Jeux video multijoueur en ligne : application basée sur la technologie blockchain

Encadrant : **M. Potop-Butucaru,**

Etudiants : **A. Atsain, I. Boubrik, M. R. Kadri, D. Phan**

Table des matières

1 Cahier des charges

Le projet a pour objectif de mettre en œuvre un service basé sur l'architecture présentée dans le document [1], permettant la création d'un jeu en ligne utilisant la technologie blockchain. Il est important de noter que le choix du service pourrait être sujet à des modifications après la publication de ce rapport. L'évaluation se concentrera principalement sur les exigences fonctionnelles du service à implémenter.

1.1 Exigences fonctionnelles et exigences non-fonctionnelles

Cette section est dédiée aux exigences fonctionnelles :

1. En ce qui concerne les données
   - La blockchain stocke publiquement et de manière immuable les données échangées entre les services, avec un accès public, une impossibilité de modification, et une structure en blocs numérotés.
   - Pour automatiser les communications au sein de la blockchain, il sera nécessaire d'implémenter des smart contract chargés de gérer ces échanges de manière automatique.
2. En ce qui concerne la mise en œuvre du service
   - Le service doit être en mesure d'établir des communications avec d'autres services spécifiques définis dans l'architecture :
     - Il doit pouvoir approuver ou rejeter un smart contract en fonction des données reçues. Un service sans exigence de signature ne devrait ni approuver ni refuser le contrat (le simple fait de le posséder constitue déjà une anormalie).
     - Il doit seulement traiter des données qui sont attribuées à son service.
     - Il doit être en mesure de transmettre un smart contract au service destiné.
     - Seul le service Publication, à le droit de valider définitivement une smart contract et d'écrire sur la blockchain
   - Les échanges doivent être sécurisés, et pour ce faire, ils seront automatisés à l'aide des smart contracts, une technologie associée à la blockchain.

Dans le cadre de notre projet, les différentes exigences non-fonctionnelles que nous avons identifiées sont les suivantes :

- Performance : Le module doit être capable de traiter efficacement et rapidement plusieurs requêtes provenant d'un service "entrant".
- Sécurité : Il est impératif d'assurer la sécurité des communications entre les divers services existants, car c'est l'objectif fondamental de l'architecture présentée.

1.2 Contraintes Technique

La concrétisation de ce projet nécessitera l'utilisation de technologies spécifiques. Nous avons ainsi approfondi nos connaissances sur les blockchains, y compris les smart contracts qui leur sont associés.

2 Plan de développement

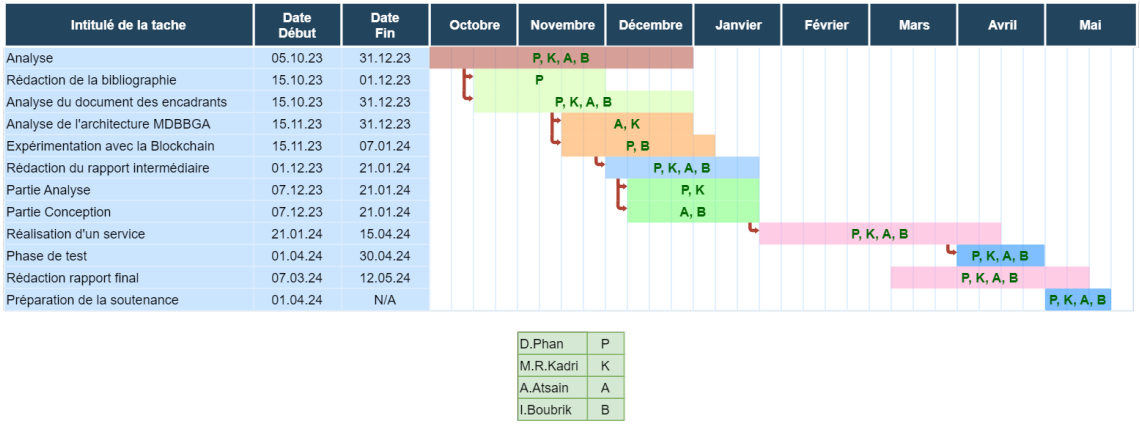


Figure 1 : Diagramme de Gantt du projet

Pour pouvoir mener à bien ce projet, il nous était indispensable d'analyser les termes de notre sujet pour pouvoir realiser une bibliographie adaptée à nos besoins, et mieux comprendre et analyser le document Blockchain-based Massively Multiplayer Online Game framework and Modular decentralized blockchain-based game architecture. Ce document nous a été remis par les encadrants, notre première tâche a été de le comprendre et de présenter ses éléments aux encadrants.

Par la suite, avec les données que nous avions, nous avons effectué des recherches plus approfondies sur l'architecure MDBBGA ainsi que sur des expérimentations sur la blockchain.

Avec toutes ces données, nous avons pu commencer la rédaction de notre rapport intermédiaire Nous nous sommes donc partagés les tâches pour être plus efficace, tout d'abord Kadri et Phan se sont occupés de la partie bibliographie, analyse pendant que Atsain et Boubrik se sont occupés de la partie conception.

Pour la suite du projet, en ayant choisi un service, il nous faudra donc le réaliser. La réalisation du service s'effectuera, suivie de la phase de test qui nous donnerons assez d'éléments pour rédiger notre rapport final et préparer notre soutenance.

3 Bibliographie

Cette partie a pour but de présenter les bibliographies utilisées dans le cadre de notre projet, divisé en trois parties distinctes. La première section explore "l'architecture des jeux en ligne", suivie de "la blockchain" et enfin "la sécurité et les méthodes de triche". L'objectif principal est de fournir des résumés concis de chaque bibliographie, offrant ainsi une compréhension ciblée dans le contexte de notre sujet (jeux vidéos et blockchain).

La bibliographie est une liste organisée des sources documentaires, telles que des livres, articles ou autres références, utilisées ou citées dans le cadre d'un travail académique ou d'une recherche. Elle offre une trace structurée des ressources consultées pour soutenir les informations présentées dans le document. Notez que des modifications peuvent être apportées à cette dernière entre le rendu intermédiaire et le rendu final.

Les références bibliographiques suivantes offrent une réponse préalable aux projets qui nous ont été assignés, à savoir "les jeux vidéo et la blockchain". Elles visent à élaborer une architecture adaptée pour les jeux en ligne intégrant la blockchain en tant que technologie, tout en utilisant les smart contracts pour automatiser divers services. L'objectif est de préciser les raisons justifiant l'utilisation de cette architecture, son applicabilité aux jeux en ligne, ainsi que les avantages et inconvénients associés à cette approche.

La première section vise à analyser les termes du sujet "blockchain et jeux en ligne" et à élaborer une architecture appropriée pour les jeux en ligne. Les articles devront présenter ou orienter vers une architecture adéquate, identifier les éventuels défauts, et proposer des solutions le cas échéant.

Le premier article constitue le document de référence pour notre projet, décrivant l'architecture décentralisée adaptée aux blockchains pour l'espace de stockage. L'ensemble de l'analyse du projet est fondé sur ce document.

[1] ->Architecture supportant le projet

Ce second article introduit une architecture destinée aux jeux MMOG. Il propose une initiation au sujet en exposant une architecture de base, tout en explorant les contraintes imposées par le choix d'un jeu en ligne et les diverses contraintes qui en découlent. De plus, l'article aborde des architectures adaptées aux jeux vidéo, ce qui confère une pertinence directe à notre champ d'étude.

[2] ->Architecture pour les jeux MMOG

Le troisième article expose une architecture intégrant la blockchain en tant que technologie. Il examine les avantages et inconvénients de l'utilisation de cette technologie au sein de l'architecture. Ce document s'inscrit également dans le contexte de notre étude,

puisqu'il aborde les jeux en ligne et explore le marché des jeux mobiles.

[3] ->Architecture basée sur la blockchain

Ce document réalise une analyse approfondie de l'architecture peer-to-peer et met en lumière ses avantages et inconvénients dans le contexte des jeux MMOG. Il souligne également sa nature évolutive. De plus, le document expose les principaux défauts de cette architecture en les classifiant.

[4] ->Architecture P2P pour les jeux MMOG

Ce document expose les lacunes des architectures existantes en les classant selon leurs problèmes pour une analyse approfondie. En complément, le document suivant propose des solutions aux problèmes évoqués précédemment en utilisant les blockchains comme espace de stockage. Cette approche renforce la sécurité des caractéristiques clés d'une architecture P2P traditionnelle, répondant ainsi aux besoins spécifiques des jeux de type MMOG.

[5] ->Solution pour les problèmes d'une architecture P2P
[6] ->Architecture P2P avec espace de stockage en blockchain

Le document à suivre propose des instruments pour définir la portée d'application d'une architecture P2P dans un domaine donné. Cela facilite la délimitation des éléments nécessaires et applicables, distinguant ainsi ceux qui ne le sont pas.

[7] ->Approche pour déterminer un couvrage des types de jeux pour la blockchain

La seconde section a pour objectif d'examiner de manière approfondie la solution proposée, à savoir une architecture Peer-to-Peer intégrant la blockchain en tant qu'espace de stockage. L'analyse se concentrera sur la pertinence de l'utilisation de la blockchain dans le contexte des jeux vidéo, en mettant en lumière les avantages potentiels tels que l'intégration de smart contract. De plus, cette partie abordera les méthodes de triche dans les jeux en ligne, mettant ainsi en évidence les vulnérabilités potentielles de cette architecture..

Le document initial effectue une classification en quatre niveaux et établit un consensus entre le réseau et l'application de la blockchain. L'objectif de cette classification est de mettre en évidence les points faibles tout en fournissant des orientations pour remédier à ces faiblesses.

[8] ->Présentation de la blockchain, de ses domaines d'application, etc...

En utilisant le document suivant, une comparaison est établie entre les différentes block-

chains existantes avec leurs algorithmes de consensus. Cela permet de les classer par ordre de préférence en fonction de chaque type de jeu MMOG et d'utiliser cette classification pour réaliser des analyses.

[9] ->Etude des différents types de blockchain et comparatifs

Le document suivant effectue une analyse détaillée des diverses formes de blockchains, classant à la fois leurs performances et les modèles qu'elles établissent. Grâce à cette analyse, il devient possible de développer une technique pour résoudre le problème lié à l'utilisation d'une blockchain spécifique.

[10] ->Analyse des performances des systèmes basés sur la blockchain

Le document suivant présente une architecture et diverses applications de la blockchain dans l'industrie des jeux vidéo. Bien que l'accent principal soit mis sur l'utilisation de la blockchain dans les jeux vidéo, le document explique également en détail l'architecture associée. Ce document sert principalement de complément aux autres document associés à la blockchain ou encore l'architecture du jeu en ligne.

[11] ->Application de la blockchain dans le domaine des jeux vidéos

La section suivante examine et expose diverses vulnérabilités observables liées à l'utilisation des smart contracts. De plus, elle présente un exemple d'attaque dirigée contre les smart contract. Le document propose également une méthode de détection des vulnérabilités des smart contract appelée le modèle SCVSN (Smart Contract Vulnerability Detection Model Based on Siamese Network). Ce modèle définit des étapes pour évaluer la sécurité des smart contract, suivies d'une analyse approfondie de la performance de ce modèle.

[12] ->Vulnérabilité des smart contract

Les deux livres suivants servent de ressources pour la blockchain et les smart contracts. Ils sont des guides pédagogiques pour mettre en œuvre des smart contracts avec la technologie Ethereum, offrant un support tant pour la phase de développement que pour la compréhension de leur implémentation.

[13] ->Implémenter des smart contract avec Ethereum et Solidity
[14] ->Implémenter des smart contract avec Solidity pour un système décentralisé

Le document qui suit effectue la mise en œuvre d'un jeu au tour par tour en utilisant une architecture Peer-to-Peer. Il aborde la recherche de parties, le déroulement d'une partie et la synchronisation entre les utilisateurs du jeu. Le jeu est conçu avec un système anti-triche, et l'ensemble de cette présentation peut servir de support à la blockchain.

[15] ->Exemple d'un jeux de stratégie en ligne utilisant de la blockchain

Ce document examine une nouvelle méthode de consensus sur les blockchains fondée sur le protocole Brylitte (BCP), qui est employé pour soutenir un écosystème de jeux MMOG. Il propose une analyse approfondie du protocole BCP ainsi que des travaux précédents dans ce domaine. Les algorithmes de BCP sont également présentés, accompagnés d'une simulation permettant d'observer le fonctionnement de cet algorithme.

[16] ->Présentation d'un exemple de support pour les jeux de type MMOG

Le document suivant expose les désavantages associés à l'utilisation d'une architecture Peer-to-Peer et vise à identifier ses vulnérabilités, offrant ainsi des pistes pour résoudre ces problèmes de sécurité. Un document ultérieur aborde spécifiquement ces questions de sécurité dans le contexte d'une architecture P2P. Il présente les aspects sécurisés permettant de défendre cette architecture contre des attaques telles que le DDoS. De plus, il analyse cette nouvelle architecture en restreignant son domaine de détection des fraudes.

[17] ->Sécurisé une architecture P2P
[18] ->Blockchain contre la triche dans les jeux en ligne

Ce document examine en détail les différentes méthodes de triche couramment employées dans les jeux MMOG et les catégorise. Cette classification a pour but de simplifier l'identification des vulnérabilités spécifiques au genre de jeu sélectionné pour l'implémentation, tout en suggérant des solutions potentielles pour contrer ces diverses méthodes de triche.

[19] ->Analyse des méthodes de triche

Ce document propose une nouvelle approche exploitant la blockchain comme technologie pour résoudre les multiples problèmes de sécurité découlant des jeux en ligne et de l'utilisation de l'architecture Peer-to-Peer. Des technologies telles que les contrats intelligents sont employées pour les éléments critiques, et deux méthodes de transfert sont mises en œuvre pour sécuriser les transactions. Ce document met en évidence les solutions élaborées et démontre des performances supérieures à celles d'une architecture classique.

[20] ->Architecture P2P utilisant de la blockchain pour lutter contre la triche dans les jeux en ligne

Les bibliographies suivantes sont indépendantes et ne sont associées à aucune section spécifique.

Le lien suivant est une page Wikipédia décrivant le botnet, un type d'attaque. Dans un botnet, des appareils infectés par des logiciels malveillants sont transformés en bots, qui sont ensuite contrôlés par un cybercriminel.

[21] ->Présentation des attaques botnet

Le document suivant décrit une architecture de réseau basée sur le SDN, adoptant une méthode hybride pour intégrer les politiques de réseau et automatiser l'installation des règles de flux sur les dispositifs de réseau.

[22] ->Architecture réseau basée sur SDN

Le dernier document examine les différentes formes de tricherie dans les jeux en ligne, ainsi que les défis spécifiques qu'elles posent pour les architectures en réseau P2P et C/S. Il classe ces tricheries par type et analyse leur impact. Le texte aborde également les solutions potentielles pour contrer la tricherie dans les systèmes C/S, tout en soulignant leurs limites en termes de scalabilité et de gestion de la sécurité.

[23] ->Analyse des différentes formes de triches dans des jeux en ligne

4 Analyse

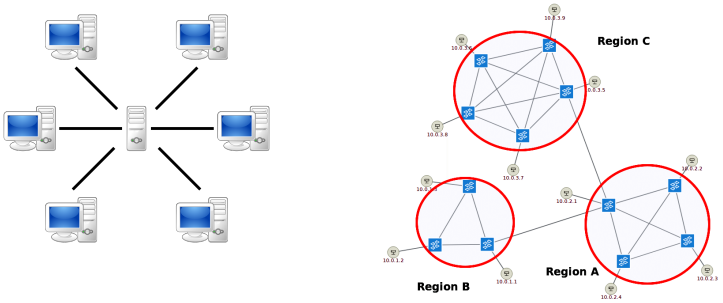
4.1 Introduction

Dans les jeux vidéo modernes, l'aspect multijoueurs et sa fiabilité joue un rôle crucial dans l'expérience des joueurs. Les architectures classiques utilisées pour ce mode sont centralisées et sont donc confrontées à des défis en matière d'évolutivité et de sécurité, en particulier avec la croissance de la base de joueurs et de la complexité des interactions dans le jeu. Dans cette partie Analyse, nous allons examiner le potentiel de la technologie blockchain pour les jeux multijoueurs avec les modèles Peer-to-peer basés sur la blockchain afin de résoudre les problèmes d'évolutivité et à renforcer la sécurité des jeux vidéo et en particulier les MMO.

4.2 Architecture des jeux vidéos multijoueurs en ligne

Un jeu vidéo est un jeu électronique qui permet au joueur d'interagir avec un environnement virtuel à travers des interfaces homme-machine (Manette, souris, etc.) et de générer une réponse visuelle, auditive ou haptique sur une interface de sortie adapté (Écran, casque audio, etc.). Il existe différentes catégories de jeux vidéo, chacun offrant des styles de jeu et de narration distincts : action, aventure, jeu de rôle, simulation, sport, stratégie, etc. Chaque catégorie présente des défis et des objectifs uniques, qui définissent l'expérience du joueur dans le monde virtuel.

De nombreux jeux vidéo proposent des modes multijoueurs, permettant la participation simultanée de plusieurs joueurs, qu'ils fassent partie de la même équipe ou de l'équipe adverse, ou qu'ils agissent en solo. Les expériences multijoueurs peuvent être classées en deux styles différents : Joueur contre Joueur ou PvP (De l'anglais "Player vs Player) et Joueur contre l'Environnement ou PvE (De l'anglais "Player vs Environment). En PvP, les joueurs s'affrontent entre eux, tandis qu'en PvE, ils coopèrent contre les PNJ (Personnages Non-Joueurs) contrôlés par l'IA (Intelligence Artificielle) du jeu.



(a) Architecture Client-serveur centralisée [21]

(b) Architecture multiserveurs [22]

Figure 2 : Représentation d'une architecture C/S et d'une architecture mutliserveur

Pour l'implémentation du mode multijoueur, l'approche la plus répandue au début était l'architecture client-serveur. Celle-ci consiste à utiliser un seul serveur central chargé d'héberger le monde virtuel du jeu, de gérer les interactions entre les joueurs et les événements du jeu et d'assurer la synchronisation de tous les clients connectés. Bien qu'elle soit efficace pour les jeux dont la population est réduite, ce type d'architecture rencontre des problèmes d'évolutivité à mesure que la popularité du jeu augmente. Le serveur central peine à gérer efficacement la demande croissante, entrainant des problèmes de lag, de latence et une dégradation générale de l'expérience de jeu. En plus, de par sa nature centralisée, ce type d'architecture est vulnérable aux attaques de déni de service (Distributed Denial of Service ou DDoS). Les développeurs de jeux ont donc opté pour des architectures multiserveurs qui consiste à répartir les fonctionnalités du jeu sur plusieurs serveurs, dont chacun s'occupe d'une région ou d'une fonction spécifique. Bien que cela permette de résoudre certains problèmes d'évolutivité, elle introduit des complications telles que le verrouillage géographique (Region lock) des clients et l'augmentation des coûts opérationnels.

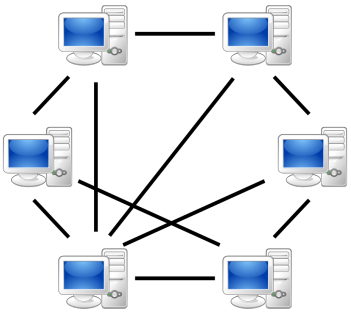


Figure 3 : Architecture Peer-to-peer [21]

Pour venir à bout de ces limitations, nous avons enfin l'architecture Peer-to-Peer (P2P). Il s'agit d'un système décentralisé dans lequel les joueurs communiquent directement entre eux plutôt que de dépendre d'un serveur central. Si les architectures P2P offrent des avantages tels qu'une meilleure évolutivité et moins de dépendance à l'infrastructure des serveurs, elles posent également des problèmes, notamment dans le contexte de la sécurité. L'absence d'une autorité centrale pour superviser les événements en jeu rend les systèmes P2P vulnérables à la triche.

En effet, certains joueurs cherchent souvent à obtenir un avantage déloyal ou à manipuler l'environnement du jeu pour influencer les mécanismes du jeu. Les méthodes de triche les plus courantes consistent à utiliser des logiciels tiers, à exploiter des bugs dans le jeu ou à s'engager dans des pratiques contraires aux règles du jeu. Il est possible de

classer ces attaques par type d'attaque comme suggéré par Yahyavi et al.[4] : Interruption de la dissémination de l'information, action illicite en jeu ou accès non autorisé aux informations. Mais aussi, au niveau auquel elles interviennent, comme suggéré par Webb et al. [23] : au niveau du jeu, au niveau du programme (Application), au niveau des protocoles ou au niveau de l'infrastructure.

Une solution pour remédier à ce problème est de combiner l'architecture P2P avec la technologie de la Blockchain. Celle-ci pourrait constituer un registre transparent et inaltérable pour stocker les événements du jeu. En utilisant la blockchain pour stocker les données cruciales du jeu, nous renforçons la sécurité en créant un environnement transparent qui réduit considérablement le risque de Triche.

4.3 Blockchain

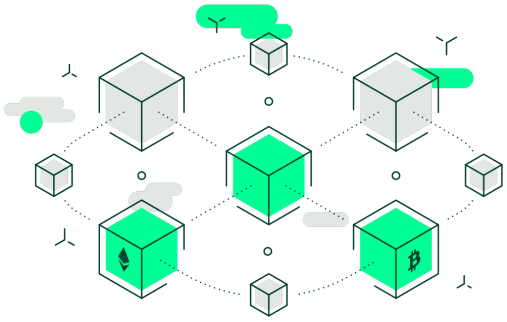


Figure 4 : Représentation d'une blockchain

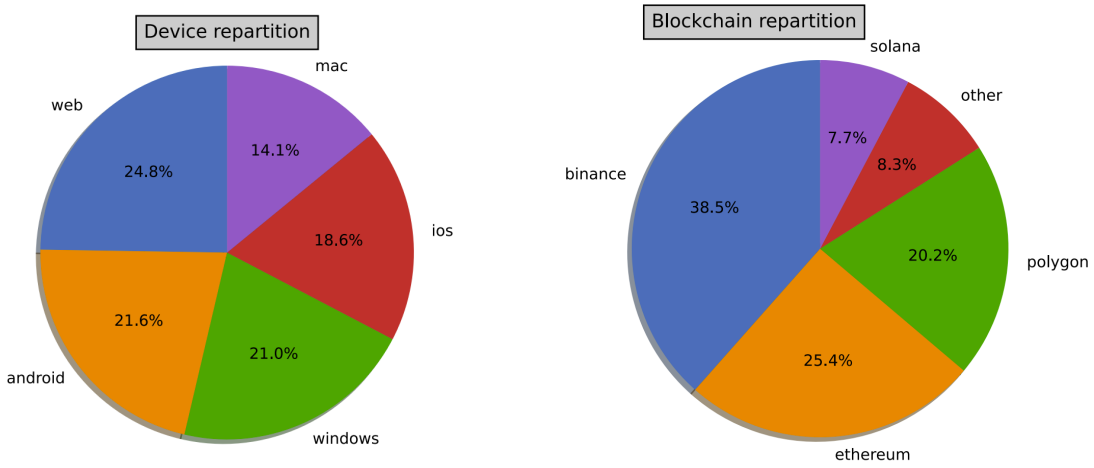
4.3.1 Présentation

La technologie blockchain est un système de registre décentralisé et distribué qui révolutionne l'enregistrement des transactions à travers un réseau informatique en toute transparence et sécurité. Au-delà du domaine des crypto-monnaies, diverses plateformes de blockchain offrent des caractéristiques uniques. Ethereum, par exemple, facilite les applications décentralisées (DApps) et les contrats intelligents (smart contracts), introduisant de la programmabilité dans la blockchain. Parmi les autres plateformes de blockchain dignes d'intérêt, citons Binance Smart Chain (BSC), conçue pour des applications décentralisées très performantes, et Polygon, qui met l'accent sur l'évolutivité et l'interopérabilité, et qui est compatible avec Ethereum. Les algorithmes de consensus jouent un rôle crucial en garantissant le consensus du réseau sur la validité des transactions. Bitcoin s'appuie sur la preuve de travai (Proof of Work ou PoW), où les nœuds effectuent des calculs complexes pour valider les transactions, tandis qu'Ethereum passe à la preuve d'enjeu (Proof of Stake ou PoS), un modèle basé sur la mise en jeu de crypto-monnaie par les validateurs en guise de garantie.

4.3.2 Les jeux vidéos et la Blockchain

L'utilisation la plus populaire de la technologie Blockchain est dans le domaine des finances et des paiements avec des monnaies numériques bien connues comme Bitcoin, Ethereum, Tether, Solana, etc. Et ce, en raison de la transparence, de l'immutabilité et de la décentralisation de son système qui n'est contrôlé par aucune entité centrale, ce qui renforce la confiance parmi les utilisateurs.

Toutefois, une autre utilisation possible de cette technologie serait, comme mentionné précédemment, dans les jeux vidéo en ligne, plus particulièrement les jeux MMO, où certains développeurs ont déjà incorporé la technologie Blockchain dans leurs jeux. Cette technologie peut être employée pour sécuriser la propriété et les transactions des ressources dans le jeu, empêcher la fraude et la triche et garantir la transparence. De plus, la blockchain facilite les échanges et la monétisation des biens virtuels entre les différentes plateformes de jeu pour les utilisateurs.



(a) Architecture Client-serveur centralisée [1]

(b) Architecture multiserveurs [1]

Figure 5 : Blockchain and device distribution

Les jeux basés sur la blockchain occupent un espace considérable sur diverses plates-formes ; les plates-formes Web, Windows et Android étant les plateformes privilégiées pour ce type de jeux comme l'illustre la figure 4(a). En ce qui concerne les blockchains les plus utilisées, Binance arrive en tête de file, suivie par Ethereum, Polygon et solana, comme le montre la figure 4(b). Les genres de ces MMO sont variés, on y trouve des jeux de rôle (RPG), des jeux de stratégie, des jeux de collection, etc.

5 Conception

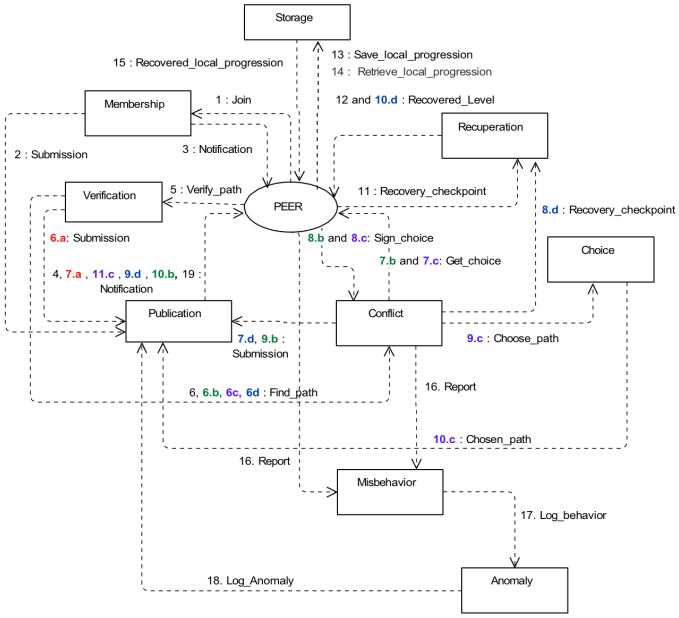


Figure 6 : Architecture MDBBGA [1]

L'architecture que nous allons utiliser pour l'application de la technologie blockchain dans les jeux vidéo est la Modular Decentralized Blockchain Based Game Architecture (MDBBGA)[1]. Il s'agit d'une architecture générale d'un jeu narrative fonctionnant avec la blockchain, mais peut être adaptée pour fonctionner pour d'autres types de jeux. Elle est constituée de plusieurs services essentiels qui sont : Membership, Verification, Publication, Conflict, Choice, Recuperation, Misbehaviour et Anomaly. Ces services sont présentés plus en détail ci-dessous :

- Membership : Ce service est chargé de fournir à l'utilisateur un identifiant unique et l'identifiant du jeu auquel il souhaite accéder. S'il s'agit de sa toute première connexion, le système les crée à partir d'un identifiant propre au client (Par exemple, son adresse dans la blockchain) et enregistre la paire identifiants spécifique au jeu et identifiant client dans la blockchain. Sinon, le système retournera les identifiants correspondants à l'identifiant du client qui ont été créés lors de sa première connexion. Le client les enregistre en local pour pouvoir directement reprendre sa partie dans le jeu.

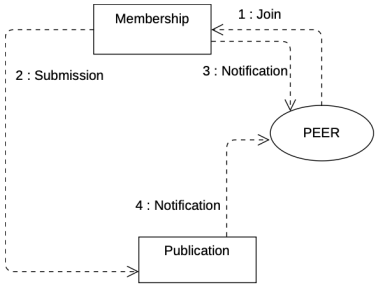


Figure 7 : Architecture du service Membership [1]

- Verification : Permet de vérifier pour un utilisateur souhaitant accéder au niveau suivant si son progrès rempli les conditions nécessaires pour pouvoir y accéder. Si ce n'est pas le cas, il fait appel au service conflict pour déterminer et résoudre le problème rencontré (Choix multiple possible pour le niveau, tentative de triche, cul-de-sac, etc.).

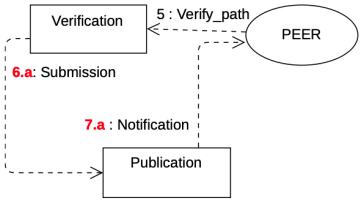


Figure 8 : Architecture du service Verification [1]

- Publication : Le seul service autorisé à écrire sur la blockchain. Il est appelé par les autres services souhaitant enregistrer une modification sur la blockchain (Passage à un autre niveau, enregistrement d'un nouvel utilisateur, etc.).
- Conflict : Ce service est appelé par le service vérification pour résoudre le problème survenue lorsque le joueur essaie de progresser dans le jeu. Il peut s'agir de plusieurs cas ; le joueur doit choisir un niveau pour avancer (Appeler le service Choice), le joueur ne peut plus avancer (Appeler le service Recuperation), le joueur tente une action illégale (Appeler le service Misbehavior), etc.

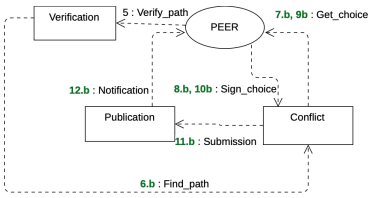


Figure 9 : Architecture du service Conflict [1]

- Choice : Dans le cas où plusieurs choix de progression sont possibles, ce service est appelé pour donner au joueur le choix de sélectionner manuellement son scénario, ou de le lui sélectionner automatiquement en se basant sur les choix des autres joueurs.

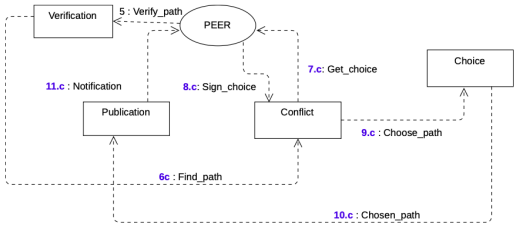


Figure 10 : Architecture du service Choice [1]

- Recuperation : Dans le cas où joueur se retrouve dans un niveau où il ne peut plus avancer dans le scénario, ce service est appelé pour le ramener à un niveau où il peut faire un choix de progression différent.

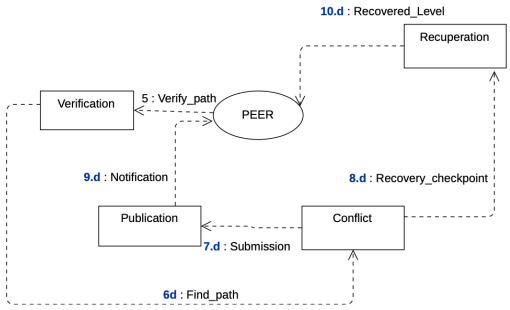


Figure 11 : Architecture du service Recuperation[1]

- Misbehaviour : Ce service est appelé pour signaler le comportement suspect d'un joueur, soit par le service Conflict ou par un autre joueur. Il fait ensuite appel au service Anomaly pour déterminer s'il s'agit véritablement d'une attaque ou d'une tentative de triche.

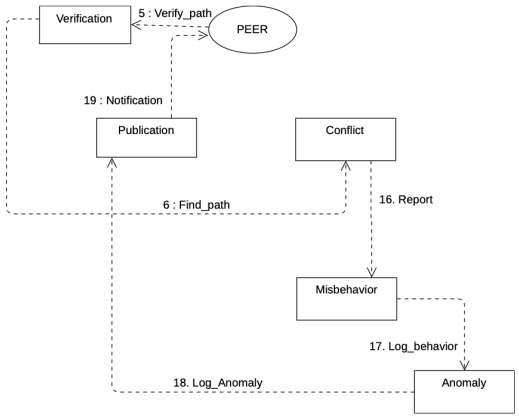


Figure 12 : Architecture du service Misbehaviour[1]

- Anomaly : Permet de déterminer si un comportement suspect signalé s'agit d'une attaque/tentative de triche ou s'il s'agit d'un faux positif. Il enregistre ensuite un rapport sur l'incident dans la blockchain.

Après avoir étudié tous les services disponibles dans le document Blockchain-based Massively Multiplayer Online Game framework and Modular decentralized blockchain-based game architecture, nous avons décidé de nous concentrer sur le service Verification dont l'implémentation nous semble la plus intéressante et qui sera implémenté en tant que smart-contract Ethereum dans la partie suivante.

6 Compte rendu

6.1 Le déroulement

Le projet consistait à coder un smart contract en Solidity exécutable sur la blockchain Ethereum qui mettrait en œuvre l'algorithme de vérification présenté (figure 13).
L'objectif principal était de créer un mécanisme permettant de valider les étapes franchies par un client dans un jeu, et de déterminer si la progression du client d'un niveau à un autre est légitime ou non en utilisant une table d'état de jeu globale appelée ALL_GAME_STATE.
La première étape a été d'étudier attentivement l'algorithme fourni et de comprendre son fonctionnement. L'algorithme décrit plusieurs services distincts : un service de publication pour valider les progressions légitimes, un service de conflit pour gérer les progressions non valides, et un service de misbehaviour pour signaler les comportements inappropriés.

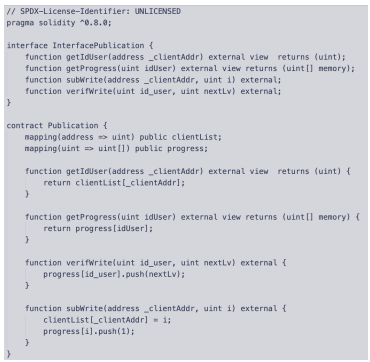
Algorithm 2 Verification service

```
1:  Local variable :
2:  ALL_GAME_STATE table of all game states inside the game, it is used to verify
    the validity of the stage of the game when a client initiates a request to level up to
    the next level.
3:
4:  Function :
5:  - check(game_state) :
6:  verify if all actions done until the current stage of the game are legit
7:
8:  - advance(game_state_1, game_state_2) :
9:  verify if the level up from game_state_1 to game_state_2 is legit.
10:
11:
12: Upon receiving Verify_path(id_user, id_game, current_Lv, next_Lv) from the
    application do :
13:     if check(current_Lv) == SUCCESS and advance(current_Lv, next_Lv) ==
    SUCCESS then :
14:         deliver Submission(current_Lv, next_Lv) to Publication service
15:     elif check(current_Lv) == SUCCESS and advance(current_Lv, next_Lv) ==
    FAILURE then :
16:         deliver find_path(id_user, id_game, current_Lv) to Conflict service
17:     elif check(current_Lv) == FAILURE and advance(current_Lv, next_Lv) ==
    SUCCESS then :
18:         deliver Report(id_user, id_game, current_Lv, next_Lv) to Misbehaviour
    service
19:     else :
20:         deliver Report(id_user, id_game, current_Lv, next_Lv) to Misbehaviour
    service
21:     endif
22: enddo
23:
24:
25:
26: check(current_Lv): The idea is to use a recursive function to verify from the last
    block to the block where id_user and id_game of the client appear for the first
    time. The condition to stop the function and to unstack is to find the block where
    the client initialized its game.
27:
28: check(current_Lv) will call check(precedent_Lv_before (current_Lv)) and this one
    will call check(precedent_Lv_before (precedent_Lv_before (current_Lv))) and so
    on, until check call id_user and id_game.
29:
30: The process of stacking is the verification itself, If the stack goes without any
    issue, the current stage of the game is legit.
31:
32: advance(current_Lv, next_Lv) : verify if the request to level up from current_Lv to
    next_Lv is with in ALL_GAME_STATE table. If it is the case the request is legit,
    otherwise there is a call to Conflict service.
```

Figure 13 : Algorithm Verification Service [1]

6.2 La réalisation

Après avoir bien assimilé l'algorithme, nous avons commencé à coder le smart contract en Solidity. Nous avons d'abord défini un contrat "Publication" (figure 14) qui contient les mappages et les fonctions nécessaires pour stocker et récupérer les informations sur les clients et leur progression dans le jeu.
Ensuite, nous avons créé un contrat "Membership" (figure 15) qui interagit avec le contrat "Publication" pour gérer les identifiants des clients, récupérer leur niveau de progression et ajouter de nouveaux clients.
Le cœur du projet résidait dans le contrat "Verification" (figure 16), qui implémente la logique de l'algorithme de vérification. Nous avons codé les fonctions "check" et "advance" pour vérifier la légitimité des actions effectuées par un client jusqu'à un certain niveau, et déterminer si la progression vers un niveau supérieur est autorisée.



```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

interface InterfacePublication {
    function getIdUser(address _clientAddr) external view  returns (uint);
    function getProgress(uint idUser) external view returns (uint[] memory);
    function subWrite(address _clientAddr, uint i) external;
    function verifWrite(uint id_user, uint nextLv) external;
}

contract Publication {
    mapping(address => uint) public clientList;
    mapping(uint => uint[]) public progress;

    function getIdUser(address _clientAddr) external view  returns (uint) {
        return clientList[_clientAddr];
    }

    function getProgress(uint idUser) external view returns (uint[] memory) {
        return progress[idUser];
    }

    function verifWrite(uint id_user, uint nextLv) external {
        progress[id_user].push(nextLv);
    }

    function subWrite(address _clientAddr, uint i) external {
        clientList[_clientAddr] = i;
        progress[i].push(1);
    }
}
```

Figure 14 : Contract Publication

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import './Publication.sol';

contract Membership {
    uint index = 0;

    address addressInterface;
    InterfacePublication P = InterfacePublication(addressInterface);

    function setAddress(address _addressInterface) external {
        addressInterface = _addressInterface;
    }

    function getClientId(address _clientAddr) public view  returns (uint) {
        return P.getIdUser(_clientAddr);
    }

    function getClientLvl(uint clientId) public view  returns (uint) {
        uint[] memory tmp = P.getProgress(clientId);
        return tmp[tmp.length - 1];
    }

    function newClient(address _clientAddr) public {
        P.subWrite(_clientAddr, index);
        index += 1;
    }
}
```

Figure 15 : Contract Membership

Figure 16 : Contract Verification

- Contrat Verification :
  Ce contrat gère la logique de vérification des actions des joueurs. Il comporte plusieurs fonctions :
  - check(uint current_lvl) :
    Vérifie si le niveau actuel du joueur est le niveau initial ou si les actions effectuées jusqu'à ce niveau sont légitimes. Si aucun chemin valide n'est trouvé, la fonction renvoie false.

- canReachLevel(uint fromLevel, uint toLevel) :
  Détermine si le niveau "toLevel" peut être atteint depuis le niveau "fromLevel" en vérifiant si tous les niveaux intermédiaires sont accessibles.
- isInitialLevel(uint level) :
  Vérifie si le niveau donné est le niveau initial du jeu. Dans la version actuelle, le code suppose simplement que le niveau 1 est le niveau initial.
- advance(uint game_state_1, uint game_state_2) :
  Vérifie si le passage du niveau game_state_1 au niveau game_state_2 est légitime en vérifiant si game_state_2 est présent dans la liste des états de jeu valides pour game_state_1.
- verify_path(uint id_user, uint id_game, uint current_lv, uint next_lv) :
  C'est la fonction principale qui orchestre le processus de vérification. Elle appelle les fonctions check() et advance() pour valider la progression du joueur vers le niveau suivant. Si la vérification réussit, elle appelle le contrat Publication pour soumettre la progression.
- addToList(uint key, uint value) :
  Ajoute une valeur à la liste associée à une clé dans le mappage ALL_GAME_STATE.
- getList(uint key) :
  Récupère la liste associée à une clé dans le mappage ALL_GAME_STATE.
- Interface Publication :
  Définit une interface pour un contrat externe appelé Publication qui sera utilisé pour soumettre les progressions validées des joueurs.

6.3 La validation

Après avoir terminé le codage, nous avons procédé à une phase de validation pour nous assurer que le smart contract fonctionnait correctement et respectait les spécifications de l'algorithme. Nous avons créé des cas de test pour vérifier différents scénarios, notamment :

- Un client qui progresse légitimement d'un niveau à un autre
- Un client qui tente de progresser de manière invalide
- Un client qui démarre une nouvelle partie (niveau initial)
- Diverses situations de progression impliquant plusieurs niveaux

En exécutant ces tests, nous avons pu valider que le smart contract réagissait correctement dans chaque situation, en appelant les services appropriés (Publication, Conflit ou Misbehaviour) selon la légitimité des actions du client.
Une fois les tests concluants, le code a été documenté et préparé pour la livraison.

6.4 Conclusion

Défis rencontrés et leçons apprises :
L'un des principaux défis de ce projet a été la traduction de l'algorithme textuel en un code Solidity exécutable sur la blockchain. Bien que l'algorithme décrivait les étapes à suivre, il a fallu faire preuve de créativité et d'ingéniosité pour concevoir une solution adaptée à l'environnement des contrats intelligents.
Nous avons également dû relever le défi de la gestion efficace des états de jeu et de la synchronisation des données entre les différents clients. La nature décentralisée de la blockchain ajoute une complexité supplémentaire à la gestion des données partagées, ce qui nous a obligés à repenser notre approche à plusieurs reprises.
En outre, nous avons appris l'importance cruciale des tests approfondis dans le développement de contrats intelligents. Étant donné que les contrats déployés sur la blockchain sont immuables, il est essentiel de s'assurer de leur bon fonctionnement avant le déploiement, car toute erreur peut avoir des conséquences graves et coûteuses.
Enfin, ce projet nous a permis de mieux comprendre les avantages et les défis liés à l'utilisation de la blockchain et des contrats intelligents dans le domaine des jeux vidéo. Bien que cette technologie offre des opportunités intéressantes en termes de transparence, de sécurité et de décentralisation, elle présente également des défis en termes de performances, de coûts et d'adoption par les utilisateurs finaux.
En résumé, ce projet de codage d'un contrat intelligent pour la vérification de progression dans un jeu vidéo a été une expérience enrichissante et formatrice. Nous avons réussi à traduire les spécifications de l'algorithme en un code Solidity fonctionnel, tout en relevant les défis liés à la conception, au développement et à la validation de contrats intelligents sur la blockchain Ethereum.
Ce projet a renforcé nos compétences en matière de développement de contrats intelligents, de gestion des données décentralisées et de test approfondi. Nous avons également acquis une meilleure compréhension des opportunités et des défis liés à l'utilisation de la blockchain dans le domaine des jeux vidéo.
Bien que ce projet soit terminé, il ouvre la voie à de nombreuses améliorations et extensions futures, telles que l'intégration avec des interfaces utilisateur conviviales, l'optimisation des performances et l'exploration d'autres cas d'utilisation de la blockchain dans le domaine du jeu.

7 Annexe A

L'annexe suivante inclut les différentes figures présentes dans ce rapport.

Table des figures

8 Annexe B

Cette annexe est associée aux différents acronymes utilisés dans ce rapport.

Acronymes

BCP Brylitte protocole. 8

BSC Binance Smart Chain. 12

C/S Client/Server. 9, 10, 22

DApps Decentralized Applications. 12

DDoS Distributed Denial of Service. 8, 11

IA Intelligence Artificielle. 10

MDBBGA Modular decentralized blockchain-based game architecture. 4, 14, 22

MMO Massively Multiplayer Online. 10, 13

MMOG Massively Multiplayer Online Game. 5–8

P2P Peer-to-peer. 6, 8, 9, 11, 12

PNJ Personnages Non-Joueurs. 10

PoS Proof of Stake. 12

PoW Proof of Work. 12

PvE Player vs Environment. 10

PvP Player vs Player. 10

RPG Role Playing Game. 13

SCVSN Smart Contract Vulnerability Detection Model Based on Siamese Network. 7

SDN Software-Defined Networking. 9

9 Annexe C

Cette section est une annexe qui contient les annotations des différents scripts présents dans ce rapport. Le code des scripts créés est disponible dans notre dépot github[24]

Annotation du code de Contract Publication (figure 14)

- L'interface InterfacePublication définit quatre fonctions externes à implémenter dans le contrat Publication :
  - getIdUser : Retourne l'ID d'un utilisateur à partir de son adresse.
  - getProgress : Récupère le tableau de progression pour un ID utilisateur donné.
  - subWrite : Souscrit un nouvel utilisateur et initialise sa progression.
  - verifWrite : Ajoute un nouveau niveau de progression pour un ID utilisateur donné.
- Le contrat Publication implémente les fonctions définies dans l'interface :
  - clientList est une mapping qui associe une adresse à un ID utilisateur.
  - progress est une mapping qui associe un ID utilisateur à un tableau de niveaux de progression (uint).
  - getIdUser retourne l'ID utilisateur associé à une adresse donnée.
  - getProgress retourne le tableau de progression pour un ID utilisateur donné.
  - verifWrite ajoute un nouveau niveau de progression à la fin du tableau pour un ID utilisateur donné.
  - subWrite souscrit un nouvel utilisateur en initialisant son ID et en ajoutant le niveau de progression 1 à son tableau.

En résumé, ce contrat permet de gérer des utilisateurs, de leur associer un ID, et de suivre leur progression sous forme d'un tableau de niveaux. Les fonctions permettent d'inscrire de nouveaux utilisateurs, de récupérer leurs informations et de mettre à jour leur progression.

Annotation du code de Contract Membership (figure 15)

- setAddress : Cette fonction permet de définir l'adresse du contrat InterfacePublication auquel le contrat Membership interagira.
- getClientId : Cette fonction renvoie l'identifiant d'un client en appelant la fonction getIdUser du contrat InterfacePublication avec l'adresse du client passée en argument.

- getClientLvl : Cette fonction renvoie le niveau de progression d'un client donné en récupérant les données du contrat InterfacePublication via la fonction getProgress.
- newClient : Cette fonction crée un nouvel enregistrement pour un client en appelant la fonction subWrite du contrat InterfacePublication avec l'adresse du client et en incrémentant un index interne.

Le contrat utilise une variable d'état index pour gérer les enregistrements de clients. Il y a aussi une variable addressInterface pour stocker l'adresse du contrat InterfacePublication avec lequel il interagit. Le code importe le fichier Publication.sol (figure 14) qui contient probablement la définition du contrat InterfacePublication.

Annotation du code de Contract Verification (figure 16)

Ce contrat gère la logique de vérification des actions des joueurs.

- check(uint current_lvl) : Vérifie si le niveau actuel du joueur est le niveau initial ou si les actions effectuées jusqu'à ce niveau sont légitimes. Si aucun chemin valide n'est trouvé, la fonction renvoie false.
- canReachLevel(uint fromLevel, uint toLevel) : Détermine si le niveau "toLevel" peut être atteint depuis le niveau "fromLevel" en vérifiant si tous les niveaux intermédiaires sont accessibles.
- isInitialLevel(uint level) : Vérifie si le niveau donné est le niveau initial du jeu. Dans la version actuelle, le code suppose simplement que le niveau 1 est le niveau initial.
- advance(uint game_state_1, uint game_state_2) : Vérifie si le passage du niveau game_state_1 au niveau game_state_2 est légitime en vérifiant si game_state_2 est présent dans la liste des états de jeu valides pour game_state_1.
- verify_path(uint id_user, uint id_game, uint current_lv, uint next_lv) : C'est la fonction principale qui orchestre le processus de vérification. Elle appelle les fonctions check() et advance() pour valider la progression du joueur vers le niveau suivant. Si la vérification réussit, elle appelle le contrat Publication pour soumettre la progression.
- addToList(uint key, uint value) : Ajoute une valeur à la liste associée à une clé dans le mappage ALL_GAME_STATE.
- getList(uint key) : Récupère la liste associée à une clé dans le mappage ALL_GAME_STATE.
- Interface Publication : Définit une interface pour un contrat externe appelé Publication (figure 14) qui sera utilisé pour soumettre les progressions validées des joueurs.

En résumé, ce contrat semble implémenter une logique de vérification d'état et de progression pour un jeu ou une application, en utilisant des contrats intelligents pour s'assurer que les actions effectuées sont légitimes.

Références

[1] Chan Yip Hon Boris. Blockchain-based Massively Multiplayer Online Game framework and Modular decentralized blockchain-based game architecture. 2017.

[2] Mohsen Ghaffari, Behnoosh Hariri, Shervin Shirmohammadi, and Dewan Tanvir Ahmed. A Dynamic Networking Substrate for Distributed MMOGs. *IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING*, 2015.

[3] Raphael Burkert, Philipp Horwat, Rico Luetsch, Natalie Roth, Dennis Stamm, Fabian Stamm, Jan Vogt, and Marc Jansen. Decentralized Online Multiplayer Game Based on Blockchains. In *BLOCKCHAIN AND APPLICATIONS*, 2022.

[4] Amir Yahyavi and Bettina Kemme. Peer-to-Peer Architectures for Massively Multiplayer Online Games : A Survey. *ACM COMPUTING SURVEYS*, 2013.

[5] Sarmad A. Abdulazeez, Abdennour El Rhalibi, Madjid Merabti, and Dhiya Al-Jumeily. Survey of Solutions for Peer-to-Peer MMOGs. In *2015 INTERNATIONAL CONFERENCE ON COMPUTING, NETWORKING AND COMMUNICATIONS (ICNC)*, 2015.

[6] Ho Yin Yuen, Feijie Wu, Wei Cai, Henry C. B. Chan, Qiao Yan, and Victor C. M. Leung. Proof-of-Play : A Novel Consensus Model for Blockchain-based Peer-to-Peer Gaming System. In *BSCI '19 : PROCEEDINGS OF THE 2019 ACM INTERNATIONAL SYMPOSIUM ON BLOCKCHAIN AND SECURE CRITICAL INFRASTRUCTURE*, 2019.

[7] Emanuele Carlini, Laura Ricci, and Massimo Coppola. Integrating centralized and peer-to-peer architectures to support interest management in massively multiplayer on-line games. *CONCURRENCY AND COMPUTATION-PRACTICE & EXPERIENCE*, 2015.

[8] Mingli Wu, Kun Wang, Xiaoqin Cai, Song Guo, Minyi Guo, and Chunming Rong. A Comprehensive Survey of Blockchain : From Theory to IoT Applications and Beyond. *IEEE INTERNET OF THINGS JOURNAL*, 2019.

[9] Seyed Mojtaba Hosseini Bamakan, Amirhossein Motavali, and Alireza Babaei Bondarti. A survey of blockchain consensus algorithms performance evaluation criteria. *EXPERT SYSTEMS WITH APPLICATIONS*, 2020.

[10] Caixiang Fan, Sara Ghaemi, Hamzeh Khazaei, and Petr Musilek. Performance Evaluation of Blockchain Systems : A Systematic Survey. *IEEE ACCESS*, 2020.

[11] Leo Besancon, Catarina Ferreira Da Silva, and Parisa Ghodous. Towards Blockchain Interoperability : Improving Video Games Data Exchange. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, May 2019.

[12] Ran Guo, Weijie Chen, Lejun Zhang, Guopeng Wang, and Huiling Chen. Smart Contract Vulnerability Detection Model Based on Siamese Network (SCVSN) : A Case Study of Reentrancy Vulnerability. *Energies*, 2022.

[13] Akhil Mittal. *Smart Contract Development with Solidity and Ethereum.* 2020.

[14] Mayukh Mukhopadhyay. *Ethereum Smart Contract Development : Build blockchain-based decentralized applications using solidity.* 2018.

[15] Feijie Wu, Ho Yin Yuen, Henry C. B. Chan, Victor C. M. Leung, and Wei Cai. Infinity Battle : A Glance at How Blockchain Techniques Serve in a Serverless Gaming System. In *MM '20 : PROCEEDINGS OF THE 28TH ACM INTERNATIONAL CONFERENCE ON MULTIMEDIA*, 2020.

[16] Jusik Yun, Yunyeong Goh, Jong-Moon Chung, OkSeok Kim, SangWoo Shin, Jin Choi, and Yoora Kim. MMOG User Participation Based Decentralized Consensus Scheme and Proof of Participation Analysis on the Bryllite Blockchain System. *KSII TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS*, 2019.

[17] Dapeng Li, Liang Hu, and JianFeng Chu. A more efficient secure event signature protocol for massively multiplayer online games based on P2P. In *PROCEEDINGS OF THE 2016 INTERNATIONAL FORUM ON MECHANICAL, CONTROL AND AUTOMATION (IFMCA 2016)*, 2017.

[18] Sukrit Kalra, Rishabh Sanghi, and Mohan Dhawan. Blockchain-based Real-time Cheat Prevention and Robustness for Multi-player Online Games. In *CONEXT'18 : PROCEEDINGS OF THE 14TH INTERNATIONAL CONFERENCE ON EMERGING NETWORKING EXPERIMENTS AND TECHNOLOGIES*, 2018.

[19] Stefano Ferretti, Marco Roccetti, and Roberta Zioni. A Statistical Approach to Cheating Countermeasure in P2P MOGs. In *2009 6TH IEEE CONSUMER COMMUNICATIONS AND NETWORKING CONFERENCE, VOLS 1 AND 2*, 2009.

[20] Nirav Patel, Arpit Shukla, Sudeep Tanwar, Neeraj Kumar, and Joel J. P. C. Rodrigues. *GiNA* : A Blockchain-based Gaming scheme towards Ethereum 2.0. In *IEEE INTERNATIONAL CONFERENCE ON COMMUNICATIONS (ICC 2021)*, 2021.

[21] Wikipedia. Botnet, 2007.

[22] Douglas Comer and Adib Rastegarnia. Osdf : A framework for software defined network programming. 2017.

[23] Steven Webb and Sieteng Soh. Cheating in networked computer games : a review. In *ACM International Conference Proceeding Series ; Vol. 274 : Proceedings of the 2nd international conference on Digital interactive media in entertainment and arts ; 19-21 Sept. 2007*, 2007.

[24] Github. github.com/omar-kadri/pres.