# LoyaltyCandy: Unity–ICP Canister Integration Manual

## Description

This manual documents how LoyaltyCandy integrates a blockchain-enabled loyalty system using the Internet Computer (ICP) platform into a Unity game. It demonstrates how developers can authenticate users via Internet Identity (II), interact with NNS and custom game canisters to retrieve balances, submit scores, fetch leaderboard data, and distribute rewards on-chain.

## Architecture Overview

- **Unity Client:** Handles gameplay, UI, local caching, and calls canisters via EdjCase.
- **II Canister:** Manages user authentication and generates Principal IDs.
- **NNS Canister:** Provides reference/account verification data.
- **Custom Game Canisters:** Store game data, compute rankings, maintain leaderboard snapshots, and distribute weekly rewards.
- **Outputs:** Principal IDs, token balances, top scores, and weekly rewards.

All canisters are deployed to Mainnet using dfx, with Unity configured to connect to live endpoints. This manual highlights classes, methods, workflow, and Mainnet-ready deployment strategy, enabling developers to adopt ICP functionality into their own games without replicating gameplay mechanics. Limitations exist regarding on-chain updates, cycle usage, and account restrictions.

---

## Prerequisites (Project Setup First)

### 1. Clone the Repository

git clone: https://github.com/operalag/LoyaltyCandy.git

### 2. Open the Unity Project

In Unity Hub, choose **Open**, then select the folder: LoyaltyCandy.

Use Unity 2022.x / 2023.x or later (the project was developed with a recent Unity version; use the Editor version recommended in the repository if provided).

**Note:** Runtime and infrastructure prerequisites (dfx, ICP tokens, EdjCase packages, HTTPS requirements) are covered in Deployment and Unity Setup.

---

# Deployment / Mainnet Setup

## 1. Prerequisites for Deployment

Before deploying canisters, ensure the following:

- **OS:** Canisters should be deployed from Ubuntu/Linux; Unity editor can run on Windows for development and testing.
- **Project Root Path:** All LoyaltyCandy project files are under ~/LoyaltyCandy/

### Guide to Installing Ubuntu and Setting up DFINITY SDK (dfx)

#### Step 1: Download and Install Ubuntu

1. **Download Ubuntu Desktop**
   - Visit the official Ubuntu Desktop page.
   - Download the latest LTS version (e.g., Ubuntu 24.04 LTS).
2. **Create Bootable USB / Install Ubuntu**
   - For physical machines: create a bootable USB using balenaEtcher, boot from it, and follow the instructions.
   - For Virtual Machines: use VirtualBox or VMware, mount the ISO, and follow the instructions.
3. **Detailed Instructions:** See Install Ubuntu Desktop

#### Step 2: Install DFINITY SDK (dfx) on Ubuntu

1**. Open Terminal**

2**. Install dfx**

sh -ci "$(curl -fsSL https://sdk.dfinity.org/install.sh)"

3. **Verify Installation**

dfx --version

Refer to DFINITY Developer Guide for detailed instructions and troubleshooting.

## 2. Canisters Used by LoyaltyCandy

**II Canister (Internet Identity)**

- **Purpose:** Handles login, registration, and authentication via Internet Identity (II). Generates Principal IDs.
- **Unity Code:** NNS_II.cs — Unity MonoBehaviour script that interacts with the II canister and manages login/registration for the player.
- **Path in project:** ~/LoyaltyCandy/Assets/SweetSugar/Scripts/ICP/NNS_II/NNS_II.cs
- **Notes:** Uses IIClientWrapper internally to communicate with the II canister.

**NNS Canister**

- **Purpose:** Provides reference/account verification data.
- **Path:** //wsl.localhost/Ubuntu/home/<user>/nns-dapp (replace <user> with your Ubuntu username)

**Custom Game Canister (Climate Wallet / LoyaltyGame)**

- **Purpose:** Stores player scores, calculates leaderboard rankings, maintains snapshots, and distributes weekly rewards in IRC1 tokens.
- **Unity Code:** ICPClient.cs — Unity MonoBehaviour script handling all interaction with the custom game canister.
- **Path in project:** ~/LoyaltyCandy/Assets/SweetSugar/Scripts/ICP/ICPClient.cs
- **Canister Files:**
    - **main.mo** → Located under**:** LoyaltyCandy\ICP-components\icp-canister\climate-wallet\src\climate-wallet-backend\main.mo
        - **Motoko backend logic** for the LoyaltyGame canister.
        - Implements:
            - Score submission and validation.

- Leaderboard ranking and weekly reward distribution.
- On-chain IRC1 token transfers for rewards.

- **climate-wallet-backend.did** → Located under**:** LoyaltyCandy\ICP-components\icp-canister\climate-wallet\.dfx\local\canisters\climate-wallet-backend\climate-wallet-backend.did
  - **Candid interface** file auto-generated by DFX.
  - Defines all exposed canister methods (submitScore, getRanking, claimReward, etc.).
  - Used by Unity (via ICPClient) to auto-generate strongly-typed client code.

**Notes:**

- This is the **backend authority**: Unity never stores or calculates scores/rewards locally, it only displays what the canister provides.
- Any updates or fixes to reward logic must be made here, recompiled, and redeployed to the IC.
- Ensures fairness and prevents client-side tampering, since all calculations (rankings, rewards, balances) run **on-chain**.
- All canisters are deployed to **Mainnet** for production use, but development and testing can be done on a **local replica** first.

## 3. Local Deployment (Testing)

**Open Ubuntu and follow these steps in the terminal:**

**Step 1: Start Local Replica**

```
cd ~
 dfx start --clean --host 0.0.0.0:8080 --background
```

Initializes the local replica for canister deployment.

**Step 2: Install NNS Canisters**

```
cd ~/nns-dapp
 dfx nns install
```

Sets up II and other NNS canisters required.

**Step 3: Deploy Custom Game Canister**

```
cd ~/LoyaltyCandy/ICP-components/icp-canister/climate-wallet
 dfx deploy
```

Save the canister IDs for Unity configuration. (Saving canister IDs steps are covered under Mainnet deployment)

## 4. Mainnet Deployment (Production)

**Open Ubuntu and follow these steps in the terminal:**

1. **Navigate to Canister Folder**

```
cd ~/LoyaltyCandy/ICP-components/icp-canister/climate-wallet
```

2. **Set Mainnet Network Variable**

```
export DFX_NETWORK=ic
```

3. **Deploy Canisters**

```
dfx deploy --network ic
```

4. **Get Mainnet Canister IDs**

```
dfx canister --network ic id <canister-name>
```

Obtain canister IDs from this command, console output after deployment, or from the saved file: ~/.dfx/ic/canister_ids.json.

5. **Save the IDs for Unity Configuration**

```
~/.dfx/ic/canister_ids.json
```

**Note:** Every score submission, leaderboard update, or reward claim consumes ICP cycles.

## 5. Funding Canisters with Cycles

**Purpose:**
On mainnet, all canisters need cycles (like fuel) to run. If a canister runs out of cycles, it stops working. Local deployments don't require cycles.

**Where to run commands:**
Go to the **climate-wallet canister folder** before funding:

cd ~/LoyaltyCandy/ICP-components/icp-canister/climate-wallet

**How to get and add cycles (mainnet only):**

1. **Get ICP tokens** in your NNS wallet (https://nns.ic0.app).

2. **Convert ICP → cycles** using the NNS dapp or dfx wallet commands.

   - Example (via dfx wallet):

     dfx wallet --network ic top-up <CANISTER_ID> <ICP_AMOUNT>

3. **Deposit cycles into the canister:**

dfx canister --network ic call climate-wallet-backend deposit_cycles '(<CYCLES_AMOUNT>)'

Replace <CYCLES_AMOUNT> with how many cycles you want to send.

4. **Check the cycle balance of the canister:**

dfx canister --network ic info climate-wallet-backend

5. Repeat this process for any other custom game canisters deployed in LoyaltyCandy.

**Important Note:**

- Cycles are consumed every time the game calls the canister (e.g., **submitting scores, updating leaderboards, distributing weekly rewards**).
- Always keep the canisters funded to avoid downtime.

**Reference:**
DFINITY Docs – Cycles

## 6. Configure Unity with Mainnet Canisters

1. **Locate Canister IDs**
   - After deploying to Mainnet, canister IDs are saved in ~/.dfx/ic/canister_ids.json.
   - You can also obtain the IDs from the console output after deployment.
2. **Open Unity Project**

   ~/LoyaltyCandy
3. **Configure in Unity Scene**
   - In the **game** scene, there is a folder named **ICPConnector**.
   - Inside this folder is a child object called **ICPClient**.
   - On the **ICPClient** object, the script **NNS_II** is attached.
   - Configure canister IDs, network URL, and other settings in the Inspector for Mainnet.
4. **Save and Rebuild**
   - Save changes in Unity.
   - Rebuild the project to connect to the live Mainnet canisters.

---

# Unity Setup / ICP Integration

## 1. Authentication / Login Panel

- **Class:** NNS_II
- **Path:**~/LoyaltyCandy\Assets\SweetSugar\Scripts\ICP\NNS_II/NNS_II.cs
- **Methods:**

  RegisterUser() → Registers a new user with Internet Identity.

  LoginUser() → Logs in the existing user and fetches Principal ID.

  RegisterCoroutine() → Handles asynchronous registration process.
- **Usage:** Authenticates players, retrieves Principal ID, stores locally, required for score submission, leaderboard retrieval, and reward claiming.

## 2. EdjCase C# Client / Canister Integration

- **Class:** ICPClient
- **Path:**~/LoyaltyCandy/Assets/SweetSugar/Scripts/ICP/ICPClient.cs
- **Methods:**

  Read(…) → Reads stored values from the canister.

  Set(…) → Writes or updates values in the canister.

  GetRanking(int before, int after, short rank) → Fetches leaderboard data.

  SubmitScore(Principal player, uint score) → Submits a player score.

  ClaimReward(Principal player) → Claims weekly rewards for the player.

- **Setup in LoyaltyCandy:**

  using EdjCase.ICP.Agent;
  using EdjCase.ICP.Candid.Models;

- **Usage:** Handles all Unity–ICP communication, submits verified scores, retrieves leaderboard snapshots, checks token balances, claims weekly rewards securely, uses async calls.

## 3. Reward / Balance Panel

- **Class:** WeeklyRankingRewardManager
- **Path:**~/LoyaltyCandy/Assets/SweetSugar/Scripts/WeeklyReward/WeeklyRankingRewardManager.cs
- **Methods:**

  ShowRewardPopup(int rank, BigInteger rewardAmount) → Displays reward panel with rank and tokens.

  ClaimReward() → Confirms reward claim and updates balance.

- **Usage:** Shows earned weekly rewards instantly, updates local UI and token balance, backend ensures tokens credited securely on-chain.

## 4. Leaderboard / Score Panel

- **Classes:** WeeklyRankingRewardManager, ICPClient
- **Methods:**

  GetRanking(int before, int after, short rank) → Retrieves leaderboard data.

  SubmitScore(Principal player, uint score) → Submits and validates player score.
- **Usage:** Leaderboard synced with canisters, ensures only authenticated, verified scores, provides immediate visual feedback.

## 5. Automatic C# Client Generation

**Purpose:**
The .did files in the backend canisters describe all public methods (Candid interface) of each canister. Instead of manually writing HTTP calls, we use EdjCase.ICP.Generator to generate strongly-typed C# client classes automatically. This ensures Unity can call canister methods (like SubmitScore, GetRanking, ClaimReward) with type safety and no boilerplate code.

**Workflow:**

1. **Generate C# client classes from .did files using EdjCase:**

- dotnet EdjCase.ICP.Generator.dll generate <path-to.did> --out <output-folder>

**Example for LoyaltyGame:**

- dotnet EdjCase.ICP.Generator.dll generate LoyaltyCandy/ICP-components/icp-canister/climate-wallet/.dfx/local/canisters/climate-wallet-backend/climate-wallet-backend.did --out GeneratedClients/

2. **Import the generated files into Unity project:**

   You can place them anywhere under the Assets folder, for example:

   ~/LoyaltyCandy/Assets/SweetSugar/Scripts/ICP

3. **Use the generated client in Unity code:**

- var agent = new Agent(...);

- var gameClient = new ClimateWalletBackendClient(agent);
- var leaderboard = await gameClient.GetRankingAsync(0, 10);

**What is Generated in LoyaltyCandy:**

- A strongly-typed C# client for each .did file.
- Each canister method becomes an async C# method with correct parameters and return types.
- Methods generated for LoyaltyGame include:
    - GetGameData()
    - RegisterPlayer(string name, bool isMale)
    - UpdatePlayerScore(uint newScore)
    - ReadScore()
    - GetCurrentGlobalRanking()
    - GetWeeklyRanking(uint before, uint after, short rank)
    - CheckAndMaybeDistributeReward()
    - RewardClaimed()
    - GetCanisterAccountAddressHex()

**Unity Wrapper (ClimateWalletApiClient):**

- **Path:** LoyaltyCandy/Assets/SweetSugar/Scripts/ICP/ClimateWallet/ClimateWalletApiClient.cs
- Wraps the generated client to provide Unity-friendly coroutines, events, offline handling, leaderboard updates, and reward logic.
- Exposes the generated methods as coroutines or async methods for easy use in game scripts.

**Usage:**

- Removes the need to manually write wrappers or serialization logic.
- Guarantees that Unity always stays in sync with the backend Motoko canister interface.
- If the backend .did changes, regenerate the client and the wrapper will work with the updated methods automatically.

## 6. End-to-End Testing

1. **Play the Game**
   - Open the game and navigate to the Settings menu.
2. **Access Profile**
   - In Settings, locate the Profile section.
   - Wallet and Leaderboard icon buttons will be visible.
3. **Register / Login**
   - Click the Wallet icon button.
   - Click Register.
   - If deployment and configuration are correct, you will be registered and logged in automatically.
4. **Verify Data**
   - After login, view your ICP balance in the wallet.
   - On the Leaderboard, check your position.
   - The game remembers your login for future sessions.
5. **Weekly Reward Testing**
   - The weekly reward logic is implemented in the Motoko canister code at: LoyaltyCandy/ICP-components/icp-canister/climate-wallet/src/climate-wallet-backend/main.mo
   - For testing, you can modify the “Sunday ID” in the canister code to simulate a new week.
   - This will trigger the reward distribution logic so you can verify the reward calculation, leaderboard updates, and token distribution without waiting for an actual Sunday.

## Limitations

- **On-chain update delays:** Updates to scores, rankings, or rewards on the leaderboard may not appear immediately because all changes are processed on-chain. There can be a short delay between when a player submits a score or claims a reward and when it is reflected in the Unity client UI.
- **ICP cycle usage:** Each canister call consumes ICP cycles; excessive calls increase costs.
- **WebGL / HTTPS requirement:** II login requires HTTPS and mainnet endpoints; self-signed certificates not supported.
- **Single account per device:** Only one device login is supported; the same account cannot be used simultaneously on multiple devices.
- **Verified scores only:** Only canister-verified scores considered for ranking and rewards.