Interactive Game Design by Smart Contracts
Supervised by Dr. John Yuen

COMP 4801
Final Report

Date of submission:
14 April, 2019

LI Kin Yeung (Ron)
UID: 3035326013
Abstract

The age of digital disruption provides opportunities for game industry to integrate with new elements and technologies. Blockchain is one of the emerging technologies that integrates with games in recent years. However, high transaction latency for public blockchain becomes a barrier in designing interactive games. Layer 2 solutions are introduced to mitigate latency. They also allow extensive scaling of the blockchain applications without worrying about the security issues.

By incorporating Ethereum and Loom network architecture with a turn-based battling game, the transaction latency will potentially be reduced significantly. The game maintains high interactivity, while still keeping the blockchain network safe and secure. Ethereum also features smart contracts, allowing for flexible game design. This project values the realization of layer 2 solutions in blockchain games and explores the feasibility of scaling blockchain applications with those solutions. By incorporating the concept of non-fungible tokens and smart contracts in game development, it unleashes the power of highly scalable and interactive gameplay. Finally, a blockchain turn-based battling game prototype has been constructed.
Acknowledgement

I would like to express my deepest appreciation to my final year project supervisor, Dr. John Yuen, for his professional guidance and continuous support in each consultation throughout the academic year. He also gave invaluable advice on blockchain security and data communication. I also thank Dr. Chow for his time in assessing and give comments to my project during the first presentation.

In addition, I would also like to extend my sincere thanks to Mr. Cezar Cazan for his constructive advice in writing project plan and progress report. He provided a one-on-one consultation on writing support, assisting me in structuring the documentations for my project.

Finally, I would like to extend my gratitude to the Loom SDK team and the developers and contributors in the Loom SDK Telegram group. Their responses are invaluable in making this project success.
# Table of Contents

Abstract ........................................................................................................................................ i
Acknowledgement ......................................................................................................................... ii
List of Figures .................................................................................................................................. v
Abbreviations ................................................................................................................................. vi

1 Introduction ................................................................................................................................. 1
   1.1 Background ............................................................................................................................. 1
   1.2 Outline ..................................................................................................................................... 3

2 Objectives ..................................................................................................................................... 4

3 Related Works .............................................................................................................................. 5
   3.1 Introduction of blockchain games ............................................................................................ 5
      3.1.1 CryptoKitties ..................................................................................................................... 5
      3.1.2 Fishbank .......................................................................................................................... 6
      3.1.3 Total Poker ...................................................................................................................... 6

4 Justification on Framework Choices ............................................................................................ 6
   4.1 Overview of Root Blockchain Layer ......................................................................................... 6
   4.2 Root Blockchain Layer Selection ............................................................................................. 6
   4.3 Overview of Layer 2 Solutions .................................................................................................. 7
      4.3.1 Raiden Network ................................................................................................................ 7
      4.3.2 Loom Network .................................................................................................................. 8
   4.4 Layer 2 Solution Selection ......................................................................................................... 8

5 Methodologies .............................................................................................................................. 8
   5.1 Introduction .............................................................................................................................. 8
   5.2 Game Setting ............................................................................................................................ 9
   5.3 Game Mechanism .................................................................................................................... 9
   5.4 Design .................................................................................................................................... 10
      5.4.1 User Interface Design ....................................................................................................... 10
      5.4.2 Smart Contract Design .................................................................................................... 12
      5.4.3 Security Design ............................................................................................................... 13
         5.4.3.1 Overview ................................................................................................................... 13
         5.4.3.2 Arithmetic Overflow Avoidance ................................................................................. 13
         5.4.3.3 Access Control .......................................................................................................... 14
         5.4.3.4 Digital Signature ..................................................................................................... 14
      5.4.4 Data Flow Design ............................................................................................................. 15
List of Figures

Fig. 1 Simplified Bitcoin Block Chain by Blockgeeks [2] .......................................................... 2
Fig. 2 The CryptoKitties marketplace by M. Greg [4] ................................................................. 5
Fig. 3 Selected section of blocks in battle ...................................................................................... 9
Fig. 4 The battle system .................................................................................................................. 10
Fig. 5 The inventory system ........................................................................................................... 11
Fig. 6 The marketplace system ....................................................................................................... 12
Fig. 7 Data Flow Diagram of the game ......................................................................................... 15
Fig. 8 Architecture of the battling game ....................................................................................... 16
Fig. 9 DAppChain directory (Left) and Unity Client directory (Right) ......................................... 18
Fig. 10 Deploying smart contracts ................................................................................................. 19
Fig. 11 Wireframe of a character .................................................................................................. 20
Fig. 12 Actual rendering of a character ......................................................................................... 21
Fig. 13 Different models of hat ...................................................................................................... 22
Fig. 14 FluffyToken Smart Contract ............................................................................................... 23
Fig. 15 DNA Generator .................................................................................................................. 24
Fig. 16 Validity Check Smart Contract .......................................................................................... 25
Fig. 17 Menu Module .................................................................................................................... 26
Fig. 18 Top-up Module .................................................................................................................. 27
Fig. 19 Marketplace Module .......................................................................................................... 27
Fig. 20 Inventory Module - Add .................................................................................................... 28
Fig. 21 Inventory Module - Remove ............................................................................................... 29
Fig. 22 Game Play Module - before ............................................................................................. 29
Fig. 23 Game Play Module - after ................................................................................................. 30
# Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABIs</td>
<td>Application Binary Interfaces</td>
</tr>
<tr>
<td>DPos</td>
<td>Delegated Proof-of-Stake</td>
</tr>
<tr>
<td>ERC-721</td>
<td>Ethereum Requests for Comments 721</td>
</tr>
<tr>
<td>HP</td>
<td>Health Points</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>PoS</td>
<td>Proof-of-Stake</td>
</tr>
<tr>
<td>PoW</td>
<td>Proof-of-Work</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Calls</td>
</tr>
<tr>
<td>SDK</td>
<td>Standard Development Kit</td>
</tr>
</tbody>
</table>
1 Introduction

Due to the rise of new technologies, the gaming world has been revolutionizing in recent years, giving players a brand new experience throughout the gameplay. Some game companies research different new technologies to unblock the bottleneck of the market, blockchain is no exception. In fact, there is an existing pioneer in the blockchain game industry called CryptoKitties. It has been a great success with total transaction amount of 32 million USD by April, 2018 [1]. It indicates that blockchain technology has great potential for game development.

However, public blockchain suffers from latency. Each transaction takes a long time for validation which is undesirable for gaming experience. In light of this issue, some layer 2 solutions have been developed on top of the root blockchain layer to support swift transaction and scalable network. This project focuses on the realization of layer 2 solutions by building a turn-based battling game using smart contracts.

1.1 Background

Blockchain is a distributed ledger technology that allows stakeholders to hold a transaction history in the network. In public blockchain, each transaction is validated via a consensus algorithm. The most famous one is called Proof-of-Work, the use of enormous computational power to validate trust-less transactions. Each block in the blockchain corresponds to a transaction which is cryptographically hashed with a mathematical puzzle. People who try to solve the puzzle are called miners. The miner gets some rewards like cryptocurrencies which are newly generated once the puzzle is solved. Then, the transaction is said to be validated and the block is attached to the current
As shown in Fig. 1, each block contains the hash of the previous block to avoid block violation. The transaction is then immutable and the entire network is tamper resistant. In general, the more the computation power, the higher the possibility of getting the rewards.

Despite greater transparency of the blockchain network, one of the famous applications is tokenization. Blockchain technology has been gaining currency these days due to the invention of Bitcoin, which is the first cryptocurrency launched in the world. The founder of Bitcoin, Satoshi Nakamoto proposed a peer-to-peer electronic cash system, promoting anonymous value transfers without third party authorities. Apart from Bitcoin, another blockchain platform called Ethereum focus on decentralized applications that run smart contracts. Smart contracts are sets of well-defined functions and instructions deployed in the blockchain. Every participant in the blockchain network can perform some tasks like initiating a transaction via making calls to smart contracts. Ether, which is the cryptocurrency in Ethereum is used in those transactions. Each transaction costs gas which is the price for running the smart contracts. They are self-executing once deployed in the network, thus a high quality smart contract design is of paramount importance in blockchain applications, preventing vulnerabilities and attacks.
The beauty of blockchain and cryptocurrency realizes transparent value transfer and asset tokenization, facilitating a new era of game development. Blockchain game is not a new concept. The first blockchain game called HunterCoin was launched in 2014, where players competed with each other and collected coins back to their campsites [3]. It turns a traditional mining concept into gaming experience, players can move around, combat with other players, as well as collecting resources. Many blockchain games are currently being developed, facilitating the integration of blockchain game and game economy.

1.2 Outline

This report first explains the concept of blockchain and its applications. Then, it highlights the justifications on framework choices and the methodologies adopted in this project. Next, details of implementation are explained with figures. Finally, recommendations for future work are made to reflect on project improvements.
2 Objectives

The goal of this project is to apply blockchain technology in game design, and evaluate existing layer 2 solutions to solve the latency problem of public blockchain. Moreover, security is also a main focus in blockchain game. A secure trading system prevents fraudulent transactions and exploitations. It is an essential element especially in games that involve rare item collection. The secure trading atmosphere is one of the key features to help build a sustainable game economy.

From the traditional view of public blockchain, transactions are validated via a consensus algorithm called PoW, requiring huge power consumption. In order to improve the transaction speed of peer-to-peer network, layer 2 solutions are introduced to provide a secure and scalable infrastructure on top of the root blockchain layer. The project aims to utilize suitable layer 2 solutions for game development to reduce transaction latency.

Apart from the latency issues mentioned above, the project aims to safeguard in-game transaction during the gameplay. During the gameplay, each move is recorded in a block. The smart contract helps to identify the player who makes a move before committing it to the blockchain. It is important to avoid perpetrators to take advantage of the gameplay, safeguarding the players’ rights.

Finally, a good game design caters for all audiences. Striking a balance between veteran players and new players is imperative, especially for games that involve scarce and powerful items. A sustainable game should be free to play without superfluous advantages offered to the paid players. The project aims to ensure fairness of the gameplay, regardless of nature of the players.
3 Related Works

3.1 Introduction of blockchain games

There are different types of blockchain games in the current market, focusing on different features and values. Some games feature crypto-collectibles where players collect rare items and trade with other players. Some games build traditional game using blockchain architecture to safeguard the in-game transaction, like online casino. These existing blockchain games offer insights into the design of my project.

3.1.1 CryptoKitties

CryptoKitties is a crypto-collectible game where people can buy virtual cats using cryptocurrencies. There are infinite combinations of cats since they are generated by non-fungible tokens. Each cat is unique and is tied to a token with specialized appearance, pattern and accessories. Players can trade with the others using cryptocurrencies and breed a new cat using two cats they owned.
3.1.2 Fishbank

Fishbank is a multiplayer fish battling game. Players can collect different fishes with various abilities. In spite of collecting fishes, they can battle with other players and hunt other fishes to growth their weights. There are also items to protect the fishes from attacks for certain period of time.

3.1.3 Total Poker

Total Poker is a mobile poker game based in blockchain architecture, featuring secure and fair gameplay. Deck operations are decentralized and cards are randomly distributed. Players can create or join a game room anonymously. The latest update of the game features free gameplay to attract more players to join the poker community.

4 Justification on Framework Choices

4.1 Overview of Root Blockchain Layer

There are two mainstream root blockchain layers in the marketplace: Bitcoin and Ethereum. Bitcoin is an electronic cash system, emphasizing on peer-to-peer anonymous value transfer. The Bitcoin network is highly secured via PoW. Another platform called Ethereum focuses on building decentralized applications via smart contracts. Its underlying security is safeguarded by PoS. Validators are chosen in a deterministic way depending on the amount of cryptocurrencies they own. A portion of cryptocurrencies, that is the stake, is taken from the validators. When the validators complete their works, they can gain transaction fees from it. If they approve a fraudulent transaction, the stake will be added to the transaction fees. It ensures true validation when the stake is higher than the transaction fees.

4.2 Root Blockchain Layer Selection

Ethereum is used as the root blockchain layer for this project. Different from Ethereum, Bitcoin network focuses on lightweight transaction like value transfer. As game transaction often contains complex structures, Bitcoin network is not suitable for transferring high volume of data which overloads the block size in the blockchain.
Instead, Ethereum can model those structures in smart contracts. It also allows customized consensus algorithms and contracts, providing greater flexibility in designing the game. Moreover, the average transaction speed of Ethereum is 13 times faster than that of Bitcoin [5]. In case there are many transactions being sent to the base blockchain layer, Ethereum has a greater performance as compared to the Bitcoin network.

4.3 Overview of Layer 2 Solutions

Layer 2 solutions are used to scale a public blockchain network. The base of the blockchain is a layer 1 protocol that relies on itself for security via proof-of-work. With large hash power generated by miners, it prevents fraudulent transactions and double-spending attacks. Layer 2 solutions sacrifice security in exchange for faster transaction throughput, and rely on layer 1 for security. Participants in the network need to periodically check relevant transactions on layer 2. If there are suspicious transactions, participants can request for auditing by giving evidences to the smart contracts residing in layer 1[6]. After processing the request, the system can then punish the cheaters in layer 2 by returning their funds back to the owner in layer 1. Therefore, layer 2 solutions allow faster transaction throughput with a mechanism to safeguard participants from exploitation. The layer 2 solutions listed below are the most popular frameworks in Ethereum.

4.3.1 Raiden Network

Raiden network is a scaling solution for Ethereum applications. It supports off-chain instant payments which is secured by blockchain smart contracts. By creating a state channel between two participants, they can transact many times until timeout is reached. The final result after subsequent transactions will be sent to the blockchain for validation, freeing up some blockchain bandwidth. It can accommodate large amount of transactions instantly with small transaction fees, facilitating the scalability of Ethereum transactions.
4.3.2 Loom Network

Loom network aims to provide a scalable infrastructure for Ethereum blockchain applications. The Loom SDK produces DAppChain which are layer 2 blockchains based in Ethereum. The Loom SDK has a large spectrum of support to various environments such as JavaScript, Phaser, Go and Unity. Different from proof-of-work in the root blockchain, pluggable consensus and smart contracts can be applied to DAppChain to optimize network scalability. It specifically supports a consensus called delegated proof-of-stake (DPoS), enabling flexible scaling of online games and social applications. The DAppChain utilizes Plasma-based relays for secure value transfer in the underlying Ethereum platform.

4.4 Layer 2 Solution Selection

Loom network is used for scaling the Ethereum network in this project. It is tailor-made for blockchain game development as it supports different game development platforms like Unity and Phaser. Also, DAppChain in Loom network is highly scalable using pluggable consensus and smart contracts, allowing for flexibility implementation [7]. Apart from SDK support, the underlying transaction mechanism is also a determining factor. Unlike Loom network, Raiden network requires a small amount of transaction fees. It is undesirable for players to pay a certain amount of fees whenever they make a move in the gameplay. Therefore, Loom network is best suited for blockchain game development due to its cost-free transactions.

5 Methodologies

5.1 Introduction

This project follows agile software development methodology. The agile approach focuses on adaptive development, allowing for change management throughout the system development lifecycle. The following sections first introduce the game setting and mechanism, and then outline the design, implementation and testing of this project.
5.2 Game Setting

The style of the landscapes, backgrounds and characters is cartoonish. The proposed theme is turn-based battling with puzzle solving. Each character is unique, as there are different combinations of colors and body parts. The purchased characters can be viewed in the inventory and can be added to form a team for battling. A valid team comprises three characters. The goal of the players is to collect as many characters as possible, and utilize them for battling. The game is therefore a mixture of crypto-collectibles and battling elements.

5.3 Game Mechanism

The game is a turn-based game where players strategically attack the enemy or avoid the characters from being attacked. Each player chooses three characters for a battle with the enemy. Each character has a speed attribute, indicating the order of battling. The player plays a puzzle game during the battle by choosing blocks with the same color. The player initiates a normal attack when choosing two consecutive blocks of the same color (see orange blocks in Fig.3). The player can also deal huge damage to the enemy by choosing 2x2 blocks with the same color. If there is no matching block, the player can choose a single block, like the red block at top-right corner in Fig.3, dealing only a small amount of damage. The actual damage done to each character is different since the defense property is taken into consideration. Characters can dodge attacks when the final damage is lower than its defense.

![Fig. 3 Selected section of blocks in battle](image)
In the battling scene, the enemy and a team of three characters take turns to initiates attacks according to their speed. The faster the speed, the higher the chance of first initiation of attack. The battling ends when the enemy’s HP reaches zero or the team’s HP reaches zero.

5.4 Design

This project design primarily includes user interface design, security design and architectural design. In order to have a better control over models, game objects, prefabs and UI elements, all design elements are primitives and they are built from scratch.

5.4.1 User Interface Design

The following part demonstrates the layouts of the three systems: the battle system, the inventory system and the marketplace system.

![Diagram of the battle system with enemy turn status, player turn status, properties status, and inventory elements.]

Fig. 4 The battle system
The game is designed in landscape mode because it provides more space for displaying a team of three characters and the enemy. As shown in Fig. 4, the battle system is divided into three regions, the upper part shows the turn status, indicating the current turn. It also shows properties during battling, for example, dodging and dealing damage to the enemy. The middle part shows the team of three and the enemy with their health bars on top. Finally, the lower part shows the puzzle game, where the player can interact with it using mouse clicks.

Fig. 5 shows the high level structure of an inventory system. It is divided into three parts. The middle part indicates a list of characters the player owned. When the player clicks on a particular character, the upper part appears. The upper part mainly shows character information, including HP, attack, defense, as well as a uniquely generated integer string called DNA. On the right bottom corner of the upper part, the “Add” button is used to add the selected character to the team. The lower part shows the left arrow and right arrow, allowing the player to browse the inventory list. Finally, the player can return to main menu by pressing the “Back” button on the top right corner.
The marketplace allows players to purchase characters used in battling. The middle part shows some characters currently available in the marketplace. The purchase function is also available in this system. Smart contract handling the transfer function is invoked when pressing the “Buy” button. Once the purchase is successful, the character will appear in players’ inventory system (see Fig.5). Character information is shown when clicking a character. The UI design of this system is changed to align with the inventory system which enhances consistency between these two systems.

5.4.2 Smart Contract Design

In order to maintain interactive gameplay, smart contracts are designed to handle lightweight transaction. Each module in Unity should only contain some small function calls to smart contracts for atomic transaction. On the other hand, smart contracts themselves have to be designed with considerations. Each function in smart contracts should only perform small tasks; otherwise, the response time may be slow.
Moreover, access modifiers should be used appropriately to provide levels of abstraction. External and internal access modifiers should be used in the first place, since they do not copy arguments to memory. It saves space and computation time especially when there is large number of incoming function calls.

5.4.3 Security Design

5.4.3.1 Overview

The Loom network is secured by the underlying Ethereum network. The game logic is deployed in the DAppChain to support fast and interactive gameplay. The idea of Loom network is that attackers are disincentivized, since they cannot benefit at all by hacking in the DAppChain. Instead, validators of the DAppChain earn rewards for securing the network.

Moreover, the Unity SDK developed by Loom network provides identity management. It validates players’ identity in the gameplay. Since the game logic is running on DAppChain, the player identity is checked using digital signatures in each transaction being sent to the blockchain which enhances security.

Apart from security guaranteed by Loom network, smart contract design is vital to prevent fraudulent transactions. Simple methods like input field checking, are used to prevent unexpected behaviors from happening. For example, add an access modifier “view” for functions that does not mutate states. The caller therefore cannot change the state of the variable defined in the smart contract.

5.4.3.2 Arithmetic Overflow Avoidance

In Solidity, “uint” is a typical data type used for unsigned integers. Attackers may make use of arithmetic overflow or underflow to assign unexpected value to functions in smart contracts. For example, adding 257 to an uint8 value of 0 will result in 1, leading to an overflow problem. In order to tackle this problem, a safe math library called “SafeMath.sol” is used. When there is transaction regarding important information or data, like value and token transfer, the library is used for arithmetic calculation.
5.4.3.3 Access Control

In Unity, some classes are designed to be singleton as only one instance of the class is allowed. For example, there is only one instance of enemy class, battle team class and game board class in each battle. It restricts the instantiation of objects, and avoid unexpected behaviour from happening, for example, two game boards in a battle. They are independent and can be easily accessed via the static instance variable which loose coupling.

In Solidity, data is encapsulated via getter and setter methods with public and private access modifiers respectively. Moreover, Solidity provides two more access modifiers, namely internal and external. Internal is similar to private whereas external is similar to public. The difference is that, both internal and external access modifiers, only read from the arguments (if there is any). Public and private access modifiers allocate memory for arguments, and thus it is less efficient.

5.4.3.4 Digital Signature

Loom network provides digital signatures for each transaction committed in the blockchain. Each transaction contains critical information, for example, player assets and battling turns. Therefore, digital signature is applied for every transaction within the Loom network to avoid perpetrators to sniff information, preventing unexpected result from happening in smart contracts. As smart contracts store game states and important assets, using digital signature to authenticate messages sent to the smart contracts is of paramount importance in keeping data consistent.
5.4.4 Data Flow Design

According to Fig. 7, players can make a request to marketplace, inventory and battling modules. In most cases, players first enter marketplace to view the available characters. When a player request for purchase, the checkout function is invoked. RPC indicating a “purchase message” is sent to the DAppChain, and finally reach the destination smart contract called “FluffyToken”. It checks the balance of the player and transfers ownership of the token if that player has enough balance.

After purchasing characters in marketplace, players can view them in the inventory. Whenever, the player requests for adding a particular character, the information is stored in local storage. Since the player already own those characters, it is unnecessary to make an additional call only for saving a team. Therefore, it is more suitable to keep team data in local storage.
When the player requests for a battling game, the Unity application checks whether there are 3 members in the team. If the team contains 3 members, the battle scene is rendered. When the player is ready, click start to begin the battling. For every move the player makes, the “move message” is digitally signed for security. It then goes through the validity check to determine whether it is a valid move. No matter it is a valid move or not, result is sent back to the Unity application for updating UI. The entire battling cycle repeats until one side is defeated.

5.4.5 Architectural Design

The architecture of this project consists of three components, the Unity application, Loom network and Ethereum network.

![Architecture of the battling game](image)

The Unity application contains the user interface for players and interacts with the DAppChain directly in the Loom network. The Loom network has transfer gateways, connecting to the smart contracts and Ethereum network. This project focuses on a DAppChain application and the battling logic is implemented in smart contracts. Whenever a player makes a move in the gameplay, it is validated using smart contracts via DAppChain. The result is send back to Unity application for updating the user interface, for example, initiating an attack. Most transactions are going through DAppChain for efficiency. However, transactions such as transferring tokens back to
Ethereum wallets are required for greater security. The overall gaming experience is not affected since most of the data communication of the game resides in DAppChain.

When the blockchain starts, a localhost RPC HTTP server is set up, listening at port 46658. The data communication between the Unity application and the DAppChain is through RPC. Each transaction submitted from the Unity application to the DAppChain is signed for security. Some functions mutate states of the smart contracts while some only view the states. The result is sent back to the Unity application for updating the UI.

Apart from the HTTP request-response cycle between Unity application and the DAppChain, there are internal communications within the Unity application, for example, function calls between classes and data extraction from game objects. They are encapsulated and provide levels of abstraction and information hiding.

Apart from external and internal data communications, Loom network provides transfer gateways to Ethereum root blockchain layer. Although the transfer gateways are out of the scope, it is worth noting that DAppChain applications are highly scalable without sacrificing security. The consensus algorithm used in Loom network is DPoS where validators get rewarded for their contributions in securing the network. Therefore, DAppChain applications are highly secured.

### 5.5 Implementation

#### 5.5.1 Overview

The implementation phase started in mid-October. This project is divided into modules and sub-modules; each individual part is self-complete for easier integration. The following sub-sections introduce the implementation of blockchain setup, game asset modelling, smart contract implementation, as well as Unity modules. Finally, a fully functional battling game prototype running in DAppChain has been developed.
5.5.2 Loom DAppChain Environment

After selecting Loom network as sidechain technologies for this project, setting up a basic Loom DAppChain development environment is the first step. By following the official tutorial [8] on setting up a blockchain architecture with Loom network, Truffle and Unity, the Unity client application is basically configured for sidechain communication. However, the backbone DAppChain application has missing files and cannot be initialized. After integrating an example of Loom Truffle Provider [9] with the Unity client application, the entire application, including Unity and DAppChain, can be set up (see Fig. 8).

Fig. 9 DAppChain directory (Left) and Unity Client directory (Right)

In the “dappchain” directory, there are files and scripts for defining the blockchain network. The public key and private key for the creator are generated via a command “./loom init”. The loom executable is also used to bring up the blockchain network via a command “./loom run”. The contracts directory stores smart contracts which are written in Solidity (version 0.4.24). The migration directory contains JavaScript files that take smart contracts as input, and export ABIs as JSON files. Then, it makes a copy of all smart contract ABIs with their contract addresses to the “unityclient” directory. In the “unityclient” directory, it is basically a normal Unity project setting with assets and scripts. Those ABIs and addresses stored in “unityclient” directory are used to interact with smart contracts deployed in DAppChain.
When deploying the smart contracts to the DAppChain via a command “yarn:deploy reset”, the DAppChain sends back RPC HTTP response upon request (see Fig. 9 left side). On the right side of the terminal, it shows which JavaScript file is currently running. The corresponding smart contract defined in that JavaScript file is deployed to the blockchain network.

![Deployment Image](image)

**Fig. 10 Deploying smart contracts**

### 5.5.3 Game Asset Modelling

All models used in this project are primitives and built from scratch in order to have better control of the game objects. Blender is used for character and body parts modelling. It is a powerful tool for creating customized models, textures and armatures. Other UI elements such as, sliders, health bars, layout, game board and textures are constructed in Unity built-in graphics library.
5.5.3.1 Character Modelling

Fig. 11 Wireframe of a character

This project features random character generation, every character has unique features, colours and body parts. The purpose of generating unique characters is to align with non-fungible tokens. Each token has a unique string representing the character. In order to create seemingly infinite number of combinations, different armatures are defined in the character for body part generation (see Fig.10 grey parts). For example, the Hair armature is used to locate the transform position of a hat prefab. During instantiation of the character, the transform of different armatures can be used to instantiate different body parts anchored to the corresponding armatures. The
primitive shape of the character is a sphere, with extended ears and legs. The orange wireframe (see Fig. 10) indicates the actual model rendering (see Fig. 11).

5.5.3.2 Body Parts Modelling

As mentioned above, different body parts are randomly chosen to form different combinations of characters. There are 6 body parts in a character, namely hat, eyes, ears, nose, mouth and tail. Each body part has 10 variations. Together with RGB values and alpha values of those body parts, forming seemingly infinite number of combinations. Fig. 12 shows different models of hat, and their names are set with
indices. This practice is useful when processing game objects in batch instantiation. The randomization of RGB values and alpha values is done in Unity.

![Different models of hats](image)

Fig. 13 Different models of hat

### 5.5.4 Smart Contracts

One of the cruxes of this project is smart contracts. Smart contracts contain lightweight functions, allowing authorized player to get states from them or make changes to smart contract states'. This project mainly implements 2 major smart contracts, namely Fluffy Token and Validity Check. Fluffy Token is a non-fungible token smart contract with a self-defined structure, indicating the special properties and meanings of the token. Validity Check is the main game logic of the puzzle game. When a player makes a move, validity checking is done in this smart contract.

There are also other smart contracts borrowed from external libraries to aid game development. For example, the Safe Math library prevents arithmetic overflow. Simple Store contract is also used for debugging purposes in the beginning of smart contract development.
5.5.4.1 Fluffy Token Smart Contract

Fig. 14 shows the extracts from the Fluffy Token smart contract. In the beginning, there are a few mappings storing token ownership, player balance, as well as indices to
Fluffy structure. A Fluffy structure basically defines appearance and battle statistics of a character. The appearance data is stored in the smart contract so that Unity can instantiate the same character in different scenes.

```solidity
function generateRandomDna(string _str) public view returns(uint){
    uint rand = uint(keccak256(abi.encodePacked(_str)));
    return rand % dnaModulus;
}
```

Fig. 15 DNA Generator

Apart from defining the appearance data, each character should have a special string uniquely representing itself. A random string of varied lengths and characters is passed as the arguments of “generateRandomDna” function (see Fig. 15). By using keccak256() hashing algorithm, it returns a long random unsigned integer. With the help of modulus, self-defined length of integers can be obtained.
5.5.4.2 Validity Check Smart Contract

The Validity Check smart contract mainly responsible for three tasks. First, it checks whether the player selects the correct number of marbles (see function `checkSelectedMarble` in Fig. 16). The result is sent back to the Unity application.

Next, the sequence of the selected marbles is checked. The two arrays are the x-coordinates and y-coordinates of those selected marbles. After calculating the distance, the correct sequence can easily be checked. The result is also sent back to the Unity application.
Finally, the colour of the selected marbles has to be the same in order to fulfil the requirement of a valid move. The function takes the hash code of colour array and the result is sent back to Unity for updating UI.

5.5.5 Unity Application

This project consists of 5 modules in the Unity application. Each module has more one scripts for handling in-game data communication, smart contract invocation and asset instantiation. Scripts that are responsible for handling UI are also important in designing game flow. However, this sections only focus on the implementation related to main components.

5.5.5.1 Menu Module

The Menu Module is the beginning of the application where players can browse to other modules by clicking on buttons. There is a checking for the team list. The player cannot access the Game Play Module until the player owns a team of 3 characters. The middle part also shows the in-game currency owned by the player.
5.5.5.2 Top-up Module

The Top-up Module allows players to add funds to the game for purchasing characters. As for game prototypes, the currency value is stored in local storage. This module should be handled in external payment gateways in real production.

5.5.5.3 Marketplace Module
In the Marketplace Module, a function call is made to the DAppChain for obtaining a list of available character. Then, the list of available characters is instantiated in the middle part. Players can browse through the list by clicking the left and right buttons. When players click on a character, the character information panel is shown in the centre of the screen. Information includes level, HP, attack, defence, speed, price and DNA. Players can click “Buy” button to purchase a character. This invokes an external function call to a transfer function resides in Fluffy Token smart contract.

5.5.5.4 Inventory Module

In the Inventory Module, a function call is made to the DAppChain for obtaining the player’s inventory list. The layout design is similar to the Marketplace module for consistency. When the player clicks a character, the character information panel is shown. There is also an “Add” button on the bottom right corner (see Fig.20). This triggers an internal function call for saving a character in a team. The team information is stored in a transparent game object for usage in different scenes. It also adds a level of security compared to Unity PlayerPrefs, which is only a key-value data store.
The red tick on the top right corner (see Fig. 21) indicates the current character has been already added to the team. After adding a character into the team, the button changes to “Remove” for removing the newly added character. This action is also an internal function call.

5.5.5.5 Game Play Module
In the Game Play Module, enemy and the team list of the player are instantiated with their respective health bars (see Fig. 22). When the player clicks “START”, the battle starts.

In Fig. 23, the label near the top of the screen indicates the current turn. When the player clicks on a marble, a marble is selected. When the player clicks “OK” button on the bottom right corner, a function call is made to the Validity Check smart contract (see Fig. 16).

![Game Play Module](image)

Fig. 23 Game Play Module - after
5.6 Testing

The testing phase starts right after the implementation phase starts. The project is divided into several modules to simplify the testing procedures. Unit testing is crucial to avoid unexpected situation from happening. It is easier to track the problems and bugs by testing each individual module separately.

Apart from unit testing, testing of smart contracts is vital to avoid unnecessary behavior. Once the smart contracts are deployed, it is difficult to make changes and is subject to loss and failure. A tool called Truffle is used for smart contract development, testing and deployment. Truffle compiles all smart contracts and deploys to the local DAppChain.

Apart from smart contract testing, testing of modules in Unity is carried out. With the use of log message and variable assignment, the logic flow can be easily checked. The most difficult part of the logic is the game loop. During the implementation phase, infinite loop happened in the game logic part. By applying more debug messages, the game flow is successfully tested and functioned normally.

Moreover, user acceptance test is also carried out. During the initial release of the game, it is important to know whether the user interfaces and the gameplay are acceptable for the players. There are different metrics for assessment, for example, user interface design, gameplay organization, graphics and responsiveness. After collecting some feedbacks from beta testers, improvements have been made.
6 Limitations and Difficulties

Although Loom network is used to scale the blockchain game, the transaction speed to the Ethereum network is still slow. It is limited by the nature of public blockchain network. In order to ensure a high level of security, mining is required which results in transaction latency. Therefore, the transactions to the Ethereum network are reduced as many as possible for optimization. Most of the transactions are accommodated in DAppChain for higher transaction speed.

Moreover, the Loom SDK was released on June, 2018. It is a newly launched development tool for developing applications under Loom network. There are only a few online tutorials and implementations. It is difficult to understand how to communicate with different components in Loom network at first. In order to cope with this problem, testing of smart contracts and thorough understanding of the documentations are crucial in developing this project.

On the other hand, there was a problem encountered when testing a DAppChain example with Unity and Truffle frameworks in GitHub. There were missing files and the ABIs were not successfully exported. Therefore, I try to set up the development environment from the beginning by viewing the Truffle framework documentation on my own. After the initial setup, the example was compiled successfully without any error.
7 Future Plan

First, the current turn-based battling lacks animation due to time constraints. Animation of characters help the players to understand the current state of the game. Also, it adds aesthetics to the game.

Second, the character information panel shows the DNA of the character to distinguish between characters. A function about changing the name of the character is essential so that the players can easily recognize a list of characters. Then, a new mapping of integer to string has to be defined in the Fluffy Token smart contract.

Moreover, the current battling mechanism only allows players to play with the generated enemy. A player-to-player mode should be supported to enhance the interactivity of the gameplay. Also, playing with another player requires more skills which in turn enhance gaming experience.

Finally, the smart contracts’ availability should be tested. A smart contract deploying in the DAppChain supports 500-1000 transaction per seconds, according to a developer in the Loom SDK Telegram group. It is essential to test whether the smart contract is fault tolerant. If all smart contracts pass the test, the DAppChain application is ready for production.
8 Conclusion

By and large, game design in public blockchain suffers from latency. Various layer 2 solutions are being developed to support high scalability of the blockchain network. They also help reduce the transaction latency in the public blockchain via sidechains.

Having discussed the rationale behind framework choices, Ethereum and Loom network are used for implementing the turn-based battling game. Ethereum supports pluggable consensus and smart contracts, assisting in structuring the game. Loom network provides direct support to the game development platforms, saving time for testing and deployment. The Loom network also provides example games for developers. Moreover, thorough design analysis of the game has been completed with illustrations (see Section 5.4). Details of the implementation are explained with data flow diagram.

Finally, a fully functional blockchain game prototypes has been constructed. However, there are still rooms for improvements such as, animation, and gameplay interactivity. In general, the game prototypes meet the objectives mentioned above. Having completed this project, sidechain technologies are the next force in shaping the development of the gaming industry.
References


[4]. M. Greg, “Do you Really Own Your CryptoKitties?”. [Figure]. Available: https://medium.com/@gmcmullen/do-you-really-own-your-cryptokitties-d2731d3491a9 [Accessed Nov. 21, 2018]


