SAPIENZA
UNIVERSITÀ DI ROMA

DEPARTMENT OF COMPUTER SCIENCE

PuzzleBlock

BLOCKCHAIN AND DISTRIBUTED LEDGER TECHNOLOGIES

Professors:
Di Ciccio Claudio

Students:
Barreca Federico
Bello Giuseppe
Biagioli Luca
De Sio Ilaria

Academic Year 2023/2024

Contents

1 Preface

1.1 Brief presentation

This paper aims to explain the functionality, features, and implementation of **PuzzleBlock**, a decentralized game application (DApp) designed to show the fundamental aspects and potential of blockchain technology through engaging gameplay in **Unity** [1], based on **Ethereum** [2]. In this journey, users will progress through various levels, with the opportunity of purchasing **Non-Fungible Tokens** (NFTs). Players from around the world can compete, enhancing the learning experience through the integration of competitive spirit with educational value.

1.2 Team members and main responsabilities

Planning and subsequent changes related to the entire project were systematically preceded by explanatory conference calls to ensure constant updates. In addition, macro tasks (such as Solidity and JavaScript, Token Eng. and Architectural Design) were organized into specific subtasks in order to optimize the workflow.

1. **Barreca Federico**: Game Design, Architectural Design, IPFS Engineering, report reviewer;
2. **De Sio Ilaria**: Token & Contract Engineering, Game Design, Architectural Design, report reviewer;
3. **Bello Giuseppe**: JavaScript & Solidity Development, diagrams, report reviewer and optimization;
4. **Biagioli Luca**: Solidity Development, report reviewer, diagrams and back-end integration;

1.3 Outline of the report

The structure of the report is organized as follows:
The first part is the **Background** section (*See Section 2*), introducing the Blockchain technology, its functionalities, and the fundamental principles upon which Puzzleblock is built, and outlines the application's domain. Following in the **Presentation of the Context** section (*See Section 3*), the objectives of the DApp are explained, specifying the type of blockchain used in the production of the project. In the **Architecture** section, the software architecture details are described with all necessary diagrams. Afterward, in the **Implementation** section (*See Section 6*), a demonstration sample of the DApp is showcased, and all technical specifications and technologies used in its implementation are discussed. In conclusion, the **Known Issues and Limitations**

section (*See Section 7*) addresses any identified problems and constraints encountered during the project's execution.

2 Background

2.1 Blockchain Technology

In 2008, Satoshi Nakamoto introduced the first decentralized blockchain with Bitcoin, offering a new way to conduct digital transactions without central authority. This approach, detailed in "Bitcoin: A Peer-to-Peer Electronic Cash System" [3] utilized a method for secure timestamping and a difficulty adjustment mechanism to maintain consistent block production, pioneering the concept of an independent digital currency. In summary, it is possible to distill the essence of blockchain technology into two fundamental concepts:

- Is an open and **distributed ledger**, meaning it operates on a decentralized network of nodes. These nodes work together to maintain a consistent version of the ledger. In the context of blockchain, a **ledger** is a record of transactions that is append-only and ordered, without any deletions, comprising a series of blocks. These blocks, which are chronologically ordered, serve as units of time and allow users to see **transparently** all activities conducted on chain.

- Blockchain technology facilitates the efficient and **verifiable** recording of transactions between two parties in a manner that is both **permanent** and immutable. Transactions are recorded by being stored in **blocks**, which are linked together through the inclusion of the previous block's hash in the subsequent block's header. Due to the immutable nature of the ledger, once transactions are verified, they are permanently stored, ensuring a reliable and unalterable record of transactions.

2.1.1 Ethereum Blockchain

Ethereum, like Bitcoin, uses a decentralized ledger system to record transactions across a network of computers. Ethereum not only processes transactions but also serves as a platform for decentralized applications (DApps) and smart contracts.
Ethereum works similarly to Bitcoin, with a distributed network of nodes validating and recording transactions. Ethereum introduces **smart contracts**, which are self-executing agreements written in code. Smart contracts on the **Ethereum Virtual Machine** (EVM) allow developers to create decentralized applications for financial services, gaming, and other purposes.
Ethereum's native cryptocurrency, **Ether** (ETH), fuels transactions and enables smart contract deployment. Ethereum's Proof of Stake (PoS) consensus mechanism replaces

the previous Proof of Work (PoW) model. This transition seeks to enhance scalability and energy efficiency.

2.1.2 Types of Blockchain

Blockchains can be categorized based on their level of permission in the consensus process or by their degree of visibility.

Regarding consensus permission, the classifications are:

- **Permissionless blockchain**: These are fully decentralized systems. Operating on the principle of being "trustless," they allow all nodes to engage in the consensus mechanism, offering complete anonymity to every participant.
- **Permissioned blockchain**: These are controlled networks where an administrator grants certain nodes the authority to handle consensus and verify data.

In terms of visibility, blockchains are classified as:

- **Public blockchain**: In these networks, any node can suggest new blocks and access previous ones. Bitcoin and Ethereum are notable examples.
- **Hybrid blockchain**: This type combines the features of both public and private blockchains to meet the unique requirements of a project. Being a tailored solution, its structure can differ from one blockchain to another.
- **Private blockchain**: Typically employed for enterprise solutions, access to these blockchains is limited to specifically chosen nodes.

2.1.3 Token Standards

Another important and distinctive feature of the Ethereum blockchain is its ability to facilitate the creation of customized assets, known as Tokens, through smart contracts. Regarding tokens there are several standards with different functionalities:

- **ERC20**: is the most widely used and recognized token standard on Ethereum. It's primarily used for **Fungible assets**, meaning each token is identical to another in value and properties;
- **ERC721**: introduced to facilitate the creation of **Non-Fungible Tokens** (NFTs), enables the representation of unique assets on the Ethereum blockchain. Each token is distinct and can hold a different value based on its attributes, these cannot be interchanged with other equivalent ones;
- **ERC1155**: offers a **Multi-token** standard that allows the creation of fungible and non-fungible tokens within the same contract. We chose to use this standard to have greater flexibility , management and efficient transfer of a wide range of asset types within a single contract architecture.

2.2 Application Domain

In this part we will explain in what context our DApp fits and why we decided to create it as a solution to the problem we examined.

We hear a lot about blockchain as an innovative, widely used technology with great potential, but few people really know about it and it is rarely taught. However, its technical complexities and the perceived inaccessibility of its concepts have created a knowledge gap, which not only hinders the adoption and evolution of blockchain technology, but also limits its benefits.

Recognizing this, we were motivated to create a DApp that would serve as a **bridge between the world of blockchain and people outside the field**, making it not only understandable but also engaging, independent of their technical background.

By making blockchain accessible to a broader audience, we are helping to promote a more **inclusive digital future** in which the potentiality of blockchain can be harnessed across different sectors to benefit society.

3 Presentation of the context

3.1 Aim of PuzzleBlock

PuzzleBlock aims to be a fun and educational platform aimed at gamers and students curious about the intricacies of blockchain technology (based on it). By navigating through its levels carefully on certain themes of the blockchain itself, players not only immerse themselves in challenging puzzles, but also gain a deeper understanding of how blockchain works in a fun and **interactive way**.
The opportunity to purchase NFT introduces players to the concept of digital ownership, further enriching their knowledge in the area of cryptocurrencies as well. While competing with other players from around the world, players experience firsthand the importance of consensus mechanisms and decentralized decision-making.
The most important goal of PuzzleBlock is to **bridge the gap between complex blockchain concepts and the general public by making this cutting-edge technology accessible to all** through the universal language of gaming. In this way, it fosters a global community of enthusiasts ready to participate in the blockchain technology revolution with more **consciousness** and understanding.

3.2 Use Cases

In Figure below the use cases of our DApp are presented; we have three possible actors that can behave in the following way:

- **Admin**: This is the user with the highest privileges within the DApp. The Admin has the following responsibilities:
  - **Deploy Contracts**: Publishes smart contracts on the blockchain that govern the logic of the DApp. This operation is executed only once by the software development team **when the game is published**.
  - **Mint Assets**: Mint new units of digital assets or create new collections purchasable by the users.
- **Non Registered-User**: These kind of users are not yet registered within the DApp. The only action that they can perform is:
  - **Register**: Sign up or register with the DApp to become a registered user, meaning that their profile is initialized and stored on the IPFS and its corresponding address saved into the PuzzleBlock Smart contract's state.
- **Registered User**: these type of users are already registered and can perform a variety of actions within the DApp:
  - **Play**: playing a game and in the meantime improve their knowledge of the mechanisms underlying blockchain technology;
  - **Claim rewards**: when the user completes a level, he can choose to claim its temporary reward or try to improve his earnings continuing his run playing other levels;
  - **Buy Assets**: Purchase digital assets that can be used within the DApp;
  - **Buy/Use PowerUps**: Purchase or use Power-Ups, which are temporary improvements or benefits that can help the user in the game activity.

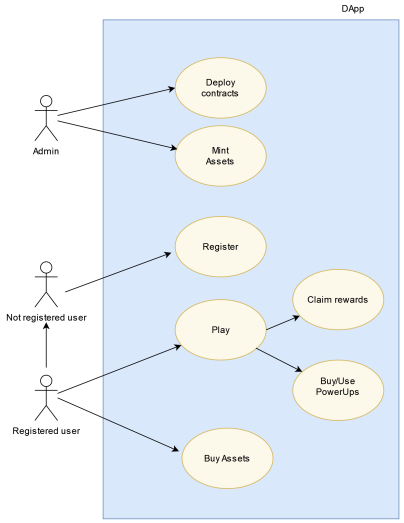

Figure 1: PuzzleBlock Use Cases

3.2.1 Scenarios

These are different Scenarios in our system with different actions performed:

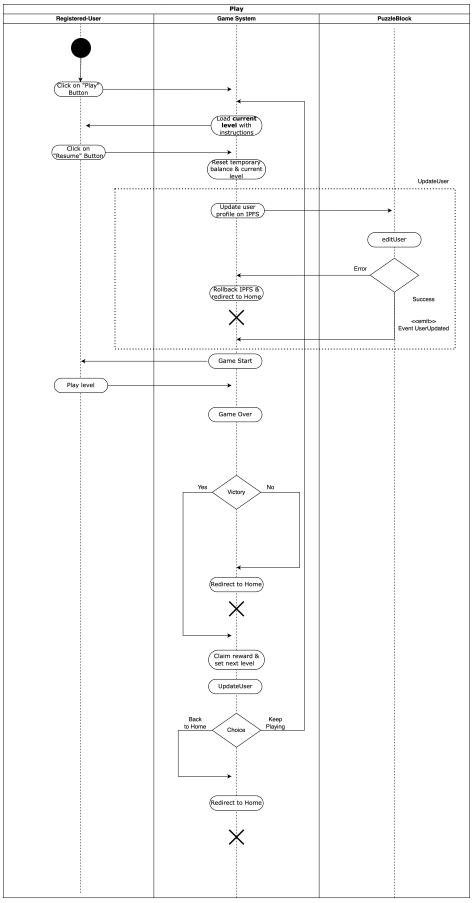

Figure 2: Play Sequence Diagram

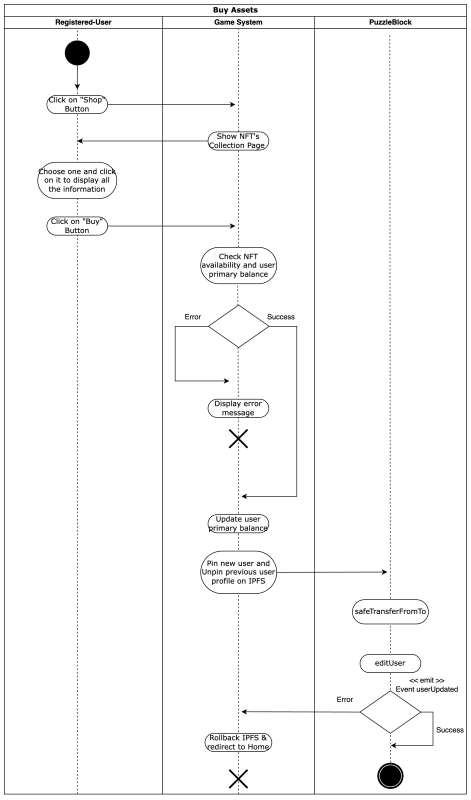

Figure 3: BuyAssets Sequence Diagram

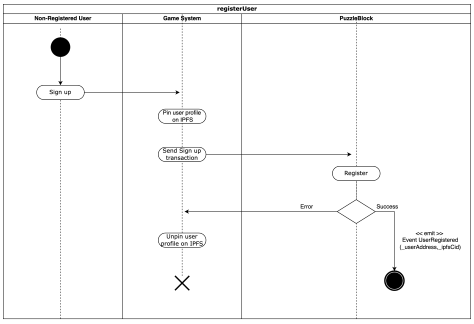

Figure 4: Register User Sequence Diagram

3.3 Why use blockchain?

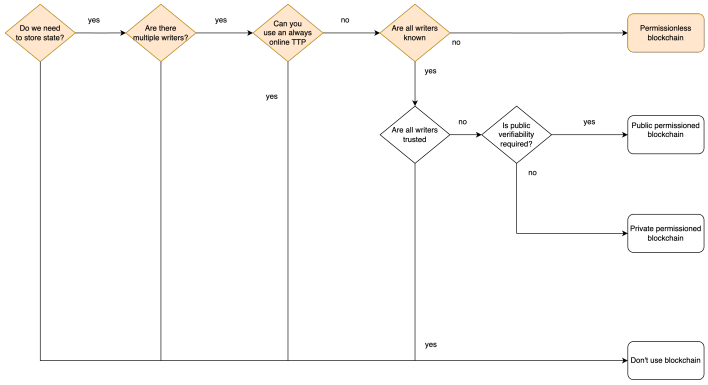

Figure 5: Blockchain Flowchart (Wüst & Gervais)

To determine our project's need for a blockchain and its type, we used the Wüst & Gervais model [4] (in Figure 5) and this revealed that: It's important for our project to maintain a secure, immutable, and transparent environment, since we need to store information securely; We have a wide range of users who participate in various activities, such as playing levels, complete multiple runs or purchasing NFTs; We cannot depend on an online Trusted Third Party (TTP) because our goal is to develop a decentralized system; The identity of our users is not known to us, with anyone owning an Ethereum address being able to join our platform. Therefore, we will employ a permissionless blockchain, specifically Ethereum, to meet these needs.

4 Game Design

4.1 Gamification

The playful component embedded in our game comes from a sophisticated Game Design process. In fact, we developed the game with a solid foundation and clear objectives, treating it as a legitimate game in its own right. The design of the game was the result of a collaboration with a **Gamification** course, with its definition:

"Gamification is an innovative educational approach that aims to increase learner motivation and engagement by incorporating elements of game design into educational contexts." [5]

Gamification has its roots in the studies of famous scholars and theorists of learning, pedagogy and philosophy, including Piaget, Huyzinga (famous for his 1938 "Homo Ludens"), Bateson, Caillois and Morris, author of 1967's "The Naked Ape."
Despite its growing popularity and the mixed results of its application in educational contexts, this concept has also been applied in the commercial, advertising and social sectors. To provide a more detailed understanding of our work, we have attached our **Game Design Document** (GDD) [6], a formal tool widely used in the game industry that comprehensively describes the technical and design aspects of a game project.

4.2 Game System

As illustrated in the **Application Flowchart** presented in Figure 6, once logged in, the user is directed to the main page. From this, he has the option of navigating to the shop to purchase NFTs or Powerups, to be used in the various levels, or he can choose to start a new run or continue a previous one. If the user loses or decides to interrupt his game run, he can easily return to the main page. In addition, after accumulating a certain number of wins, (where each one unlocks a next level), he can complete the game run.

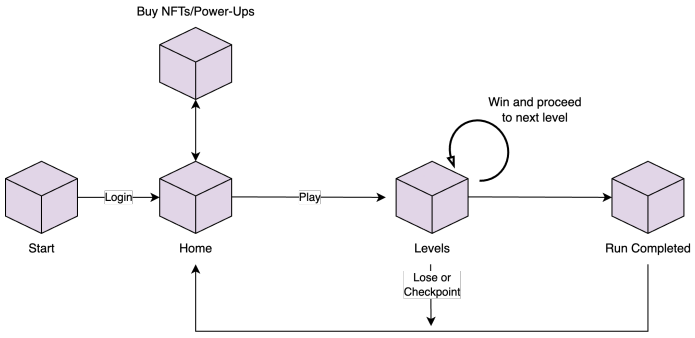

Figure 6: Application Flow

4.3 Reward System

Regarding the reward system as can be seen in the Figure 7, initially starting from the first level, in case of a win, the user will be rewarded with a reward equal to the score times a certain multiplier for that specific level. We can generalize and implement this formula for each level, as can be seen in the Figure 8.

Each level features a **different score multiplier** characterized by an increasing trend. As each level has to be completed within a time limit, we define the level score

$$R_1 = S_1 \times M_1$$

where:

$R_1 = 1^\circ$ Level reward

$S_1 = 1^\circ$ Level score

$M_1 = 1^\circ$ Level score multiplier

Figure 7: First Level

$$R_n = T_b + (S_n \times M_n)$$

where:

$R_n = n$-th level reward

$S_n = n$-th level score

$T_b =$ Temporary Balance

$M_n = n$-th level score multiplier

Figure 8: n-th Level

as the **remaining amount of time** (in seconds) on the timer. Upon completing the n-th level, where n is not the final level, if victorious, the reward R_n will be the updated Temporary balance T_b. For the final level, the reward will be added to the Primary balance, thus resetting the Temporary balance to zero. In case of a loss, the Temporary balance is reset to zero as well, and the player must restart from the first level. The Temporary balance is always zero at the first level, which explains why it can be omitted from the formula.

4.4 Gameplay

4.4.1 Level 1 : Hash Functions

The first level provides the player with a generic introduction to the blockchain and addresses the topic of hashing. Given a four-digit integer **BlockData** in the range $[1000, 9999]$, a hash function **F**, and a criterion **C**, the objective is to insert an integer **Nonce**, within the time limit, such that:

$$C(F(BlockData + Nonce)) = true \tag{1}$$

The hash function and the criterion might change with different difficulties.

4.4.2 Level 2 : Transactions and Double-Spending

The second level consists in the validation of a transaction by the player. The player must decide whether the transaction is valid or not valid. To do this, they must consider all the transaction data (such as input data, output, addresses and signature), verifying their correctness and coherence. The objective is to correctly validate all transactions within the time limit. As the difficulty increases, the number of transactions increases, the available time decreases, and the player must also consider the temporal consistency between transactions.

4.4.3 Level 3 : Branching and Consensus

The third level addresses branching and consensus, drawing inspiration from Ethereum's LMD Ghost algorithm[2]. The blockchain tree is then displayed with some missing blocks, and each block is characterized by an 'Attestations' number. Additionally, there are some blocks that need to be inserted into the empty spaces. The player must choose and drag the available blocks to complete the tree in such a way that, according to the visiting algorithm, a certain highlighted path is followed.

4.5 Variable Difficulty

Our game offers three levels of difficulty: **easy**, **medium**, and **hard**. Each difficulty level entails changes to the gameplay: one common and others unique to each level. Regarding the common change, it involves a reduction in the time available to solve the puzzle proposed by the level, which will be **60 seconds**, **40 seconds**, and **20 seconds**, respectively.
The first level undergoes alterations in the formulation of the hash function and criteria.
The second level witnesses an increase in the number of transactions to be validated, proportionate to the difficulty level. Additionally, for the highest difficulty, the player must consider time progression and interaction between transactions.
The increase in difficulty in the third level entails entering more blocks to solve the puzzle.
We propose determining the difficulty based on the average points of users as follows:

Easy mode triggers when

$$AveragePoints < MinPoints \tag{2}$$

Medium mode triggers when

$$MinPoints \leq AveragePoints < MaxPoints \tag{3}$$

Hard mode triggers when

$$AveragePoints \geq MaxScore \tag{4}$$

Figure 9: Difficulty trigger system

Consequently, the system adjusts to prevent excessively rapid reward growth, while emulating one of the fundamental mechanisms of Proof of Work difficulty self-adjustment within Bitcoin's network traffic [3]. The points balance of each user is reset at the beginning of each month.

4.6 Power-Ups

Between levels, the player can choose whether to spend the tokens from Temporary balance obtained in the current run to purchase Power-ups (see Figure below 10). These power-ups can be used to act on the timer, the puzzle, or the point multiplier.

Figure 10: PowerUps

The amount that can be purchased is limited to 3 for each type of Power-up. At the end of a run, unused power-ups are saved for the next run.

- The "**Time Amethyst**" influences the flow of time by freezing the timer for 10 seconds with each use, and the effect stacks.
- The "**Potion x5**" applies an additional $\times 5$ multiplier at the end of each level won. If a level is lost, the potion used is not refunded. Additionally, only one unit of potion can be used per level.
- Differently from the aforementioned power-ups, the effect of the "**Hints Grimoire**" provides the player with visual hints which change depending on the current level puzzle.

5 Software Architecture

5.1 DApp Architecture

In this section we show in detail the architecture of PuzzleBlock and all the design choices behind it.
The entire project is open source and can be accessed on **GitHub** [7](the main repository and the submodule of Unity) and following the README.md file, there are all the instructions for using and installing the software.

Below is possible to see the Figure 11 with the complete diagram of the architecture.

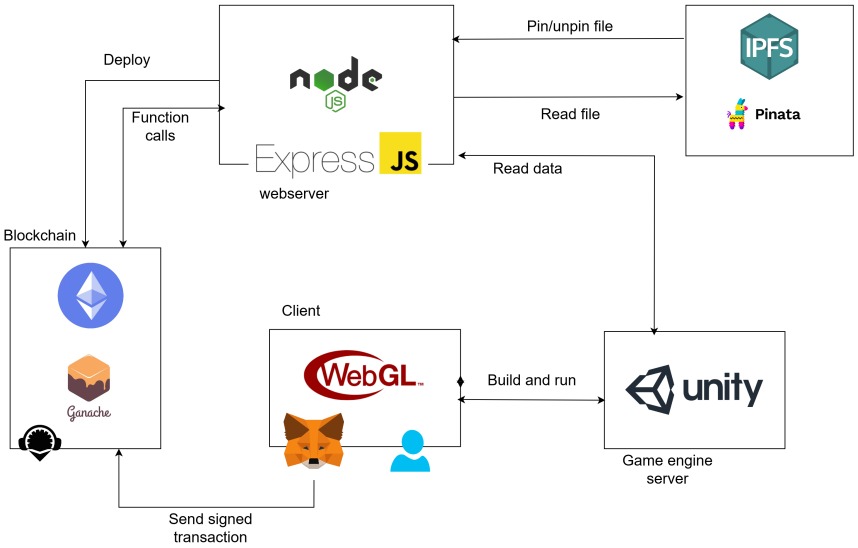

Figure 11: PuzzleBlock Architecture Diagram

5.2 Tools used

For the realization of PuzzleBlock we relied on **Node.js** [8] and **Express.js** [9] to create a Webserver that allows us to handle POST requests. We used **Remix**[10] as the editor to write and compile the contracts with solidity compiler 0.8.20 (with EVM version set to "London" and enable optimization setted to 200) and also we use the package **@openzeppelin/contracts** in smart contracts to manage some technical aspects regarding the standards of the NFTs.
To manage files on IPFS instead, we chose to use **Pinata** [11] that makes simple to store and retrieve media files on it and in the user-interactive frontend part, we used **Unity**[1] to build a **WebGL application** [12] in order to interact with the Metamask browser extension via the Metamask **Ethereum provider API** [13] . Thanks to C# properties, we could design the Unity project with a **MVC Pattern**-like approach [14], which allowed us to organize and distribute our team's work very efficiently.

5.3 Smart Contracts

5.3.1 PuzzleContract

In Figure 12 is shown the class diagram for the contract PuzzleContract. This contract provides all functionalities related to the user, defining a unique structure that allows us to associate an Ethereum address with each User object.
Furthermore, two events are defined:

1. **UserRegistered**: Highlights the registration of a new user by emitting the user's address and their IPFS CID.

2. **UserUpdated**: Issued when a user's data is updated, with the same parameters as the UserRegistered event.

Additionally, functions are defined for registering a new user within the contract by saving their address and IPFS CID, retrieving their information, and modifying the data of an existing user.

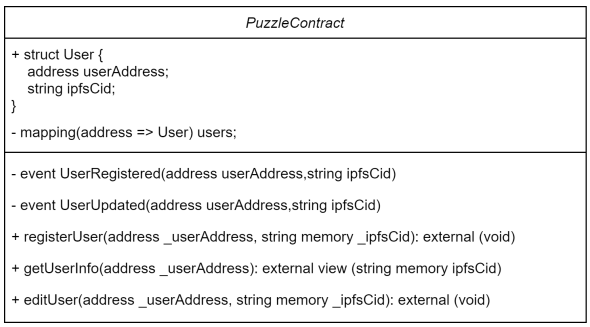

Figure 12: Class Diagram of PuzzleContract

5.3.2 GameAsset

In Figure 13 is shown the class diagram for the contract GameAsset to manage and manipulate digital assets.
This contract includes the properties and methods provided by OpenZeppelin, such as **Ownable**, **ERC1155**, and **Strings**, leveraging these standards to implement security, traceability, and interoperability features.
An IPuzzleContract interface was also defined, the GameAsset contract incorporating the editUser method, facilitating interaction with the IPFS ecosystem to modify user information.

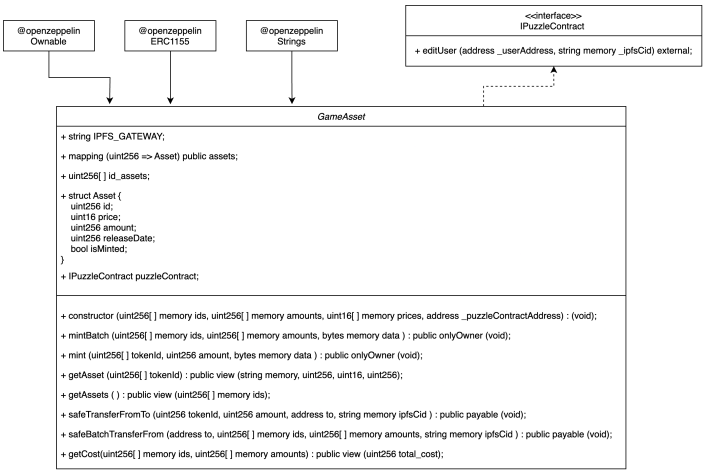

Figure 13: Class Diagram of GameAsset

The internal architecture of the contract is enhanced by the definition of a data structure called **Asset**, designed to encapsulate the essential attributes of managed

asset collections, including unique identifiers, prices, available quantities, release date, and minting status.
The contractual logic is exposed through a suite of methods that allow:

1. the **creation** of new collection;
2. the **minting** of new assets in **single** or **batch**;
3. **obtain information** about a specific asset or all;
4. make a **secure transfer** of one or more assets to another address;

6 Implementation

6.0.1 Login & Home page

Initially, the user navigates from the **Login page** (See Figure 14), bypassing the conventional credential input method, and directly accesses the **Main Home page** (See Figure 15) via the MetaMask extension popup.

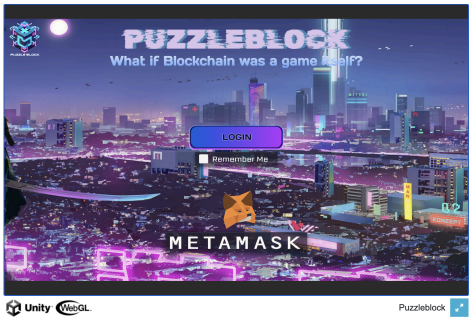

Figure 14: Login Page

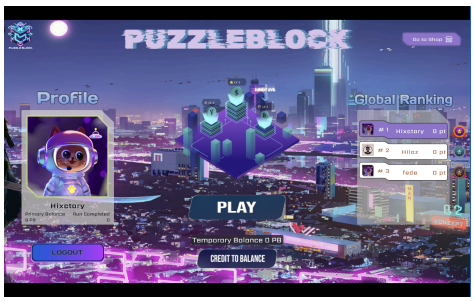

Figure 15: User HomePage

Within this interface, users are afforded the capability to execute various operations. These operations encompass initiating a new run, transferring achieved credits from the secondary balance to the primary balance, or utilizing credits for purchases within the shop.

6.0.2 Shop

Upon entering the **Main page of the Shop** (See Figure 16), the user has the ability to navigate through the various sections available. He can explore the **Collection Page** of NFTs (See Figure 17) for sale by accessing the **detailed panel of each NFT** by a simple click (See Figure 18), this panel provides information such as description, price, and rarity of the NFT.

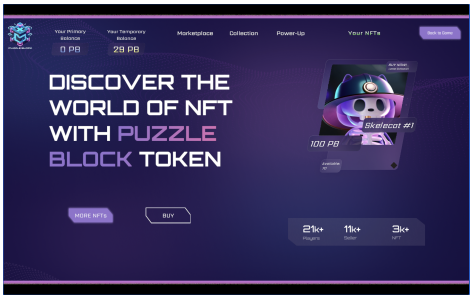

Figure 16: Shop Main Page

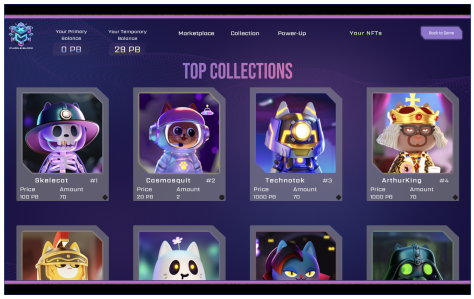

Figure 17: Shop Collection Page

In addition, the user has the option of viewing previously **purchased NFTs Page** (See Figure 19) and setting one of them as his profile picture. Finally, within the **PowerUps Page** (See Figure 20), it is possible to acquire the powerups needed to pass the various levels of the game.

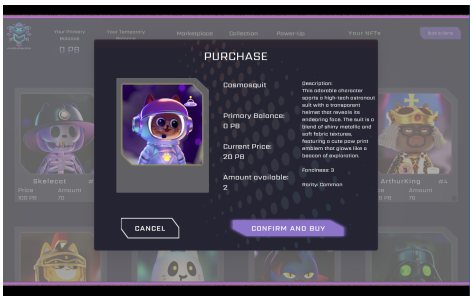

Figure 18: Shop Buy Page

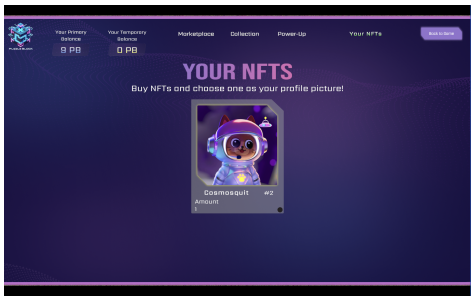

Figure 19: Shop My NFTs Page

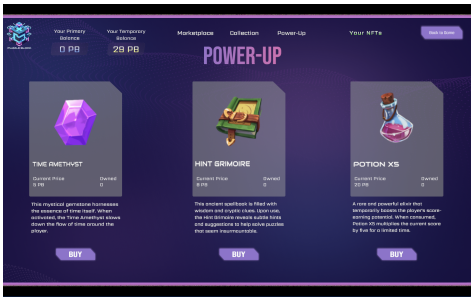

Figure 20: Shop PowerUp Page

6.0.3 Level 1

This is the first level, initially the user is on the **Introductory Page** (See Figure 21) to allow him to better understand the context of the level he is going to play on the **first Game Page** (See Figure 22).

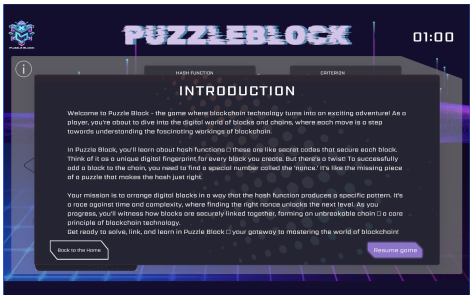

Figure 21: LV1 Intro Page

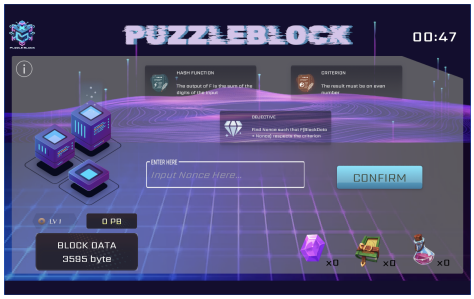

Figure 22: LV1 Game Page

6.0.4 Level 2

Let's change context and start the **Second level Intro Page** , which has transactions and double spending as context (See Figure 23), after the user understands how it works they can proceed and play on the next page (See Figure 24).

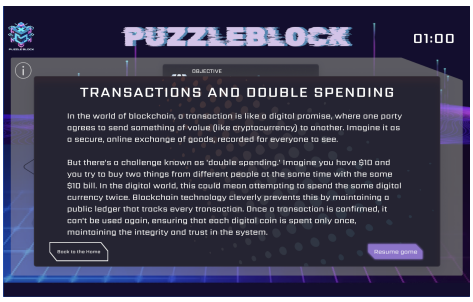

Figure 23: LV2 Intro Page

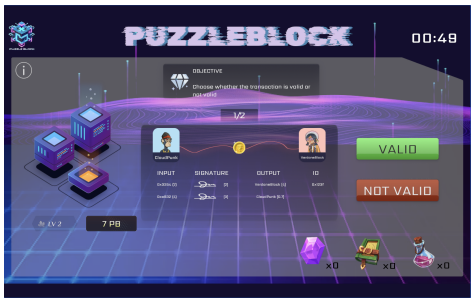

Figure 24: LV2 Game Page

6.0.5 Level 3

Subsequently the last level with its intro regarding **Consensus** (See Figure 25), and the game page associated (See Figure 26) with it.

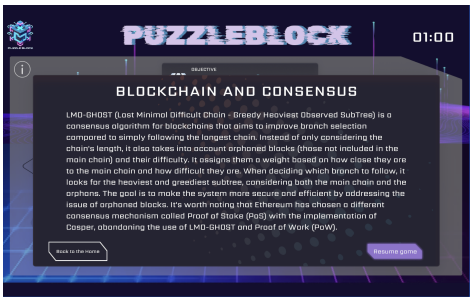

Figure 25: LV3 Intro Page

Figure 26: LV3 Game Page

7 Known issues and limitations

7.1 Metamask Unity API issue

The lack of a complete and updated Metamask library for Unity, which prevents a flexible login system through the Metamask web extension, has been a significant obstacle for us. However, we overcame this problem by using Unity's system for compiling **jslib libraries**. This method allows us to establish communication via messages between Unity code written in C# and the WebGL API for 3D graphics, thus facilitating the interaction between HTML and JavaScript content within the Web browser. It is also important to note that the aforementioned **messaging system is bidirectional**, which allows for effective response management and robust error handling. This bidirectional solution ensures that it is possible not only to send requests from Unity code to the Web environment, but also to receive and manage responses in Unity, creating a smooth and reliable user experience.

7.2 Countermeasures against cheating

7.2.1 WebGL vulnerabilities and browser security

Although WebGL itself shows some vulnerabilities, when applications run within the browser environment, they are subject to the security policies and restrictions of the browser. Unity WebGL builds can be decompiled, and the source code can potentially be reverse-engineered. To mitigate this risk, **code obfuscation techniques** [15] make it more challenging for attackers to comprehend and alter the code.

7.2.2 Unfair user behaviour

Additionally, we have developed a **penalty system** to ensure proper user behavior. According to the game rules, the user accepts the outcome, whether it be victory or defeat. Therefore, at the beginning of each level, the user signs a transaction that temporarily sets the conditions for defeat. At the end of the level, in case of victory, the user is required to sign another transaction that overrides the previous defeat condition with that of victory, crediting the temporary balance earned and proceeding to the next level. **The user may attempt to cheat** by reloading the page to restart the level from the beginning in order to retain the temporary balance and level checkpoint. Alternatively, the user may choose not to accept the end-of-level transaction to retry the level and achieve a better score. **These scenarios are avoided through a preventive defeat state** that the user must accept to start the level.

7.3 Cost analysis

Another issue we encountered was related to the costs arising from the deployment and transactions of smart contracts. To ensure the availability and maintenance of the application, a **balanced allocation of costs** between the platform manager and individual users is necessary. Therefore, we have distributed the costs as follows:

Table 1: Cost attribution

	Platform	User
Smart Contracts Deployment	◯	
IPFS Provider	◯	
Node.js Server Maintenance	◯	
User Registration Transaction		◯
BeginLevel Transaction		◯
FinishLevel Transaction (Only if win)		◯
Temporary-to-Primary Balance Transfer Transaction		◯
Profile Picture Setup Transaction		◯
NFT Purchase Transaction		◯
Power Up Purchase Transaction		◯

8 Conclusions and future remarks

In summary, the development of our application aimed at **infotainment through gamification** has been a rewarding journey, successfully guiding users through the study and exploration of blockchain and cryptocurrencies. The software itself exhibits a complex yet **robust structure**, ensuring **reliability** and functionality across its various components. One of the key strengths of our project lies in the efficient organization of teamwork, facilitated by available technologies and the modular design of the application. Despite the significant volume of assets to produce, our team **managed to optimize the development process and maintain coherence throughout**.

Furthermore, the addition of new levels featuring diverse themes and content, categorized into beginner, intermediate, and advanced levels, enhances user engagement and progression within the application and blockchain knowledge. This multi-tiered approach caters

to users of varying expertise, ensuring a tailored and enriching experience for all. The Smart Contract managing the NFTs also features an interface, already implemented, for the **buying and selling of assets among users** and also the purchase of multiple assets within a sort of shopping cart. In addressing security concerns, measures such as code obfuscation on Unity could be implemented to strengthen security. Additionally, the **potential integration of oracles**, facilitated by event handling for communication with the blockchain, could enhance the reliability and functionality of our application.

In conclusion, our application represents a significant achievement in the realm of infotainment and game-based learning, offering users a dynamic platform for exploring the world of blockchain and cryptocurrencies. With continued dedication to innovation and improvement, we look forward to further advancing our project to meet the evolving needs of our users and the ever-changing landscape of technology.

References

[1] Unity Real-Time Development Platform — 3D, 2D, VR & AR, 2024. `https://unity.com/`.

[2] Ethereum: A Decentralized Platform for Applications, 2024. `https://ethereum.org/`.

[3] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. `https://web.archive.org/web/20140320135003/https://bitcoin.org/bitcoin.pdf`.

[4] Karl Wüst and Arthur Gervais. Do you need a blockchain?, Accessed in 2024. `https://eprint.iacr.org/2017/375.pdf`.

[5] Christo Dichev and Darina Dicheva. Gamifying education: What is known, what is believed and what remains uncertain: A critical review. *International Journal of Educational Technology in Higher Education*, 14(1), Feb 2017.

[6] Federico Barreca, Ilaria De Sio, Giuseppe Bello, and Luca Biagioli. Game Design Document PuzzleBlock, Created in 2024. `https://github.com/iladesio/PuzzleBlock-BlockchainProject-Server/blob/main/GDD_PuzzleBlock.pdf`.

[7] Federico Barreca, Ilaria De Sio, Giuseppe Bello, and Luca Biagioli. Puzzleblock-GitHub-Repository, 2024. `https://github.com/iladesio/PuzzleBlock-BlockchainProject-Server`.

[8] NodeJS, Accessed in 2024. `https://nodejs.org/en`.

[9] ExpressJS, Accessed in 2024. `https://expressjs.com/`.

[10] Remix IDE, Accessed in 2024. https://remix.ethereum.org.

[11] Pinata, Accessed in 2024. https://www.pinata.cloud/.

[12] Build your WebGL application, Accessed in 2024. https://docs.unity3d.com/Manual/webgl-building.html.

[13] Metamask, Accessed in 2024. https://docs.metamask.io/wallet/reference/provider-api/.

[14] ASP.NET MVC, Accessed in 2024. https://learn.microsoft.com/it-it/aspnet/core/mvc/overview?view=aspnetcore-8.0.

[15] Unity Code Obfuscator, Accessed in 2024. https://assetstore.unity.com/packages/tools/utilities/obfuscator-pro-89589.

[16] OpenZeppelin Docs. ERC20 Token Standard, Accessed in 2024. https://docs.openzeppelin.com/contracts/4.x/erc20.

[17] OpenZeppelin Docs. ERC721 Token Standard, Accessed in 2024. https://docs.openzeppelin.com/contracts/2.x/api/token/erc721.

[18] OpenZeppelin Docs. ERC1155 Token Standard, Accessed in 2024. https://docs.openzeppelin.com/contracts/3.x/erc1155.