NFT Geno Car

Zhaomeng Wang	Ziqian Wang	Zifan Chen	Ziping Chen	Xiaohang He
zw2801	zw2697	zc2628	zc2431	xh2509

Repo Link: https://github.com/Oliverpapa/6883final

Abstract

In this paper, we show the whole process of designing and building an NFT Geno Car production. We used Remix to edit and deploy our smart contract. We also used it to perform transactions when we needed to update our contract or mint one NFT Geno Car to our own address. We used OpenSea testnet to display our storage of our NFT Geno Cars. We also built our own web application that could visualize the Geno Car, test its performance, and rank them using a global leaderboard.

1. Introduction

NFT is an inseparable and unique digital certificate that can be mapped to a specific asset, and the relevant rights content of the specific asset, historical transaction flow information, etc. are recorded in the label information of its smart contract, and on the corresponding blockchain. Generate a unique code that cannot be tampered with for that particular asset, ensuring its uniqueness and authenticity. NFT realizes the assetization of virtual items, so that digital assets have tradable entities.

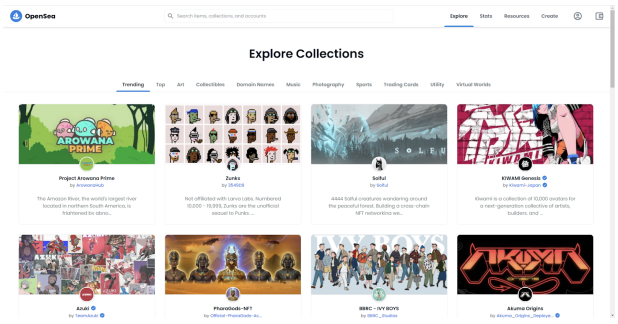

Figure 1: NFT Marketplace on OpenSea

In recent years, the most common NFT assets are undoubtedly NFT images, and most people have participated in related transactions. However, in the opinion of our group, although the existing NFT images on the market have their unique artistic value, they do not have much practical value. We want to publish a series of NFT images with entertainment features. Each image is not only an individual's unique asset, it can also be used for some entertaining applications on the website we developed. Therefore, we designed and built the NFT Geno Car.

2. Originality and Novelty

2.1. Genetic Algorithm

Our project is inspired by the genetic algorithm below.

Genetic Algorithm

```
has_top_candidate = False
while True:
  for n time:
    candidate_list += GenerateRandomObj()
  if has_top_candidate:
    candidate_list.append(offspring
      ,first_candidate,second_candidate)
  top_two_score = [0,0]
  candidate1 = candidate2 = None
  for current_candidate in candidate_list:
    score = evaluate(current_candidate)
    if score > top_two_score[0]:
      candidate1 = current_candidate
      top_two_score[0] = score
      continue
    if score > top_two_score[1]:
      candidate2 = current_candidate
      top_two_score[1] = score
  offspring = NewEmptyObj()
  for feature in all features in offspring:
    feature = RandomSelect_50_50(
      candidate1.feature,candidate2.feature)
    feature = RandomSelect_95_5(feature
      ,RandomFeatureValue())
  has_top_candidate = True
```

Our project uses simplified cars' feature combinations as

NFT content, we call it Geno Car. Each Geno Car represents a simplified two-dimensional car in the form of a picture. Just like how the genetic algorithm generates the candidates randomly. Each new Geno Car will have the following random but bounded features to determine their shape and structure: ***wheel_radius, wheel_density, wheel_vertex, chassis_density,*** and ***vertex_list***. Everytime a Geno Car is minted, we will use the random number mechanism to assign random values to these features of this Geno Car, and store these features as formatted string into this NFT's MetaData as attributes. One possible feature configuration could look like following:

```
'WR':'04','WD':'48','WV':'1_6_2_3_7_5_
0_4','CD':'2011','VL':'01_12_17_05_14_
02_07_12_19_06_11_04'"
```

And, if the buyer wants to get the picture corresponding to a certain Geno Car, the picture could be visualized via our web application using that Geno Car's MetaData.

Similar to evaluating candidates in the genetic algorithm via heuristic function, our web application will use the same randomly generated bumpy 2D route to test the distant one Geno Car can travel before stucked. If users want to know how their Geno Car is performing, just upload the MetaData of that Geno Car to our web application, and you can get the distance/score of that Geno Car. The web application also has a leaderboard for all the tested Geno Cars.

Our Geno Car also supports merge. Instead of randomly merging features of top two candidates in each generation to generate the "offspring" in genetic algorithms, we give the choice to users. A user could choose any two Geno Cars it owns as "parents" and give up its ownership to these two Geno Cars to gain a new Geno Car. The generation of this new Geno Car is similar to the generation of "offspring" in genetic algorithms but without the mutation part.

2.2. Smart Contract

For the smart contract section, we used ERC721 as our template.

2.2.1 Transaction

In the module of user transaction users can do some partial reveal that means selectively reveal attributes about their NFT. For instance, they can prove that a certain attribute is greater than X, without revealing such attribute. For the seller, this is effective in driving up speculation while maintaining leverage over information asymmetry. The buyer can also filter tha attribute they want to a certain extent for example, the sum of certain attributes must be greater than Y. Transaction can be made once all verifications are passed. In this situation user A creates her car and keeps these properties secret from herself(off-chain) and submits its hash on the chain (submission proof). A few hours later many other players such as B also create their cars. As the game progresses, many people have realized it is useful to have large tires and a short body, and A happens to have one of these cars. However because the information is hidden others are not aware of it. Seeing that her car was desirable, she decided to sell it. In an effort to create speculation and movement in the marketplace, A partially reveals (using SNARKs) that her cart does have a tire size over 7, but does not reveal the exact data (realistically she should be able to reveal any proven combinations). This generates interest from buyers and B decides to bid 1ETH. Meanwhile, A retains some leverage and information asymmetry by not revealing the entire NFT.

Seeing that B's bid is fair, A accepts the bid and sells her role. When she accepts the bid, she surrenders the rights to her Geno Car and is also limited to revealing all three attributes of her Geno Car at the same time. B is very happy he knows that the tire size attribute is higher than 7 and is happy to find out that it is actually 9 and thinks he got a good deal. A, too, thinks she got a decent deal because her cart's other attributes such as height and shape are poor (but she did not need to reveal it to B), she suspects that in the future people will find that the height and shape of the car will be more important, so she is happy to sell it.

2.2.2 Random Generator

We use the VRFConsumerBase.sol to interact with the Chainlink VRF and get our random numbers for each feature of each Geno Car.

We define Geno Car's attributes in the Feature structure and create a list of attributes. We want to keep track of the requestId so that when the random number is generated, we can map it to the Geno Car we are creating. Once the Chainlink node has finished processing the request, it will call the fulfillRandomness function. This function will generate the attributes, add the car to the list, and mint the NFT.

```
198     function fulfillRandomness(bytes32 requestId, uint256 randomNumber)
199         internal
200         override
201     {
202         uint256 newId = features.length;
203         uint256 WR = (randomNumber % 100);
204         uint256 WD = ((randomNumber % 10000) / 100 );
205         uint256 CD = ((randomNumber % 1000000) / 10000 );
206         uint256 VL = ((randomNumber % 100000000) / 1000000 );
207         uint256 WV = ((randomNumber % 10000000000) / 100000000 );
208
209         features.push(
210             Feature(
211                 WR,
212                 WD,
213                 CD,
214                 VL,
215                 WV,
216                 requestToFeatureName[requestId]
217             )
218         );
219         _safeMint(requestToSender[requestId], newId);
220     }
```

Figure 2: Random Generation

2.2.3 Merge

For each time one address wants to merge two Geno Cars owned by it. The contract will check if two Geno Car is owned by this address and if there is enough fee in this transaction to perform merge. After checking this information via required functions, the smart contract will gather Metadata from these two Geno Cars and extract their feature values. Then, these two Geno Cars will be removed from that address, and one newly generated Geno Car will be minted which merges features of two removed Geno Car. The newly generated Geno Car will then sent to that address.

2.3. Marketability

In our policy, each address can only own up to five Geno Cars at any time. Therefore, when an address has already owned five Geno Cars and still wants to get a new Geno Car, it could 1. sell one it owns first to make the space for a new Geno Car or 2. merge two selected Geno Cars it owns to generate a new car. Latter will cause two selected Geno Car removed from that address and a new generated Geno Car that merged from that two selected Geno Car will be added to that address.

In order to ensure the marketability of our issued NFTs, we will just remove the two Geno Cars that merged and not reissue these two, which will be one of the strategies to balance the market. First, as long as we remove the already fused Geno Car from the market, we can somewhat avoid the phenomenon of the market overpaying for the already existing high performance Geno Car. Because people will merge two high performance cars for a higher performance Geno Car, a significant portion of the good performance cars will be removed because of the fusion. In this case, people will be more likely to acquire a new high performance Geno Car by obtaining a new NFT. Secondly, this will also increase the fun of our product and make people focus more on the Geno Car with great randomness that has not been tapped yet.

3. Showcase of your DAPP

We deployed our smart contract via Remix and complete our purchase on it as well.
To mint a Geno Car, we connected our wallet to Remix and ran a transact like following in Figure 3:

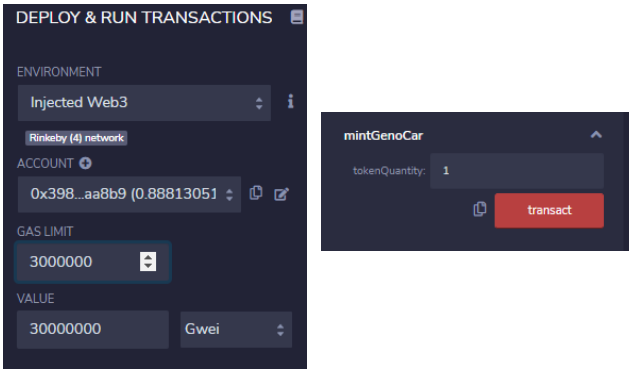

Figure 3: Run a Transaction

After we successfully finished the transaction, we got a confirmation log like following in Figure 4:

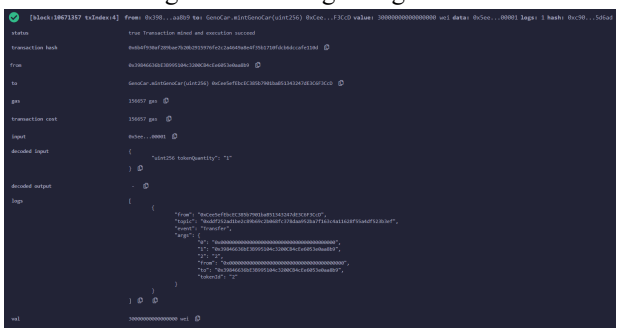

Figure 4: Contract Log Showing Successfully Mint a Geno Car

We could check our storage on testnets.opensea.io after we link our walet to it. The Figure 5 below shows the Geno Car we mint.

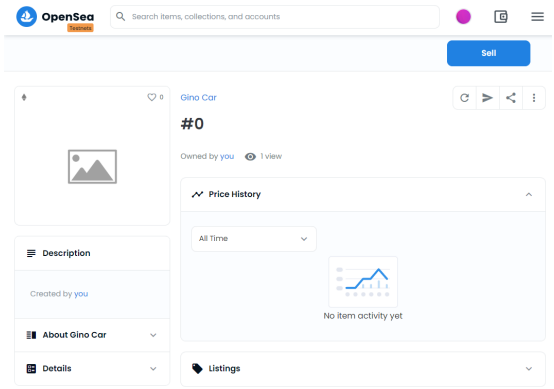

Figure 5: Geno Car in Under Our Wallet/Address

We could also get MetaData as json file using NFT's URI, the Metadata of our third Geno Car is shown below in Figure 6:


```
{"name": "Your Geno Car #2", "description": "Geno Car", "dna":
"0de3c8add53f3a8cd2c6a6553dad9f3a646255121", "attributes":
[{"type": "WR", "value": "04"}, {"type": "WD", "value": "48"},
{"type": "WV", "value": "1_6_2_3_7_5_0_4"}, {"type": "CD",
"value": "2011"}, {"type": "VL", "value":
"01_12_17_05_14_02_07_12_19_06_11_04"}]}
```


Figure 6: MetaData of Geno Car #2

After we get Geno Car features, we will be able to get the picture of our Geno Car and test it on our web application.

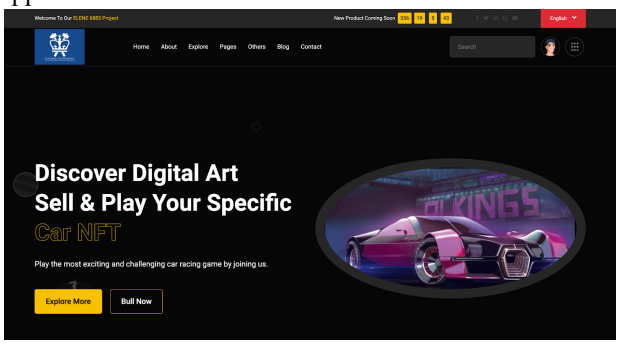

Figure 7: Homepage of Our Web Application

The Figure below shows how to check your NFT Geno Car picture by using its MetaData. The web application will provide a preview of the car in 2D after the user type in the MetaData of their Geno Car.

Figure 8: Geno Car Visualization via Its MetaData

Our web application will offer other functionalities, such as live bidding, selling leaderboard, and user dashboard in the future. Demos of these functionalities are shown in Figure 9 - Figure 11.

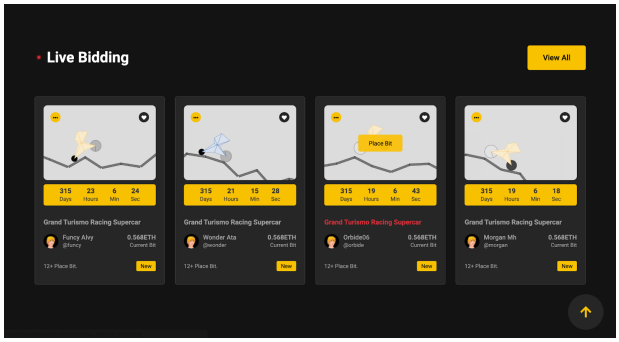

Figure 9: Live Bidding Demo in Web Application

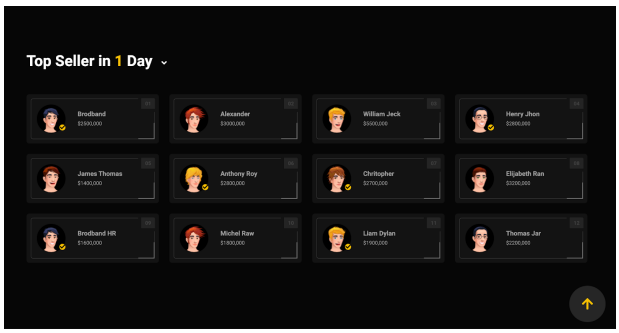

Figure 10: Selling Ranking Demo in Web Application

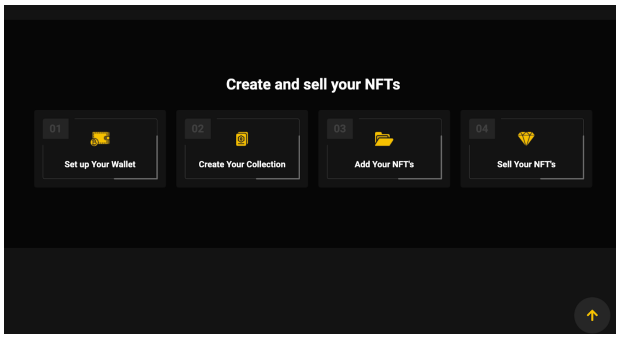

Figure 11: User Dashboard Demo in Web Application

4. Conclusion

Our project successfully implemented the random generation of geno cars using the genetic algorithm, and encapsulated it as an NFT. The smart contract we built includes mint NFT, setting the number of NFT, setting the maximum number of NFT each address can hold, and implementing conditional transactions between users. We successfully deployed the contract on the chain, interacted with the Metamask wallet, and successfully saw the NFTs we mined on the OpenSea test network. Our NFT smart app has good front and back-end interaction, so users can check the NFT they own, the performance of their own smart carts, and trade at any time. (some functionalities for our web application have not been implemented yet, just have the demo)

References

[1] https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol

[2] https://github.com/PatrickAlphaC/dungeons-and-dragons-nft

[3] https://github.com/kevinz917/zk-NFT/blob/main/contracts/NFT.sol

[4] https://rednuht.org/genetic_cars_2/

[5] http://www.bootstrapmb.com/item/12196