Imperial College London

COURSEWORK

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Principle of Distributed Ledgers

Author:
Alexandre Allani (CID: 01797836)

Date: February 16, 2020

1 Introduction

In this report, I will describe the different details of my Cryptomon's implementation. I will try to emphasize the following points:

- Game logic (backend) on the blockchain
- Details on the User Interface
- Security measures

You will find the code a README.md file which explains all the steps to create the private blockchain (using ganache-cli), migrate the contracts (using Truffle) and run the UI (using npm).

2 Game Logic

To create the game logic I coded two contracts and on library :

- *Cryptomons.sol* : Contains buying, exchanging, breeding logic.
- *FightingRoom.sol* : Contains all fighting logic of the game.
- *myLib.sol* : Contains pure helpers, and it is used to reduce the deployment cost in gas

The next subsections are a description of the buying, exchanging, breeding and fighting logic.

2.1 Buying

The function described in this section belongs to the contract Cryptomons.sol In this game there are four ways of obtaining a new Pokemon:

- Buying from the store
- Buying from a player
- Exchanging with a Pokemon with another one
- Breeding two Pokemons

At the beginning of the game, there are 8 Pokemons available in the store, each costing 1 ETHER. The transaction with the store is simple: a player buys a card from the store simply by calling the function *buyCard* and adding to the transaction a price of 1 ETHER. After that, the card bought by the player is not anymore available in the store and belongs to him.

If a player wants to sell one of his Pokemon, he can put it on the market by calling the function *makeSellable* (*makeUnsellable* has the opposite effect). Other players can then make an offer to buy this Pokemon with *makeOffer*. Finally the owner of this pokemon can check the available offers with *getLengthListBuyer getAddressBuyer* and *getOffer*. The first function gets the length of the array containing all the possible buyers. The second function gets the addresses of these buyers. And finally, the last function returns the offer of a specific buyer. If the player owning the Pokemon finds an interesting offer he can accept it with *acceptOffer*. The player who made the offer finally pays for the accepted price with *buyCardFromPlayer*. The Pokemon's owner can accept multiple offers, however, only the first one to pay will get the Pokemon. The following schema describes the different steps:

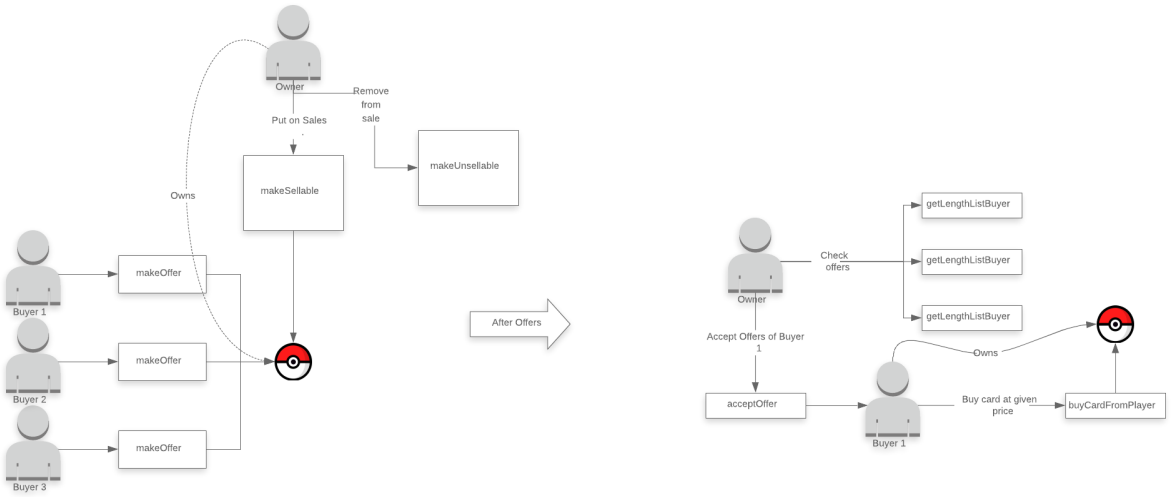

Figure 1: Buy card from player flow

2.2 Exchanging

The function described in this section belongs to the contract Cryptomons.sol Exchanging follow the same principle. However, since there is no money involved (except the gas price of the different transactions), the exchange is immediate. A card owner first selects one of his Pokemon A and Pokemon B he wants to exchange with and makes an offer with *makeOfferExchange*. The owner of Pokemon B now can check all the exchange offer he has on Pokemon B with *getLengthListExchange getAddressExhchange* and *getOfferExchange*. These functions are equivalent to the buying ones but for exchange. Finally, if Pokemon B's owner finds an offer that suits its expectations, he can accept the offer with *acceptOfferExchange*. The exchange is immediate, no need for Pokemon A's owner to accept again the trade.

2.3 Breeding

The function described in this section belongs to the contract $Cryptomons.sol$ Breeding is the last way of having a new Pokemon. To be able to breed, a user needs to own at least 2 Pokemons. These Pokemons can be of the same type, however.
A player can start breeding by selecting two of its Pokemon (A and B) and use the function $startBreeding$. A and B should not be fighting or breeding, otherwise $startBreeding$ will throw an error. After a certain amount of time (in seconds) which is based on this formula:

$$time = max(levelof(A), levelof(B)) * 60$$

The player can then collect the result of its breeding via $collectBreeding$. A new Pokemon is then created. All of its properties (species, attack, defense, and health points) are the mean of its parent.

2.4 Fighting

The function described in this section belongs to the contract $FightingRoom.sol$ Fighting in my game is round-based. Each Pokemon has two possible actions, $attack$ and $defend$. $attack$ remove HP from the opponent life bar and $defend$ reduce the damage taken by the Pokemon. The fight ends when one of the two pokemon has a no more HP left. At the end of the Fight the winner gain XP points. If the winner has enough XP he can gain a level. A gain in level basically improves the statistics of the Pokemon.

2.4.1 Start a Fight

A player can start a fight by selecting one of its Pokemon (A) and an opponent (B) from other players. A and B should be different Pokemon and owned by two different players. Moreover, both Pokemons should not be either breeding or fighting. As in the real game Pokemon, the owner is forced to do the fight. The fight starts by $startFight$ (from $Cryptomons.sol$). This creates an instance of $FightRoom.sol$.

2.5 Fight

A pokemon has the following statistics:

- typ : Water, Plant, Air (Flying) and Fire.
- hp : Health Point.
- attack : Damage that can deal the pokemon
- def : Linked to the number of turn in defense mode
- lvl : Level of the Pokemon

- bonusCritic : Chance of doing double damage
- dodgeChance : Chance of dodging an attack

The following alterations can be applied after each attack:

- Fire : High Tick damage applied over short time. The tick damage is based on the attack statistic
- Poison : Low Tick damage applied over a long time. The tick damage is based on the attack statistic
- Soaked : Higher chance to miss an attack. The chance is based on the opponent lvl statistic
- In Air : Higher chance to receive a critic (double damage). The chance is based on the opponent lvl statistic
- In Def : Higher chance of dodging an attack. The chance is based on the def statistic

The fight is round based. The first one to play is decided by chance. A Pokemon can have an advantage of type (see Appendix). If A has an advantage of type, the following advantages are applied to A :

- More Attack Points (+10%)
- More Defense Points (+10%)
- More Health Points (+10%)

A Pokemon can defend himself with $actionDefend$. This removes all the alterations the Pokemon had. Moreover, it gives to the Pokemon the defense alteration. According to its def statistic, the pokemon remains with this alteration a certain number of turns. With this alteration, the attacks of the opponent are reduced by a fixed percentage.

A Pokemon can also attack with $actionAttack$. This action deals damage to the opponent's pokemon based on the attack statistic. Each $actionAttack$ has a chance of applying an alteration listed above. The number of turn for each alteration is based on the lvl statistic.
Let's imagine that A won the fight (ie B had a negative value of hp). Then Pokemon A gains XP. The XP that Pokemon A gains depends on the difference of level A has with B. The bigger the difference the less A gains XP. After reaching 1000 XP, Pokemon A gains a level. This increases the attack and def statistic by 10.

3 User Interface

In this section I will present the relevant tabs of my User Interface. You can find all possible interactions with my UI in the ”videos” folder. In case my UI doesn't work on your computer, I have recorded how it is supposed to work.

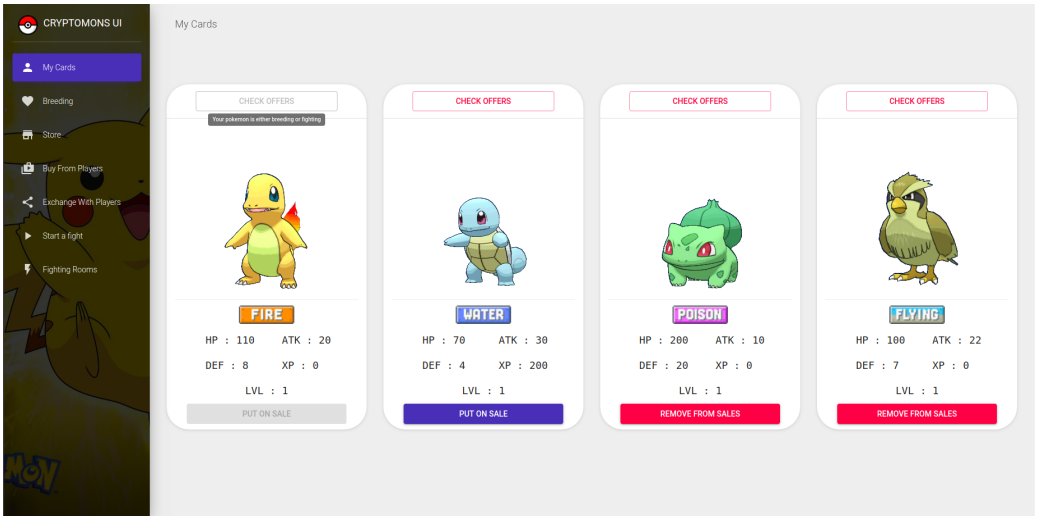

Figure 2: My Cards tab

This tab presents all the cards owned by the player. Check offers button open a Dialog that presents the exchange and priced offers (see Appendix>Offers and Exchange). These buttons might be disabled if the Pokemon is either Fighting or breeding

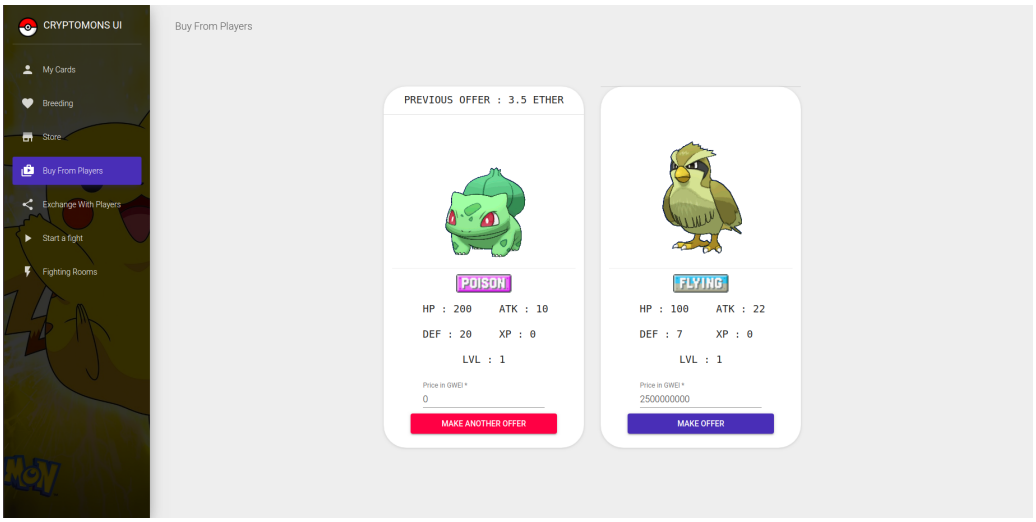

Figure 3: Buy from players tab

In these tabs, you can make priced offers on available Pokemons. If an offer has already been made, you will see the previous offer in the card header. Otherwise, you can make an offer in GWEI. Finally if your offer has been accepted, you will

see in the header (Appendix>Accepted offer) an accept offer button. By clicking on it and paying the required amount, the transaction is finished and you own the Pokemon.

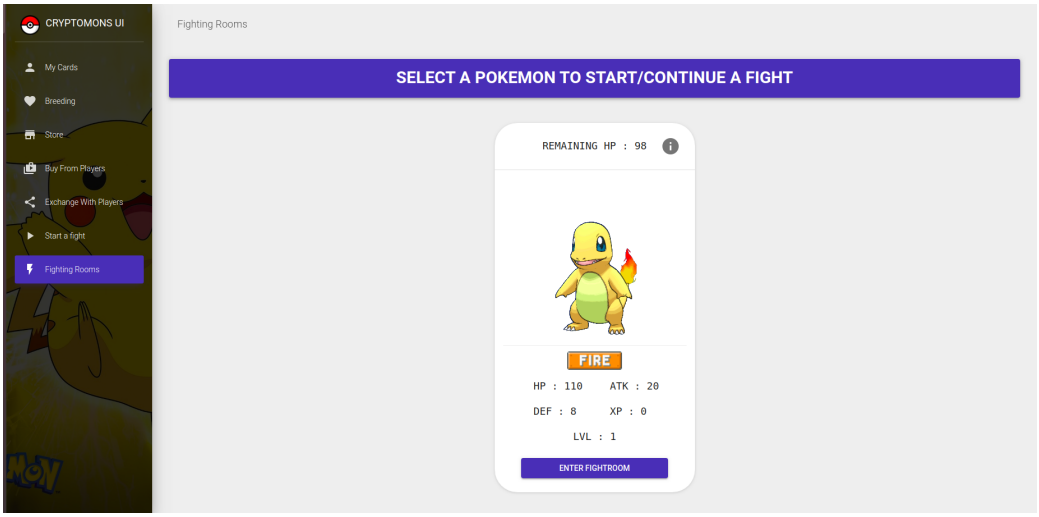

Figure 4: Fighting Rooms

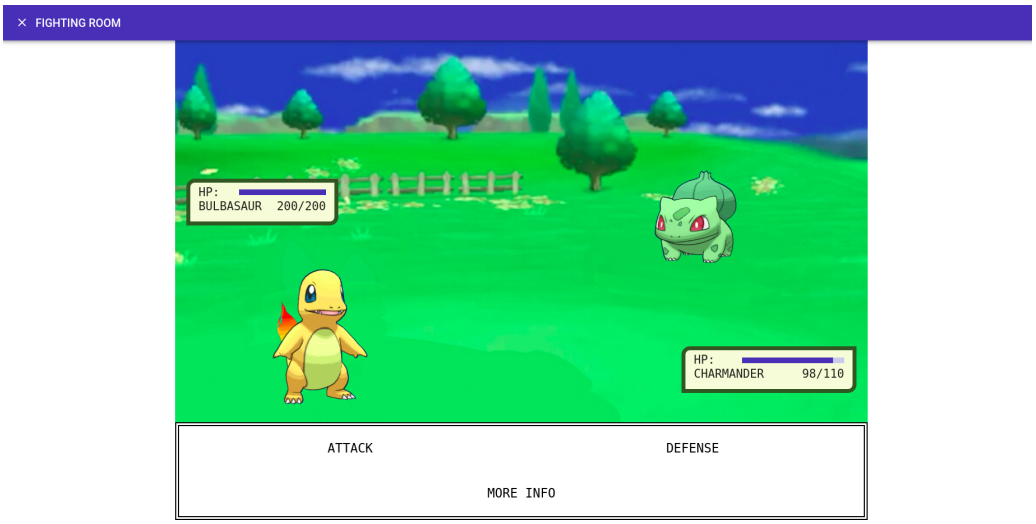

Figure 5: Fight

In the fighting room tab, you can select a Pokemon that is already in Fight and go to its fighting room. The Information button in the card header shows the statistics of the Pokemon and its opponent. By clicking on ”Enter Fightroom”, you can access to the fight room (**Figure 5**). If it is your turn your Pokemon can Attack and Defend based on the button you click on. The ”More Info” button also gives the statistics of your Pokemon and its opponent.
Breeding, Store, Exchange with Players, Start a fight are pretty straightforward, and easy to understand, so I will not present them in this report.

4 Security measures

In this section, I will try to tackle the different difficulties I have encountered while coding the solidity contracts and how I tried to overcome them. I will also discuss

the possible abuse in my games.

4.1 Re-entrancy attacks

I have only two functions in my solidity codes that involve transactions with money (except for gas price). These functions are *buyCard* and *buyCardFromPlayer*. Both of these functions with the modifier cannot be victims of re-entrancy attacks. Since the player is directly paying to the script or the player, money is not in transition, hence no possible re-entrancy attacks are possible.

Other important functions are *gainXP* from *Cryptomons.sol* and *endFight* from *FightRoom.sol*. If both functions are called multiple times, then a Pokemon can have an infinite level. However, with the modifiers, these functions can only be runned once during a fight. Indeed, there are 2 flags, *hasEnded* and *hasExpired* that ensure this. *hasEnded* checks that the fight has already ended, otherwise *endFight* (which gives XP and up a level) should not be called. Moreover, after one call to *endFight* the flag *hasExpired* is raised. If it is, then *endFight* cannot be called anymore.

4.2 Authorized use of functions

Most transaction functions in my solidity code are linked to a modifier or a requirement. This is to ensure that the function is used only in the considered case.
For instance, the *startBreeding* function from *Cryptomons.sol* should only be called by the card owner. Moreover, both Pokemon should not be breeding nor fighting. And the Pokemons should also be different. You can see in the code the list of modifiers restraining to this usage only. Hence someone playing to my Cryptomon game without the UI, will not have an advantage.

4.3 Randomness and Time-Based functions

I use a lot of randomness in my code, especially in fightings.

4.3.1 In fighting rooms

In fighting room manipulating chance can be quite harmful, and lead to some Pokemon always applying alteration and doing critic attacks or dodging all the opponent attacks. I tried reducing chance manipulation as much as possible.
My pseudo-random function is based on a nounce (which is specific to each FightingRoom contract), the transaction hash and the block timestamp. A miner can indeed manipulate the chance, however, he has no incentives to do so. Indeed the only parameter that a miner could have an influence on is the block timestamp. He can compute a hash with a given timestamp to predict the next probabilities involved in the fight. However, this would require some computation power. Also, this would require that the miner wait a certain amount of time to push the block onto the blockchain.

However, on the Etherum blockchain, a block is generated every 15 seconds. This implies that the miner has at most 15 different couples (timestamp, transaction hash) to manipulate the chance. Plus the more time he waits, the more likely he will lose its block.
Finally, it is possible for the miner to not do any action for a block, and wait for the next block to have more chance. This is indeed a workaround. After waiting for a long time, the player can manipulate the chance.

4.3.2 In Breeding

I also use the block timestamp in Breeding. As said in the previous subsection, a miner can gain 15 seconds at most on the breeding by changing the block's timestamp. This is at risk of losing the block. Considering the time taken to breed, this is probably not worth doing so.

4.4 Possible abuses

Since anyone can start a fight, it is possible that a player starts a fight with a lot of Pokemon and never ends the fight by not playing on the game. I did not have the time to implement a time out which would be a solution to this problem. After a moment, probably a day, if someone did not make an action, its Pokemon would lose the fight immediately. However, such abuse would also lead the Pokemon owner not being able to play the game.

The number of Pokemon available in the store is limited, 8 at the moment. Only the manager of the game can create new Pokemon by calling $CreateCard$ function. The manager is a trusted party. If he wants, he can totally destroy the economy of the game by creating a lot of new Pokemon at a low price. Or he can also create Pokemons with high statistics and play with it.

The game is also unbalanced between the Pokemons, and I think it would need some modifications.

5 Appendix

5.1 Advantage of Type

For Air Type :

- Air > Fire
- Air < Plant
- Air ∼ Water
- Air ∼ Air

For Fire Type :

- Fire > Water
- Fire < Air
- Fire ∼ Plant
- Fire ∼ Fire

For Water Type :

- Water > Plant
- Water < Fire
- Water ∼ Air
- Water ∼ Water

For Plant Type :

- Plant > Air
- Plant < Water
- Plant ∼ Fire
- Plant ∼ Plant

5.2 Offers and Exchange

After clicking the Check Offers button in the tabs ”My Cards”:

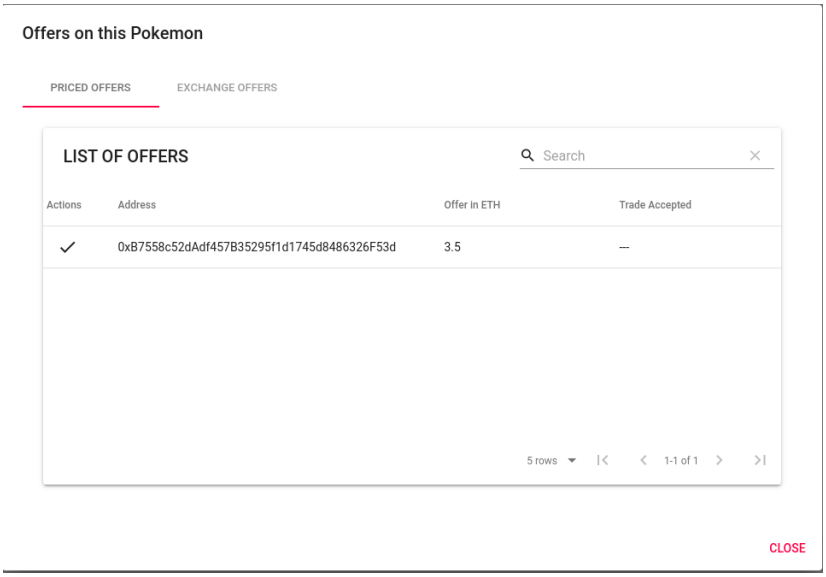

Figure 6: Offer Dialog.

After clicking Check Offers button in the tabs ”My Cards”:

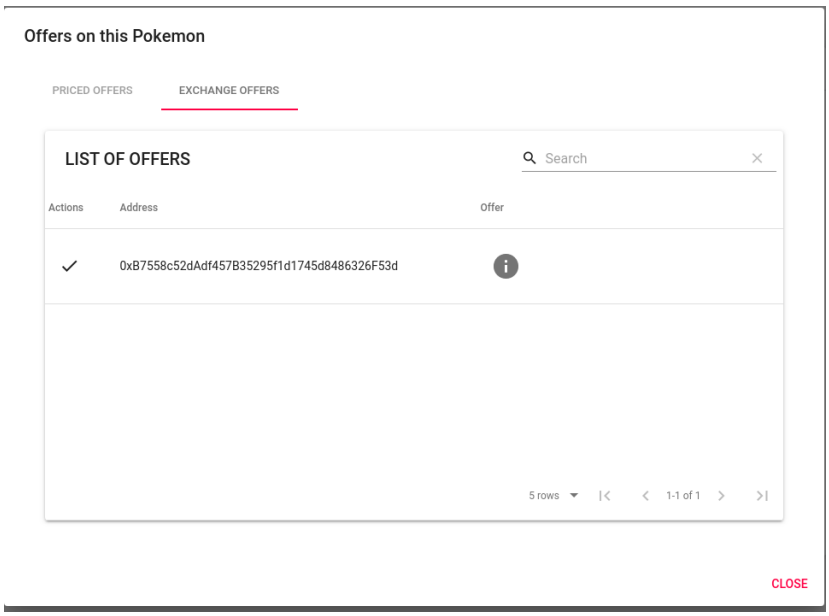

Figure 7: Exchange Dialog

In the Exchange Dialog, you can check the card statistics by clicking on the information button:

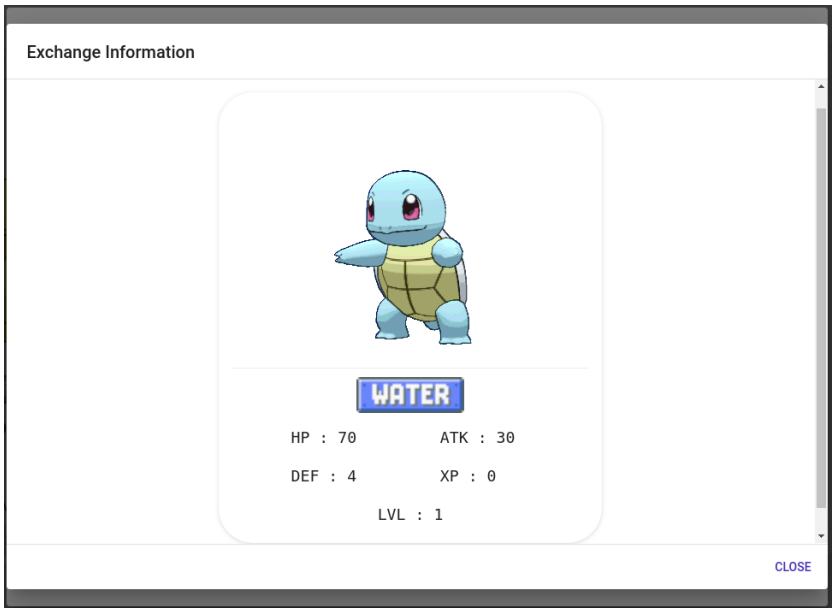

Figure 8: Exchange Dialog

5.3 Accepted offer

Figure 9: Accepted button in the Buy from Player