher un client de truquer les informations qu'il envoie afin de tricher sur sa position, son nombre de points de vie, ses actions, etc.. Le fait que ces informations passent obligatoirement par le serveur permet à celui-ci de traiter ces données afin de vérifier si elles ne sont pas falsifiées.

Le principal défaut de ce système est l'hébergement des serveurs, avec des centaines de milliers de joueurs actifs simultanément à travers le monde, chaque partie de jeu doit avoir un serveur attitré tournant constamment et capable de gérer tous les clients connectés tout en offrant une expérience fluide. En effet, plus un jeu est coûteux en performance, plus les serveurs dédiés à ce jeu ont besoin de ressources pour héberger des parties.

SERVEUR D'ÉCOUTE

Le serveur d'écoute est une architecture similaire au serveur dédié à l'exception que celui-ci est aussi un client, voir figure 1.4. En effet, le principe du serveur d'écoute réside de=ans le fait qu'un des clients héberge la partie et s'occupe de faire le transfert et traitement des données entre tous les clients y compris lui-même. Cette architecture est aussi très populaire car elle évite que des serveurs dédiés soient toujours en train de fonctionner en attendant de nouveaux joueurs. Cela fait de grandes économies pour les développeurs puisque ce sont les clients qui s'occupent d'héberger.

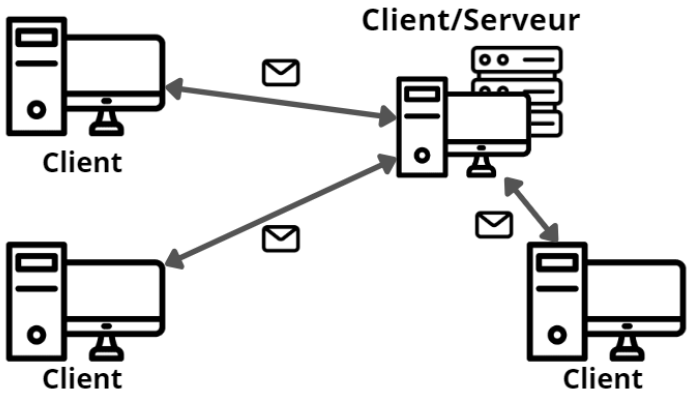

FIGURE 1.4 : Représentation de l'architecture serveur d'écoute.

Le défaut principal de ce modèle est que tous les clients doivent faire confiance à celui qui héberge la partie, c'est pourquoi cette architecture est souvent utilisée pour les jeux coopératifs et non compétitifs. Le second défaut important de ce modèle est que si l'hôte de la partie quitte le jeu, la partie se termine pour tous les clients. Certains jeux essaient de pallier à ce problème en faisant une ré-allocation dynamique de l'hôte lorsque celui-ci se déconnecte mais c'est un processus difficile, non adapté à l'architecture et très peu fiable. Le dernier défaut qui peut arriver, dépendamment de la manière dont le jeu est développé, est la perte en fluidité du jeu de l'hôte. Puisque le joueur hébergeant la partie doit traiter toutes les interactions entre les joueurs, celles-ci peuvent parfois être plus coûteuses que ce que la machine de l'hôte est capable de simuler à des cadences de rafraîchissement raisonnables [7]. Ce manque de fluidité peut impliquer un inconfort dans l'expérience de jeu de l'hébergeur mais aussi chez les clients, pour qui le traitement de leurs actions à travers le réseau sera ralenti, voir inexacte.

Pair-à-Pair (P2P)

Le P2P consiste à ce que chaque client se partage les uns entre les autres les données, voir figure 1.5. Dans cette architecture, les données passent rarement par des intermédiaires tels que des serveurs auxiliaires. Tous les clients connectés au réseau jouent le rôle de « nœuds », ils sont à la fois client et serveur et peuvent en effet s'envoyer directement les informations ce qui simplifie le trafic. Ce système est dynamique et les nœuds changent constamment au cours de l'évolution du réseau. Ce genre d'architecture se fait de plus en plus rare dans le domaine du jeu vidéo car cela implique que chaque utilisateur fasse confiance à l'ensemble des entités connectées au réseau puisqu'il n'y a aucun système administratif centralisé tel qu'un serveur

7. 30-60Hz

dédié. De plus, l'irrégularité de la connexion et des performances de chaque client peut avoir un impact direct sur l'expérience de tous les utilisateurs du réseau.

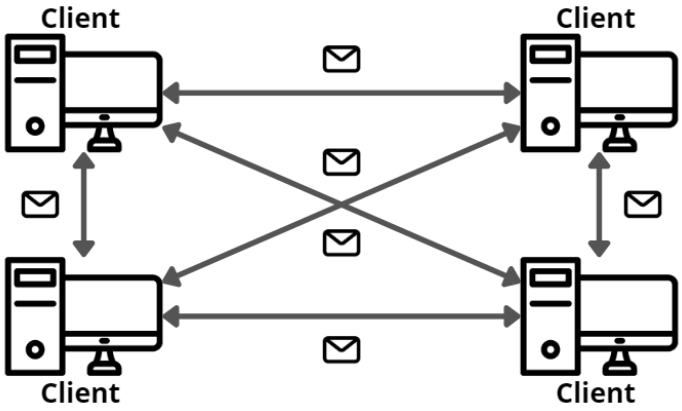

FIGURE 1.5 : Représentation de l'architecture Pair-à-Pair.

1.3 L'ARCHITECTURE PROGRAMMATION ORIENTÉE OBJET

La POO est un paradigme de programmation qui utilise des « objets » pour concevoir des applications et des logiciels. Ces objets peuvent contenir des données, sous la forme de champs (souvent appelés attributs ou propriétés), et du code, sous la forme de procédures (souvent appelées méthodes). La POO est basée sur plusieurs concepts clés : encapsulation, abstraction, héritage et polymorphisme.

1.3.1 CONCEPTS PRINCIPAUX

1. **Encapsulation** : Ce principe consiste à regrouper les données (attributs) et les méthodes qui manipulent ces données au sein d'une même unité appelée « classe ». L'encapsulation permet de cacher l'implémentation interne des objets et de ne rendre accessibles que

les méthodes nécessaires à leur utilisation, assurant ainsi une meilleure modularité et protection des données [14].

2. **Abstraction** : L'abstraction permet de simplifier les systèmes complexes en modélisant les classes de manière à représenter uniquement les caractéristiques essentielles tout en cachant les détails d'implémentation. Les objets sont définis par des interfaces qui précisent ce qu'ils peuvent faire sans indiquer comment [15].
3. **Héritage** : L'héritage est un mécanisme par lequel une classe (dite classe dérivée ou sous-classe) peut hériter des attributs et des méthodes d'une autre classe (dite classe de base ou super-classe). Cela permet de réutiliser le code existant et de créer des hiérarchies de classes [16].
4. **Polymorphisme** : Le polymorphisme permet à des objets de différentes classes d'être traités comme des objets d'une même classe de base commune. Il permet d'utiliser des méthodes dans des contextes variés, grâce à la surcharge (plusieurs méthodes avec le même nom mais des signatures différentes) et à la redéfinition (une méthode dans une sous-classe qui remplace celle de la super-classe) [17].

1.3.2 HISTORIQUE ET ÉVOLUTION

La POO a été conceptualisée dans les années 1960 et 1970, principalement grâce aux travaux sur le langage de programmation Simula, considéré comme le premier langage orienté objet. Plus tard, Smalltalk, développé dans les années 1970 par Alan Kay et ses collègues au Xerox PARC, a popularisé le concept d'objets et influencé de nombreux langages modernes comme C++, Java et Python [18].

1.3.3 AVANTAGES ET INCONVÉNIENTS

La Programmation Orientée Objet (POO) présente à la fois des avantages et des inconvénients intrinsèques, souvent liés aux mêmes caractéristiques fondamentales de ce paradigme. Ces qualités ainsi que ces défauts sont décrits dans les deux sections ci-dessous.

AVANTAGES

La POO favorise la ré-utilisabilité du code grâce à la modularité des classes et des objets, permettant leur utilisation dans différents programmes et projets [19]. Cette modularité contribue également à l'extensibilité des systèmes, facilitant l'ajout de nouvelles fonctionnalités par l'intermédiaire de classes dérivées sans modifier le code existant de manière significative [20]. De plus, le principe des classes et des objets est intuitif pour de nombreux développeurs, améliorant la compréhension et la maintenance du code [19].

INCONVÉNIENTS

Cependant, ces mêmes caractéristiques peuvent aussi engendrer des inconvénients. Par exemple, l'héritage multiple peut complexifier la maintenance et la compréhension du code, en raison des interactions compliquées entre les classes et des ambiguïtés potentielles [21]. La POO peut également encourager la création de hiérarchies de classes profondes, rendant le code rigide et difficile à modifier sans affecter d'autres parties du système [22]. Cette complexité peut conduire à une prolifération de classes, compliquant la structure globale du programme et rendant le débogage plus ardu.

En ce qui concerne la programmation concurrente, la POO ne la favorise pas naturellement. L'implémentation de modèles de concurrence dans un cadre orienté objet peut être

complexe et sujette à des erreurs, notamment en raison des effets de bord et des dépendances implicites entre les objets [23]. De plus, la surutilisation des objets pour tout type de données peut entraîner une surcharge mémoire et une diminution des performances, surtout dans les systèmes où l'efficacité est cruciale [24]. Ces inconvénients soulignent que les avantages de la POO peuvent également se transformer en limitations significatives, en particulier dans des contextes où les performances et la simplicité sont essentielles. Une des caractéristiques inhérentes de la POO ayant un impact important sur les performances est la V-Table.

1.3.4 LA V-TABLE

Une V-table (ou table des méthodes virtuelles) est une structure de données utilisée pour implémenter le polymorphisme dynamique. La V-table est une table de pointeurs vers les méthodes virtuelles des objets. Chaque classe possédant des méthodes virtuelles possède une V-table, et chaque instance de la classe contient un pointeur vers cette V-table. Lorsque une méthode virtuelle est appelée sur un objet, le programme utilise ce pointeur pour accéder à la V-table et obtenir l'adresse de la méthode à exécuter. Cela permet de résoudre les appels de méthodes à l'exécution plutôt qu'à la compilation, permettant ainsi le polymorphisme .

Cependant, l'utilisation de V-tables peut entraîner des problèmes de performance. Les appels de méthodes virtuelles via une V-table nécessitent un accès indirect en mémoire, ce qui est moins efficace que les appels directs de fonctions. Chaque appel de méthode virtuelle introduit une indirection supplémentaire, ce qui peut entraîner des ratés de cache, ce qui signifie que le processeur essaie d'accéder à des données qui ne sont pas présentes dans le cache et doit alors les recharger depuis la mémoire RAM ayant un temps de latence extrêmement conséquent.

En outre, les optimisations du compilateur, telles que l'*inlining* [8], sont plus difficiles à appliquer aux méthodes virtuelles, ce qui limite les opportunités d'optimisation. De plus, dans des contextes où les performances et l'efficacité de la mémoire sont critiques, comme les systèmes embarqués ou les jeux vidéo, ces inefficacités peuvent s'accumuler et avoir un impact significatif sur les performances globales du système.

1.4 L'ARCHITECTURE PROGRAMMATION ORIENTÉE DONNÉE

La programmation orientée donnée (POD) est un paradigme de programmation qui met l'accent sur la définition brute de ce que devrait faire un programme : recevoir des données en entrée, les transformer et les renvoyer [25]. Elle se concentre sur la structuration et la manipulation des données plutôt que sur les objets et les méthodes comme dans la programmation orientée objet. Dans ce paradigme, les données sont considérées comme l'élément central, et les algorithmes sont conçus pour traiter ces données de manière efficace. La Conception Orientée Données (COD) est une méthode de conception peu enseignée dans le milieu académique et pourtant très répandue dans le milieu des jeux à gros budgets [26].

De nos jours, la COD n'a toujours pas une définition précise unanime de par le manque de littérature académique à son sujet, ce patron de conception est surtout connu dans le milieu restreint des studios de jeux à gros budget et sa définition diffère souvent selon les développeurs [26]. Là où tous semblent être en accord sur ce qu'est la COD, c'est sur la prise de décision basée uniquement sur toutes sortes de données d'entrée tout en mettant un accent sur les performances au sens large [27].

8. Technique d'optimisation de code où le corps d'une fonction est directement inséré dans le code où la fonction est appelée.

1.4.1 CONCEPTS PRINCIPAUX

1. **Structuration des Données** : En POD, les données sont structurées de manière optimale pour leur traitement. Les structures de données sont choisies en fonction de leur capacité à être manipulées efficacement par les algorithmes utilisés [28].
2. **Séparation des Données et des Comportements** : Contrairement à la POO où les données et les comportements sont encapsulés dans des objets, la POD sépare strictement les données des fonctions qui les manipulent[29].
3. **Accès Séquentiel et Localité** : La POD favorise l'accès séquentiel aux données et optimise la localité des références, améliorant ainsi les performances en exploitant mieux les caches processeur [30].
4. **Immutabilité** : Les données sont souvent traitées comme immuables, ce qui signifie qu'elles ne sont pas modifiées après leur création. Cela permet de simplifier la compréhension et la prédiction du comportement des programmes [31].

1.4.2 HISTORIQUE ET ÉVOLUTION

La POD trouve ses racines dans les premières méthodes de programmation et les structures de données introduites dans les années 1960 et 1970. Le livre de Niklaus Wirth, *Algorithms + Data Structures = Programs* [28], est l'une des premières œuvres majeures à mettre en avant l'importance de la structuration des données dans le développement de logiciels efficaces.

Avec l'évolution des architectures matérielles et l'augmentation des volumes de données à traiter, la POD a gagné en importance. Les systèmes massivement parallèles et les applications de traitement de données à grande échelle, comme les bases de données et les moteurs de recherche, ont adopté ce paradigme pour optimiser les performances.

1.4.3 AVANTAGES ET INCONVÉNIENTS

La POD présente plusieurs avantages, en particulier en ce qui concerne les performances. L'un des principaux avantages de la POD est son efficacité en termes de gestion du cache processeur. En structurant les données de manière contiguë en mémoire, la POD minimise les ratés de cache, ce qui améliore significativement les performances d'accès aux données. Cette approche favorise également le traitement par lots et l'optimisation des boucles de traitement, réduisant ainsi le coût des accès mémoire par rapport à la programmation orientée objet qui peut fragmenter la mémoire et entraîner des accès inefficaces.

Cependant, la POD présente aussi des inconvénients. Elle peut rendre le code plus difficile à lire et à maintenir, en particulier pour les développeurs habitués aux paradigmes de la POO [28]. La POD exige une discipline stricte dans la structuration des données et des algorithmes, ce qui peut augmenter la complexité du développement. De plus, l'absence d'abstractions de haut niveau comme celles offertes par la POO peut rendre certaines tâches de programmation plus ardues et moins intuitives [32].

1.5 L'ARCHITECTURE ENTITÉ-COMPOSANT-SYSTÈME

L'ECS est un patron de développement logiciel basé sur la Programmation Orientée Données (POD) et qui suit le principe de composition, contrairement à la POO traditionnelle qui utilise l'héritage. Nous explorons ses spécificités dans les sous-sections suivantes.

1.5.1 CONCEPTS PINCIPAUX

Le principe de composition de l'ECS signifie qu'une entité est définie par les composants qui lui sont associés et non par hiérarchisation. Cette caractéristique permet notamment d'affecter et de retirer dynamiquement les différents traits qui composent une entité, ce qui

fait le cœur de la flexibilité de l'ECS. Cet avantage inhérent au patron ECS est ce pourquoi il est principalement utilisé dans le domaine du jeu vidéo de par sa complexité logicielle et le nombre conséquent d'entités dynamiques à traiter.

Cependant la notion d'ECS porte souvent à confusion car celle-ci est aussi utilisée pour décrire une méthode de programmation plutôt qu'une architecture complète. En effet, malgré l'utilisation de la POO, si une classe possède des instances dynamiques permettant de définir ses traits, certains programmeurs appelleront cela ECS. Cette dénomination n'est pas forcément fausse mais elle ne fait pas référence au concept étudié dans cette recherche. Pour mieux assimiler le concept d'ECS il faut comprendre que ce que l'on définit « entité » ou n'importe quel autre aspect de l'ECS ne peut être une classe : un composant n'est pas une classe, un système n'est pas une classe. La raison principale étant qu'une classe est une caractéristique inhérente à la POO car celle-ci peut être instanciée en objet or en ECS la notion d'objet n'existe pas [33].

1.5.2 HISTORIQUE ET ÉVOLUTION

Dans le domaine du jeu vidéo, ce paradigme s'est démocratisé dans les années 2000 suite à la parution de la nouvelle génération de consoles de salons telles que la *Playstation* 3 [9] qui, de par leurs architectures processeur, ont nécessité de trouver des alternatives à la POO afin que les studios de développement puissent continuer de proposer des jeux toujours plus complexes [33] [27].

9. https://www.playstation.com/en-ca/playstation-history/2007-ps3-ps-vita/, Saisi le 9 juillet 2025

Plusieurs cadriciels utilisant ce paradigme sont aujourd'hui disponibles pour différents langages comme *Flecs* [10] et *Entt* [11] pour du C++ ou encore Bevy [12] pour du Rust. En effet, utiliser un ECS nécessite la programmation d'une architecture qui puisse correctement profiter de ses avantages en mettant en place la gestion des entités, des composants et des systèmes. Ceux-ci se démarquent les uns des autres selon leurs performances et leurs fonctionnalités telles que la manière dont on peut filtrer les entités, les systèmes d'événements intégrés, etc.

1.5.3 ARCHITECTURE

Concrètement, un cadriciel ECS fonctionne en définissant des entités auxquelles on attribue des composants. Afin de gérer ces couples entités/composants, on crée des systèmes venant modifier leurs propriétés. Ces notions clés du fonctionnement de l'ECS sont décrites ci-dessous.

ENTITÉ

Une entité, à elle seule, ne représente rien de concret. Ce n'est en réalité qu'un simple identifiant qui se limite souvent à un entier non signé. Les entités servent de support aux composants, c'est à dire que l'on va attribuer un ou plusieurs composants à une entité. Par exemple un « personnage », dans un ECS, ce ne sera qu'un nom arbitraire qu'on aura donné à notre entité numéro un ou deux ou etc. Cependant c'est lorsqu'on lui attribuera un composant que notre personnage prendra sens. Si on souhaite que notre personnage puisse avoir une position dans le monde et subir la gravité, il suffit alors de lui ajouter les composants adéquats.

10. https://github.com/SanderMertens/flecs, Saisi le 9 juillet 2025
11. https://github.com/skypjack/entt, Saisi le 9 juillet 2025
12. https://crates.io/crates/bevy_ecs, Saisi le 9 juillet 2025

C'est là une des principales forces de flexibilité d'un ECS et la raison pour laquelle c'est un patron de conception intéressant dans le domaine du jeu vidéo. Les entités sont donc une simple liste permettant d'identifier les acteurs d'un environnement qui sont définis par leurs composants.

COMPOSANT

Comme vu au-dessus, les composants sont des caractéristiques que l'on souhaite attribuer à une entité. Ce sont de simples données qui n'ont pour but que de décrire des comportements par leurs valeurs. Si notre entité « personnage » veut se déplacer, il est possible de lui ajouter un composant « movement » qui va avoir par exemple comme information une vitesse de déplacement « speed ». On peut aussi lui ajouter un composant *transform* qui aura comme information une position et une orientation. Le composant ne va pas décrire comment ces valeurs vont être utilisées, il se contente de les énumérer pour ensuite être rattaché à une entité comme présenté dans le pseudo-code suivant.

```
// déclaration du composant Movement
Movement{
    float Speed;
};

// déclaration du composant Transform
Transform{
    Vector3 Position;
    Quat Rotation;
};

// création de l'entité
Entity = CreateEntity();

// attribution des composants à l'entitié
Entity.AddComponents(
    (Mouvement, Transform){
        Mouvement = {SomeSpeed};
        Transform = {SomeVector, SomeQuat};
```

```
        }
    );
```

La répartition des informations stockées dans des composants reste à la discrétion de chaque développeur et dépend des besoins du programme. Par exemple les composants « Movement » et « Transform » de notre personnage pourraient très bien être réduis à un seul composant « Locomotion » qui combinerait les informations des deux composants comme dans le pseudo-code présenté ci-dessous. Cependant, les composants sont habituellement gardés sous leurs formes les plus simples et les plus intuitives afin de les rendre le plus versatile possible.

```
    Locomotion{
        float Speed;
        Vector3 Position;
        Quat Rotation;
    };
```

SYSTÈME

Les systèmes sont des fonctions qui vont donner vie à l'ensemble des couples entités/composants. Ils vont décrire et exécuter les règles spécifiques à un seul ou un ensemble de composants. Pour cela, les systèmes fonctionnent souvent par filtre, c'est à dire qu'ils vont prendre en paramètre un ensemble d'entités qui seront au préalable filtrées par composants. Si on souhaite programmer un système qui ajoute une mécanique de gravité, on va alors récupérer toutes les entités qui possèdent les composants gravité et *transform*. Une fois les entités récupérées, on va pouvoir itérer à travers dans le but de modifier la position de leur *transform* après avoir calculé leur nouvelle position (voir pseudo-code ci-dessous). Les systèmes sont donc les

fonctions qui vont faire vivre le jeu et modifier dynamiquement les valeurs des composants de chaque entité.

```
ComputeGravity(Gravity& g, Transform& t){
    t.z -= g.force;
}

main() {

    // pour chaque entité avec les composants Gravity et Transform
    ecs.system<Gravity, Transform>()
    // on leur applique le système ComputeGravity
    .each(ComputeGravity);

    // le système sera executé à chaque tick
    while (ecs.progress()) { }
}
```

1.6 LE CACHE PROCESSEUR

Le cache processeur est une mémoire de petite taille mais très rapide, située à proximité ou intégrée directement dans le processeur (processeur). Sa fonction principale est de stocker temporairement des copies des données et des instructions les plus fréquemment utilisées par le processeur, afin de réduire le temps d'accès moyen aux données et d'améliorer les performances globales du système. Dans les langages de programmation compilés comme C++, ce sont les compilateurs qui tentent d'optimiser au maximum le code d'un programme afin de le rendre le plus prédictible possible pour le processeur. Cela permet ainsi de stocker dans le cache toutes les données et les instructions cœur d'un programme.

1.6.1 FONCTIONNEMENT DU CACHE PROCESSEUR

Cette section présente les aspects principaux du fonctionnement du cache processeur permettant des bases de compréhension solides. Toutefois, des notions plus avancées sur le fonctionnement du cache processeur et de son utilisation sont disponibles en annexe, voir A.

CACHE D'INSTRUCTION ET CACHE DE DONNÉES

Un cache d'instruction et un cache de données sont deux types distincts de mémoire cache intégrés dans les processeurs modernes pour améliorer l'efficacité et la rapidité d'exécution des programmes. Le cache d'instruction est une mémoire rapide utilisée pour stocker les instructions que le processeur prévoit d'exécuter prochainement. En stockant ces instructions, le cache réduit les délais associés à l'accès à la mémoire principale, permettant au processeur de récupérer et d'exécuter les instructions plus rapidement. Ce cache est crucial pour l'optimisation des boucles de programme et des branches conditionnelles, où les mêmes instructions peuvent être exécutées de manière répétitive [34].

D'un autre côté, le cache de données est une mémoire rapide dédiée au stockage des données fréquemment utilisées par les programmes en cours d'exécution. En réduisant le temps nécessaire pour accéder aux données de la mémoire principale, le cache de données améliore la performance des opérations de lecture et d'écriture. Cette optimisation est particulièrement bénéfique pour les applications gourmandes en données, telles que les calculs scientifiques et le traitement d'image, où les mêmes données peuvent être accédées de manière répétitive [35].

Les caches d'instruction et de données travaillent ensemble pour minimiser les temps de latence et maximiser le débit de traitement, en utilisant des techniques comme l'association de cache et les politiques de remplacement pour gérer efficacement le contenu du cache [36][37].

HIÉRARCHIE DES CACHES

Le cache processeur est organisé en niveaux hiérarchiques, généralement appelés L1, L2, et L3 :

1. **Cache L1** : Le plus petit et le plus rapide, directement intégré dans le noyau du processeur. Il est subdivisé en cache d'instructions (L1i) et cache de données (L1d).
2. **Cache L2** : Plus grand et légèrement moins rapide que le cache L1, il est souvent partagé entre plusieurs cœurs de processeur.
3. **Cache L3** : Le plus grand et le plus lent des caches intégrés, il est partagé par tous les cœurs du processeur.

1.6.2 AVANTAGES DU CACHE PROCESSEUR

Le cache processeur permet d'éviter des accès récurrents à la mémoire vive qui, en comparaison, est généralement 20 fois plus lente que la mémoire cache [38]. Cela fluidifie et accélère significativement les temps d'exécution des programmes. À titre indicatif, la figure 1.6 représente approximativement combien d'objets peuvent être traités en même temps par un processeur dépendamment du type de mémoire.

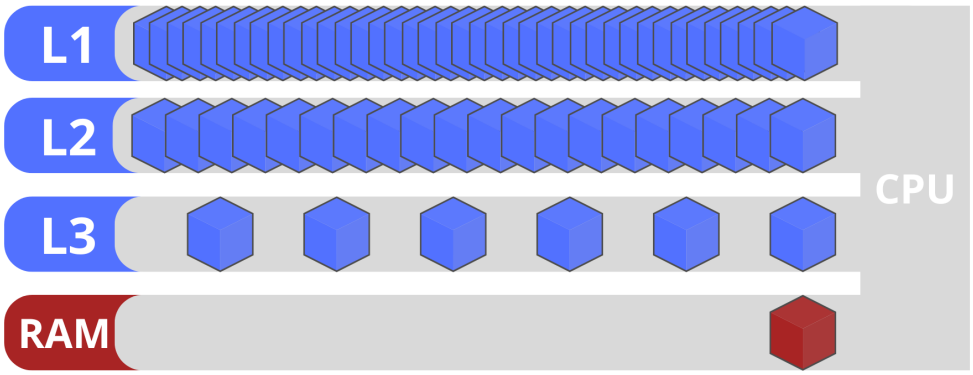

FIGURE 1.6 : Représentation de la vitesse de traitement processeur dépendamment de la mémoire.

1.6.3 PROBLÈMES ET LIMITATIONS

Si les ordinateurs actuels ont toujours besoin de RAM pour fonctionner, c'est que le cache processeur est encore une mémoire extrêmement coûteuse à produire, on parle de composants 50 à 1000 fois plus petits que la mémoire vive [38], nécessitant des machines de pointe pour les produire. Le cache processeur ne représente pas de problèmes d'un point de vue logiciel, mais connaît actuellement des limitations physiques quant à sa production.

1.7 EFFICIENCE DE L'ECS SUR L'UTILISATION DU CACHE PROCESSEUR

L'ECS, au-delà d'être un paradigme de programmation très flexible pour des logiciels aussi dynamiques et complexes que les jeux vidéo, étant basé sur de la programmation orientée données, il en possède son plus gros avantage : une gestion du cache processeur réfléchie. En

effet, les ECS utilisent deux principales méthodes d'optimisation afin d'optimiser les capacités du processeur, l'itération à travers des tableaux et la vectorisation du code.

1.7.1 ITÉRATION

Afin de rendre l'exécution du code plus efficient pour le processeur, les ECS utilisent les tableaux à mémoire contiguë en tant que piliers d'exécution. L'avantage est qu'itérer au travers de listes forme un patron d'exécution très prédictible. Une prédictibilité dont le processeur prend avantage car cela lui permet de charger dans le cache processeur l'ensemble des données qu'il suppose devoir accéder en un seul cycle d'accès à la RAM, réduisant ainsi considérablement le temps d'accès aux données.

Cette règle de conception des ECS permet d'éviter l'accès aléatoire à la mémoire comme en POO où les objets sont alloués séparément et leur accès en mémoire ne peut être prédit par le processeur. Si les accès mémoire ne peuvent être prédits, le processeur aura besoin d'effectuer de nombreux cycles d'accès à la RAM afin de charger dans le cache les données qu'il n'aura pas pu prédire et dont il aura besoin durant l'exécution du programme.

1.7.2 VECTORISATION

Itérer au travers de tableaux contigus en mémoire permet aussi au compilateur d'insérer des instructions Single Instruction, Multiple Data (SIMD). Ce genre d'instructions permet d'utiliser une seule opération sur plusieurs données avec un temps d'exécution égal à celui d'une instruction pour une seule donnée, réduisant ainsi considérablement les temps d'exécution et réduisant même la consommation énergétique jusqu'à 94% [39]. Cependant, les instructions SIMD ne peuvent être implémentées par le compilateur uniquement si le code est vectorisé. La vectorisation consiste à ce que les éléments de code sur lesquels on souhaite

faire des opérations soient contigus en mémoire. Cela signifie que dans une liste, les éléments contenant un même type de données doivent se suivre. Voir le pseudo code présenté ci-dessous.

```
{
    struct A{};
    struct B{};

    struct Pair{A;B};

    Pair Array[3] = {{A},{A,B},{B}};

    //[A]
    //[A][B]
    //   [B]
}
```

1.8 L'INFORMATIQUE VERTE

L'informatique consomme une quantité significative d'énergie et contribue à l'augmentation de l'effet de serre. Un article de 2018 dans le Journal du CNRS estime que les technologies numériques (ordinateurs, centres de données, réseaux, etc.) consomment 10 % de l'électricité mondiale : 30 % par les équipements terminaux, 30 % par les centres de données, et 40 % par les réseaux [40].

L'informatique verte, ou *green computing*, désigne l'ensemble des pratiques visant à utiliser les produits informatiques de manière efficace et responsable en tenant compte de leur impact environnemental. Ces pratiques couvrent tout le cycle de vie des produits, de la conception au recyclage. Ce concept a émergé dans les années 1990 avec le programme américain « Energy Star » [41], visant à réduire les émissions de gaz à effet de serre grâce aux économies d'énergie. Ce programme concernait déjà à l'époque les équipements informatiques tels que les moniteurs, les ordinateurs et les serveurs.

Ce mouvement a pris de l'ampleur avec la montée des préoccupations écologiques. De nombreuses recherches ont été menées, faisant de l'informatique verte une tendance majeure dans l'industrie [42]. Cette préoccupation est particulièrement importante pour les centres de données, dont la consommation énergétique a doublé entre 2000 et 2005 [43]. Ces avancées ont été encouragées par la sensibilisation du public, principalement grâce aux ONG environnementales comme le *WWF* et Greenpeace. Greenpeace publie régulièrement des rapports sur l'impact écologique du secteur informatique, contribuant ainsi à accroître la prise de conscience et à favoriser les évolutions dans ce domaine.

Cette recherche s'inscrit dans la continuité des études sur l'informatique verte en se concentrant sur la conception de logiciels, une composante clé qui joue un rôle crucial dans la réduction de l'empreinte énergétique des systèmes informatiques. Les logiciels peuvent être optimisés pour utiliser les ressources matérielles de manière plus efficace, ce qui réduit la consommation d'énergie des processeurs, de la mémoire et des disques durs [44]. Des techniques telles que l'optimisation des algorithmes, la gestion dynamique de l'énergie et la virtualisation permettent de diminuer la consommation énergétique des centres de données et des dispositifs individuels [45].

Comme vu dans les sections ci-dessus, l'utilisation des ECS offre une meilleure gestion du cache processeur ce qui permettrait, en outre, aux développeurs de logiciels de contribuer au *green computing* en adoptant des pratiques de codage efficientes et en minimisant les cycles d'instructions inutiles. Cela améliore non seulement les performances, mais réduit également l'impact environnemental [46]. En complément des initiatives matérielles telles que l'utilisation de composants à faible consommation d'énergie et le recyclage des équipements électroniques, l'utilisation de l'architecture ECS semble pouvoir contribuer au mouvement de l'informatique verte. Elle peut influer notamment sur la consommation énergétique du grand nombre de serveurs dédiés présents dans le domaine du jeu vidéo. Cette recherche ambitionne

d’analyser l’impact de cette architecture en espérant que l’ECS permette de réduire les besoins énergétiques des serveurs de jeux.

CHAPITRE II

ETAT DE L'ART

2.1 PROCESSUS DE RECHERCHE ET DE SÉLECTION

Le processus de recherche inclut tout type de documents (travaux, mémoires, thèses et volumes) et repose sur les mots clés suivants : ***Entity-Component-System* / ECS** ainsi que sa dénomination alternative ***Entity-Component-System-Architecture* / ECSA** ; Les mots clés faisant référence au jeu vidéo et aux logiciels : ***video-game(s)***, ***game(s)***, ***software*** ; Les serveurs et le multijoueurs : ***server(s)***, ***multiplayer***, ***online*** ; et enfin la consommation énergétique : ***energy***, ***power***, ***consumption***, ***greencomputing***.

Les recherches sont effectuées sur les moteurs de recherche suivants : ***Google Scholar*** pour un aperçu global de la majorité des ressources disponibles sur le sujet. **IEEEXplore** pour des documents de bonne fiabilité et le moteur **Sofia** de la bibliothèque de l'Université du Québec à Chicoutimi pour approfondir les recherches, notamment à l'aide des différents volumes traitant du sujet.

Afin d'être sélectionné, un document doit traiter de la consommation ou de l'impact énergétique des logiciels de façon à pouvoir statuer sur la consommation énergétique de diverses architectures de programmation. Bien que l'informatique verte devienne une préoccupation importante dans le domaine de la recherche informatique et que des travaux actuels voient le jour, la période de sélection des documents se doit de rester très ouverte car les travaux traitant de la consommation énergétique relative à la conception logicielle, qui plus est dans le domaine du jeu vidéo, sont limités. Si le résumé d'un document semble répondre aux critères ci-dessus, son introduction ainsi que sa conclusion sont analysées afin d'en déduire si la sélection du document est pertinente.

2.2 PRÉSENTATIONS DE TRAVAUX

Suite à l'application du processus de recherche et de sélection, cinq travaux ont été sélectionnés, étudiés et analysés afin d'en rédiger un résumé concis venant compléter l'état de l'art dans le domaine de l'optimisation énergétique logicielle. Le premier travail intitulé *Understanding the Impact of Object-Oriented Programming and Design Patterns on Energy Efficiency* [47] explore l'impact de la programmation orientée objet et des patrons de conception sur l'efficacité énergétique des logiciels. Le second travail *Entity Component System Architecture for Scalable, Modular, and Power-Efficient IoT-Brokers* [48] développe un broker en utilisant l'ECSA à des fins de comparaison énergétique par rapport à un broker populaire non ECS. Le troisième travail *An Energy-Aware Programming Approach for Mobile Application Development Guided by a Fine-Grained Energy Model* [49] analyse le coût en performance de différentes opérations énergétiques. Le quatrième travail *Game Action Based Power Management for Multiplayer Online Game* [50] propose deux approches écoénergétiques basées sur la prédiction de l'activité de jeu afin d'ajuster dynamiquement le mode de consommation de la carte réseau du client. Le dernier travail *Energy-aware software : Challenges, opportunities and strategies* [51] étudie l'impact énergétique de différents aspects de la programmation logicielle tels que le parallélisme ou encore les compilateurs utilisés.

Chaque présentation intègre une brève *introduction*, un résumé détaillé de la *méthodologie* appliqué à chaque étude, la synthétisation des *résultats*, une *conclusion* basée sur les retours des chercheurs et enfin une partie *discussion* faisant une critique objective du travail et apportant des indications quant à son apport en informations dans le cadre de cette recherche.

2.2.1 UNDERSTANDING THE IMPACT OF OBJECT-ORIENTED PROGRAMMING AND DESIGN PATTERNS ON ENERGY EFFICIENCY

Ce travail [47] explore l'impact de la programmation orientée objet et des patrons de conception sur l'efficacité énergétique des logiciels dans leur globalité. Bien que la modularité et la réutilisabilité de la POO soient reconnues, ses mécanismes d'abstraction peuvent engendrer une surcharge énergétique et de performance. Les auteurs analysent empiriquement quatre caractéristiques clés de la POO (héritage, polymorphisme, liaison dynamique, surcharge) et cinq patrons de conception (décorateur, façade, poids-mouche, prototype, patron de méthode).

MÉTHODOLOGIE

Les expériences ont été réalisées sur le système *Marcher*, un cluster de calcul haute performance financé par la *National Science Foundation*. La consommation énergétique a été mesurée en temps réel grâce à une API développée sur mesure. Cette API intègre des données issues de multiples interfaces (Intel RAPL, NVIDIA Management Library, Intel MICAccess API et une carte d'acquisition de données énergétiques). Les mesures se concentrent sur la consommation du CPU et de la DRAM, car les autres composants, comme le disque et le GPU, ont montré des variations négligeables.

Les expérimentations ont étudié l'impact énergétique des caractéristiques de la POO et des patrons de conception en comparant des programmes avec et sans les éléments ciblés.

La programmation orientée objet se distingue par plusieurs caractéristiques essentielles qui influencent directement les performances et l'utilisation des ressources. Parmi celles-ci, l'héritage joue un rôle central. Afin d'en examiner les impacts, trois types de programmes ont été développés et comparés. Dans le premier, une classe dérivée hérite de pas moins de

onze classes parentales, ce qui engendre une construction en cascade des objets. Un second programme a été conçu sans recourir à l'héritage, chaque classe étant alors indépendante. Enfin, une troisième approche alternative consiste à intégrer toutes les fonctionnalités au sein d'une seule et même classe.

Le polymorphisme, autre pilier fondamental de la POO, a également fait l'objet d'une analyse approfondie. Deux formes principales ont été explorées. D'un côté, une organisation dans laquelle une classe dérivée héritait de plusieurs classes parentales, aboutissant à une structure hiérarchique complexe. De l'autre, un mécanisme d'accès aux membres reposant sur des pointeurs dirigés vers une classe de base, enrichi par l'emploi de fonctions virtuelles associées à une liaison dynamique. La surcharge d'opérateurs, bien qu'élégante dans sa conception, soulève également des questions d'efficacité. Pour en mesurer l'impact, un programme exploitant des opérateurs surchargés tels que `+`, `-`, ou `*`, a été comparé à une alternative où ces opérateurs étaient remplacés par des fonctions explicites comme `Add()` pour l'addition.

Les patrons de conception offrent des solutions élégantes et structurées aux problématiques de développement. Le motif du décorateur, par exemple, a été analysé en confrontant une approche basée sur des classes décoratrices dynamiques à une autre reposant exclusivement sur l'héritage. Le patron de façade a été évalué sur ses bénéfices liés à l'unification des interfaces. Par ailleurs, le motif du poids-mouche a aussi été étudié pour son efficacité à mutualiser les ressources et réduire le nombre d'objets nécessaires. L'étude du patron prototype, quant à elle, se penche sur la mise en lumière des avantages et inconvénients énergétiques du clonage d'objets par rapport à leur création directe. Enfin, le patron de méthode a été exploré à travers une comparaison entre une implémentation utilisant des classes abstraites pour normaliser les algorithmes et une version s'affranchissant de toute abstraction.

Pour chaque expérimentation, le temps d'exécution et la consommation énergétique ont été mesurés. Le temps d'exécution en secondes représente la durée totale que le programme prend pour être exécuté au complet. La consommation énergétique est mesurée en Joules et représente l'énergie cumulée consommée par le CPU et la DRAM pendant l'exécution. Les résultats ont été obtenus en supprimant tout code non pertinent (par exemple, des instructions *cout* en C++) afin de minimiser les biais et de s'assurer que les mesures reflètent fidèlement l'impact des caractéristiques ou motifs testés.

RÉSULTATS

Les résultats de cette recherche offrent des indications pratiques et théoriques sur l'utilisation de la POO et des motifs de conception, en mettant en lumière leurs effets divers sur la performance et l'efficacité énergétique.

En ce qui concerne les caractéristiques de la POO, l'héritage s'est révélé avoir un impact négligeable sur les performances et la consommation énergétique. Ce mécanisme n'apparaît pas comme un levier critique dans les stratégies d'optimisation énergétique, ce qui permet de l'utiliser sans appréhension majeure.

Le polymorphisme, notamment à travers les fonctions virtuelles et la liaison dynamique, présente une situation plus nuancée. Bien que ces techniques entraînent une légère dégradation des performances, de l'ordre de 5,9 %, et une augmentation de la consommation énergétique avoisinant 6,5 %, ces effets restent acceptables dans de nombreux contextes. Toutefois, une gestion prudente s'impose, en particulier dans les systèmes où les ressources sont limitées.

La surcharge d'opérateurs, en revanche, pose davantage de problèmes. Elle entraîne des impacts significatifs sur les performances et la consommation énergétique, principalement en

raison de la création fréquente d'objets temporaires. Cette pratique, bien qu'attrayante sur le plan de la lisibilité du code, génère des coûts non négligeables en temps et en énergie, ce qui en fait un choix discutable pour des systèmes où l'efficacité est cruciale.

Quant aux motifs de conception, leurs effets varient considérablement selon les cas. Le décorateur, par exemple, se révèle particulièrement inefficace, avec une dégradation massive des performances, atteignant près de 566 %, et une augmentation de la consommation énergétique de l'ordre de 595 %. Ce motif apparaît clairement comme un choix à éviter dans les applications sensibles à l'énergie.

À l'opposé, le patron de façade offre des avantages modestes mais notables. En simplifiant les interfaces et en réduisant la complexité, il permet de légères améliorations des performances et de la consommation énergétique, le rendant pertinent pour des systèmes cherchant à combiner clarté et efficacité.

Le motif du poids-mouche, quant à lui, se distingue par des gains significatifs, atteignant environ 50 % tant en performances qu'en efficacité énergétique. Cette approche s'avère particulièrement adaptée dans des contextes où la réduction du nombre d'objets est un objectif primordial.

Le prototype, de son côté, a démontré un impact minimal sur l'énergie et les performances, tout en améliorant considérablement la réutilisabilité des objets. Cependant, il implique des coûts initiaux liés à la mise en œuvre des méthodes de clonage, un facteur à prendre en compte lors de son adoption.

Enfin, le patron de méthode, bien que légèrement pénalisant pour les performances et la consommation énergétique (respectivement 3,6 % et 0,8 %), offre des bénéfices significatifs en

termes de réutilisabilité et d'extensibilité du code. Ces qualités en font une solution avantageuse dans les projets où la maintenance et l'évolution sont des préoccupations majeures.

CONCLUSION

Ce travail répond à une lacune dans la littérature existante concernant l'impact énergétique des caractéristiques de la programmation orientée objet (POO) et des patrons de conception. Les auteurs apportent une analyse systématique des implications énergétiques des approches POO et des motifs de conception. Ils démontrent que l'application appropriée des patrons peut améliorer l'efficacité énergétique, tandis que d'autres, comme le décorateur, sont énergivores. L'étude souligne l'importance d'adopter des pratiques de développement conscientisées sur le plan énergétique, surtout pour les appareils limités en batterie. Les résultats offrent des recommandations pratiques pour choisir des motifs et des paradigmes POO adaptés à des contextes énergétiques spécifiques.

DISCUSSION

Le travail propose une étude empirique approfondie sur l'impact des caractéristiques de la programmation orientée objet (POO) et des motifs de conception sur l'efficacité énergétique des logiciels, apportant des éclairages précieux sur un domaine encore relativement peu exploré. Son approche quantitative et son analyse ciblée de motifs de conception, tels que *flyweight* et *decorator*, fournissent des résultats exploitables pour les développeurs cherchant à optimiser la consommation énergétique.

Cependant, la dépendance de l'étude à un ensemble limité de benchmarks et de scénarios, reposant sur une seule plateforme (le système *Marcher*), limite la généralisation de ces

conclusions. Bien que le travail souligne les effets négatifs de certaines fonctionnalités de la POO, comme la surcharge d'opérateurs, il n'explore pas suffisamment d'autres paradigmes, tels que la programmation fonctionnelle ou procédurale, qui auraient pu servir de base de comparaison.

Par ailleurs, le travail se concentre fortement sur les aspects techniques de la consommation énergétique, sans approfondir les compromis liés à l'utilisabilité ou à la maintenabilité, des éléments cruciaux en ingénierie logicielle. Malgré ces limites, ce travail met en lumière les arbitrages entre performance et efficacité énergétique dans la POO et ouvre la voie à des recherches futures qui pourraient étendre ces résultats à des contextes plus larges et diversifiés.

Dans le cadre de cette recherche, ce travail apporte une base comparative importante. Il souligne les limitations de la POO et justifie l'exploration de paradigmes alternatifs comme l'ECS, qui peut contourner ces contraintes grâce à une meilleure gestion des données et une réduction des accès aléatoires en mémoire. Ces *insights* permettent de positionner l'ECS comme une alternative prometteuse pour des architectures serveur plus économes en énergie.

2.2.2 ENTITY COMPONENT SYSTEM ARCHITECTURE FOR SCALABLE, MODULAR, AND POWER-EFFICIENT IOT-BROKERS

Ce travail [48] propose une nouvelle architecture basée sur le paradigme ECS pour la conception de *broker* IoT (Internet of Things) modulaires, évolutifs et économes en énergie. L'objectif principal est de relever les défis liés à la gestion de la performance, de la scalabilité et de l'efficacité énergétique dans des systèmes IoT massivement interconnectés. En s'appuyant sur un modèle de messagerie de type publication/abonnement, la solution permet une gestion simplifiée des sessions, des sujets, et des mécanismes de sécurité et de confiance, tout en

surpassant les implémentations open-source traditionnelles en termes de performance et de consommation énergétique.

MÉTHODOLOGIE

Le travail propose l'utilisation de l'architecture ECS pour améliorer la modularité, la scalabilité et l'efficacité énergétique des brokers IoT. Pour leur implémentation, les auteurs ont choisi la bibliothèque EnTT, une solution légère et moderne en C++ adaptée à l'ECS, et la bibliothèque réseau Asio pour gérer les opérations d'entrée/sortie.

Dans le cadre de la solution proposée par ce travail, l'implémentation de l'architecture ECS est structurée comme suit :

1. **Entités** : Chaque entité peut correspondre à un appareil, une session ou un sujet de communication. Ces identifiants permettent une gestion flexible et évolutive.
2. **Composants** : Les composants encapsulent les données associées aux entités. Par exemple le *SessionComponent* contient les informations nécessaires pour gérer une session client, comme un socket réseau, tandis que le *TopicComponent* stocke les données sur un sujet, telles que son nom ou sa hiérarchie.
3. **Systèmes** : Les systèmes appliquent des logiques spécifiques aux entités qui possèdent certains composants. Cette séparation permet une exécution parallèle efficace et un découplage des responsabilités, rendant l'architecture plus maintenable.

L'architecture est conçue pour des fonctionnalités cruciales des brokers IoT, à savoir la gestion des sessions, des sujets, de la bande passante, ainsi que la sécurité.

La gestion des sessions repose sur une modélisation des clients en tant qu'entités, chacune dotée d'un composant spécifique, le *SessionComponent*. Lorsqu'un client se connecte, une nouvelle entité est créée dynamiquement, et le composant correspondant est ajouté pour suivre la session de manière efficace.

Pour ce qui est de la gestion des sujets, ceux-ci, ainsi que leurs sous-sujets, sont également représentés sous forme d'entités. Afin de minimiser la bande passante utilisée, les noms de sujets, souvent longs, sont remplacés par des identifiants uniques de 4 octets. Chaque sujet est enrichi d'un composant répertoriant ses abonnés, ce qui permet une diffusion rapide et optimisée des messages à travers le système.

L'optimisation de la bande passante est au cœur de cette architecture, et la substitution des noms de sujets par des identifiants numériques joue un rôle clé. Cette approche simplifie la structure des messages en évitant les opérations de parsage complexes, tout en réduisant la surcharge liée aux métadonnées. Par exemple, un nom de sujet de 72 octets peut être remplacé par un identifiant unique de seulement 4 octets, réduisant ainsi significativement la consommation de ressources réseau.

La sécurité et la confiance sont également intégrées dans cette architecture grâce à l'ajout de composants dédiés. Un *AccessComponent* est chargé de gérer les droits d'accès, tandis qu'un *TrustComponent* permet de suivre le score de confiance des clients. Ce dernier peut être ajusté en fonction des actions des utilisateurs, par exemple lorsqu'un client tente d'accéder à un sujet sans y être autorisé. Dans de tels cas, le système peut soit modifier son score de confiance, soit interrompre sa session, renforçant ainsi la sécurité globale du broker.

Les mesures relevées lors des tests incluent les vitesses de lecture et d'écriture du broker ainsi que sa consommation énergétique. Les résultats du broker **NetMQ** développé par les

auteurs sont comparés aux mêmes mesures effectuées sur le broker open-source *Mosquitto* déjà bien établi dans le domaine.

RÉSULTATS

Les graphiques comparatifs révèlent trois aspects essentiels liés aux performances des brokers étudiés. Concernant le débit d'écriture (*Write Speed*), **NetMQ** se distingue nettement en surpassant systématiquement **Mosquitto** avec une performance supérieure d'environ **112%**. Cette différence s'explique principalement par l'absence de mécanismes de balance de charge, tels que le *message dropping*, dans NetMQ, ce qui permet une évaluation complète des coûts de traitement. De plus, la gestion optimisée des messages, rendue possible grâce à l'architecture ECS, joue un rôle important dans ces résultats.

En ce qui concerne le débit de lecture (*Read Speed*), les deux brokers affichent des performances similaires. Les variations observées demeurent mineures et sont influencées principalement par le nombre de clients connectés.

Sur le plan de la consommation énergétique (*Power Consumption*), **NetMQ** se montre significativement plus efficace, consommant environ **35% moins d'énergie** que **Mosquitto**. Cette réduction peut être attribuée à plusieurs facteurs, notamment l'optimisation du protocole de communication, qui limite les données redondantes, ainsi que l'utilisation d'identifiants de sujets de seulement quatre octets au lieu de chaînes complètes, ce qui réduit la bande passante nécessaire. En outre, les avantages inhérents à l'architecture ECS contribuent également à améliorer l'efficacité énergétique et l'évolutivité des traitements.

Ces résultats soulignent plusieurs points clés. Tout d'abord, **NetMQ** démontre une grande évolutivité en maintenant des performances stables même avec un nombre croissant de

clients. Ensuite, la réduction de la consommation d'énergie et de bande passante, essentielle dans les environnements IoT, confère à NetMQ un avantage considérable en matière d'efficacité énergétique. Enfin, l'architecture ECS se révèle être un atout majeur, permettant une transmission de données à la fois plus rapide et plus efficace, tout en soutenant les exigences de performance brute des applications modernes.

CONCLUSION

Les auteurs concluent que l'architecture ECS offre un cadre puissant pour le développement de courtiers IoT modernes. Des benchmarks supplémentaires sont suggérés pour évaluer son potentiel face à d'autres paradigmes et dans des cas d'utilisation étendus, incluant le traitement avancé des données et des mesures de sécurité renforcées.

DISCUSSION

Le travail propose une évaluation intéressante de l'architecture ECS appliquée aux brokers IoT, notamment à travers la comparaison entre NetMQ et Mosquitto. Les résultats montrent des gains substantiels en performance et en efficacité énergétique pour NetMQ, ce qui témoigne du potentiel de l'ECS dans ce contexte. Cependant, la comparaison présente certains biais méthodologiques. Tout d'abord, Mosquitto, étant un broker largement utilisé, est conçu pour supporter des charges importantes avec un grand nombre de clients et des opérations complexes. Il n'est pas optimisé spécifiquement pour une consommation énergétique minimale, ce qui fausse partiellement l'interprétation des résultats. De plus, le travail indique que NetMQ omet une étape essentielle dans le processus de réception des messages, présente dans Mosquitto.

Cette omission fausse la mesure réelle des performances, notamment en termes de débit d'écriture. Ces limitations méthodologiques rendent difficile une comparaison parfaitement équitable et suggèrent que les conclusions devraient être nuancées. Malgré ces points, l'étude met en lumière les avantages de l'ECS. En particulier sa capacité à simplifier la gestion des sessions et des sujets tout en améliorant la modularité et l'efficacité énergétique des brokers IoT. Pour affiner ces résultats, une comparaison impliquant un ensemble plus diversifié de brokers, ainsi qu'une prise en compte des spécificités d'implémentation, serait bénéfique.

Dans le cadre de cette recherche, ce travail est particulièrement pertinent car il montre l'adaptabilité de l'ECS dans un domaine parallèle au jeu vidéo. Les principes qui sous-tendent les gains d'efficacité observés dans les brokers IoT peuvent potentiellement être appliqués aux serveurs multijoueurs. Cette étude démontre que l'ECS est non seulement appropriée pour des systèmes complexes, mais qu'elle peut aussi être optimisée pour répondre à des contraintes énergétiques strictes.

2.2.3 AN ENERGY-AWARE PROGRAMMING APPROACH FOR MOBILE APPLICATION DEVELOPMENT GUIDED BY A FINE-GRAINED ENERGY MODEL

Ce travail[49] propose une approche de programmation écoénergétique destinée aux développeurs d'applications mobiles. Cette approche est guidée par un modèle énergétique finement granulé, ce qui signifie que l'énergie consommée par le code est analysée à un niveau très détaillé. L'objectif est de permettre aux programmeurs d'analyser et d'optimiser l'énergie consommée par leur code, en se concentrant sur des blocs coûteux identifiés par un modèle basé sur des « opérations énergétiques ». L'approche est testée sur un moteur de jeu Android codé en Java, avec des gains énergétiques allant de 6,4 % à 50,2 % dans différents scénarios.

MÉTHODOLOGIE

L'approche repose sur une méthodologie structurée en trois étapes principales : *modélisation énergétique*, *analyse énergétique* et *optimisation du code*. Tout d'abord, la *modélisation énergétique* se base sur la création d'un modèle au niveau du code source, en s'appuyant sur les opérations énergétiques. Ces opérations correspondent aux unités élémentaires du programme qui consomment de l'énergie, telles que les invocations de méthodes, les opérations arithmétiques ou les comparaisons logiques. Le modèle établit un lien entre ces opérations et leur coût énergétique, évalué en fonction de leur fréquence d'exécution, permettant ainsi une analyse précise de leur impact.

Ensuite, l'*analyse énergétique* utilise ce modèle pour attribuer la consommation d'énergie aux différents blocs de code, en fonction des opérations qu'ils contiennent. Cette approche fine identifie les blocs les plus énergivores, qui sont ensuite classés en fonction de leur coût énergétique total. Celui-ci est calculé comme le produit du coût énergétique par opération et du nombre d'exécutions de chaque opération.

Enfin, l'*optimisation du code* cible spécifiquement les blocs les plus coûteux. Cette phase inclut plusieurs stratégies, notamment la réduction des opérations coûteuses, comme le remplacement d'invocations fréquentes de méthodes par des implémentations intégrées (*method inlining*), la réorganisation du code pour déplacer des calculs invariants hors des boucles (*loop-invariant code motion*), et l'utilisation de fonctions de bibliothèque natives optimisées, en substitution aux implémentations en code source.

Les expérimentations ont été menées sur une carte Odroid-XU+E dotée de deux processeurs ARM Cortex-A15 cadencés à 1,1 GHz. Ce dispositif dispose d'un outil intégré de surveillance énergétique, mesurant en continu la tension et le courant des processeurs à une

fréquence de 30 Hz. Le logiciel analysé, *Cocos2d-Android*, est un moteur de jeu utilisé pour des applications interactives, notamment des jeux ou des simulations.

Pour tester l'approche, trois scénarios représentatifs ont été définis : dans ***Click & Move***, un sprite se déplace en réponse aux clics de l'utilisateur ; dans ***Orbit***, le sprite effectue une rotation en trois dimensions avec un arrière-plan animé ; et dans ***Waves***, le sprite oscille en taille tandis que l'arrière-plan imite une vague en mouvement. Afin d'assurer la reproductibilité des tests, les séquences d'entrée utilisateur ont été simulées à l'aide de l'outil *Android Debug Bridge* (ADB). Parallèlement, un graphe de flux de contrôle du code source a été exploité pour varier systématiquement l'exécution des blocs et affiner la précision du modèle énergétique.

Ce modèle décompose la consommation énergétique totale (E) en trois composantes distinctes (voir l'équation ci-dessous). La première, associée aux opérations énergétiques (EnergyOps), est calculée en multipliant le coût énergétique de chaque opération (Cost_{opi}) par son nombre d'exécutions ($N_e(opi)$). La deuxième composante correspond aux fonctions de bibliothèque (LibFuncs), dont la consommation est déterminée de manière similaire. Enfin, la troisième composante, appelée *Idle Cost*, représente l'énergie consommée par le système lorsqu'il est au repos. Les coûts énergétiques sont estimés à l'aide d'une analyse de régression basée sur des cas d'exécution variés, offrant ainsi une évaluation détaillée et fidèle des besoins énergétiques du logiciel.

$$E = \sum_{opi \in \text{EnergyOps}} \text{Cost}_{opi} \cdot N_e(opi) + \sum_{func_i \in \text{LibFuncs}} \text{Cost}_{func_i} \cdot N_e(func_i) + \text{Idle Cost},$$

RÉSULTATS

Parmi les 120 opérations analysées, les invocations de méthodes et les références à des tableaux se révèlent être les plus énergivores. Ces résultats mettent en évidence la pertinence d'une analyse finement granulée pour guider les efforts d'optimisation. Les résultats obtenus pour chacun des scénarios révèlent des gains énergétiques significatifs, détaillés dans les deux paragraphes suivants.

Dans le scénario ***Click & Move***, où le sprite se déplace en fonction des clics de l'utilisateur, l'énergie consommée a été réduite grâce à plusieurs optimisations ciblées. La combinaison de blocs conditionnels a diminué le nombre de comparaisons et d'instructions de contrôle, tandis que le *method inlining* a permis d'intégrer directement une méthode interne fréquemment invoquée dans le code appelant, évitant ainsi les appels répétés. Enfin, la réorganisation des boucles a consisté à déplacer les calculs invariants hors de celles-ci, supprimant leur réévaluation inutile. Ces ajustements ont conduit à une baisse de 6,4 % de la consommation énergétique, sans affecter les fonctionnalités du programme.

Dans le scénario ***Orbit***, où le sprite tourne sur un arrière-plan animé, des gains énergétiques significatifs ont également été enregistrés. L'utilisation de fonctions de bibliothèque natives a permis de remplacer une implémentation en Java interprétée, entraînant une réduction spectaculaire de 50,2 % des coûts énergétiques. Par ailleurs, le déroulement des boucles (*loop unrolling*), qui consiste à dupliquer le corps d'une boucle, a réduit le nombre de comparaisons et de sauts conditionnels nécessaires. Ces améliorations mettent en lumière l'importance des optimisations natives offertes par les bibliothèques.

Enfin, dans le scénario ***Waves***, où le sprite et l'arrière-plan oscillent, des optimisations similaires ont été appliquées pour obtenir des gains énergétiques significatifs. Le *method*

inlining, combiné au déplacement de code, a permis d'intégrer directement les méthodes fréquemment invoquées dans des boucles imbriquées, éliminant ainsi les coûts d'appel récurrents. Comme dans le scénario précédent, l'utilisation intensive des fonctions de bibliothèque a également remplacé les calculs interprétés, augmentant ainsi la performance globale. Ces améliorations ont conduit à une réduction de 27,7 % de la consommation énergétique du processeur, tout en renforçant la fluidité des animations. Ces résultats illustrent l'efficacité des techniques d'optimisation ciblées pour améliorer les performances énergétiques, en particulier dans des applications interactives et gourmandes en ressources comme les jeux.

Scénario	**Optimisations principales**	**Réduction énergétique**
Click & Move	Combinaison de conditions, *Inlining*, boucles optimisées	6,4 %
Orbit	Fonctions de bibliothèque, déroulement de boucles	50,2 %
Waves	*Inlining*, réorganisation du code, fonctions de bibliothèque	27,7 %

TABLEAU 2.1 : Résumé des optimisations et gains énergétiques par scénario.

Outre les gains énergétiques, l'approche a également amélioré les performances des applications. Par exemple, dans le scénario *Waves*, le taux de rafraîchissement des animations a augmenté de 36 Hz à 60 Hz, offrant une meilleure expérience utilisateur sans surcharger les ressources du processeur.

Ces résultats démontrent que l'approche proposée permet d'identifier et d'exploiter efficacement des opportunités d'optimisation énergétique dans des applications interactives. Les gains obtenus illustrent l'importance d'une analyse détaillée au niveau du code source pour maximiser l'efficacité énergétique tout en préservant ou en améliorant les performances globales du système.

CONCLUSION

Cette approche écoénergétique aide les développeurs à écrire du code plus performant et économe en énergie, tout en offrant une meilleure expérience utilisateur. Le travail souligne la nécessité d'adopter des pratiques de développement conscientes de l'énergie dès la phase de conception des logiciels. Les travaux futurs incluront l'application de cette méthode à d'autres domaines et architectures logicielles.

DISCUSSION

Le travail propose une approche apportant des bénéfices significatifs, notamment en permettant aux développeurs de cibler des blocs coûteux en énergie et de réaliser des gains d'énergie impressionnants, atteignant jusqu'à 50,2 % dans certains scénarios. Cependant, certaines limites méritent d'être soulignées. La méthodologie repose principalement sur des cas d'exécution spécifiques et sur un moteur de jeu unique, ce qui limite la généralisation des résultats à d'autres types d'applications ou d'environnements. De plus, bien que l'accent soit mis sur l'efficacité énergétique, le travail ne discute pas suffisamment des compromis potentiels, tels que la complexité accrue du code ou la perte de lisibilité due à des optimisations comme l'inlining de méthodes. Par ailleurs, l'utilisation d'une plateforme unique (Odroid-XU+E) peut restreindre l'applicabilité des conclusions à des architectures matérielles différentes.

Malgré ces points, l'étude constitue une avancée précieuse en établissant un lien clair entre les opérations au niveau du code source et leur impact énergétique. Elle ouvre la voie à des recherches futures visant à étendre ces résultats à des domaines plus variés et à évaluer les compromis liés à l'optimisation énergétique. Pour cette recherche, l'apport de ce travail réside dans la méthodologie qu'il propose pour évaluer l'impact énergétique des décisions

de conception. Ces techniques peuvent être transposées à l'évaluation des architectures ECS dans les serveurs multijoueurs, permettant d'analyser de manière fine les gains énergétiques apportés par l'ECS par rapport à la POO.

2.2.4 GAME ACTION BASED POWER MANAGEMENT FOR MULTIPLAYER ONLINE GAME

Ce travail [50] se focalise sur les jeux pour la plateforme mobile dont la puissance n'a cessé d'augmenter ces dernières années, lui permettant d'exécuter des jeux complexes comme les MOG (Multiplayer Online Game). Cependant, la consommation d'énergie nécessaire à ces jeux dépasse les améliorations technologiques des batteries, posant un problème majeur. Cette étude explore des stratégies de conservation d'énergie adaptées aux MOG déjà connus afin de mettre en avant leurs limitations. Les chercheurs proposent ensuite deux solutions basées sur de la prédiction permettant de réduire le coût énergétique des jeux tout en préservant une expérience utilisateur qualitative. Les solutions explorées et proposées par ce travail se concentrent sur l'économie d'énergie au niveau réseau (transfert/réception de trames, gestion de la carte réseau, etc.).

MÉTHODOLOGIE

Les chercheurs ont mis en place des tests incluant quatre clients de jeux mobiles, un serveur de jeu central et des connexions sans fil utilisant la norme 802.11b. Les tests sont configurés pour capturer des mesures précises de consommation énergétique, en mettant l'accent sur les performances de la carte réseau sans fil (WNIC). Les auteurs ont choisi **Quake II** comme jeu de test car son code source est librement disponible, permettant une personnalisation approfondie. Le jeu est représentatif des jeux en temps réel, offrant des

scénarios de test variés. De plus, le jeu est compatible avec les plateformes mobiles. Tous les tests sont effectués sur des sessions de 5 minutes de jeu.

Le travail étudie en premier lieu trois approches déjà connues visant à réduire la consommation énergétique des applications. La première approche testée est l'**abandon de trame** par le client qui consiste à ce que le client abandonne volontairement des trames réseau dynamiquement dépendamment du niveau de batterie restant.

La seconde approche étudiée est l'**ajustement du seuil de Dead Reckoning** (DR) qui consiste à changer dynamiquement la valeur du DR dépendamment du niveau de batterie restant. Le Dead Reckoning est une technique qui consiste à lisser les déplacements d'un joueur distant. Un faible seuil de DR nécessite plus de trames réseau mais reflète avec plus d'exactitude les mouvements du joueur, tandis qu'un seuil élevé nécessite moins de trames mais est plus enclin à l'inexactitude de la retranscription des mouvements.

La dernière approche testée est la **réduction de trames serveur** qui consiste à réduire la quantité de trames réseau envoyées par le serveur dépendamment du niveau de batterie du client. Les chercheurs proposent deux méthodes prédictives, l'une basée sur de la *prédiction statistique* et l'autre basée sur la *prédiction des actions de jeu*.

La méthode statistique consiste à prédire les niveaux d'activité du joueur afin d'ajuster dynamiquement le nombre de ressources allouées à la carte réseau. Ces niveaux d'activités sont basés sur un modèle linéaire portant sur l'observation de l'historique des actions du joueur et sont calculés à intervalles réguliers. Lorsque l'activité prédite est faible, la carte réseau est mise en mode veille ou à faible consommation. Afin de déterminer si un taux d'activité est considéré comme faible, le modèle prend en paramètre un seuil d'activité, son dépassement permet de déterminer si la carte reste active ou passe en mode économique. Les périodes critiques, comme les actions rapides de tir, sont maintenues en mode haute consommation.

La méthode basée sur la prédiction des actions de jeu est complémentaire à la prédiction statistique car elle repose sur la corrélation observée par les chercheurs entre le niveau d'activité du joueur et l'état du jeu. Cette approche segmente les différents types d'activités en jeu, comme marcher, courir, tirer, se cacher, et ajuste les modes de consommation énergétique en conséquence. Lors d'intervalles où des activités moins critiques se produisent (par exemple, marcher ou se cacher) la WNIC est basculée en mode faible consommation. Lors des phases critiques comme les combats, la carte réseau passe en mode haute consommation, évitant ainsi une dégradation de la qualité de l'expérience utilisateur.

RÉSULTATS

Les trois premières approches connues (abandon de trame, ajustement du seuil de Dead Reckoning, réduction de trames serveur) sont indépendantes du contexte du jeu. En effet, cela les rendrait inexploitables dans le cadre de réelles économies d'énergies puisque ces méthodes devraient être amplifiées nuisant alors fortement à l'expérience utilisateur. Par exemple, un trop gros nombre d'abandon de paquet pourrait nuire à la fidélité de la retranscription de la position des joueurs ennemis à travers le réseau ou même annuler des actions *gameplay* comme des tirs ou la récolte de soins.

Les méthodes prédictives (prédiction statistique, prédiction des actions de jeu) réalisent quant à elles des économies d'énergie importantes tout en conservant un bon confort de jeu pour l'utilisateur. L'approche statistique offre une économie d'énergie moyenne d'environ **25.3 %**. Pour une application mobile, ces résultats peuvent prolonger la durée de vie typique de la batterie pour un jeu continu d'environ 0,6 heure sans dégradation notable de l'expérience utilisateur. L'approche basée sur les actions de jeu montre qu'en activant sélectivement les modes de faible consommation pendant les périodes d'actions moins critiques, comme marcher

et se cacher, des économies d'énergie supplémentaires ont été réalisées sans compromettre les moments critiques du jeu tels que tirer ou poursuivre. Sans effectuer aucune optimisation sur les périodes d'action critique, les chercheurs estiment des économies globales de **21 %**.

CONCLUSION

L'étude démontre que des techniques de gestion de l'énergie limitant dynamiquement la carte réseau, en particulier celles exploitant les actions en jeu et les prédictions d'activité, peuvent réduire considérablement la consommation d'énergie des MOG sur la plateforme mobile. Bien que les deux approches aient permis des économies substantielles, la méthode basée sur les actions de jeu a montré un potentiel d'optimisation plus élevé en ciblant les phases de jeu non critiques.

Les travaux futurs se concentreront sur l'intégration de techniques de gestion énergétique de l'affichage et du processeur et sur le développement d'une API de jeu compatible avec les contraintes énergétiques pour faciliter une adoption plus large de ces stratégies. Les résultats soulignent la faisabilité d'améliorer les performances des batteries sans compromettre l'expérience des jeux multijoueurs.

DISCUSSION

La proposition d'utiliser des algorithmes de prédiction pour ajuster dynamiquement l'utilisation des ressources, notamment via la gestion de l'état des cartes réseau sans fil, est particulièrement pertinente et innovante. Les résultats montrent des gains mesurables en termes d'économie d'énergie, ce qui représente un pas significatif vers des jeux mobiles plus durables.

Cependant, ce travail présente des limitations notables. Tout d'abord, les expérimentations sont basées sur un échantillon restreint de configurations matérielles et logicielles, notamment le jeu Quake II et un matériel spécifique (PDA iWAVE et cartes réseau D-Link). Cela limite la généralisation des résultats à d'autres contextes de jeux ou plateformes matérielles. De plus, l'accent mis sur la consommation énergétique des cartes réseau, bien que pertinent, occulte d'autres sources potentielles d'optimisation, comme le processeur ou l'affichage. Enfin, bien que les mécanismes proposés réduisent la consommation énergétique, ils impliquent des compromis potentiels sur l'expérience utilisateur, tels que des latences accrues ou des dégradations mineures de la qualité de jeu, qui ne sont pas explorées en profondeur.

Par ailleurs, bien que cette approche contribue à une optimisation énergétique sur le plan logiciel, elle se limite essentiellement à la réduction de la consommation d'énergie côté client. Néanmoins, elle met en évidence un aspect crucial : le coût énergétique associé au traitement des trames réseau, un facteur omniprésent dans le fonctionnement des serveurs de jeu.

2.2.5 ENERGY-AWARE SOFTWARE : CHALLENGES, OPPORTUNITIES AND STRATEGIES

Ce travail[51] explore les problématiques liées à la consommation énergétique des systèmes informatiques modernes. Avec l'augmentation des exigences énergétiques, notamment dans les systèmes d'exascale, les chercheurs se concentrent sur des solutions logicielles, en complément des approches matérielles traditionnelles. Cette étude met en lumière l'importance d'un logiciel conscient de l'énergie pour optimiser les performances tout en réduisant l'empreinte énergétique.

MÉTHODOLOGIE

Les auteurs ont adopté une démarche expérimentale pour étudier la consommation énergétique des logiciels parallélisés. Leur méthodologie s'est déroulée en plusieurs étapes.

Premièrement, des capteurs de puissance ont été installés entre la source d'alimentation et les systèmes informatiques analysés. Ces capteurs ont permis de mesurer avec précision la tension et le courant de la ligne d'alimentation à intervalles réguliers. Les données collectées ont ensuite été centralisées sur un serveur pour une analyse détaillée.

Deuxièmement, les expériences ont été réalisées à l'aide de benchmarks issus de la suite *NAS Parallel Benchmarks*. Ces tests, conçus pour représenter différentes charges de travail computationnelles, comprenaient des applications présentant des profils variés en termes d'accès mémoire, de ratio calcul/communication et de durée d'exécution, afin de garantir la fiabilité des mesures.

Troisièmement, une variété de configurations a été explorée. Cela inclut :

— Le nombre de *threads*, en testant des scénarios de sous-utilisation, d'utilisation optimale et de sur-utilisation des cœurs disponibles.

— Les fréquences d'horloge des processeurs, en ajustant les niveaux pour observer l'impact sur l'énergie consommée et le temps d'exécution.

— Les compilateurs, notamment GNU et Intel, afin d'évaluer leur influence sur les performances énergétiques des logiciels.

Enfin, les données expérimentales ont été analysées en corrélant les mesures de puissance avec des métadonnées expérimentales telles que les paramètres de configuration. Cette analyse a permis d'identifier les interactions complexes entre les paramètres matériels et logiciels, et de quantifier leur impact sur la consommation énergétique.

RÉSULTATS

Les résultats montrent que la consommation énergétique augmente avec le nombre de *threads*, tandis que les temps d'exécution tendent à diminuer. Cependant, un excès d'utilisation des *threads* peut entraîner une hausse significative des coûts énergétiques sans amélioration proportionnelle des performances. L'efficacité énergétique et les performances sont également fortement influencées par des facteurs tels que le choix des compilateurs, la fréquence d'horloge ou encore la taille des données traitées. Par exemple, les benchmarks impliquant davantage de transferts de données ont enregistré une consommation énergétique jusqu'à trois fois supérieure à celle des benchmarks avec moins de volume à traiter.

CONCLUSION

Les conclusions soulignent la complexité de l'optimisation énergétique dans les systèmes parallélisés. Une configuration inappropriée peut non seulement dégrader les performances, mais aussi augmenter considérablement la consommation énergétique. Les auteurs appellent au développement d'outils de profilage énergétique et de logiciels adaptatifs capables d'optimiser les interactions entre matériel et logiciel pour réduire la consommation d'énergie tout en maintenant les performances. Ce travail ouvre des perspectives pour des algorithmes plus efficaces et conscients de l'énergie dans des contextes variés.

DISCUSSION

Le travail apporte une contribution notable en analysant l'impact de paramètres tels que le nombre de *threads*, la fréquence d'horloge, et les choix de compilateurs sur la consommation énergétique. Ces résultats offrent des perspectives pratiques aux développeurs et aux concep-

teurs de systèmes cherchant à équilibrer performance et efficacité énergétique. L'accent mis sur l'importance des métriques standards et des benchmarks énergétiques est particulièrement pertinent pour encourager des pratiques industrielles plus conscientes de l'énergie.

Cependant, le travail présente certaines limites. Tout d'abord, les expériences reposent sur des configurations matérielles spécifiques (plateformes Zen et Skyray) et des benchmarks limités, ce qui restreint la généralisation des conclusions à d'autres architectures ou types de logiciels. Ensuite, bien que le lien entre la consommation énergétique et le parallélisme soit exploré, il manque une analyse approfondie des compromis entre efficacité énergétique et performance, notamment dans des scénarios à forte intensité de données. Par ailleurs, le travail n'aborde pas suffisamment les implications des optimisations proposées sur la complexité du code et la facilité de maintenance. Enfin, les résultats, bien que significatifs, ne prennent pas pleinement en compte l'impact des systèmes d'exploitation ou des interactions utilisateurs sur la consommation énergétique des applications. Ce travail fournit une base solide pour comprendre les enjeux de la conception logicielle consciente de l'énergie, mais gagnerait à intégrer une évaluation plus large des architectures et des applications afin de renforcer la portée de ses recommandations.

Dans le cadre de cette recherche, ce travail fournit un cadre théorique précieux pour intégrer des stratégies énergétiques dans le développement logiciel. Il soutient que l'ECS, par ses caractéristiques intrinsèques, répond à plusieurs de ces recommandations. Cela renforce l'hypothèse que l'ECS peut réduire non seulement les coûts énergétiques, mais également les complexités associées à la gestion des ressources dans des environnements de serveurs multijoueurs.

2.3 DISCUSSION SUR L'ÉTAT DE L'ART

L'analyse de la littérature existante a mis en évidence plusieurs aspects fondamentaux relatifs à l'impact des architectures logicielles sur la consommation énergétique et les performances des logiciels. La Programmation Orientée Objet (POO) est aujourd'hui encore largement adoptée, mais plusieurs études ont démontré ses limitations en termes d'efficacité mémoire et de consommation énergétique, notamment en raison de la fragmentation mémoire et des surcoûts liés aux V-Tables. À l'inverse, l'architecture ECS, en supprimant ces structures hiérarchiques et en favorisant une organisation optimisée des données, semble offrir une alternative plus performante. Son application dans des environnements nécessitant une gestion rigoureuse des ressources a déjà démontré une réduction significative de la consommation énergétique, tout en améliorant la modularité et l'efficacité des traitements.

Par ailleurs, diverses stratégies d'optimisation logicielle, telles que la gestion fine des cycles processeur et de la mémoire, ont prouvé leur impact sur la réduction de la consommation d'énergie, notamment dans des domaines comme les applications mobiles. Ces approches, bien que développées pour d'autres contextes, sont transposables aux serveurs de jeux et renforcent l'intérêt d'une architecture pensée pour maximiser la localité des données et minimiser les accès mémoire inutiles, comme le propose l'ECS.

Dans le domaine des jeux multijoueurs, des méthodes d'adaptation dynamique de la consommation énergétique en fonction de l'activité des joueurs ont été explorées, mais restent principalement ancrées dans des architectures traditionnelles. L'ECS pourrait intégrer ces stratégies de manière plus efficace, en optimisant la distribution des tâches et en réduisant les traitements inutiles.

Enfin, plusieurs recommandations théoriques en matière de développement logiciel écoénergétique convergent vers des principes inhérents à l'ECS, tels que l'optimisation de

l'accès mémoire, la réduction des cycles d'instructions superflus et l'utilisation d'instructions SIMD pour accélérer les traitements. Ces éléments renforcent l'hypothèse selon laquelle l'ECS pourrait constituer une solution plus efficiente pour améliorer les performances et réduire l'empreinte énergétique des serveurs de jeux multijoueurs.

Les études analysées démontrent qu'une architecture logicielle bien pensée peut avoir un impact direct sur la consommation énergétique des serveurs. L'ECS apparaît comme une alternative prometteuse, capable d'améliorer à la fois les performances et l'efficacité énergétique en s'appuyant sur une approche optimisée de la gestion des entités et des données. Toutefois, ces conclusions reposent principalement sur des résultats obtenus dans des contextes variés (IoT, applications mobiles, gestion de l'énergie dans les jeux), et aucune étude ne s'est spécifiquement intéressée à l'application de l'ECS aux serveurs de jeux multijoueurs.

C'est précisément l'objectif de cette recherche : mesurer de manière rigoureuse les bénéfices potentiels de l'ECS dans un environnement de serveur de jeu. Pour ce faire, la partie expérimentale qui suit mettra en place un protocole visant à comparer les performances et la consommation énergétique d'un serveur implémenté en POO et d'un serveur utilisant une architecture ECS. En s'appuyant sur des métriques de performance et de consommation énergétique, cette analyse permettra de déterminer si l'ECS représente une alternative viable et plus écologique pour le développement des infrastructures multijoueurs en ligne.

Ainsi, la discussion menée dans ce chapitre souligne l'importance d'une approche expérimentale afin de valider ou d'infirmer les bénéfices théoriques de l'ECS dans le cadre des serveurs de jeux multijoueurs. Le chapitre suivant détaillera la méthodologie adoptée pour cette analyse, en décrivant le cadre expérimental, les outils de mesure et les critères d'évaluation retenus.

CHAPITRE III
EXPÉRIMENTATION

Ce chapitre présente l'expérimentation réalisée dans le cadre de cette recherche, visant à évaluer de manière objective les performances et la consommation énergétique d'un serveur de jeu en comparant une implémentation basée sur la programmation orientée objet (POO) à une implémentation utilisant l'architecture Entité-Composant-Système (ECS). L'expérimentation repose sur une série de tests rigoureusement conçus et exécutés selon une méthodologie uniforme, garantissant ainsi la reproductibilité et la validité des résultats obtenus.

3.1 MÉTHODOLOGIE

Afin d'assurer l'équité des mesures et des opérations logiques entre les deux architectures étudiées, les sous-sections suivantes détaillent respectivement l'architecture logicielle mise en place, garantissant une parité des opérations entre la programmation orientée objet (POO) et l'architecture Entité-Composant-Système (ECS), ainsi que l'environnement d'exécution du programme. Cette dernière section inclut les spécificités matérielles du dispositif sous test (DUT) ainsi que celles de l'équipement de mesure énergétique (EET) employé.

3.1.1 ARCHITECTURE LOGICIELLE

Dans le cadre des expérimentations, un projet a été développé à l'aide d'*Unreal Engine* [52], un moteur de jeu largement utilisé dans l'industrie du jeu vidéo AAA [13]. *Unreal Engine* ne repose pas nativement sur une architecture ECS, de ce fait, ce choix de moteur reste pertinent dans la mesure où le jeu développé constitue une surcouche logicielle au moteur

13. Jeux vidéo à gros budget et à grande échelle de production

lui-même. Ainsi, l'analyse des performances entre une implémentation en Programmation Orientée Objet (POO) et une implémentation en ECS demeure indépendante du moteur, dans la mesure où les deux versions sont comparées sous des conditions strictement identiques et sur une même version du moteur.

L'absence d'une architecture ECS native dans *Unreal Engine* rend pertinent le développement d'un prototype de jeu exploitant la structure classique du moteur en POO ainsi qu'une alternative reposant sur un ECS personnalisé. Cette approche permet d'évaluer de manière rigoureuse l'impact de l'ECS sur la performance et la consommation énergétique du serveur de jeu, en maintenant un cadre expérimental contrôlé et reproductible.

Les deux versions du jeu ont été développées sur une branche *Github* de la version source 5.4 du moteur. Cette décision se justifie par le fait qu'à la date du 9 juillet 2025, les versions pré-compilées d'*Unreal Engine* fournies par *Epic Games* ne permettent pas la génération d'une version *serveur* d'un jeu, rendant ainsi l'utilisation du code source du moteur indispensable pour la mise en place des tests. Ce prototype de jeu, intitulé *Space Game*, vise à simuler, de manière très simple, une quantité importante de vaisseaux spatiaux se déplaçant aléatoirement dans l'espace et tirant des missiles de manière aléatoire.

Afin de pouvoir comparer de manière équitable, les deux versions étudiées se doivent d'avoir des règles d'opérations logiques similaires. Dans cette optique, le prototype de jeu développé intègre les exactes mêmes règles dans sa version POO et sa version ECS.

Les règles du jeu sont les suivantes. À l'initialisation du monde, **1000** vaisseaux sont instanciés dans l'espace de manière aléatoire entre les positions vectorielles 3.1 et 3.2 avec $\Delta r = 1000$ bien que la valeur de Δr soit purement arbitraire.

$$\overrightarrow{O_{\min}} \begin{pmatrix} -\Delta r \\ -\Delta r \\ -\Delta r \end{pmatrix} \tag{3.1}$$

$$\overrightarrow{O_{\max}} \begin{pmatrix} \Delta r \\ \Delta r \\ \Delta r \end{pmatrix} \tag{3.2}$$

Lors de leur instanciation, chaque vaisseau se voit attribuer aléatoirement un nombre flottant appelé ***AISeed*** compris entre 0.1 et 1.0 interprété comme une variable permettant d'ajuster le comportement des vaisseaux en multipliant les différents vecteurs et délais de leurs actions par la valeur de ***AISeed*** (e.g $\overrightarrow{Vel} = \overrightarrow{Dir}.Speed.\boldsymbol{AISeed}$). Chacune de ces variables est générée à partir d'un même flux de valeurs aléatoires établi via une graine de génération (*seed*) identique aux deux versions du prototype afin de garantir l'équité des tests. À chaque *tick* [14], chaque vaisseau se déplace dans la direction vers laquelle il fait face (axe X relatif) à une vitesse de ***AISeed***.350 cm/s tout en appliquant une rotation constante définie aléatoirement lors de leur initialisation. La constante de rotation est définie avec un *Pitch* (rotation sur l'axe Y relatif) et un *Roll* (rotation sur l'axe X relatif) égaux à ***AISeed***.180 degrés/s ainsi qu'un *Yaw* (rotation sur l'axe Z relatif) égal à zéro degrés. Chaque vaisseau se voit aussi attribuer un délai, appelé ***FireDelay***, entre chaque tir de missile compris entre 5.0 et 10.0 secondes. Cette variable ***FireDelay*** est multipliée par ***AISeed***. Toutes les ***FireDelay*** secondes, le vaisseau tire un missile en lui appliquant une rotation initiale égale à celle du tireur au moment du tir. Chaque missile se déplace à chaque *tick* à une vitesse de 500 cm/s en suivant la direction vers laquelle il fait face (axe X relatif). Au bout d'un délai de cinq secondes après apparition, le

14. Boucle complète du processus de calcul physique, *gameplay* et graphique

missile s'auto-détruit. Dans cette version basique de *Space Game*, les missiles n'ont aucune interaction de quelque nature qu'il soit avec le reste des entités du monde.

Pour ce prototype de jeu, l'architecture POO est structurée de la manière suivante en respectant le *Gameplay framework* [15] d'Unreal : Les vaisseaux, dits *Sapceships*, héritent de la classe *APawn* qui intègre la forme et les fonctions de mouvement des vaisseaux. Les *Sapceships* sont contrôlés par un *AAIController* qui intègre les fonctions de contrôle des vaisseaux (e.g simuler des inputs afin de faire bouger le *Spaceship* contrôlé). Une classe héritant de *AGameMode* s'occupe d'instancier les IA à l'entrée en jeu du niveau. Les missiles, quant à eux, sont des *AActor* implémentant leur propre logique de déplacement.

L'architecture ECS utilise un *SpaceshipsManager* héritant de la classe *AActor* afin de pouvoir être instancié dans le monde, c'est dans cette seule et unique classe que tout l'ECS est implémenté en utilisant *Flecs* [16], un cadriciel ECS open-source. L'ECS est composé de 5 *Components* :

— ***FSpaceShip*** comprenant un *Id* ainsi qu'une vitesse de *translation* et de *rotation* par défaut dont les valeurs seront respectivement de 350 et 180 et seront écrites tel quel (350 et 180) dans le reste de la méthodologie afin de simplifier la compréhension.

— ***FTransform*** comprenant une *position* et une *rotation* ;

— ***FSpaceshipAI*** comprenant la *AISeed*.

— ***FWeapon*** comprenant le *FireDelay* ainsi qu'une variable permettant de stocker le délais restant entre chaque tirs appelée *FireCooldown* ;

— ***FBullet*** comprenant la *position* du missile, son vecteur de *direction* et une variable permettant de stocker le temps restant avant la destruction du projectile.

15. https://dev.epicgames.com/documentation/en-us/unreal-engine/gameplay-framework-in-unreal-engine, Saisi le 9 juillet 2025

16. https://www.flecs.dev/flecs/, Saisi le 9 juillet 2025

Les entités instanciées sont toutes composées d'un *SpaceShip*, un *Transform*, une *IA* et un *Weapon*. Lorsqu'un vaisseau tire, une nouvelle entité est créée se composant d'une *Bullet*. L'ECS implémente un total de 3 systèmes : un système de mouvement IA permettant à toutes les entités comprenant un *Spaceship*, un *Transform* et une IA de se mouvoir dans l'espace selon les règles du jeu établies plus haut ; un système de tir *IA* permettant à toutes les entités ayant un *Transform*, un *Weapon* et une *IA* de tirer selon les règles statuées au-dessus ; et enfin un système de missile permettant à chaque *Bullet* de se mouvoir selon les règles du jeu.

La version POO et ECS du jeu intègre l'exacte même logique dans leurs architectures respectives. La version POO initialise les variables dans l'objet à sa création, l'ECS les initialise lors de la création des entités comme présenté dans le code suivant.

```
// POO
//Dans la fonction appelée à la création
{
    //Initialisation de AISeed entre 0.1 et 1.0
    AISeed = Clamp(RandomFloat(),0.1,1.f);

    //Initialisation de FireDelay entre 5 et 6 secondes
    FireDelay = RandomFloatInRange(5.f,10.f);

    //Mise à jour de FireDelay en appliquant le facteur AISeed
    FireDelay = FireDelay*AISeed;
}
```

```
// ECS
//[...]
    //Création de l'entité
    auto Spaceship = ECSWorld->entity();

    //Initialisation de AISeed entre 0.1 et 1.0
    const float AISeed = Clamp(RandomFloat(),0.1,1);

    //Attribution de AISeed à l'entité créée via le composant
    //FSpaceshipAI{float AISeed}
    Spaceship.set<FSpaceshipAI>({
        AISeed
    });
```

```
    //Initialisation de FireDelay entre 5 et 6 secondes
    const float FireDelay = RandomFloatInRange(5.f,10.f);
    //Mise à jour de FireDelay en appliquant le facteur AISeed
    FireDelay = FireDelay*AISeed;

    //Attribution de FireDelay à l'entité créée via le composant
    //FWeapon{float FireDelay}
    Spaceship.set<FWeapon>({
        FireDelay
    });
//[...]
```

La version POO applique les actions des vaisseaux dans la fonction *Tick* qui est appelée à chaque *tick*, la version ECS applique les actions des vaisseaux dans des systèmes appelés à chaque *tick*, voir le code ci-dessous.

```
// POO
//Dans la fonction Tick
{
    //Applique la rotation
    SpaceshipRotation.Yaw += AISeed * 180.f * DeltaTime;
    SpaceshipRotation.Roll += AISeed * 180.f * DeltaTime;

    //Applique la translation
    SpaceshipLocation += AISeed * SpaceshipRotation.Vector()
    * 350.f * DeltaTime;

    //Timer pour le tir
    if(FireCooldown <= 0)
    {
        Fire();
        FireCooldown = FireDelay;
    }
    else
    {
        FireCooldown-=DeltaTime;
    }
}
```

```
// ECS
// Dans le système de mouvements (T => Transform)
{
    // Rotation
```

```
    T.Rotation.Yaw += AI.Seed * 180.f * DeltaTime;
    T.Rotation.Roll += AI.Seed * 180.f * DeltaTime;

    // Translation
    T.Location += AI.Seed * T.Rotation.Vector() * 350.f * it.delta_time();
}

// Dans le système de tir (W => Weapon)
{
    if(W.FireCooldown <= 0)
    {
        Fire();
        W.FireCooldown = W.FireDelay;
    }
    else
    {
        W.FireCooldown-=DeltaTime;
    }
}
```

Et il en est de même pour tous les systèmes logiques implémentés dans chaque version du prototype de jeu. L'intégralité du code du prototype est disponible sur *Github* [17].

3.1.2 ENVIRONNEMENT

Toutes les expériences ont été réalisées sur un ordinateur de bureau alimenté par une prise de 115V avec la configuration suivante : Windows 11, processeur AMD Ryzen 9 5900X [18], carte mère MSI MPG B550 GP [19], 64 Go de RAM cadencés à 3200 MHz, SSD NVMe et une carte graphique nVidia RTX 3080 [20].

Afin de mesurer la consommation électrique du dispositif testé (*Device Under Test* - DUT), un appareil de mesure de la consommation énergétique a été développé dans le cadre

17. https://github.com/fuegoglol/ECS-EnergyAnalysisProject-MasterThesis, Saisi le 9 juillet 2025

18. https://www.amd.com/en/products/processors/desktops/ryzen/5000-series/amd-ryzen-9-5900x.html, Saisi le 9 juillet 2025

19. https://www.msi.com/Motherboard/MPG-B550-GAMING-PLUS, Saisi le 9 juillet 2025

20. https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3080-3080ti/, Saisi le 9 juillet 2025

de cette recherche. Ce système utilise un capteur de courant SCT013 pour mesurer l'intensité du courant circulant entre la source d'alimentation et le DUT. Un microcontrôleur ESP32-C6-DevKitC-1 a été employé pour convertir la tension de sortie du capteur SCT013 en mesures de puissance en watts (W). Le prototype du système de surveillance a été assemblé sur une *breadboard* (voir Figure 3.2) en respectant le schéma électrique présenté à la Figure 3.1. Le code ESP32 utilisé pour la conversion en puissance est le suivant.

```
#include <math.h>
#include "EmonLib.h"

const int ADC_INPUT = A4;
EnergyMonitor emon1;
double Irms, thePower;

void setup()
{
  Serial.begin(9600);
  emon1.current(ADC_INPUT, 30);
}

void loop()
{
  Irms = emon1.calcIrms(1480);
  thePower = Irms*115.0;
  Serial.printf("%d.%d\n",(int)thePower,((unsigned int)(thePower*100))%100);
  delay(30);
}
```

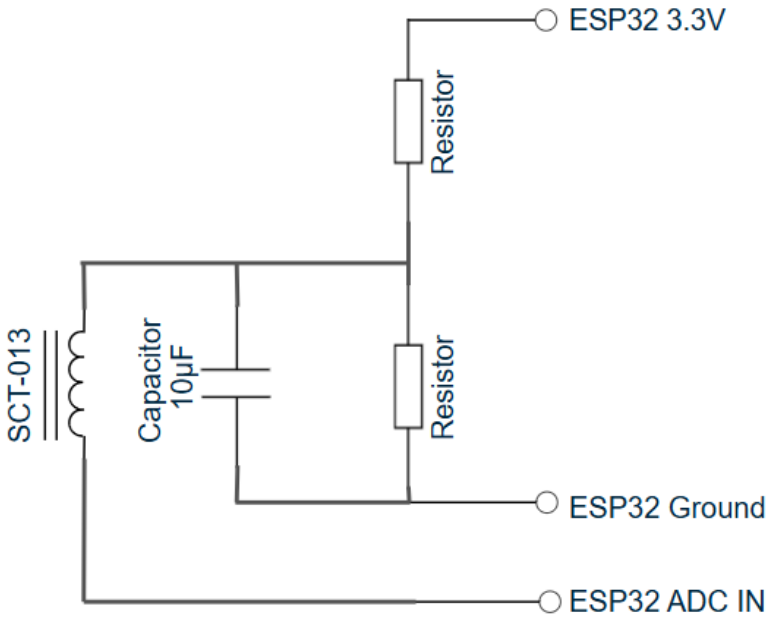

FIGURE 3.1 : Schéma électrique de l'appareil de mesure

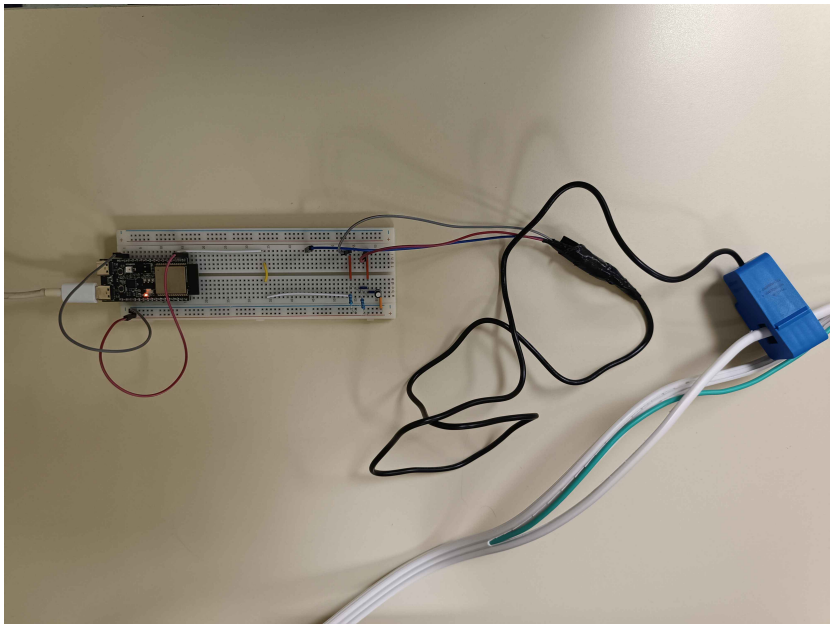

FIGURE 3.2 : Appareil de mesure de la consommation du DUT

3.1.3 MESURES

Dans le cadre de cette recherche, l'unité de mesure Joules par image (J/image) est utilisée afin d'évaluer simultanément la consommation énergétique et la performance du dispositif sous test (DUT). Étant donné que les serveurs de jeu fonctionnent à un taux de rafraîchissement constant, cette métrique permet de comparer l'efficience des deux architectures étudiées pour une fréquence d'images donnée. Voir le travail intitulé « *Profilage énergétique dans les jeux : introduction d'une métrique de consommation d'énergie basée sur l'image* » en annexe B pour de plus amples informations.

Pour établir une corrélation entre le taux de rafraîchissement (IPS) des jeux testés et la consommation énergétique (W), un plugin (disponible sur *Github* [21]) développé dans le cadre de cette recherche est utilisé. Cet outil est programmé pour *Unreal Engine* lui permettant alors d'être intégré à n'importe quel projet utilisant ce même moteur. Ce plugin permet d'obtenir en temps réel la consommation énergétique du DUT, telle que mesurée par notre outil de mesure (ESP32/SCT013), via une connexion série à travers un port COM. À la fin de l'exécution du jeu, le plugin stocke les données enregistrées dans un fichier CSV respectant un format prédéfini, voir Table 3.1.

TABLEAU 3.1 : Format des prises de mesures de consommation (données d'exemple)

Temps (s)	Puissance (W)	Images par seconde (IPS)
1.88	1100	120

Pour traiter les résultats collectés à partir des fichiers CSV, un outil de visualisation des données (disponible à la même adresse *Github* que le plugin) est utilisé. Ce programme, développé dans le cadre de cette recherche, permet de générer des graphiques affichant les valeurs moyennes, maximales et minimales mesurées, converties en Joules par image (J/image).

21. https://github.com/fuegoglol/JPFMonitoringPlugin_UE5, Saisi le 9 juillet 2025

Afin de mesurer le comportement de chaque processus logique durant l'exécution, l'outil *Unreal Insights* est exécuté en parallèle de l'instance du serveur permettant de récupérer un ensemble de données télémétriques sur le temps d'exécution et les ressources allouées à chaque processus logique.

3.2 TESTS

Les expérimentations ont été menées sur les deux versions du jeu *Space Game*, à savoir l'implémentation en Programmation Orientée Objet (POO) et celle utilisant l'architecture Entité-Composant-Système (ECS). Afin d'évaluer de manière rigoureuse l'impact de ces deux paradigmes sur les performances et la consommation énergétique des serveurs de jeux, les tests ont été effectués dans des conditions contrôlées et sous plusieurs fréquences d'exécution.

L'une des particularités des serveurs de jeux est leur fonctionnement à un taux de rafraîchissement fixe, généralement de 30, 60 ou 120 images par seconde (IPS). Plus la fréquence d'images est élevée, plus le serveur doit mobiliser ses ressources de calcul pour garantir un maintien stable de la fréquence d'images afin d'éviter toute dilatation du déroulement des actions de jeu. À l'inverse, lorsque la fréquence d'images est basse, le serveur peut potentiellement entrer en état d'inactivité volontaire une fois l'ensemble des calculs requis pour un cycle terminé. Cette relation entre charge de calcul, efficience et gestion énergétique constitue un élément central de l'étude.

Dans ce contexte, les tests ont été réalisés sous quatre configurations de fréquence d'images distinctes : 30 IPS, 60 IPS, 120 IPS et 240 IPS. L'ajout du cas 240 IPS permet d'explorer les limites des deux architectures en simulant une charge de calcul extrême, mettant en évidence les capacités d'optimisation et d'adaptabilité des deux approches.

Pour chaque configuration de fréquence d'images, une batterie de **200** tests a été exécutée, comprenant **100** exécutions sur la version POO et **100** exécutions sur la version ECS. Afin de garantir des conditions d'expérimentation équitables, les exécutions des deux versions du jeu ont été effectuées de manière alternée et non simultanée. Chaque test individuel a une durée de **120** secondes, soit **2** minutes, impliquant un total de **400** minutes (**6** heures et **40** minutes) par batterie de tests.

Pour assurer une allocation optimale des ressources au serveur de jeu et éviter toute interférence externe susceptible d'altérer les mesures, tous les processus non essentiels au système d'exploitation Windows ont été arrêtés avant chaque batterie de tests. De plus, l'ensemble des tests a été entièrement automatisé grâce à un outil développé en interne, garantissant une exécution rigoureuse et contrôlée des sessions de serveur, sans chevauchement entre elles. Dans ces conditions expérimentales, le serveur fonctionne de manière isolée, ne recevant aucune interaction extérieure et se limitant exclusivement à la simulation des entités du jeu, en l'occurrence les vaisseaux spatiaux. Cette méthodologie permet ainsi de mesurer précisément les performances et la consommation énergétique des deux implémentations.

CHAPITRE IV
ANALYSES DES RÉSULTATS

4.1 30 IMAGES PAR SECONDE

4.1.1 ANALYSE DE LA CONSOMMATION

À une fréquence de **30 images par seconde**, les résultats montrent une tendance claire : l'implémentation ECS consomme significativement moins d'énergie par image que sa contre-partie en POO. Le graphique correspondant (Figure 4.1) illustre une diminution nette de la consommation énergétique en Joules par image pour la version ECS du jeu. Bien que les pics de consommations maximums enregistrés entre les deux versions semblent homogènes, la consommation moyenne de l'ECS égale la consommation minimum de la POO à une valeur approximative de **44** Joules/image, notant un écart de 3 joules par rapport à la moyenne de la POO se situant à **47**.

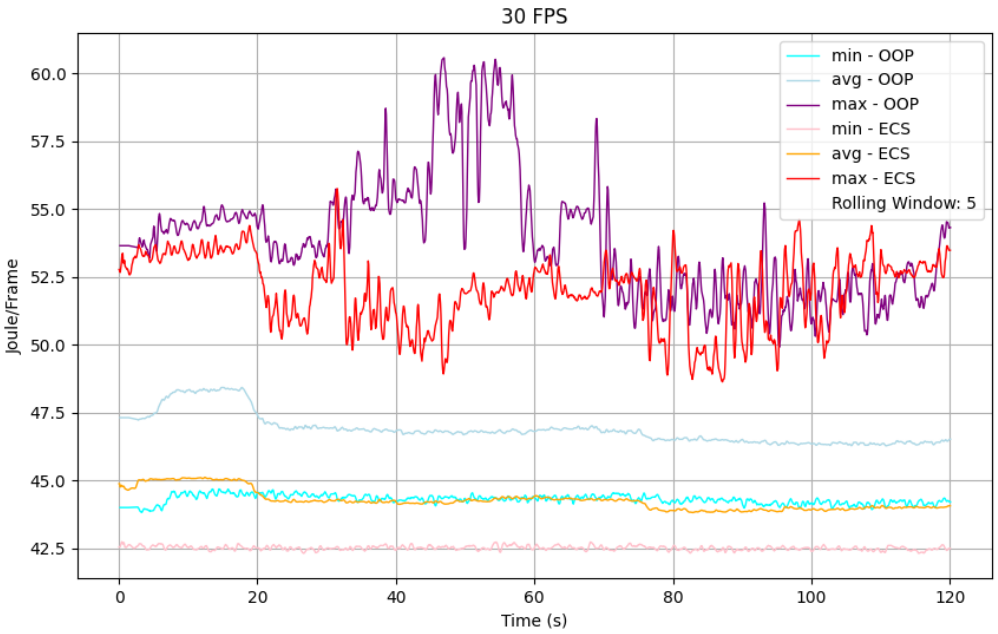

FIGURE 4.1 : *Benchmark* Joules/image POO vs ECS 30 IPS

Ces résultats indiquent qu'à fréquence d'images fixe et relativement basse, les deux versions du jeu présentent une gestion des ressources distincte, bien que soumises à une charge de calcul équivalente. La consommation minimale observée dans la version POO tend à se rapprocher de la consommation moyenne enregistrée dans la version ECS.

Par ailleurs, les courbes moyennes des deux versions présentent une dynamique relativement similaire, ce qui appuie le fait que les traitements effectués sont de même nature avec des optimisations énergétiques distinctes. Il est également à noter que la version ECS enregistre des valeurs moins disparates, y compris dans ses valeurs maximales, traduisant ainsi une plus grande stabilité énergétique sur l'ensemble des exécutions.

4.1.2 ANALYSE DE L'EFFICIENCE

À une fréquence d'images relativement basse, les serveurs disposent de marges d'inactivité plus importantes, ce qui rend l'efficience énergétique fortement dépendante de la durée d'exécution des traitements logiques au sein d'une image. Les mesures collectées via l'outil *Unreal Insight* révèlent que, dans le cas de l'implémentation orientée objet (cf. Figure 4.2), le traitement logique lié à la mise à jour des actions des vaisseaux occupe environ **51.4 secondes** sur les **2 minutes** de simulation prévus (3602 images pour 2 minutes de simulation à 30 IPS). Cela représente un temps d'utilisation par image moyen de **14 ms** sur les **33.3 ms** disponibles pour chaque image à 30 IPS, soit près de **42%** de la durée totale. Cela implique que le serveur demeure inactif pendant approximativement **58%** du temps avec une durée d'inactivité totale de **1 minute 8 secondes**.

FIGURE 4.2 : *Profiling* de l'ensemble des images pour la version POO 30 IPS

À l'inverse, pour la version fondée sur l'ECS (cf. Figure 4.3), le temps d'exécution logique est significativement réduit, s'établissant à environ **300µs**, soit à peine **1%** de la durée d'une image. Cela représente une durée d'activité de **1.1 secondes** sur les **2 minutes** de

simulation prévue. Le serveur reste donc inactif pendant **1 minute et 57 secondes** soit près de **97.5%** du cycle, ce qui révèle un très faible taux d'occupation dans ce contexte.

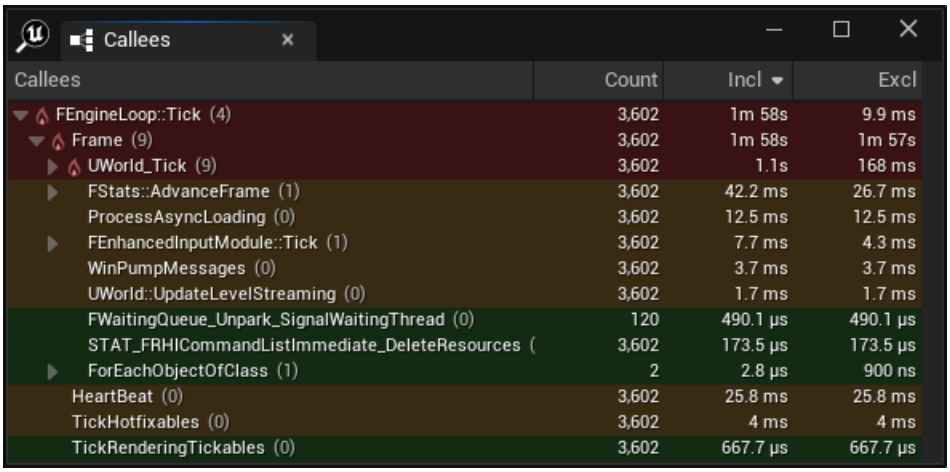

FIGURE 4.3 : *Profiling* de l'ensemble des images pour la version ECS 30 IPS

Ainsi, bien que la version ECS présente une efficacité de traitement brute nettement supérieure avec un temps d'exécution environ **46 fois** plus rapide que celui de la version POO , cette performance se traduit par une utilisation extrêmement partielle du temps alloué à chaque image. Dans un contexte de fréquence d'images basse, l'implémentation ECS apparaît donc moins efficiente en termes d'exploitation des ressources temporelles disponibles, malgré ses performances brutes supérieures.

4.2 60 IMAGES PAR SECONDE

4.2.1 ANALYSE DE LA CONSOMMATION

À une fréquence de **60 images par seconde** (Figure 4.4), l'écart de consommation énergétique entre les deux versions du jeu se révèle plus marqué que dans le cas précédent. La version POO enregistre une consommation moyenne par image supérieure à celle de

l'ECS, avec une différence avoisinant les 3 Joules par image. Il est également notable que la consommation minimale observée dans la version POO dépasse désormais la moyenne de la version ECS de plus d'un Joule par image. En outre, les valeurs maximales ne présentent plus de chevauchement significatif entre les deux architectures : la moyenne des pics de consommation est supérieure de plus d'un Joule par image pour la version POO par rapport à l'ECS, accentuant ainsi la distinction entre les deux comportements énergétiques.

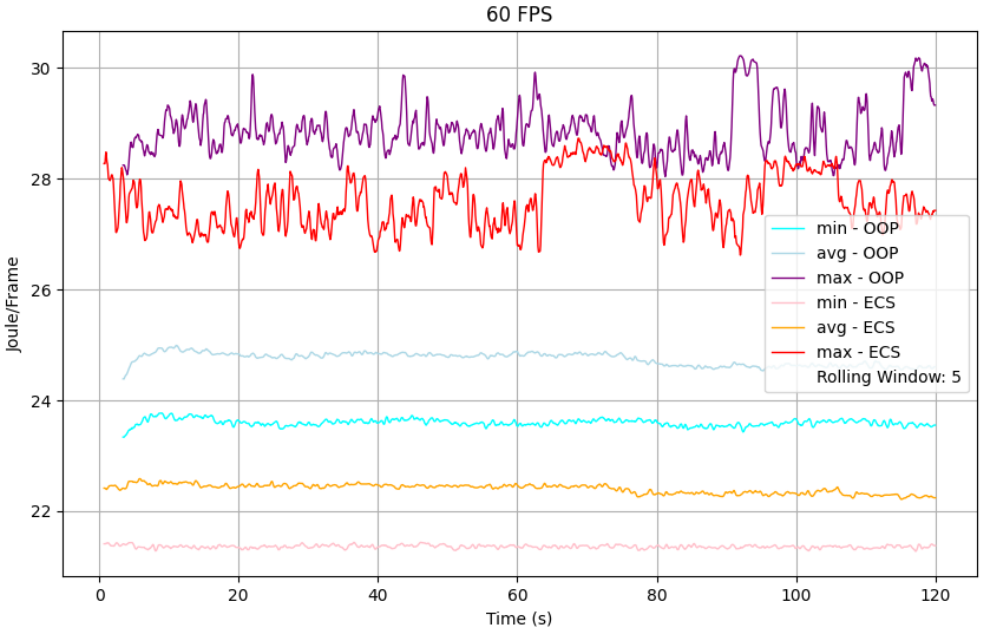

FIGURE 4.4 : *Benchmark* Joules/image POO vs ECS 60 IPS

Ces résultats suggèrent qu'à une fréquence d'image standard, les différences de gestion des ressources entre les deux versions du jeu deviennent plus prononcées que celles observées à une fréquence plus basse, malgré une charge de calcul équivalente. La consommation minimale mesurée dans la version POO reste de manière constante supérieure à la consommation moyenne de la version ECS. Contrairement aux observations faites à 30 IPS, les courbes

moyennes des deux architectures ne suivent plus une dynamique parallèle, ce qui laisse supposer une répartition différente des traitements pour satisfaire les exigences d'un framerate plus élevé. En outre, les mesures indiquent une plus grande stabilité dans cette configuration, notamment en ce qui concerne les valeurs maximales, qui présentent moins de dispersion que précédemment. Cela témoigne d'une meilleure régularité énergétique dans les deux versions lors des exécutions à 60 IPS.

4.2.2 ANALYSE DE L'EFFICIENCE

À une fréquence d'images de 60 IPS, les serveurs disposent encore de marges d'inactivité, bien que plus réduites qu'à 30 IPS, ce qui améliore l'efficience énergétique des deux architectures au sein d'une image. Les mesures obtenues via l'outil *Unreal Insight* indiquent que, dans le cas de l'implémentation orientée objet (cf. Figure 4.5), le traitement logique relatif à la mise à jour des actions des vaisseaux s'étale sur environ **13 ms** des **16.7 ms** disponibles pour chaque image. Cela représente un temps d'exécution de **1 minute et 32 secondes** sur les **2 minutes** de simulation prévus (7201 frames pour 2 minutes de simulation à 60 IPS). Ce temps d'exécution représente désormais près de **78 %** de la durée totale de l'image, ne laissant qu'environ **22 %** du temps dans un état inactif.

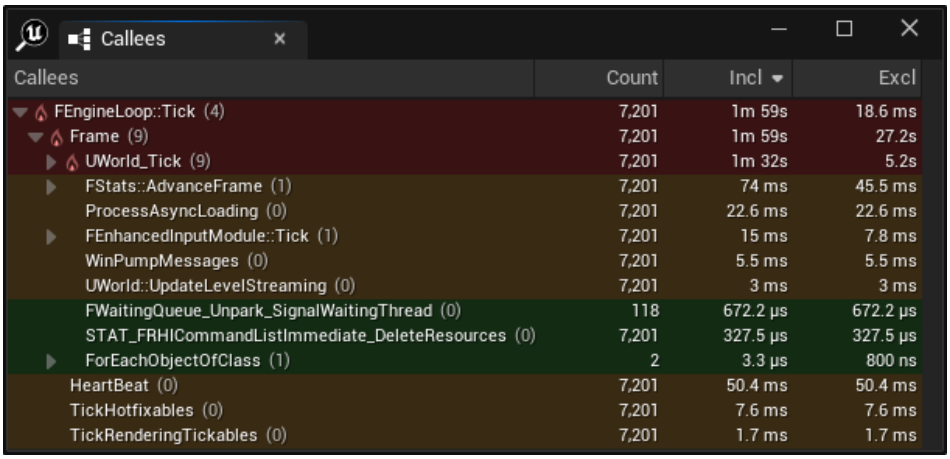

FIGURE 4.5 : *Profiling* de l'ensemble des images pour la version POO 60 IPS

En ce qui concerne la version basée sur l'ECS (cf. Figure 4.6), le temps d'exécution logique reste proche de celui relevé à 30 IPS, avec une valeur atteignant approximativement **264 µs** par image soit **1.9 secondes** sur les **2 minutes** de simulation prévus. Cela correspond à environ **1.6 %** de la durée totale de l'exécution, laissant toujours le serveur inactif durant près de **98.4%** du cycle.

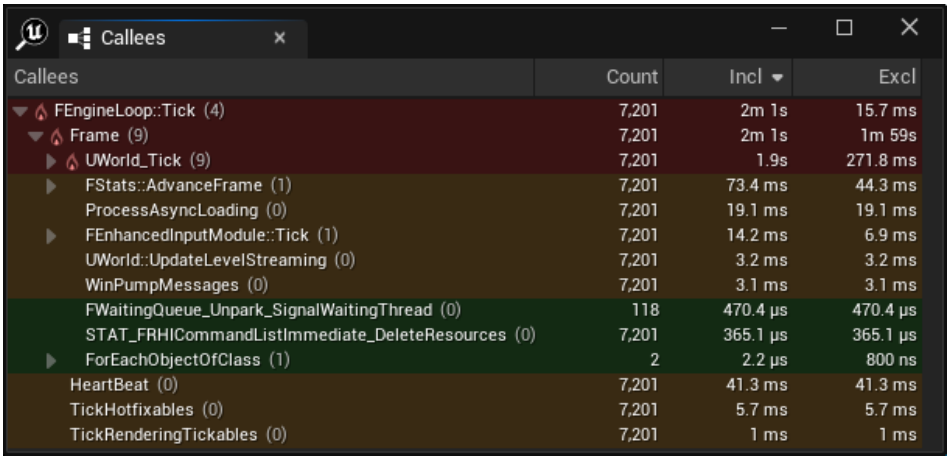

FIGURE 4.6 : *Profiling* de l'ensemble des images pour la version ECS 60 IPS

En comparaison avec une fréquence d'images plus basse (30 IPS), l'efficacité respective des deux architectures n'évolue que marginalement, ce qui s'explique par le fait que la charge de calcul simulée reste globalement égale peu importe le taux de rafraîchissement. Il est toutefois intéressant de noter que la version orientée objet semble atteindre une configuration proche de son seuil d'occupation optimal, mobilisant la majeure partie du temps de calcul disponible à 60 IPS. À l'inverse, l'implémentation ECS reste encore largement en deçà de cette limite, avec un temps d'exécution très inférieur à la durée allouée à chaque image.

4.3 120 IMAGES PAR SECONDE

4.3.1 ANALYSE DE LA CONSOMMATION

Les données issues de la Figure 4.7 révèlent une concentration plus marquée des valeurs de consommation pour les deux architectures à **120 IPS**, avec des moyennes et des minima globalement proches entre elles. La tendance générale observée précédemment se confirme néanmoins : la version POO maintient des niveaux de consommation supérieurs à ceux de la version ECS tout au long des tests. L'écart entre les courbes reste relativement constant, bien que la version POO affiche ici une consommation maximale sensiblement plus proche de sa propre moyenne. À l'inverse, la version ECS conserve un écart proportionnel plus important entre sa consommation moyenne et sa valeur maximale, cette dernière demeurant proche de la consommation moyenne observée dans la version POO. Ainsi, si la POO se caractérise par une répartition plus resserrée autour de sa moyenne, l'ECS continue de présenter des performances énergétiques globalement inférieures, tout en conservant une marge supérieure entre ses différentes valeurs mesurées.

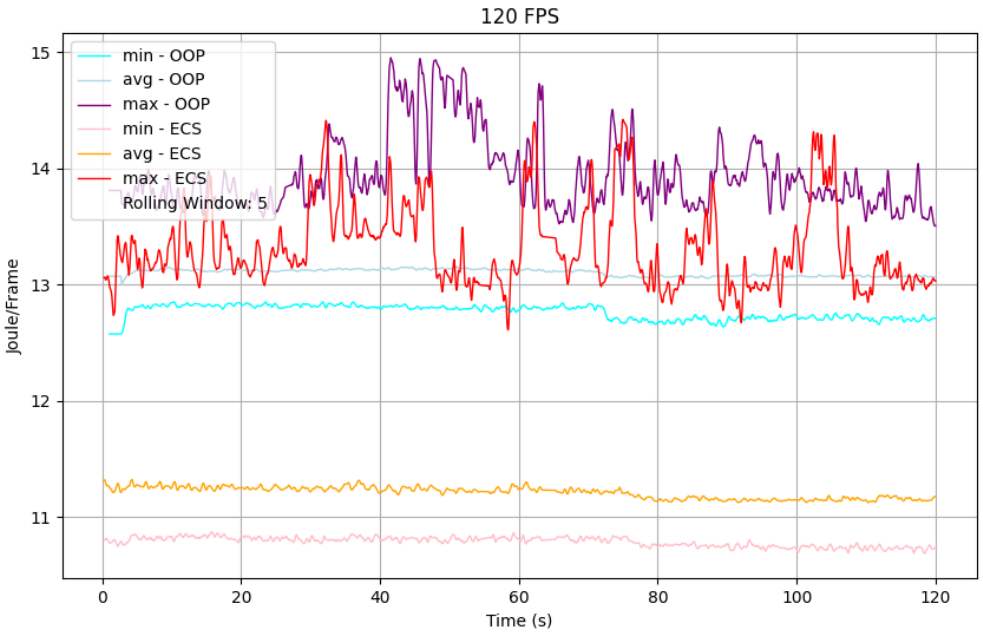

FIGURE 4.7 : *Benchmark* Joules/image POO vs ECS 120 IPS

Les résultats obtenus à une fréquence d'image élevée indiquent une évolution notable dans la gestion des ressources entre les deux versions du jeu, et ce malgré une charge de calcul équivalente. La consommation minimale observée dans la version POO demeure systématiquement supérieure à la consommation moyenne relevée dans la version ECS. Toutefois, les courbes moyennes des deux architectures présentent une évolution similaire, marquée notamment par une légère diminution des consommations moyennes et minimales autour de $t = 75$ s, suggérant une répartition énergétique comparable entre les deux versions pour maintenir une fréquence d'image soutenue. Par ailleurs, les mesures montrent une stabilité accrue du côté de la POO, dont les valeurs minimale, moyenne et maximale sont globalement rapprochées, traduisant une meilleure régularité de consommation dans cette configuration.

À l'inverse, la version ECS conserve un écart important entre ses valeurs moyennes et maximales, ces dernières restant nettement plus élevées. Ainsi, bien que la version ECS continue d'enregistrer des niveaux de consommation inférieurs, la version POO se distingue ici par une plus grande constance dans son comportement énergétique à 120 IPS.

4.3.2 ANALYSE DE L'EFFICIENCE

À une fréquence d'images élevée de 120 IPS, la version orientée objet du serveur ne dispose plus d'aucune marge d'inactivité ; plus encore, le temps requis pour traiter une image logique excède la durée maximale théorique allouée à chaque image, ce qui conduit à une dilatation temporelle du jeu afin de tenter de maintenir le taux de rafraîchissement ciblé, en l'occurrence 120 IPS. Les mesures obtenues via l'outil *Unreal Insight* montrent que, dans le cas de l'implémentation orientée objet (cf. Figure 4.8), le temps d'exécution de la logique des vaisseaux est d'environ **11.5 ms** par image (pour un total de 14402 images, soit l'équivalent de 2 minutes de simulation à 120 IPS). Or pour un taux de rafraîchissement de 120 IPS, une image a une durée totale de **8.33 ms**, soit une valeur inférieure aux **11.5 ms** de temps de calcul des actions des vaisseaux. Cette situation implique que le moteur doit répartir les charges d'exécution pour conserver une cadence perçue régulière, en ajustant dynamiquement la durée des frames au-delà de la limite théorique.

FIGURE 4.8 : *Profiling* de l'ensemble des images pour la version POO 120FPS

Cette dilatation temporelle est confirmée par les mesures globales d'exécution, où la simulation, censée durer **2 minutes**, atteint en réalité **2 minutes et 47 secondes**, soit une augmentation de près de **40 %** du temps total. Ce phénomène impacte directement l'expérience utilisateur, en introduisant un décalage entre le temps réel et le temps simulé, et traduit une inefficience critique de cette architecture dans ce contexte.

En revanche, pour la version fondée sur l'ECS (cf. Figure 4.9), le temps d'exécution logique reste extrêmement faible, s'établissant aux alentours de **236 µs**, ce qui représente environ **2.8 %** de la durée totale d'une image. Ainsi, le serveur demeure inactif pendant un total d'environ **8.1 ms** sur les **8.33 ms** alloués à l'image. Toutefois, une légère dilatation temporelle est également observée, avec une durée totale de simulation atteignant **2 minutes et 5 secondes**, soit un écart de l'ordre de **5 secondes** par rapport à la durée attendue. Ce décalage, bien plus modéré, pourrait être attribué à des limitations techniques propres au moteur, indépendantes de la logique implémentée via l'ECS.

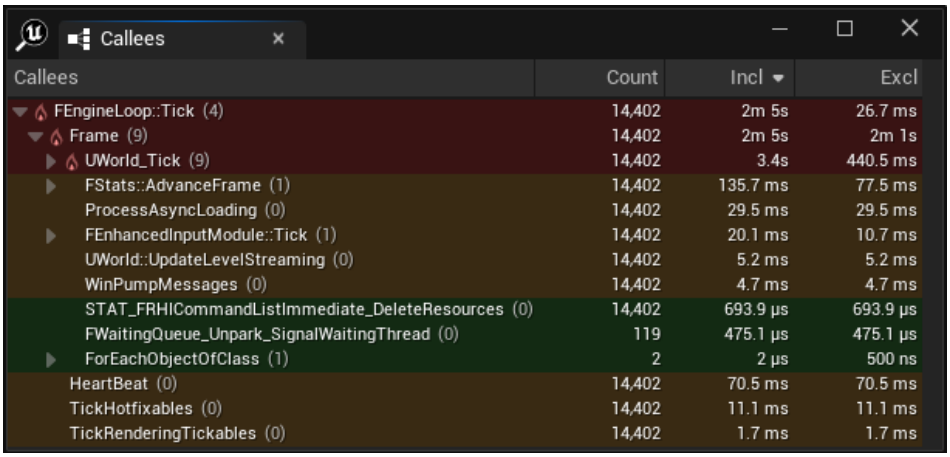

FIGURE 4.9 : *Profiling* de l'ensemble des images pour la version ECS 120FPS

Dans ce contexte, les deux architectures présentent des comportements nettement différenciés. L'implémentation orientée objet excède sa capacité de traitement et provoque une dégradation sensible du rythme de jeu, tandis que l'ECS conserve une efficacité d'exécution élevée, avec un temps de calcul par image représentant une fraction marginale (2.8 %) du temps total alloué.

4.4 240 IMAGES PAR SECONDE

4.4.1 ANALYSE DE LA CONSOMMATION

La Figure 4.10 illustre qu'à une fréquence de **240 images par seconde**, les deux versions présentent une évolution continue de leur consommation énergétique, bien que celle-ci s'inscrive dans une plage de valeurs plus resserrée. La version orientée objet (POO) maintient des niveaux de consommation systématiquement supérieurs à ceux observés pour la version ECS. Comme dans les autres contextes d'exécution, les valeurs maximales de consommation des deux architectures tendent à se chevaucher. Néanmoins, les mesures enregistrées pour la version ECS présentent une dispersion moindre et une moyenne légèrement inférieure à celle de la POO. Malgré ce rapprochement des valeurs maximales, les mesures associées à l'ECS

demeurent globalement inférieures, avec un écart moyen d'environ **un Joule/image** en faveur de l'ECS tant pour les moyennes que pour les valeurs minimales observées.

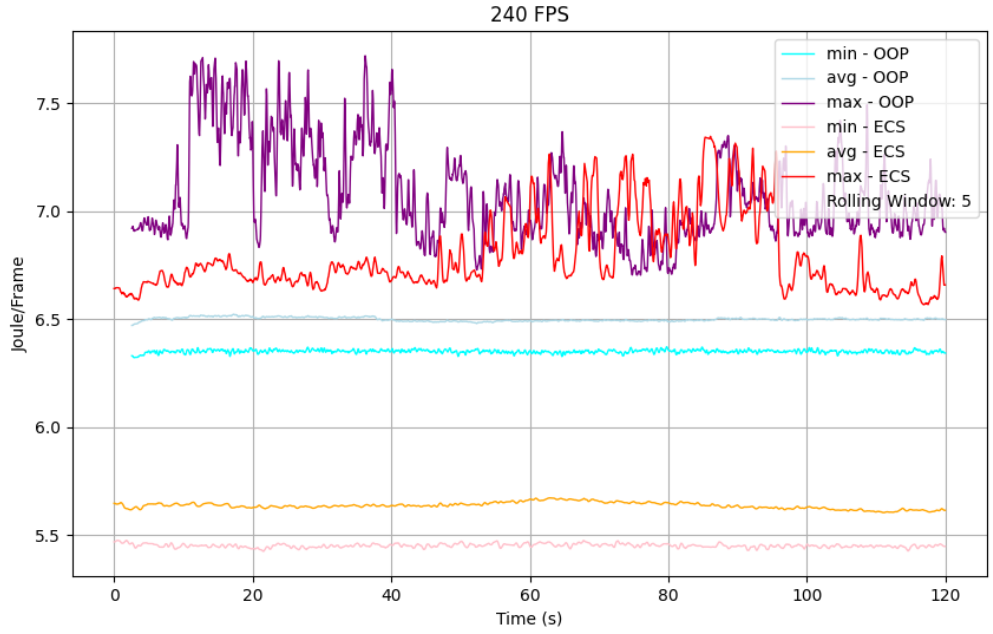

FIGURE 4.10 : *Benchmark* Joules/image POO vs ECS 240 IPS

Ces résultats suggèrent qu'à une fréquence d'image très élevée, les différences de gestion des ressources entre les deux versions du jeu connaissent une nouvelle évolution, malgré une charge de calcul demeurant constante. La consommation minimale observée dans la version orientée objet (POO) reste systématiquement supérieure à la consommation moyenne mesurée pour la version ECS, confirmant ainsi une tendance déjà identifiée aux fréquences inférieures. À 240 IPS, les mesures mettent en évidence une amélioration de la stabilité énergétique pour la version ECS, qui présente des valeurs maximales légèrement plus régulières et moins dispersées que celles de la POO.

À l'inverse, la version POO semble rencontrer davantage de difficultés à maintenir une consommation maximale stable, traduisant une variabilité plus importante dans ses performances énergétiques sous haute contrainte temporelle. Cette évolution témoigne d'une légère supériorité de la régularité énergétique de l'ECS dans ce contexte extrême, bien que les deux architectures conservent, en termes de consommation moyenne et minimale, une forte stabilité globale tout au long des campagnes de tests.

Ainsi, si les écarts absolus restent modérés, la capacité de l'ECS à maintenir une consommation maximale mieux maîtrisée pourrait constituer un atout supplémentaire en environnements critiques où la prédictibilité énergétique est un facteur déterminant.

4.4.2 ANALYSE DE L'EFFICIENCE

À une fréquence d'images de 240 IPS, le temps de traitement d'une image pour la version orientée objet du serveur excède de manière significative la durée maximale théorique allouée à chaque image, fixée à **4.16 ms**. Cette situation entraîne une dilatation temporelle beaucoup plus importante que celle observée à 120 IPS, dans une tentative du moteur de maintenir le taux de rafraîchissement cible. Les mesures obtenues via l'outil *Unreal Insight* révèlent que, pour l'implémentation orientée objet (cf. Figure 4.11), le traitement logique des actions des vaisseaux atteint environ **11 ms** par image, pour un total de **28806** images sur une session théorique de 2 minutes à 240 IPS.

FIGURE 4.11 : *Profiling* de l'ensemble des images pour la version POO 240FPS

Compte tenu de la durée cible d'une image à 240 IPS (**4.16 ms**), le temps de traitement observé est environ **2.6** fois supérieur à la durée théorique, nécessitant une réorganisation dynamique des charges d'exécution par le moteur afin de tenter de conserver une cadence visuellement acceptable. Cette dilatation est confirmée par l'analyse globale, où la simulation censée durer **2 minutes** s'étend finalement sur **5 minutes et 21 secondes**, soit une augmentation de près de **167.5 %** du temps prévu. Ce phénomène engendre un décalage majeur entre le temps réel et le temps simulé, affectant de manière critique l'expérience utilisateur et révélant une inefficience extrême de cette architecture dans un contexte de charge élevée.

En comparaison, la version fondée sur l'ECS (cf. Figure 4.12) conserve un temps d'exécution logique extrêmement faible, s'établissant aux alentours de **215 µs**, ce qui représente environ **5.16 %** de la durée totale d'une image. Le serveur reste ainsi inactif pendant environ **3.95 ms** sur les **4.16 ms** disponibles pour chaque image. Toutefois, une légère accélération temporelle est observée, la durée totale de simulation se réduisant à **1 minute et 50 secondes**, soit un écart de l'ordre de **-10 secondes** par rapport à la durée attendue. Ce phénomène, bien plus modéré, pourrait s'expliquer par des optimisations internes propres au moteur, indépendantes de l'architecture ECS elle-même.

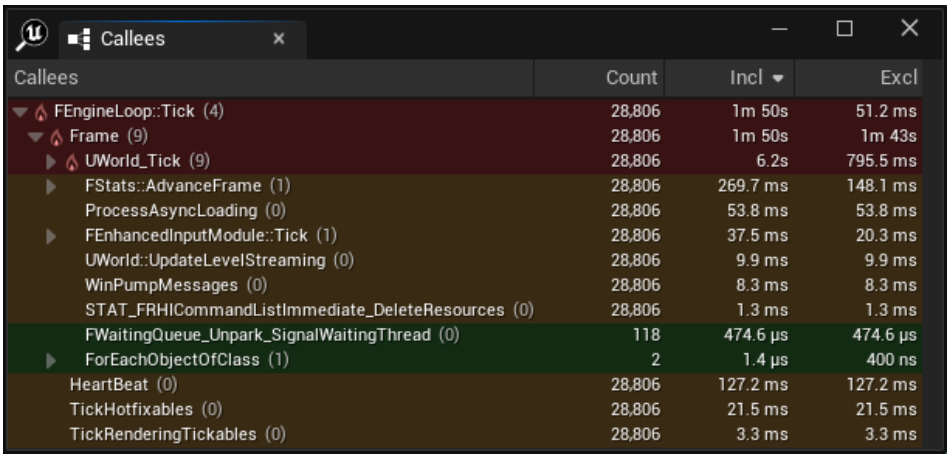

FIGURE 4.12 : *Profiling* de l'ensemble des images pour la version ECS 240FPS

Dans ce contexte, les deux architectures manifestent des comportements nettement différenciés : tandis que l'implémentation orientée objet dépasse largement sa capacité de traitement, provoquant une dégradation majeure, voire rendant l'expérience de jeu injouable, la version ECS maintient une efficacité d'exécution élevée tout en étant encore loin de son seuil d'efficience maximum, avec un temps de calcul par image représentant une fraction marginale (**5.16 %**) de la durée totale allouée.

4.5 DISCUSSION SUR LES RÉSULTATS

L'ensemble des résultats expérimentaux obtenus dans cette étude démontre de manière concluante la supériorité de l'architecture ECS par rapport à la POO, tant en termes de consommation énergétique que d'efficience d'exécution, dans un contexte de serveurs de jeux multijoueurs en ligne. Tout d'abord, sur l'aspect de la consommation énergétique, il est observé que la version ECS du projet consomme systématiquement moins d'énergie par image que la version orientée objet, et ce pour toutes les fréquences d'images testées (30, 60, 120 et 240 IPS). Cet écart se manifeste dès les faibles fréquences, mais tend à devenir encore plus significatif lorsque la charge de calcul par seconde augmente, comme c'est le cas aux

fréquences de 120 IPS et 240 IPS. L'ECS parvient à maintenir une consommation énergétique inférieure, avec des écarts moyens d'environ un Joule par image dans certaines conditions expérimentales, traduisant une meilleure exploitation des ressources matérielles.

Au niveau de l'efficience temporelle, les résultats montrent également une nette supériorité de l'ECS. À faible fréquence d'images (30 IPS), bien que les deux architectures présentent de larges phases d'inactivité, la version ECS réalise les traitements logiques en un temps extrêmement réduit par rapport à la durée allouée à chaque image. Cet avantage devient encore plus critique à haute fréquence : à 120 IPS et surtout à 240 IPS, la version orientée objet dépasse les capacités de traitement du serveur, entraînant une dilatation importante du temps de simulation (**+40%** à 120 IPS et **+167.5%** à 240 IPS), ce qui détériore sévèrement la qualité de service. À l'inverse, la version ECS parvient à maintenir un traitement logique rapide, avec une occupation temporelle minimale, garantissant le respect de la fréquence d'images cible sans compromettre la fluidité du *gameplay*.

L'analyse des données met également en lumière une plus grande stabilité énergétique de l'ECS, particulièrement à haute fréquence. Les valeurs de consommation maximales pour l'ECS présentent une dispersion plus faible que celles relevées pour la POO, notamment à 240 IPS, ce qui démontre une meilleure prévisibilité des performances énergétiques. Cette stabilité est un atout majeur pour le dimensionnement et la planification des ressources dans un environnement de serveurs de jeux massivement multijoueur.

Sur le plan des implications industrielles, ces résultats indiquent que l'utilisation de l'architecture ECS pourrait permettre d'optimiser la densité d'instances serveur par machine physique, tout en réduisant la consommation énergétique globale. Cette capacité à maximiser l'efficience des serveurs s'inscrit dans une dynamique actuelle de réduction de l'empreinte

carbone des infrastructures numériques (*green IT*), particulièrement critique dans un contexte de croissance continue du marché des jeux en ligne.

En résumé, les résultats obtenus confirment que l'adoption de l'architecture ECS permet de répondre simultanément aux exigences de haute performance et d'efficience énergétique des serveurs de jeux multijoueurs modernes. La capacité de l'ECS à offrir une consommation réduite, une stabilité renforcée et une adaptabilité à des framerates élevés en fait une alternative stratégique de premier ordre face aux approches classiques orientées objet. Ces observations ouvrent ainsi la voie à de nouvelles perspectives de recherche et d'optimisation dans le domaine du développement serveur pour les jeux vidéo.

CHAPITRE V
DISCUSSION GÉNÉRALE

5.1 SYNTHÈSE DES RÉSULTATS ET CONSTATS MAJEURS

L'ensemble des analyses réalisées tout au long de ce mémoire a permis de dresser un constat clair : l'architecture ECS surpasse significativement la POO sur le plan de l'efficacité énergétique et du temps de calcul, dans le contexte d'un serveur de jeu multijoueur en temps réel. Cette conclusion se fonde sur des séries de tests menés à différentes fréquences d'images (IPS), qui ont permis d'observer l'évolution du comportement des deux architectures.

Les données ont démontré que, pour une session de jeu standard, la version ECS consomme systématiquement moins d'énergie que la version POO. Cette économie énergétique s'explique en grande partie par l'organisation mémoire optimisée, la réduction du nombre de sauts d'instruction et la meilleure localité des données dans l'approche ECS, qui minimise les appels redondants et maximise l'usage du cache processeur.

De plus, les temps de calcul associés aux tâches critiques du jeu, comme la mise à jour des entités (vaisseaux, projectiles, etc.), ont montré des écarts considérables : à 240 IPS, l'architecture POO affiche un temps de traitement d'environ 11.5 ms par image, bien au-delà du budget de 4.16 ms imposé par la cadence, provoquant un ralentissement marqué de la simulation. À l'inverse, l'ECS se contente d'environ 215 µs pour ces mêmes opérations, soit à peine 5.16 % de la durée d'une image, lui permettant de maintenir aisément la cadence cible, sans effort particulier du moteur.

5.2 EXPLORATION DES LIMITES : COMPORTEMENT À PLEINE CHARGE

Malgré ces résultats favorables, il convient de nuancer ces observations. Les tests ont été menés dans des contextes contrôlés, où le processeur n'était pas saturé, laissant entrevoir une efficacité potentielle maximale, mais ne reflétant pas nécessairement les situations de charge extrême.

En effet, si l'on souhaite généraliser les conclusions à des environnements de production à haute densité, il devient indispensable d'observer le comportement de chaque architecture lorsque le processeur est sollicité à pleine capacité. Il est alors possible que la consommation énergétique de l'ECS augmente, sans pour autant excéder celle de la POO. Cette interrogation demeure ouverte et constitue une piste de recherche incontournable pour confirmer l'intérêt énergétique de l'ECS dans des contextes hautement concurrentiels.

5.3 POTENTIEL DE SCALABILITÉ MASSIF

L'un des apports majeurs de ce mémoire réside dans la mise en évidence du potentiel de scalabilité de l'ECS. Les performances obtenues suggèrent qu'il serait possible, pour un même budget temporel et sur un même processeur, de faire fonctionner jusqu'à une quarantaine d'instances de serveurs ECS là où une seule instance POO serait possible (dans le cadre du prototype d'expérimentation).

Ce résultat ouvre deux scénarios d'application majeurs dans l'industrie du jeu en ligne :

— **Réduction des ressources matérielles nécessaires :** en divisant le nombre de serveurs physiques requis, on peut réduire drastiquement les besoins en infrastructure, les coûts d'exploitation ainsi que l'impact écologique des centres de données.

— **Augmentation exponentielle de la capacité d'accueil :** à l'inverse, on pourrait conserver l'infrastructure actuelle tout en multipliant par une cinquantaine (dans le cadre du

prototype de test) le nombre de parties hébergées simultanément, optimisant ainsi les revenus générés par unité matérielle.

Ce gain de densité, combiné à la faible latence observée dans l'exécution des tâches critiques, positionne l'ECS comme une solution de choix pour les architectures de serveurs massivement distribués.

5.4 ASPECTS TECHNIQUES NON COUVERTS PAR LES TESTS

Bien que l'évaluation expérimentale soit robuste sur les aspects temps de traitement et consommation énergétique processeur, plusieurs dimensions techniques restent inexplorées dans ce travail :

- **Consommation réseau :** les échanges entre client et serveur n'ont pas été mesurés. Or, selon la structure des messages réseau et la manière dont les entités sont sérialisées, l'ECS pourrait introduire une surcharge ou, au contraire, permettre des transferts plus efficaces. Ce facteur est crucial dans un contexte multijoueur massivement connecté.
- **Utilisation mémoire :** si l'on suppose que l'ECS exploite mieux la mémoire cache, des mesures précises de l'utilisation des caches L1, L2 ou encore de la RAM globale auraient permis de confirmer cette hypothèse et de renforcer l'analyse énergétique.
- **Portabilité sur d'autres architectures matérielles :** les tests ont été exclusivement réalisés sur des processeurs x86. Étendre l'évaluation à des plateformes ARM, de plus en plus utilisées dans les centres de données pour leur rendement énergétique, permettrait de valider la généralité des conclusions et d'explorer de nouveaux rapports performance/consommation.

5.5 APPLICATIONS CLIENT : VERS UNE MEILLEURE EXPÉRIENCE UTILISATEUR

Les bénéfices identifiés côté serveur peuvent également être exploités dans le développement de jeux côté client. Sur les appareils mobiles ou les ordinateurs à faibles performances, la capacité de traitement allégée offerte par l'ECS pourrait se traduire par :

— Une meilleure fluidité d'exécution, réduisant les chutes de fréquence d'images et les latences.

— Une réduction de la consommation énergétique, prolongeant l'autonomie des appareils portables.

— Une capacité à intégrer davantage d'objets ou d'effets visuels en temps réel, sans compromettre la stabilité de l'application.

Dans cette optique, l'ECS s'inscrit comme un levier d'optimisation global de l'expérience de jeu, tant sur le plan serveur que client, et constitue un paradigme de plus en plus pertinent dans l'industrie vidéoludique moderne.

5.6 CONCLUSION DE LA DISCUSSION

Cette discussion générale a permis de mettre en avant les apports, limites et perspectives du travail mené. Les résultats obtenus en faveur de l'ECS ouvrent la voie à une redéfinition des architectures logicielles dans le domaine des serveurs de jeux en ligne, en intégrant les enjeux énergétiques et de scalabilité. Toutefois, pour consolider ces résultats, des analyses complémentaires doivent être entreprises sur d'autres couches techniques (réseau, mémoire, plateformes ARM) et dans des environnements de production à charge réelle.

En définitive, ce mémoire propose une fondation solide pour penser la prochaine génération de serveurs de jeux éco-performants, capables de répondre aux exigences croissantes

du marché tout en minimisant leur empreinte écologique. Il invite à reconsidérer les choix d'architecture logicielle non seulement sous l'angle de la performance, mais également dans une démarche durable et systémique.

CONCLUSION

Ce mémoire s'est inscrit dans une démarche d'analyse comparative entre deux paradigmes d'architecture logicielle — la Programmation Orientée Objet (POO) et l'Entité-Composant-Système (ECS) — appliqués au développement de serveurs de jeux multijoueurs en ligne. L'objectif principal était d'évaluer l'impact de ces deux approches sur la consommation énergétique et l'efficience des traitements logiques, notamment dans des contextes de fréquences d'images variées, allant de 30 IPS à 240 IPS.

La revue de littérature a mis en évidence les limitations énergétiques inhérentes à la POO, ainsi que les promesses de l'approche ECS pour améliorer la gestion des ressources matérielles grâce à une organisation orientée données. Ces constats théoriques ont motivé la mise en place d'une expérimentation rigoureuse, appuyée sur un projet développé sous *Unreal Engine*, permettant une comparaison directe entre les deux architectures à partir d'un même environnement technique.

Les résultats obtenus à travers les campagnes de tests sont sans équivoque. À toutes les fréquences d'images testées, la version utilisant l'architecture ECS a systématiquement affiché une consommation énergétique inférieure à celle de la version POO. Cet écart, bien que perceptible dès les faibles fréquences, devient particulièrement critique lorsque la fréquence d'images augmente, la version POO montrant alors des signes clairs de saturation du temps de traitement par image, entraînant des phénomènes de dilatation temporelle nuisant directement à la qualité de service. À l'inverse, la version ECS maintient une charge de traitement minimale, assurant une exécution stable et respectueuse des contraintes temporelles même en situation de charge extrême à 240 IPS.

Outre la consommation énergétique, l'analyse de l'efficience du temps de calcul par image a également mis en lumière la capacité de l'ECS à offrir une meilleure stabilité énergé-

tique, caractérisée par des mesures de consommation moins dispersées, en particulier dans les configurations de haute fréquence. Cette propriété est déterminante pour envisager des déploiements massifs de serveurs optimisés, permettant d'augmenter le nombre d'instances hébergées par unité matérielle tout en réduisant l'empreinte énergétique globale.

Ce travail, s'il valide les hypothèses initiales quant à l'intérêt de l'architecture ECS pour les serveurs de jeux multijoueurs, présente néanmoins certaines limites. Les tests ont été réalisés sur une simulation simplifiée de *gameplay* et dans un environnement contrôlé, sans interactions concurrentes ni variabilité réseau. De plus, les mesures ont été effectuées sur une seule configuration matérielle, ce qui pourrait influencer les résultats dans d'autres contextes techniques.

Pour les travaux futurs, plusieurs axes de recherche sont envisageables. Il serait intéressant d'étendre l'expérimentation à des simulations multijoueurs complexes avec des charges réseau réelles, afin d'évaluer l'impact de l'ECS dans des scénarios plus représentatifs d'une exploitation en production. Par ailleurs, l'étude de l'intégration d'un ECS natif au moteur *Unreal Engine* ou le portage vers d'autres moteurs de jeu pourraient également ouvrir de nouvelles perspectives d'optimisation. Il serait également judicieux d'étendre les expérimentations à d'autres architectures matérielles émergentes, telles que l'architecture processeur ARM, afin d'évaluer la généricité des résultats obtenus.

En définitive, ce mémoire confirme que l'architecture Entité-Composant-Système constitue une alternative pertinente et performante à la Programmation Orientée Objet pour le développement de serveurs de jeux multijoueurs. En conciliant efficience énergétique, stabilité et haute performance, l'ECS s'impose comme une voie prometteuse pour répondre aux défis techniques et environnementaux croissants du secteur du jeu vidéo en ligne.

BIBLIOGRAPHIE

[1] B. J. Abraham, *Digital games after climate change*. Springer, 2022.

[2] J. A. Gallego Arrubla, Y. M. Ko, R. J. Polansky, E. Pérez, L. Ntaimo, et N. Gautam, "Integrating virtualization, speed scaling, and powering on/off servers in data centers for energy efficiency," *IIE Transactions*, vol. 45, n° 10, pp. 1114–1136, 2013.

[3] K. Salen et E. Zimmerman, *Rules of Play : Game Design Fundamentals*. MIT Press, 2004.

[4] J. Juul, *Half-Real : Video Games between Real Rules and Fictional Worlds*. MIT Press, 2005.

[5] E. Adams, *Fundamentals of Game Design*. New Riders, 2014.

[6] J. Novak, *Game Development Essentials : An Introduction*. Cengage Learning, 2012.

[7] K. Collins, *Game Sound : An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design*. MIT Press, 2008.

[8] M.-L. Ryan, *Narrative as Virtual Reality : Immersion and Interactivity in Literature and Electronic Media*. Johns Hopkins University Press, 2001.

[9] H. Jenkins, "Game design as narrative architecture," *Computer*, vol. 44, n° 3, pp. 118–130, 2004.

[10] J. H. Murray, *Hamlet on the Holodeck : The Future of Narrative in Cyberspace*. MIT Press, 1997.

[11] B. N. Laboratory, "The first video game ?" 2008. [En ligne]. Repéré à : https://www.bnl.gov/about/history/firstvideo.php

[12] S. L. Kent, *The Ultimate History of Video Games : From Pong to Pokémon and Beyond... The Story Behind the Craze that Touched Our Lives and Changed the World*. Three

Rivers Press, 2001.

[13] M. J. P. Wolf, *The Video Game Explosion : A History from PONG to PlayStation and Beyond*. Greenwood Press, 2007.

[14] G. Booch, *Object Oriented Design with Applications.* Benjamin-Cummings Publishing Co., 1991.

[15] B. Liskov et J. Guttag, *Abstraction and Specification in Program Development.* MIT Press., 1986.

[16] B. Meyer, *Object-Oriented Software Construction.* Englewood Cliffs : Prentice hall, 1997.

[17] B. Stroustrup, *The C++ programming language*. Addison-Wesley, 2013.

[18] K. Nygaard et O.-J. Dahl, "The development of the simula languages," dans *History of programming languages*. Academic Press, 1978, pp. 439–480.

[19] G. Booch, *Object-Oriented Analysis and Design with Applications*. Benjamin/Cummings Publishing Company, 1994.

[20] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, et W. Lorensen, *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[21] B. Stroustrup, "The essence of data-oriented design," dans *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2013, pp. 1–9, accessed : 2024-06-14. [En ligne]. Repéré à : https://dl.acm.org/doi/10.1145/2491956.2462170

[22] R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002.

[23] D. Lea, *Concurrent Programming in Java : Design Principles and Patterns*. Addison-Wesley, 2000.

[24] H. Sutter, *Exceptional C++ Style : 40 New Engineering Puzzles, Programming Problems, and Solutions*. Addison-Wesley, 2004.

[25] J. D. Bayliss, "The data-oriented design process for game development," vol. 55, n° 5, pp. 31–38, conference Name : Computer. [En ligne]. Repéré à : https://ieeexplore.ieee.org/abstract/document/9771161

[26] P.-M. Straume, "Investigating data-oriented design," accepted : 2020-09-14T14 :08 :07Z. [En ligne]. Repéré à : https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2677763

[27] J. D. Bayliss, "Developing games with data-oriented design," dans *Proceedings of the 6th International ICSE Workshop on Games and Software Engineering : Engineering Fun, Inspiration, and Motivation*. ACM, pp. 30–36. [En ligne]. Repéré à : https://dl.acm.org/doi/10.1145/3524494.3527626

[28] N. Wirth, *Algorithms + Data Structures=programs*. Prentice-Hall Englewood Cliffs, NJ, 1976, vol. 158.

[29] R. Hickey, "The value of values," dans *Clojure Conj*, 2010. [En ligne]. Repéré à : https://www.youtube.com/watch?v=-6BsiVyC1kM

[30] C. Lattner, "Llvm and clang : Next generation compiler technology," dans *The BSD conference*, vol. 5, 2008, pp. 1–20.

[31] C. Okasaki, *Purely functional data structures*. Cambridge University Press, 1999.

[32] R. Hickey, *Data-Oriented Programming in Clojure*. Manning Publications, 2020, accessed : 2024-06-14.

[33] A. Ma, "Entity systems are the future of mmog development." [En ligne]. Repéré à : https://t-machine.org/index.php/2007/09/03/entity-systems-are-the-future-of-mmog-development-part-1/

[34] J. L. Hennessy et D. A. Patterson, *Computer architecture : a quantitative approach*. Elsevier, 2011.

[35] A. J. Smith, “Cache memories,” *ACM Computing Surveys (CSUR)*, vol. 14, n° 3, pp. 473–530, 1982.

[36] W. Stallings, *Computer organization and architecture : designing for performance*. Pearson Education India, 2003.

[37] D. McFarlin, L. Adiseshu, S. Mysore, A. H. Majidimehr, A. Kumar, et G. S. Sohi, “Pushing the limits of RTL simulation for SoC verification,” *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, n° 4, pp. 1–32, 2013.

[38] M. D. Hill, *Aspects of cache memory and instruction buffer performance*. University of California, Berkeley, 1987.

[39] M. Lorenz, L. Wehmeyer, et T. Dräger, “Energy aware compilation for dsps with simd instructions,” *ACM SIGPLAN Notices*, vol. 37, n° 7, pp. 94–101, 2002.

[40] L. Cailloce, “Numérique : le grand gâchis énergétique.” [En ligne]. Repéré à : http://web.archive.org/web/20180705150801/https://lejournal.cnrs.fr/articles/numerique-le-grand-gachis-energetique

[41] Anonyme, “Pete-energy star : Energy star program,” vol. 60, n° 9, p. pp 40. [En ligne]. Repéré à : https://www.proquest.com/openview/2c03610c5ef6bbeab9ec4a5f14d3333f/1?pq-origsite=gscholar&cbl=34757#

[42] B. Saha, Dr. B.C Roy, et M. Abul kalam Azad, “Green computing current research trends,” vol. 6, n° 3, pp. 467–469. [En ligne]. Repéré à : http://www.ijcseonline.org/full_paper_view.php?paper_id=1830

[43] J. G. Koomey, “Estimating total power consumption by servers in the u.s. and the world,” *Lawrence Berkeley National Laboratory*.

[44] S. Murugesan, “Harnessing green it : Principles and practices,” *IT Professional*, vol. 10, n° 1, pp. 24–33, 2008.

[45] J. Koomey, *Growth in Data Center Electricity Use 2005 to 2010*. Analytics Press, 2011.

[46] R. R. Harmon et N. Auseklis, “Sustainable it services : Assessing the impact of green computing practices,” *Proceedings of the 2009 40th Annual Hawaii International Conference on System Sciences*, pp. 1–10, 2009.

[47] S. Maleki, C. Fu, A. Banotra, et Z. Zong, “Understanding the impact of object oriented programming and design patterns on energy efficiency,” dans *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*, octobre 2017, pp. 1–6. [En ligne]. Repéré à : https://ieeexplore.ieee.org/abstract/document/8323605

[48] F. Pouhela, D. Krummacker, et H. D. Schotten, “Entity component system architecture for scalable, modular, and power-efficient IoT-brokers,” dans *2023 IEEE 21st International Conference on Industrial Informatics (INDIN)*. IEEE, pp. 1–6. [En ligne]. Repéré à : https://ieeexplore.ieee.org/document/10218094/

[49] X. Li et J. P. Gallagher, “An Energy-Aware Programming Approach for Mobile Application Development Guided by a Fine-Grained Energy Model,” mai 2016, arXiv :1605.05234. [En ligne]. Repéré à : http://arxiv.org/abs/1605.05234

[50] B. Anand, A. L. Ananda, M. C. Chan, L. T. Le, et R. K. Balan, “Game action based power management for multiplayer online game,” dans *Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*, ser. MobiHeld ’09. New York, NY, USA : Association for Computing Machinery, août 2009, pp. 55–60. [En ligne]. Repéré à : https://dl.acm.org/doi/10.1145/1592606.1592619

[51] A. E. Trefethen et J. Thiyagalingam, “Energy-aware software : Challenges, opportunities and strategies,” *Journal of Computational Science*, vol. 4, n° 6, pp. 444–449, novembre 2013. [En ligne]. Repéré à : https://linkinghub.elsevier.com/retrieve/pii/S1877750313000173

[52] E. G. Inc., “The most powerful real-time 3d creation tool - unreal engine,” 2024, accessed : 2024-03-18. [En ligne]. Repéré à : https://www.unrealengine.com/

[53] G. Fettweis et E. Zimmermann, “Ict energy consumption-trends and challenges,” dans *Proceedings of the 11th international symposium on wireless personal multimedia communications*, vol. 2, n° 4. (Lapland Finland, 2008, p. 6.

[54] C. Ana, “Reducing the energy use of video gaming : energy efficiency and gamification.”

Copenhagen Center On Energy Efficiency, Rapport Technique, 2020.

[55] M. V. Birk, M. A. Friehs, et R. L. Mandryk, “Age-based preferences and player experience : A crowdsourced cross-sectional study,” dans *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, ser. CHI PLAY ’17. New York, NY, USA : Association for Computing Machinery, 2017, p. 157–170. [En ligne]. Repéré à : https://doi.org/10.1145/3116595.3116608

[56] W. Tom, “New free report : Explore the global games market in 2023,” Newzoo, Amsterdam, Netherlands, Rapport Technique, 2023. [En ligne]. Repéré à : https://newzoo.com/resources/blog/explore-the-global-games-market-in-2023

[57] B. Saha, “Green computing : current research trends,” *International Journal of Computer Sciences and Engineering*, vol. 6, n° 3, pp. 467–469, 2018.

[58] S. Murugesan, “Harnessing Green IT : Principles and Practices,” vol. 10, n° 1, pp. 24–33.

[59] J. Mancebo, C. Calero, F. García, M. Á. Moraga, et I. G.-R. de Guzmán, “Feetings : framework for energy efficiency testing to improve environmental goal of the software,” *Sustainable Computing : Informatics and Systems*, vol. 30, p. 100558, 2021.

[60] R. Van Solingen, V. Basili, G. Caldiera, et H. D. Rombach, “Goal question metric (gqm) approach,” *Encyclopedia of software engineering*, 2002.

[61] B. Henderson-Sellers, “Method engineering for oo systems development,” *Communications of the ACM*, vol. 46, n° 10, p. 73 – 78, 2003.

[62] W. Goethert et J. Siviy, *Applications of the indicator template for measurement and analysis*. Citeseer, 2004.

[63] ISO/IEC, “Iso/iec 15939 : 2017 systems and software engineering–measurement process,” 2017.

[64] A. Hindle, “Green mining : A methodology of relating software change and configuration to power consumption,” *Empirical Software Engineering*, vol. 20, n° 2, pp. 374–409,

2015.

[65] K. T. Claypool et M. Claypool, “On frame rate and player performance in first person shooter games,” *Multimedia systems*, vol. 13, n° 1, pp. 3–17, 2007.

[66] J. Wang, R. Shi, W. Zheng, W. Xie, D. Kao, et H.-N. Liang, “Effect of frame rate on user experience, performance, and simulator sickness in virtual reality,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, n° 5, pp. 2478–2488, 2023.

[67] S. Liu, A. Kuwahara, J. J. Scovell, et M. Claypool, “The effects of frame rate variation on game player quality of experience,” dans *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, ser. CHI ’23. New York, NY, USA : Association for Computing Machinery, 2023. [En ligne]. Repéré à : https://doi.org/10.1145/3544548.3580665

[68] M. Claypool et K. Claypool, “Perspectives, frame rates and resolutions : it’s all in the game,” dans *Proceedings of the 4th International Conference on Foundations of Digital Games*, ser. FDG ’09. New York, NY, USA : Association for Computing Machinery, 2009, p. 42–49. [En ligne]. Repéré à : https://doi.org/10.1145/1536513.1536530

[69] J. Mancebo, F. Garcia, et C. Calero, “A process for analysing the energy efficiency of software,” *Information and Software Technology*, vol. 134, p. 106560, 2021.

[70] D. Djaouti, J. Alvarez, J.-P. Jessel, G. Methel, et P. Molinier, “The nature of gameplay : a videogame classification,” dans *Cybergames Conference*, 2007.

[71] C. Crawford, “Chris crawford on game design,” *New Riders*, 2003.

APPENDICE A

FONCTIONNEMENT AVANCÉ DU CACHE PROCESSEUR

A.1 MÉMOIRE ASSOCIATIVE

La mémoire associative, également connue sous le nom de mémoire associative par contenu (CAM, *Content Addressable Memory*), est une forme de mémoire utilisée dans les processeur pour accélérer la recherche et la récupération de données. Contrairement à la mémoire conventionnelle où les données sont accessibles via des adresses spécifiques, la mémoire associative permet la recherche de données par contenu, ce qui signifie que l'on peut interroger la mémoire pour trouver toutes les entrées qui correspondent à une valeur donnée. Cette caractéristique est particulièrement utile dans les caches processeur pour des opérations telles que la traduction d'adresses et la gestion des tables de pages, où des correspondances rapides entre les adresses mémoire virtuelle et physique sont nécessaires [34].

La mémoire associative améliore les performances du processeur en réduisant le temps de recherche des données, car toutes les lignes de la mémoire sont comparées simultanément en une seule opération de cycle d'horloge [36]. Les applications typiques de la mémoire associative incluent les caches de traduction d'adresses (TLB, *Translation Lookaside Buffer*), qui jouent un rôle crucial dans la gestion de la mémoire virtuelle en permettant des traductions rapides d'adresses mémoire [35]. De plus, cette mémoire est utilisée dans les tables de routage de réseaux et d'autres systèmes nécessitant une recherche rapide par contenu [37].

A.2 POLITIQUE DE REMPLACEMENT

La politique de remplacement dans un processeur est une stratégie utilisée pour déterminer quelles données doivent être évincées du cache lorsqu'il est plein et qu'une nouvelle

donnée doit y être placée. Cette politique est cruciale pour maintenir des performances optimales du cache en minimisant les temps d'accès aux données les plus fréquemment utilisées. Parmi les nombreuses politiques de remplacement, trois des plus courantes sont LRU (*Least Recently Used*), FIFO (*First In, First Out*) et Random :

1. **LRU** : La politique LRU (*Least Recently Used*) remplace la donnée la moins récemment utilisée dans le cache. Cette stratégie est basée sur l'hypothèse que les données récemment utilisées seront probablement réutilisées dans un avenir proche, ce qui optimise l'utilisation du cache en gardant les données actives accessibles [34].
2. **FIFO** : la politique FIFO (*First In, First Out*) évince la donnée la plus ancienne du cache, indépendamment de sa fréquence d'utilisation récente. Cette méthode est simple à implémenter mais peut ne pas être aussi efficace que LRU pour les charges de travail où les données plus anciennes peuvent encore être pertinentes [36].
3. **Random** : la politique Random remplace une ligne de cache choisie aléatoirement. Bien que cette approche soit la plus simple à mettre en œuvre, elle ne prend pas en compte les modèles d'accès aux données et peut donc conduire à des performances sous-optimales par rapport aux autres méthodes [35].

A.3 ÉCRITURE

La stratégie d'écriture dans un processeur se réfère aux méthodes utilisées pour gérer les opérations d'écriture de données dans le cache et la mémoire principale. Ces stratégies sont essentielles pour maintenir la cohérence des données entre le cache et la mémoire tout en optimisant les performances. Les deux principales stratégies d'écriture sont le *write-through* et le *write-back* :

1. *Write-through* : La stratégie *write-through* implique que chaque écriture dans le cache est immédiatement reflétée dans la mémoire principale. Cela garantit que la mémoire principale est toujours à jour avec les données les plus récentes, ce qui simplifie la gestion de la cohérence des données dans les systèmes multiprocesseurs. Cependant, cette méthode peut entraîner une augmentation significative du trafic mémoire, car chaque opération d'écriture dans le cache se traduit par une écriture correspondante en mémoire principale [34].
2. *Write-back* : la stratégie *write-back* diffère en ce que les écritures dans le cache ne sont pas immédiatement propagées à la mémoire principale. Au lieu de cela, les données modifiées sont marquées comme « sales » (*dirty*) dans le cache et ne sont écrites en mémoire principale que lorsque ces données sont évincées du cache. Cette approche réduit le trafic mémoire et peut améliorer les performances en limitant le nombre d'opérations d'écriture en mémoire principale. Cependant, elle complique la gestion de la cohérence des données, notamment dans les environnements multiprocesseurs, car les données les plus récentes peuvent résider uniquement dans le cache [36].

APPENDICE B

PROFILAGE ÉNERGÉTIQUE DANS LES JEUX : INTRODUCTION D'UNE MÉTRIQUE DE CONSOMMATION D'ÉNERGIE BASÉE SUR L'IMAGE

B.1 RÉSUMÉ

L'intégration omniprésente des technologies de l'information et de la communication (TIC) dans la vie quotidienne a intensifié les préoccupations concernant leur impact environnemental, notamment en raison de la consommation énergétique considérable des infrastructures sous-jacentes. Les jeux vidéo, en tant qu'acteurs majeurs du secteur des TIC, contribuent de manière significative à cette consommation. Alors que le marché mondial du jeu vidéo continue de croître, les pratiques de l'informatique verte, visant à réduire l'empreinte environnementale des systèmes numériques, deviennent indispensables. Toutefois, la mesure de la consommation énergétique des jeux vidéo souffre d'un manque de méthode normalisée. Ce travail propose une méthode générique de quantification de la consommation énergétique dans les jeux, fondée sur une métrique performance-énergie exprimée en joules par image (J/image). En combinant deux indicateurs indépendants du contexte — la performance du jeu mesurée en images par seconde (IPS) et la consommation énergétique en joules — cette méthode fournit un calcul générique applicable à tout scénario. Ce travail explore les systèmes de mesure existants, présente une nouvelle solution et discute de ses avantages par rapport aux méthodes actuelles de l'industrie, avant de conclure par des perspectives de recherche en matière de jeux vidéo à haute efficacité énergétique.

B.2 INTRODUCTION

Dans le contexte de la présence croissante des technologies numériques dans la vie quotidienne, le secteur des technologies de l'information et de la communication (TIC) constitue un pilier essentiel de l'économie mondiale [22]. Cette intégration quasi universelle dans presque tous les aspects de notre quotidien soulève des préoccupations majeures quant à l'impact environnemental de ces technologies [53]. Bien que ces systèmes permettent des avancées majeures, ils reposent sur une infrastructure physique massive — serveurs, centres de données, réseaux de distribution — qui consomme d'importantes quantités d'énergie [54]. Parmi les plus grands consommateurs de ressources dans le secteur des TIC, on retrouve le streaming vidéo et les jeux vidéo [23]. La popularité des jeux vidéo auprès de toutes les générations en fait l'un des principaux médias de divertissement des années 2000 [55]. En 2023, l'industrie mondiale du jeu vidéo poursuit sa croissance, générant des revenus considérables : le marché mondial du jeu a atteint 187,7 milliards de dollars en 2023, avec une projection d'augmentation à 212,4 milliards de dollars d'ici 2026 [56]. Avec une telle présence sur le marché, il devient impératif que l'industrie du jeu vidéo joue un rôle central dans le développement de l'informatique verte.

Le concept d'« informatique verte » désigne l'ensemble des pratiques et des technologies visant à réduire l'impact environnemental des systèmes informatiques [57]. Cette réduction est obtenue, entre autres, par l'amélioration de l'efficacité des infrastructures logicielles et matérielles du secteur numérique. Cette approche s'applique à l'ensemble du cycle de vie d'un appareil, de sa conception à son recyclage [58]. Une condition essentielle à la mise en œuvre de solutions d'informatique verte est la capacité à mesurer avec précision la

22. https://fr.statista.com/statistiques/570539/chiffre-d-affaires-global-des-tic-2005/, Saisi le 9 juillet 2025
23. https://karmametrix.com/blog/web-sustainability/internet-carbon-footprint-data/, Saisi le 9 juillet 2025

consommation énergétique d'un logiciel. Cependant, il n'existe pas de méthode simple et universelle permettant de quantifier la consommation énergétique d'un logiciel.

Pour aborder cette problématique, il est crucial d'identifier l'unité de mesure la plus appropriée pour évaluer l'efficacité énergétique d'un jeu vidéo. La distinction entre efficacité et efficience peut prêter à confusion. L'***efficience*** fait référence à la quantité de ***moyens*** qu'un système nécessite pour produire des ***résultats***, tandis que l'***efficacité*** renvoie au degré selon lequel un système atteint ses ***objectifs*** visés. Se reporter à la Figure B.1 pour plus de clarté. Dans cet article, nous proposons une approche générique pour mesurer la consommation énergétique d'un jeu vidéo. Selon la définition de l'efficience mentionnée ci-dessus, cette approche repose sur l'hypothèse que les ***résultats*** d'un jeu vidéo sont évalués en nombre d'images, tandis que les ***moyens*** correspondent à l'énergie fournie par le système, exprimée en joules. Par conséquent, il serait possible de mesurer la consommation énergétique d'un jeu en **joules par image** (J/image), faisant de cette mesure un indicateur pertinent de l'efficience d'un jeu vidéo. De plus, cette solution permettrait aux développeurs de prendre des décisions éclairées pour équilibrer l'efficience énergétique tout en atteignant leurs ***objectifs***, tels que la qualité de vie (QoL) de l'utilisateur final.

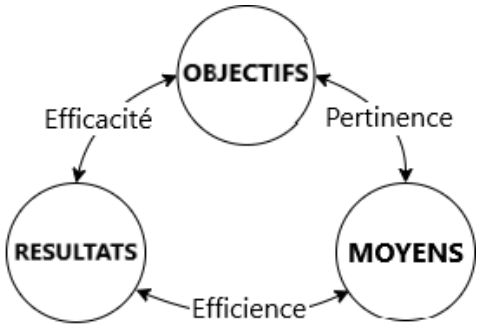

FIGURE B.1 : Efficacité, Efficience et Pertinence

Cet article est structuré en trois sections. Dans la première section, nous présentons les systèmes de calculs actuels ainsi que leurs faiblesses lorsqu'ils sont appliqués aux jeux vidéo. La deuxième section introduit notre solution. Enfin, la troisième section conclut sur la pertinence de notre solution par rapport aux méthodes actuellement employées dans l'industrie, et aborde également les pistes de recherche futures liées à notre méthode de mesure de la consommation énergétique des jeux vidéo.

B.3 DIRECTIVES ACTUELLES DE MESURE DE L'ÉNERGIE ET LEURS LIMITES POUR LES JEUX VIDÉO

Dans l'industrie du jeu vidéo, la mesure de la consommation énergétique des logiciels est devenue de plus en plus cruciale à mesure que l'impact environnemental du divertissement numérique continue de croître. Bien qu'il n'existe pas de méthodes spécifiques pour les jeux vidéo, la sous-section suivante présente les directives les plus récentes en matière de mesure de la consommation énergétique des logiciels.

B.3.1 MÉTHODOLOGIES DE MESURE

La consommation énergétique des logiciels est généralement mesurée en watts, et les avancées récentes ont permis l'introduction d'outils spécialisés visant à accroître la précision et la pertinence de ces mesures. L'une des méthodologies les plus notables est le *imagework for Energy Efficiency Testing to Improve Environmental Goals of the Software* (FEETINGS) [59]. Ce cadre consolide et améliore les directives issues de divers processus de mesure énergétique reconnus, notamment le *Goal Question Metric* (GQM) [60], le *Practical Software and Systems Measurement* (PSM) [61], le GQ(I)M GDSM [62], ainsi que des normes internationales comme l'ISO/IEC/IEEE 15939 :2017 [63]. Ces processus ont été initialement conçus pour

mesurer les performances logicielles, mais FEETINGS en étend la portée afin de se concentrer spécifiquement sur la consommation d'énergie.

Le cadre FEETINGS repose sur l'utilisation d'un *Efficient Energy Tester* (EET), capable de capturer en temps réel la consommation énergétique des composants matériels critiques tels que le processeur (CPU), la carte graphique (GPU) et la mémoire pendant l'exécution du logiciel. Cela revêt une importance particulière dans le domaine du jeu vidéo, où le traitement graphique et la puissance de calcul influencent fortement la consommation énergétique globale. En surveillant ces composants matériels, l'EET fournit non seulement la puissance utilisée par le système (exprimée en watts), mais mesure également le temps requis pour accomplir des tâches spécifiques au sein du dispositif testé (Device Under Test, DUT). Cette double mesure—consommation énergétique en watts et temps d'exécution—permet aux développeurs de calculer avec précision l'efficience énergétique pour différents segments du jeu.

L'importance de cadres comme FEETINGS est soulignée par la variation significative de la consommation énergétique selon les genres de jeux, les configurations matérielles et les pratiques de développement. De plus, des cadres comme *Green Mining* [64] visent à extraire des données de consommation énergétique à partir des dépôts logiciels afin d'identifier des tendances et de proposer des améliorations, soutenant ainsi l'objectif de FEETINGS de fournir des informations exploitables aux développeurs.

En intégrant ces méthodologies complètes, FEETINGS propose une approche plus fine et sensible au contexte pour mesurer et améliorer l'efficience énergétique des logiciels de jeux vidéo, ce qui est essentiel dans la poursuite plus large des objectifs de l'informatique verte.

B.3.2 PROBLÉMATIQUE DE LA MESURE ÉNERGÉTIQUE DANS LES JEUX VIDÉO

Le principal problème des directives actuelles pour l'analyse de la consommation énergétique dans les logiciels de jeux vidéo ne réside pas dans les protocoles de mesure eux-mêmes, mais plutôt dans les unités de mesure utilisées pour l'analyse des résultats. Toutes les directives rapportent la consommation d'énergie en utilisant des unités de type *watt par unité de temps*, telles que le watt-seconde (W.s) ou le watt-heure (W.h).

Bien que ces unités puissent être utiles dans d'autres domaines, elles sont insuffisantes pour évaluer la consommation énergétique dans les applications vidéoludiques. En effet, elles ne tiennent pas compte d'un aspect essentiel de la performance des jeux vidéo : le taux de rafraîchissement des images, ou fréquence d'images [65, 66, 67].

La fréquence d'images est une valeur clé pour les logiciels dont l'exécution dépend du temps réel, notamment les jeux vidéo, car cette unité est fortement dépendante du matériel utilisé [68]. Les études de consommation énergétique dans ce domaine ne devraient pas proposer d'évaluation sans prendre en compte la fréquence d'images. Une métrique plus appropriée pour ce type de recherche serait le **joule par image** (J/image), car elle offre une représentation plus fidèle de l'efficience énergétique d'une solution tout en considérant les performances du dispositif testé (DUT). L'utilisation de cette unité de mesure permettrait une évaluation plus pertinente de la consommation énergétique dans le contexte vidéoludique.

B.4 SOLUTION JOULES PAR IMAGE

La mesure en **joules par image** (J/image) est simple, mais extrêmement utile dans le contexte de l'analyse de la consommation énergétique des jeux vidéo. Cette métrique vise à quantifier la consommation énergétique par image calculée d'une application logi-

cielle, permettant ainsi une évaluation plus pertinente de l'efficience énergétique dans les environnements vidéoludiques.

En se concentrant sur chaque image individuellement, cette métrique réduit la dépendance au matériel en éliminant l'impact des performances du système sur le nombre d'images rendues par seconde. Pour chaque image, la métrique mesure exclusivement la consommation énergétique pour un ensemble d'instructions constant. De plus, le taux de rafraîchissement (*fréquence d'images*) est une métrique relativement simple à récupérer en développement logiciel, ce qui facilite la conversion entre les unités joules par image (J/image) et joules par heure (J/h). En effet, chaque image représente une période de temps de $\frac{1s}{frequence}$, ce qui permet de l'interpréter comme une unité temporelle.

Ce focus sur l'énergie requise pour générer des images individuelles permet une évaluation plus précise de la consommation énergétique, tout en permettant l'analyse du jeu dans sa globalité (moyenne globale en J/image) ou de segments de *gameplay* spécifiques.

Se focaliser sur les images individuelles permet également aux développeurs d'intégrer des outils externes et de scripts de suivi de la consommation énergétique par image, rendant ainsi le processus très accessible et simple à mettre en œuvre.

Pour mesurer une consommation en $J/image$, la première étape consiste à mesurer à la fois la consommation électrique (P) en watts et le taux de rafraîchissement (FPS) sur un intervalle de temps défini (Δt) en secondes. Une fois ces mesures obtenues, on peut calculer la consommation énergétique pour chaque période (i) en multipliant la puissance (P) par la durée de la période (Δt). On calcule ensuite la consommation en $J/image$ pour la période i en divisant J_i par le nombre d'images produites durant cette période, noté N_i.

$$J_i = P_i * \Delta t \tag{B.1}$$

$$N_i = FPS * \Delta t \tag{B.2}$$

$$J/image = \frac{J_i}{N_i} \tag{B.3}$$

À partir de l'équation B.3, on peut simplifier J_i et N_i par Δt et ainsi déterminer l'équation finale suivante :

$$J/image = \frac{P_i}{FPS} \tag{B.4}$$

Cette mesure permet d'uniformiser l'analyse de la consommation énergétique dans des scénarios variés. Lors de tests effectués sur différentes plateformes matérielles avec une base de code identique, les résultats reflètent l'influence du matériel sur la consommation énergétique par image.

Inversement, avec un même matériel et des bases de code différentes, la métrique met en évidence l'impact du logiciel sur cette consommation. Cette double capacité permet aux développeurs de jeux vidéo d'isoler facilement les effets du logiciel et du matériel, tout en fournissant une unité de mesure cohérente pour les comparaisons.

Pour une meilleure compréhension, nous présentons un exemple de mesure en **joules par image** (J/image) avec un intervalle de temps Δt en secondes. Les valeurs suivantes sont purement théoriques et volontairement exagérées afin d'illustrer à la fois la consommation énergétique d'un dispositif testé (DUT) et le taux de rafraîchissement d'un jeu vidéo lors d'un test, voir le tableau B.1.

TABLEAU B.1 : Exemple de consommation énergétique et *fréquence d'images*

Jeu	Images/seconde moyennes	Moyenne en Watt.s
A	60	1000
B	30	800

À partir de ces valeurs, nous pouvons calculer la consommation énergétique en **J/image** des jeux A et B à l'aide de l'équation B.4. Les équations B.5 et B.6 présentent les résultats pour ces deux exemples.

$$A : \frac{J}{\text{image}} = \frac{1000}{60} \approx 17\ J/\text{image} \tag{B.5}$$

$$B : \frac{J}{\text{image}} = \frac{800}{30} \approx 27\ J/\text{image} \tag{B.6}$$

Bien que le jeu A présente une consommation énergétique totale plus élevée que le jeu B, sa consommation par image est inférieure. Cela indique que le jeu A possède une meilleure efficience logicielle et aurait probablement une consommation énergétique plus faible que le jeu B s'ils tournaient à la même fréquence d'images.

Selon les besoins du jeu, les développeurs peuvent surveiller les étapes clés de l'exécution des images comme présenté dans la figure B.2. Les directives telles que FEETINGS proposent plusieurs méthodes pour mesurer la consommation énergétique de différents dispositifs à l'aide d'un EET (Efficient Energy Tester) tel qu'Elliot [69]. Les étapes d'une image dans les jeux vidéo peuvent impliquer divers composants matériels qui doivent être capturés par l'EET lors des tests. Pour identifier les dispositifs clés à surveiller, il est essentiel de comprendre quels composants matériels sont actifs à chaque étape d'image.

Pendant l'étape de ***listen***, tous les dispositifs utilisés pour enregistrer les entrées du joueur, les événements du système d'exploitation ou l'activité réseau doivent être surveillés.

Dans l'étape ***think***, le processeur (CPU) est généralement le composant principal chargé des calculs logiques et de physique. Toutefois, cela peut dépendre de la conception logicielle, certains jeux déléguant les calculs au GPU pour réduire la charge du CPU.

Dans la phase ***speak***, le GPU est l'élément central, chargé de rendre et d'afficher le jeu, bien que d'autres dispositifs utilisés pour fournir un retour au joueur puissent également être impliqués.

En revanche, pour les serveurs de jeu, l'étape ***speak*** est presque négligeable, car la majorité des serveurs n'effectuent aucun rendu graphique, mais communiquent uniquement par le réseau.

Étant donné que ces dispositifs sont actifs tout au long de l'exécution du jeu, leur consommation énergétique doit être prise en compte avec attention lors de la détermination de la consommation en **J/image**. Il est tout aussi important d'éviter d'inclure des dispositifs qui n'interviennent pas dans les étapes d'image afin de garantir la précision des mesures énergétiques.

B.4.1 ÉQUILIBRER QUALITÉ DE VIE ET EFFICACITÉ ÉNERGÉTIQUE

Notre solution s'aligne avec les principes de l'informatique verte, dans le but de suivre et d'optimiser l'efficacité énergétique des jeux vidéo. Cependant, bien que cette métrique repose sur la fréquence d'images, réduire simplement ce dernier ne rend pas nécessairement un jeu plus économe en énergie. Dans certains contextes, une baisse de la fréquence d'images

peut dégrader la qualité de vie (QoL) globale, impactant ainsi négativement la satisfaction et l'expérience de l'utilisateur [65].

Pour intégrer efficacement les pratiques de l'informatique verte dans un projet de jeu vidéo, il est essentiel de trouver un équilibre entre la fourniture d'une expérience utilisateur agréable (***objectifs***) et la réduction de la consommation énergétique du jeu (***moyens***). Par exemple, il serait incohérent de réduire la fréquence d'images d'un jeu à une image par seconde tout en visant une expérience dynamique et rapide, comme dans un jeu de type parkour tel que *Mirror's Edge* [24]. Cela est d'autant plus vrai que la fréquence d'images peut être interprété comme le nombre d'instances de rétroaction fournies au joueur par seconde. Trouver le bon équilibre garantit que les efforts en matière d'efficacité énergétique ne se font pas au détriment de la qualité du *gameplay*.

Comme mentionné précédemment, la fluidité du *gameplay* est un facteur clé dans la perception d'un jeu par le joueur. Chris Crawford décrit les jeux comme une forme de dialogue : « **Un processus cyclique dans lequel deux agents actifs écoutent, pensent et parlent tour à tour (au sens métaphorique)** » [70, 71]. Cette vision simplifiée du jeu s'aligne parfaitement avec les étapes logiques fondamentales d'une image dans un jeu vidéo. D'abord, le jeu ***écoute*** les entrées du joueur et les données réseau entrantes. Ensuite, il ***réfléchit*** en calculant les actions et en mettant à jour le monde du jeu en conséquence. Enfin, il ***parle*** en signalant le nouvel état du jeu via divers types de rétroaction, comme les visuels, les sons ou les vibrations haptiques.

24. https://www.ea.com/fr/games/mirrors-edge/mirrors-edge, Saisi le 9 juillet 2025

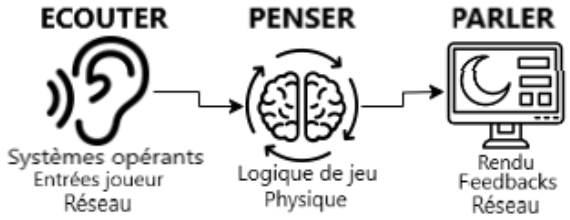

FIGURE B.2 : Étapes d'une image

B.5 CONCLUSION ET TRAVAUX FUTURS

Dans ce travail, nous abordons la préoccupation croissante concernant la consommation d'énergie des technologies de l'information et de la communication (TIC), un problème devenu pressant en raison de l'expansion rapide du secteur au cours de la dernière décennie.

Les jeux vidéo, en tant que composant important de l'écosystème des TIC, offrent une opportunité convaincante de développer des solutions d'informatique verte visant à réduire la consommation d'énergie. Cependant, pour garantir que ces solutions soient véritablement efficaces, il est essentiel de quantifier de manière précise la consommation d'énergie d'un jeu vidéo. À cet égard, nous introduisons le concept de la métrique *joules par image* (J/image).

La métrique des images par seconde (IPS) est une mesure indépendante du contexte qui s'applique universellement à tous les jeux vidéo. Cela la rend particulièrement précieuse, car elle n'est pas liée aux spécificités du matériel exécutant le jeu. Bien que le même jeu puisse tourner à des fréquences d'images différentes sur deux systèmes distincts, les opérations exécutées pour chaque image individuelle restent constantes. Cette universalité permet aux développeurs de jeux et aux chercheurs de calculer la consommation d'énergie d'un jeu de manière indépendante du matériel sous-jacent.

En exploitant cette métrique indépendante du contexte, nous garantissons que la méthode de mesure proposée est applicable dans une grande variété de scénarios, permettant ainsi de

comparer la consommation d'énergie entre différents jeux. De plus, cette métrique facilite la conversion transparente entre les joules par heure et les joules par image, car elle est intrinsèquement liée à la fréquence d'images du jeu, une métrique facilement accessible dans le développement de jeux.

Maintenant que nous avons établi une méthode générique pour mesurer la consommation d'énergie des jeux vidéo, ce calcul peut servir de base pour développer des outils qui facilitent la mise en œuvre de solutions concrètes d'informatique verte.

Dans un avenir proche, et dans le but de valider davantage cette méthode de calcul de la consommation d'énergie des jeux vidéo, plusieurs pistes de recherche méritent d'êtres explorées.

Dans un premier temps, le développement d'un outil automatisé pour calculer la consommation d'énergie en watts par image offrirait aux développeurs de jeux vidéo un moyen simple et efficace de suivre la consommation énergétique de leurs jeux. En intégrant cette approche dans des moteurs de jeux populaires, tels que *Unreal Engine*, il serait possible de généraliser les métriques d'efficacité énergétique, permettant ainsi aux développeurs de visualiser la consommation d'énergie parallèlement aux métriques de performance existantes telles que l'IPS. Cette accessibilité accrue pourrait favoriser l'adoption de pratiques plus économes en énergie au sein de l'industrie. Par exemple, bien que les moteurs actuels affichent des données en IPS, l'incorporation de joules par image pourrait fournir une vue plus complète de la performance d'un jeu.

D'autre part, pour rendre cette mesure aussi complète et précise que possible, il serait utile de développer une formule (ou une notation standard) permettant de comparer les joules par image entre deux jeux avec des fréquences d'images différentes.