Battle Game on Blockchain

Professor: **Maria Potop-Butucaru**

Students: **Emilie LIN, Massyl BENGANA**

Contents

1 Repository

All project codes and contracts are available in the following repository: https://github.com/DHinode/tamagotchi-anime

2 Introduction

This project implements a system of digital creatures inspired by the Tamagotchi and Pokémon universes. Each creature is defined and managed by smart contracts deployed on the blockchain. Its state is stored on-chain, ensuring persistence, immutability, and traceability of its evolution. Users have verifiable ownership of their creatures without a central authority.
The main interactions (feeding, working, duels) are performed as blockchain transactions. To participate in a duel, each user must pay a fixed on-chain cost. At the end of the combat, the funds are automatically transferred to the winner by the smart contract, according to deterministic and transparent rules.

3 Detailed Concept

The project is built around the AnimeGotchi smart contract, which inherits from the ERC721 standard (for the creature) and interacts with an ERC20 contract (GotchiToken) for the economy.

3.1 The Living Asset (On-Chain State)

Unlike traditional NFTs which are static images, the AnimeGotchi is a dynamic "Living Asset". Its metadata is not stored on a server but calculated in real-time by the smart contract based on a state machine . A struct `GotchiStats` stores the following mutable attributes for each Token ID:

- **Name & Level**: Identity and experience progress.
- **Strength**: Combat power used during duels.
- **Hunger (0-100)**: A gauge that fills up. If it reaches 100, the system crashes.
- **Happiness (0-100)**: A gauge that empties. If it reaches 0, the system crashes.

- **isCrazy**: A boolean flag indicating if the creature is in a "Blue Screen of Death" state.

3.2 Time Management

Since the smart contract cannot run in the background, the creature's stats are updated automatically at the start of every interaction.

- The contract checks how much time has passed since the last action.
- To speed up the demo, every 15 seconds, the creature gains 1 point of Hunger and loses 1 point of Happiness.
- If these changes push stats beyond their limits (Hunger ≥ 100 or Happiness $= 0$), the creature crashes immediately.

3.3 Sanity Check Modifier

Security is enforced by a custom modifier called `checkSanity`. This modifier is applied to every game function. It acts as a gatekeeper that:

1. Updates the stats based on the time passed.
2. Checks if the creature is still alive (not `isCrazy`).
3. If the creature has crashed, it reverts the transaction with an error, forcing the user to call the repair function.

3.4 The Gameplay Loop

The system is automatic: as time passes on the blockchain, the robot gets hungry and sad. If you wait too long, it crashes and stops working. To stay alive and become the best, you have a complete cycle :

1. **Work**: You send your robot to work, it gets tired, but you earn `tokens`.
2. **Feed**: You spend these tokens to heal your robot.
3. **Train**: You pay to increase its Level and Strength.

Finally, the most important part is The Duel. It is like a bet where you pay a fee to enter the arena. The code compares your Strength and Happiness. If you win, you take the money and gain levels; if you lose, your robot becomes very sad.

4 Game Mechanics & Economy

The game economy relies on the token and strict rules enforced by the smart contract.

4.1 Minting

The `mintGotchi` function allows a user to create a new creature for free. It initializes the stats with optimal values and records the block timestamp.

4.2 Work

The `work` function allows the robot to mine 15 tokens. However, this action increases hunger and lowers happiness, carrying a risk of burnout.

4.3 Feed

The `feed` function costs 5 tokens to perform. It completely resets hunger to 0 and restores happiness to 100%, preventing a system crash.

4.4 Train

The `train` function costs 10 tokens and increases the robot's Strength and Level. It causes a massive hunger spike (+20), so players must be careful not to overload the creature.

4.5 Reboot System

If the robot crashes due to neglect, the `rebootSystem` function is the only fix. It imposes a heavy penalty of 100 Tokens to fully repair the creature.

4.6 Duel

The `duel` function allows players to bet 50 tokens to fight another creature based on stats and luck. The winner takes the double reward, while the loser suffers a significant drop in happiness.

5 Technical Stack

The technical architecture of the project is based on the following tools:

- **Solidity 0.8.20**: Used for writing smart contracts.
- **OpenZeppelin**: We used ‘ERC721‘ for the creature, ‘ERC20‘ for the currency, and ‘Ownable‘ for access control.

6 Conclusion

This project demonstrates a functional gamified economy on the blockchain. By managing the delicate balance between working (earning) and training (spending), users learn the constraints of smart contract interactions. The implementation ensures that the game remains dynamic without requiring an external server to update the state.

7 Deployment and Testing Instructions

To test the application, we use the Remix IDE (https://remix.ethereum.org/). Follow these steps to deploy and interact with the contracts.

7.1 Setup and Compilation

1. Open Remix IDE in your browser.
2. Create two files: `GotchiToken.sol` and `AnimeGotchi.sol`.
3. Paste the code provided in Section 8 into the respective files.
4. Go to the "Solidity Compiler" tab (on the left) and toggle the auto compile button.

7.2 Deployment

1. Go to the "Deploy & Run Transactions" tab.
2. Select the environment `Injected Provider - MetaMask`.
3. `Deploy GotchiToken`:
   - Select `GotchiToken` from the "Contract" dropdown.
   - Click `Deploy`.
   - Copy the address of the deployed contract.
4. `Deploy AnimeGotchi`:
   - Select `AnimeGotchi` from the "Contract" dropdown.
   - Paste the `GotchiToken` address into the constructor input field (`_tokenAddress`).
   - Click `Deploy`.

7.3 Configuration (Critical Step)

For the game to work, the AnimeGotchi contract must be authorized to mint tokens.

1. Expand the deployed `GotchiToken` contract.
2. Find the `addController` function.
3. Paste the address of the `AnimeGotchi` contract.
4. Click `transact`.

7.4 Playing the Game

1. Create a Creature:
   - In `AnimeGotchi`, find `mintGotchi`.
   - Enter a name (e.g., "Pikachu") and click `transact`.
2. Approve Spending:
   - Before feeding or training, you must approve the contract to spend your tokens.
   - In `GotchiToken`, find `approve`.
   - Spender: Address of `AnimeGotchi`. Amount: `100000000000000000000` (a large number).
   - Click `transact`.
3. Interact: the parameter being the number corresponding to the creation of the n^{th} creature.
   - Use `work(1)` to earn tokens,
   - Use `feed(1)` to lower hunger.
   - Use `train(1)` to level up (watch out for the hunger spike!).
4. Simulate Time:
   - Wait 15-30 seconds between actions to see the stats degrade automatically.
5. How to Duel:
   - Get an Opponent: Call `mintGotchi` again to create Token ID 2 (the enemy).
   - Fight: In the `duel` function, enter:
     - `_myId`: **1** (Your fighter)
     - `_enemyId`: **2** (The target)

- Click `transact`.
- Check Result: Look at the logs in the Remix terminal.
  - If you see 'DuelResult' with your ID first, you won (Level Up + Money).
  - If you see the enemy's ID first, you lost (Happiness Drop).

8 Annexes

8.1 GotchiToken.sol

This contract manages the currency used in the game. It includes an access control list to allow the game contract to mint tokens.

GotchiToken.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract GotchiToken is ERC20, Ownable {
    mapping(address => bool) public controllers;

    constructor() ERC20("GotchiFood", "FOOD") Ownable(msg.sender) {
        _mint(msg.sender, 1000 * 10 ** decimals());
    }

    function addController(address _controller) external onlyOwner {
        controllers[_controller] = true;
    }

    function mint(address to, uint256 amount) external {
        require(controllers[msg.sender] || msg.sender == owner(), "Non autorise");
        _mint(to, amount);
    }
}
```

8.2 AnimeGotchi.sol

This is the main game logic contract. It handles the NFT attributes, the state machine, and the interaction rules.

AnimeGotchi.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

interface IGotchiToken is IERC20 {
    function mint(address to, uint256 amount) external;
}

contract AnimeGotchi is ERC721, Ownable {

    uint256 public tokenIds;
    IGotchiToken public foodToken;

    event GotchiBorn(uint256 indexed id, address owner,
   string name);
    event ActionPerformed(uint256 indexed id, string
   actionType);
    event DuelResult(uint256 indexed winnerId, uint256
   indexed loserId, uint256 amountWon);
    event SystemCrash(uint256 indexed id, string reason);
    event SystemReboot(uint256 indexed id);

    struct GotchiStats {
        string name;
        uint256 level;
        uint256 strength;
        uint256 hunger;
        uint256 happiness;
        uint256 lastInteraction;
        bool isCrazy;
    }

    mapping(uint256 => GotchiStats) public gotchis;

    uint256 constant TRAIN_COST = 10 * 10**18;
    uint256 constant FEED_COST = 5 * 10**18;
    uint256 constant WORK_REWARD = 15 * 10**18;
    uint256 constant THERAPY_COST = 100 * 10**18;
    uint256 constant DUEL_COST = 50 * 10**18;

    constructor(address _tokenAddress) ERC721("SonicGotchi",
   "SGT") Ownable(msg.sender) {
        foodToken = IGotchiToken(_tokenAddress);
    }

    function mintGotchi(string memory _name) public {
        tokenIds++;
```

```
47          _mint(msg.sender, tokenIds);
48
49          gotchis[tokenIds] = GotchiStats({
50              name: _name,
51              level: 1,
52              strength: 10,
53              hunger: 0,
54              happiness: 100,
55              lastInteraction: block.timestamp,
56              isCrazy: false
57          });
58
59          emit GotchiBorn(tokenIds, msg.sender, _name);
60      }
61
62      function _updateStatus(uint256 _tokenId) internal {
63          GotchiStats storage g = gotchis[_tokenId];
64
65          if (g.isCrazy) return;
66
67          uint256 timePassed = block.timestamp - g.
    lastInteraction;
68
69          uint256 statChange = timePassed / 15;
70
71          if (statChange > 0) {
72              g.hunger += statChange;
73
74              if (g.happiness > statChange) {
75                  g.happiness -= statChange;
76              } else {
77                  g.happiness = 0;
78              }
79              g.lastInteraction = block.timestamp;
80          }
81
82          if (g.happiness == 0 || g.hunger >= 100) {
83              if(g.hunger > 100) g.hunger = 100;
84              g.isCrazy = true;
85              emit SystemCrash(_tokenId, "Negligence fatale");
86          }
87      }
88
89      modifier checkSanity(uint256 _tokenId) {
90          if (!gotchis[_tokenId].isCrazy) {
91               _updateStatus(_tokenId);
92          }
93          require(!gotchis[_tokenId].isCrazy, "CRASH SYSTEME :
    Utilise 'rebootSystem' !");
94          _;
95      }
96
```

```
97      function work(uint256 _tokenId) public checkSanity(
      _tokenId) {
98          require(ownerOf(_tokenId) == msg.sender, "Pas a toi")
      ;
99          GotchiStats storage g = gotchis[_tokenId];
100
101         uint256 burnOutRisk = 20 + (g.hunger / 2);
102
103         if (g.happiness > burnOutRisk) {
104             g.happiness -= burnOutRisk;
105         } else {
106             g.happiness = 0;
107         }
108
109         g.hunger += 10;
110         foodToken.mint(msg.sender, WORK_REWARD);
111
112         if (g.happiness == 0 || g.hunger >= 100) {
113             g.isCrazy = true;
114             emit SystemCrash(_tokenId, "Burnout au travail");
115         }
116
117         emit ActionPerformed(_tokenId, "Work");
118     }
119
120     function train(uint256 _tokenId) public checkSanity(
      _tokenId) {
121         require(ownerOf(_tokenId) == msg.sender, "Pas a toi")
      ;
122         bool success = foodToken.transferFrom(msg.sender,
      address(this), TRAIN_COST);
123         require(success, "Fonds insuffisants ou pas d'APPROVE
      ");
124
125         GotchiStats storage g = gotchis[_tokenId];
126         g.strength += 5;
127         g.hunger += 20;
128         g.level++;
129
130         if (g.hunger >= 100) {
131             g.isCrazy = true;
132             emit SystemCrash(_tokenId, "Surmenage physique");
133         }
134
135         emit ActionPerformed(_tokenId, "Train");
136     }
137
138     function feed(uint256 _tokenId) public checkSanity(
      _tokenId) {
139         bool success = foodToken.transferFrom(msg.sender,
      address(this), FEED_COST);
```

```
140         require(success, "Fonds insuffisants ou pas d'APPROVE
    ");
141
142         GotchiStats storage g = gotchis[_tokenId];
143         g.hunger = 0;
144         g.happiness = 100;
145
146         emit ActionPerformed(_tokenId, "Feed");
147      }
148
149      function rebootSystem(uint256 _tokenId) public {
150         require(ownerOf(_tokenId) == msg.sender, "Pas a toi")
    ;
151
152         _updateStatus(_tokenId);
153
154         GotchiStats storage g = gotchis[_tokenId];
155         require(g.isCrazy, "Le systeme fonctionne
    correctement, pas besoin de reboot.");
156
157         bool success = foodToken.transferFrom(msg.sender,
    address(this), THERAPY_COST);
158         require(success, "Pas assez d'argent");
159
160         g.isCrazy = false;
161         g.hunger = 0;
162         g.happiness = 100;
163         g.lastInteraction = block.timestamp;
164
165         emit SystemReboot(_tokenId);
166      }
167
168      function duel(uint256 _myId, uint256 _enemyId) public
    checkSanity(_myId) {
169         require(ownerOf(_myId) == msg.sender, "Pas ton Gotchi
    ");
170         require(_myId != _enemyId, "Impossible contre soi-
    meme");
171
172         bool betPlaced = foodToken.transferFrom(msg.sender,
    address(this), DUEL_COST);
173         require(betPlaced, "Paiement de la mise refuse");
174
175         _updateStatus(_enemyId);
176         GotchiStats storage myG = gotchis[_myId];
177         GotchiStats storage enemyG = gotchis[_enemyId];
178
179         require(!enemyG.isCrazy, "L'ennemi a crash.");
180
181         uint256 myPower = myG.strength * myG.happiness / 100;
182         uint256 enemyPower = enemyG.strength * enemyG.
    happiness / 100;
```

```
183
184         uint256 luck = uint256(keccak256(abi.encodePacked(
    block.timestamp, msg.sender))) % 10;
185
186         if (myPower + luck >= enemyPower) {
187             myG.level++;
188             myG.strength += 2;
189             enemyG.happiness = (enemyG.happiness > 20) ?
    enemyG.happiness - 20 : 0;
190             if (enemyG.happiness == 0) enemyG.isCrazy = true;
191
192             foodToken.transfer(msg.sender, DUEL_COST);
193             foodToken.mint(msg.sender, DUEL_COST);
194
195             emit DuelResult(_myId, _enemyId, DUEL_COST * 2);
196         } else {
197             myG.happiness = (myG.happiness > 20) ? myG.
    happiness - 20 : 0;
198             if (myG.happiness == 0) myG.isCrazy = true;
199
200             enemyG.level++;
201             enemyG.strength += 2;
202
203             address enemyOwner = ownerOf(_enemyId);
204             foodToken.transfer(enemyOwner, DUEL_COST);
205
206             emit DuelResult(_enemyId, _myId, DUEL_COST);
207         }
208      }
209
210      function withdrawEarnings() external onlyOwner {
211         uint256 balance = foodToken.balanceOf(address(this));
212         require(balance > 0, "Rien a retirer");
213         foodToken.transfer(msg.sender, balance);
214      }
215
216      function getStats(uint256 _tokenId) public view returns (
    string memory Nom, uint256 Force, uint256 Faim, uint256
    Bonheur, uint256 Niveau, string memory Etat) {
217         GotchiStats memory g = gotchis[_tokenId];
218
219         if (!g.isCrazy) {
220             uint256 timePassed = block.timestamp - g.
    lastInteraction;
221             uint256 statChange = timePassed / 15;
222
223             if (statChange > 0) {
224                 g.hunger += statChange;
225                 if (g.hunger > 100) g.hunger = 100;
226                 if (g.happiness > statChange) { g.happiness
    -= statChange; } else { g.happiness = 0; }
227             }
```

```
228             }
229             string memory statusStr = (g.isCrazy || g.hunger >=
        100 || g.happiness == 0) ? "CRASH SYSTEME" : "OPERATIONNEL
        ";
230             return (g.name, g.strength, g.hunger, g.happiness, g.
        level, statusStr);
231          }
232 }
```