Publicly available variables

Total Supply

```
uint16 public constant totalSupply = 2000;
```

It indicates the total number of mintable tickets.

Minimum votes required for a refund

```
uint16 public constant minRefundVotes = 200;
```

It indicates the minimum number of refund votes. Specifically more than 200 and more than 50% of the entitled tickets have to vote for a refund to start.

Refund percentage

```
uint16 public constant refundPercentage = 85;
```

The refunded percentage. When a refund starts you will get back 85% of the ticket price.

Issued tickets counter

```
uint16 public ticketCount;
```

The total number of tickets issued until now.

Prizes

```
Prize[] public prizes;
```

A list of all the prizes. A prize is defined as follows:

```
struct Prize {
    bool paid; uint16 tokenId;
    uint16 percentAmount; uint128 weiAmount;
}
```

Where:

- **paid** stores whether or not the prize has been paid.
- ***tokenId*** stores what the winning token for that prize is.
- ***percentAmount*** stores how much was won in percentage.
- ***weiAmount*** stores how much was won in wei.

Current votes for refund

```
uint16 public refundVotes;
```

Tracks how many refund votes have been issued until now.

Current lottery state

```
LOTTERY_STATE public state;
```

The state the lottery is currently in. May be one of the following:

- **NOT_STARTED**
  Meaning the lottery has not started yet.
- **OPEN**
  JAL is open, you can buy tickets, ask for refunds…
- **PICKING_WINNERS**
  JAL is in the process of picking winners. You cannot do anything at this point until this process is completed (should only last a handful of blocks).
- **REFUNDED**
  People voted for the refund and the lottery agreed. So be it, this is the final state after the refund has been issued.
- **CLOSED**
  Lottery is closed, it means you can't buy tickets anymore, can't ask for refunds but you can still withdraw.

VRF keyhash

```
bytes32 public immutable keyhash;
```

Check ChainLink documentation.

Deploying the contract

The contract constructor signature is as follows:

```
constructor(address _vrfCoordinator, uint64 _subId, bytes32 _keyhash)
   VRFConsumerBaseV2(_vrfCoordinator)
   ERC1155("...") {...}
```

It takes the **VRF** Coordinator contract address, a **ChainLink** subscription **ID** and a key hash as arguments.
It will then:

- set the JAL **state** to NOT_STARTED
- set **100 prizes** to a default of 0.01%
- set prize **#**:

1	20.00%
2	10.00%
3	5.00%
4	2.50%
5	1.00%
6-10	0.50%

Mint 100 tickets to the owner address. Those tickets will be used as **giveaways** and as **presales** to **further fund** the project itself. As such they won't count towards the balance for the final prizes. Those tickets will be in everything **identical** to any other normally minted ticket.

Opening sales

```
function openSales() public onlyOwner {...}
```

This function is responsible for **opening** the **sales** to the **public**. It can only be called by the **owner** and can only be called once. It is **irreversible**, there is no way of closing the lottery once opened. It can then draw or enter a refunding state but cannot be reverted to its original state.

Getting ticket prices

```
function getPrice() public view returns(uint256) {...}
```

Returns the **price** of the **next ticket**. The value returned by this function is defined based on the amount of tickets already minted.
Pricing will be:

0.003 **eth**	up to **1000**
0.004 **eth**	up to **1500**
0.005 **eth**	up to **2000**

```
function getBatchPrice(uint256 count) public view returns(uint256) {...}
```

Returns the **sum of the prices** of the next ***count*** tickets. This function is based on ***getPrice().***

Eligibility

A ticket **can't win** the lottery if any of its owners has **asked for a refund**.

```
function hasAskedForRefund(uint256 id) public view returns(bool) {...}
```

It takes the ***id*** of the ticket to be checked and returns ***true*** if it was used to ask for a refund (thus making it **not eligible** for winning), ***false*** otherwise.

Minting

```
function mint() public payable {...}
```

This function is used to **mint** a **new** ticket.
Requirements:

- There must be tickets **available**.
- **JAL** must be in an **OPEN** state.
- The **amount** of eth **sent** with the call must be equal to the value returned by ***getPrice()***.

```
function mint(uint256 count) public payable {...}
```

This function is used to batch mint. You can provide it with the count of how many tickets you want to be minted.
Requirement:

- There must be at least **as many tickets available as the *count* you want to mint**.
- **JAL** must be in an **OPEN** state.
- The **amount** of eth **sent** with the call must be equal to the value returned by ***getBatchPrice(count)***.
- ***count*** must be greater than 0.
- ***count*** must not be greater than 20.

Checking owners

```
function ownerOf(uint16 id) public view returns(address) {...}
```

It allows users to check who the owner of a specific ticket is with this function.

Checking ticket balances

```
function balanceOf(address account) public view returns(uint256) {...}
```

It allows users to check how many tickets a specific address currently owns.

Transferring tickets

```
function safeTransferFrom(
    address from,
    address to,
    uint256 id,
    uint256 value,
    bytes memory data
) public override {...}
```

This function internally calls ***_safeTransferFrom(...)*** so you can check the original docs for the **ERC1155** standard.
Some key **differences** are:

- You can only move **one token per id** (duh!)
- It updates the **internal ownership** of the moved token.

Asking for refunds

```
function askForRefund(uint16 id) public {...}
```

Requirements:

- **JAL** must be in an **OPEN** state.
- ***id*** must be within the **minted tickets range**.
- The **owner** of the ticket must be the **one sending the request**.
- The token must **not have been already used to ask for a refund request**.

Your **NFT** will be automatically **staked** when you call this function. you will be allowed to **unstake** it only if the r**efund goes through.** If you ask for a refund and there were already enough requests the function will proceed with the actual refund (making **you pay for the gas** fee).

```
function getRefund(uint16 id) public {...}
```

Requirements:

- **JAL** must be in a **REFUNDED** state.
- ***id*** must be within the **minted tickets range**.
- The **owner** of the ticket must be the **one sending the request**.
- The token must **not have been already used to ask for a refund request**.

This function is a safety net. It's supposed to make it possible to **receive a refund** even if for some reason **you didn't receive it automatically**.

Unstake

```
function unstake(uint16 id) public {...}
```

Requirements:

- **JAL** must be in a **REFUNDED** state.
- ***id*** must be within the **minted tickets range**.
- The **owner** of the ticket must be the **one sending the request**.

It will transfer the token back to its owner, unstaking it.

Draw

```
function draw() public onlyOwner {...}
```

Requirements:

- **JAL** must be in an **OPEN** state.
- Must be called by the **owner**.
- **Eligible** tickets must be **more than 100.**

It sets the state to **PICKING_WINNERS**. It will submit a request for **100 random words** to our **VRF Coordinator**. With those words it will pick **100 winners** and set the state to **CLOSED**. At this point it's going to **payout** all the winners.

Owner Withdraw

```
function ownerWithdraw() public onlyOwner {...}
```

It allows the owner of the lottery to retrieve all the ETH not used to fulfill prizes.

Requirements:

- JAL must be in a CLOSED state.
- **2** (two) **weeks** should have at least passed after ***draw()*** was called.

Withdraw

```
function withdraw(uint256 id) public {...}
```

This function allows users to withdraw if for some reason they didn't receive their prize when ***draw()*** was called. It also allows the owner of the lottery to retrieve all the remaining ETH after all the payouts have been issued. After ***draw()*** has been called the owner will have to wait **2 weeks** before he can withdraw the remaining ETH, this is a sort of grace period to allow everyone to get their prize.

Requirements:

- **JAL** must be in a **CLOSED** state.
- **The prize** must **not be** already **paid**.
- **Ticket id** must be within the **minted range**.
- **Ticket owner** must be the **caller**.
- Ticket must have **won something**.