

# Auditoría XBetters smart contract

Ventura Lucena Martínez

## Resumen

El objetivo de este informe es documentar tanto potenciales fallos como mejoras sobre el contrato inteligente de XBetters, desplegado sobre la red de Ethereum en la dirección [0xD19056371236ED978aa2e23699bB5EfAd0Bc3566](#). Para ello, se usará el código fuente del archivo *XBetters.sol* alojado en la anterior dirección de Etherscan como sus respectivas librerías. En este sentido, se comenzará haciendo unos comentarios respecto a la estructura y las librerías, continuando con un estudio del cuerpo del contrato y finalizando con unas conclusiones. Por último, se añaden unos anexos con información que pueda ser de utilidad en el informe.

## 1. Estructura

Se recomienda hacer uso de la estructura de código disponible en la guía de estilo de Solidity [1], la cual permite identificar de manera eficiente las funciones según la visibilidad de estas. Por otro lado, también se sugiere hacer uso del estándar de comentarios NatSpec [2], que enriquece la documentación de código, haciendo más eficiente y sencillo su análisis.

## 2. Librerías importadas

Se analizarán de manera exhaustiva el conjunto de librerías que no correspondan a una importación desde un repositorio externo, como la del estándar ERC721A y las del directorio “opensea”, la cual evidencia una potencial modificación en el código fuente respecto al original [3] debido a que la importación viene de un fichero local. En cuanto al resto, se reflejarán si han sufrido cambios respecto a su alojamiento original respecto a los originales de los repositorios de OpenZeppelin.

### 2.1. ERC721A.sol

Para comprobar si existen cambios en el fichero se ha descargado una versión del repositorio con fecha 10 de noviembre de 2022 [4], días antes<sup>1</sup> del despliegue del contrato XBetters en la red de Ethereum bajo la transacción [0xacfd439301780ab8387a93628cb4d10b2045295ce5b9e0c35a6a985016242a0f](#) a día 16 de noviembre de 2022. Se han compilado ambos (el original y el obtenido del directorio del contrato de Xbetters) y se han comparado los *bytecodes*. Al ser distintos, se ha comparado manualmente, detectándose los siguientes cambios:

- La función `approve()` ha sido modificada: se han refactorizado las funciones `_approve` `internal` `virtual` (líneas 889 y 906) en una única, eliminando el parámetro de tipo booleano `approvalCheck`. Puede encontrarse el respectivo test en [5, 6]. Ejecución sin fallos.

---

<sup>1</sup>Se ha dejado un margen entre fechas debido a que es probable que el desarrollo haya acabado bastante antes que la fecha de despliegue.

## 2.2. Ownable.sol

Sin cambios.

## 2.3. MerkleProof.sol

Sin cambios.

## 2.4. Strings.sol

Sin cambios. La librería String.sol importa la librería Math.sol de OpenZeppelin, aunque únicamente hace uso de dos funciones: `log10()` y `log256()`. Eliminar el resto del contenido que no se usa beneficia en:

- Ahorro de espacio en la cadena de bloques. El *bytecode* de las funciones no utilizadas se está almacenando en la cadena de bloques de manera innecesaria, además de aumentar el tamaño global del contrato y hacer que su despliegue sea más costoso.
- Código más legible.
- Menos código “atacable”.

## 2.5. Operator-Filter-Registry

Nuevamente, se han comprobado la diferencia entre ficheros del directorio “opensea” con los originales a fecha de 9 de noviembre de 2022 [7]. Sin cambios. Conviene revisar los cambios de cara a las siguientes implementaciones según la auditoría de OpenZeppelin del 20 de enero de 2023 [8].

## 3. XBetters.sol

A continuación se hacen las siguientes observaciones respecto al contrato original de XBetters<sup>2</sup>.

### 3.1. Variables estado o *state variables*

#### 3.1.1. Constantes y no constantes

En las líneas 37, 38 y 39 del contrato [9] se declaran 3 variables privadas con la nomenclatura de variables constantes según la guía de estilo oficial de Solidity [10]. Se recomienda que, al no tener intencionalidad de variables constantes, declararlas con una estructura *camelCase* [11].

#### 3.1.2. Inicialización

En Solidity, las variables que no son inicializadas con ningún valor quedan inicializadas con valores por defecto que el lenguaje establece. En este sentido, algunos ejemplos son:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.8.17;
3
4 contract Default {
5     bool public a; // False
```

---

<sup>2</sup>Únicamente se mencionarán propuestas de mejora o errores. La ausencia de comentarios respecto a alguna parte del contrato implica que no se proponen cambios.

```

6      int public b; // 0
7      uint256 public c; // 0
8      address public d; // 0x0000000000000000000000000000000000000000
9      bytes32 public e; //
↪    0x0000000000000000000000000000000000000000000000000000000000000000
10     mapping(uint256 => uint256) public f; // Default value for introduced key.
11     string public g; // ""
12     // Note: Fixed arrays, not dynamic ones.
13     uint256[3] public h; // [0, 0, 0]
14
15     enum Phase {
16         Before,
17         WhitelistA,
18         WhitelistB,
19         Public,
20         Soldout,
21         Reveal
22     }
23
24     Phase public phase; // 0
25 }

```

En la línea 45 [12] del contrato se puede observar la variable `revealed` de tipo `bool` inicializada a `false`, y en la línea 60 [13] la variable `phase` del tipo `enum Phase` queda inicializada a `Phase.Before`. A la hora de desplegar el contrato, esto supone un incremento innecesario de **+2267 unidades de gas** sobre la inicialización la variable `revealed` y **+2299 unidades de gas** sobre la variable `phase`. Inicializar las variables sin asignarle ningún valor supone un ahorro del **(-5.56 %)** y **(-5.56 %)** en unidades de gas, respectivamente.

### 3.2. *setPhase()*

El contrato dispone de una variable de tipo `enum` llamada `Phase`, cuya finalidad es almacenar el estado actual del contrato respecto al ERC721A. Este tipo de variables permiten definir un conjunto de constantes con un nombre, a las cuales se les asocia un valor de tipo entero sin signo `uint256` de forma predeterminada, que va desde 0 hasta  $2^{256} - 1$  [14]:

Enum constant	Valor
Before	0
WhitelistA	1
WhitelistB	2
Public	3
Soldout	4
Reveal	5

A su vez, el contrato dispone de un *setter* para este `enum`, el cual modifica el estado de la variable. Este *setter* recibe como parámetro un entero con signo `int` que posteriormente es casteado al tipo `Phase` para establecer finalmente el valor final de la fase. La función debería de recibir un entero sin

signo `uint256`, tal y como espera una enumeración ya que, de esta manera, la función acepta valores como `-1`, los cuales **provocan una transacción fallida** debido a un **Panic error** con código `0x21` [15]. Este tipo de errores deben ser evitados en producción, tal y como se comenta en la anterior referencia:

*“(...) Properly functioning code should never create a Panic, not even on invalid external input. If this happens, then there is a bug in your contract which you should fix. Language analysis tools can evaluate your contract to identify the conditions and function calls which will cause a Panic.” [15]*

Para evitar esto, sería conveniente realizar un control de errores, si bien se puede optimizar reestructurando la función de la siguiente manera:

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity 0.8.17;
3
4  import "@openzeppelin/contracts/access/Ownable.sol";
5
6  contract PhaseTest is Ownable {
7      enum Phase {
8          Before,
9          WhitelistA,
10         WhitelistB,
11         Public,
12         Soldout,
13         Reveal
14     }
15     error InvalidPhase();
16
17     Phase public phase = Phase.Before;
18
19     // USING INT DATATYPE
20     // -----
21     // Original function
22     // Cost: 5899
23     function setPhase(int _phase) external onlyOwner {
24         // Error handling here does increment costs, due to
25         // int to uint indirectly conversion and then the
26         // necessary checks.
27         phase = Phase(_phase);
28     }
29
30     // USING PHASE DATATYPE
31     // -----
32     // Cost: 5786
33     function setPhase(Phase _phase) external onlyOwner {
34         phase = _phase;
35     }
36 }
```

```

37 // USING UINT256 DATATYPE
38 // -----
39 // Cost_1: 5856 without error handling
40 // Cost_2: 5892 with error handling using custom error and revert
41 // Deployment: 357671
42 function setPhase(uint256 _phase) external onlyOwner {
43     if(_phase > uint8(type(Phase).max))
44         revert InvalidPhase();
45
46     phase = Phase(_phase);
47 }
48
49 // Cost_3: 5889 using require
50 // Deployment: 380890
51 function setPhase(uint256 _phase) external onlyOwner {
52     require(_phase < uint8(type(Phase).max), "Invalid phase.");
53
54     phase = Phase(_phase);
55 }
56 }

```

- Se podría implementar un control de errores dentro de la **función original**. Sin embargo, al ser la más cara de las opciones con un coste de **5899 unidades de gas** y, al tener que aumentar la lógica al no poder realizar una conversión directa del tipo de dato `int` a `uint8` (para comparar rango de valores aceptados), se prefiere alguna de las propuestas siguientes.
- **Usar como parámetro el tipo de dato del `enum`** es la opción más barata con un costo de 5786 unidades de gas (-1.8%), aunque también tiene un inconveniente: recibir el tipo de dato del `enum` implica que el rango de valores ya va implícito, por lo que todavía podría recibir un valor como el comentado anteriormente (-1), causando un **Panic error** con código 0x21. Es por ello que se propone alguno de los dos últimos ejemplos como solución.
- **Usar como parámetro un entero sin signo `uint256`**: esta opción permite realizar un control de errores como máximo en dos líneas de código (más la declaración del error personalizado) haciendo los respectivos casteos de variables de forma directa. Se han contemplado el uso de 1) **revert** con errores personalizados y 2) el uso de **require**, ya que aunque parezca que funciona de igual manera, esto no es así. La diferencia reside en cuánto cuesta cada despliegue: el uso de errores personalizados en conjunción con **revert** es más barato que hacer uso de **require**. En este caso, el ahorro en el despliegue es de **23219 unidades de gas** (-6.09%).

### 3.3. Modificadores

De manera similar a lo comentado al final de la sección 3.2, se ha comparado el coste de gas en el despliegue del contrato con distintas codificaciones de los modificadores: 1) **revert** con errores personalizados y 2) el uso de **require**.

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.8.17;
3

```

```

4  contract ModifiersTest {
5
6      address private fiatMinter = 0x349560B18AF0aC8474dFa15221C5430A94A5E3C6;
7
8      error CallerIsOtherContract();
9      error CallerIsNotMinter();
10
11     // USING REQUIRE
12     // -----
13     // Cost: 142132
14     modifier callerIsUser() {
15         require(tx.origin == msg.sender, "Caller is another contract");
16         -;
17     }
18
19     modifier onlyFiatMinter() {
20         require(fiatMinter == msg.sender, "Caller is not minter");
21         -;
22     }
23
24     // USING REVERT WITH CUSTOM ERROR
25     // Cost: 91884
26     modifier callerIsUser() {
27         if (tx.origin != msg.sender) {
28             revert CallerIsOtherContract();
29         }
30         -;
31     }
32
33     modifier onlyFiatMinter() {
34         if (fiatMinter != msg.sender) {
35             revert CallerIsNotMinter();
36         }
37         -;
38     }
39
40     // Cost using modifier with require: 2322
41     // Cost using modifier with rever: 2322
42     function foo() public view callerIsUser onlyFiatMinter{}
43 }

```

El ahorro en el despliegue del contrato con los modificadores haciendo uso de condicionales y `revert` es de un (-35.35%) sobre los modificadores con `require`.

Por otro lado, se ha comprobado que ambos modificadores funcionan correctamente, haciendo especial mención a `callerIsUser()`, el cual evita que se pueda llamar a alguna función de minto a

través de otro contrato (prevención contra bots).

### 3.4. Funciones de mintageo

En esta sección se comentarán dos aspectos para cada función: 1) desempeño y 2) optimización.

#### 3.4.1. *whitelistAMint()* y *whitelistBMint()*

**Desempeño** Se ha comprobado el funcionamiento de la fase de mintageo con *whitelist* con el objetivo de verificar el correcto desempeño de la gestión de la misma haciendo uso de un *Merkle Tree*. El resultado ha sido el esperado. Para ello, se ha utilizado el ejemplo disponible en el anexo [B](#).

**Optimización** Se propone, de igual manera que en anteriores secciones, el cambio de las sentencias `require` por condicionales con la sentencia `revert` con errores personalizados:

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity 0.8.17;
3
4  contract WhitelistMintTest {
5
6      enum Phase {
7          Before,
8          WhitelistA,
9          WhitelistB,
10         Public,
11         Soldout,
12         Reveal
13     }
14
15     (...)
16
17     Phase public phase = Phase.Before;
18
19     error CallerIsOtherContract();
20
21     (...)
22
23     error WhitelistANotActive();
24     error NotWhitelistedInA();
25     error WhitelistBNotActive();
26     error NotWhitelistedInB();
27     error LimitExceeded(); // Shared with other mint functions.
28     error NotEnoughTokensLeft(); // Shared with other mint functions.
29     error UnsufficientEther(); // Shared with other mint functions.
30
31     // Cost with require: 4399403
32     // Cost with revert and custom error:
33     //      - Only with WL A: 4368770
```

```

34 //      - Both WL A and B: 4316494
35 function whitelistAMint(
36     uint _quantity,
37     bytes32[] calldata _proof
38 ) external payable callerIsUser {
39     if (phase != Phase.WhitelistA)
40         revert WhitelistANotActive();
41
42     if (!isWhitelistedA(msg.sender, _proof))
43         revert NotWhitelistedInA();
44
45     if (_numberMinted(msg.sender) + _quantity > MAX_WLA_MINTS_PER_ADDRESS)
46         revert LimitExceeded();
47
48     // Code shared with other functions.
49     if (totalSupply() + _quantity > MAX_SUPPLY)
50         revert NotEnoughTokensLeft();
51
52     if (msg.value < (whitelistAMintPrice * _quantity))
53         revert UnsuufficientEther();
54     // -----
55
56     _safeMint(msg.sender, _quantity);
57 }
58
59 function whitelistBMint(
60     uint _quantity,
61     bytes32[] calldata _proof
62 ) external payable callerIsUser {
63     if (phase != Phase.WhitelistB)
64         revert WhitelistBNotActive();
65
66     if (!isWhitelistedB(msg.sender, _proof))
67         revert NotWhitelistedInB();
68
69     if (_numberMinted(msg.sender) + _quantity > MAX_WLB_MINTS_PER_ADDRESS)
70         revert LimitExceeded();
71
72     // Code shared with other functions.
73     if (totalSupply() + _quantity > MAX_SUPPLY)
74         revert NotEnoughTokensLeft();
75
76     if (msg.value < (whitelistBMintPrice * _quantity))
77         revert UnsuufficientEther();
78     // -----
79

```



```

80     _safeMint(msg.sender, _quantity);
81 }
82
83     (...)
84 }

```

No seguir esta metodología para la función *whitelistAMint()* implica un gasto adicional de **30633 unidades de gas**, y no hacerlo con ambas funciones lo **incrementaría 52276 unidades de gas** más<sup>3</sup>, ascendiendo la suma total a un gasto de **82909 unidades de gas**.

### 3.4.2. *publicMint()*

**Desempeño** Se ha comprobado el funcionamiento de la fase de mintage público para verificar su correcto desempeño. El resultado ha sido el esperado.

**Optimización** Se propone, de igual manera que en anteriores secciones, el cambio de las sentencias **require** por condicionales con la sentencia **revert** con errores personalizados:

```

1     (...)
2
3     error PublicMintNotActive();
4
5     // Cost with require: 4316494
6     // Cost with revert and custom error: 4287661
7     function publicMint(uint _quantity) external payable callerIsUser {
8
9         // Code shared with fiatMint().
10        if (phase != Phase.Public)
11            revert PublicMintNotActive();
12        if (_numberMinted(msg.sender) + _quantity > MAX_PUB_MINTS_PER_ADDRESS)
13            revert LimitExceeded();
14        // -----
15
16        // Code shared with other functions.
17        if (totalSupply() + _quantity > MAX_SUPPLY)
18            revert NotEnoughTokensLeft();
19
20        if (msg.value < (mintPrice * _quantity))
21            revert UninsufficientEther();
22        // -----
23
24        _safeMint(msg.sender, _quantity);
25    }
26
27    (...)

```

<sup>3</sup>Esto se debe a la reutilización de algunos errores personalizados para ambas funciones. Véanse líneas de código 27, 28 y 29 del apartado “Optimización” de la sección 3.4.1.

Esto supone un gasto adicional de 28833 unidades de gas.

### 3.4.3. *fiatMint()*

**Desempeño** Se ha comprobado el funcionamiento de la fase de mintage FIAT para verificar su correcto desempeño. El resultado ha sido el esperado.

**Optimización** Se propone, de igual manera que en anteriores secciones, el cambio de las sentencias `require` por condicionales con la sentencia `revert` con errores personalizados:

```
1  // Cost with require: 4287661
2  // Cost with revert and custom error: 4217369
3  function fiatMint(
4      address _account,
5      uint _quantity
6  ) external onlyFiatMinter {
7
8      // Code shared with PublicMint().
9      if (phase != Phase.Public)
10         revert PublicMintNotActive();
11     if (_numberMinted(msg.sender) + _quantity > MAX_PUB_MINTS_PER_ADDRESS)
12         revert LimitExceeded();
13     // -----
14
15     // Code shared with other functions.
16     if (totalSupply() + _quantity > MAX_SUPPLY)
17         revert NotEnoughTokensLeft();
18     // -----
19
20     _safeMint(_account, _quantity);
21 }
```

Esto supone un gasto adicional de 70292 unidades de gas.

Estas funciones pueden refactorizarse aún más haciendo uso de modificadores que alojen el código que compartan entre sí. En este sentido, se pueden crear un total de cuatro nuevos modificadores que acorten el código en cada función y que, dependiendo de cada uno, puedan ser más legibles. Estos modificadores se llamarán en cada función de mintage sin alterar el orden original:

```
1  modifier onlyIfThereAreTokensLeft(uint _quantity) {
2      if (totalSupply() + _quantity > MAX_SUPPLY)
3          revert NotEnoughTokensLeft();
4      _;
5  }
6
7  modifier onlyIfEnoughEther(uint _quantity) {
8      if (msg.value < (whitelistAMintPrice * _quantity))
```

```

9         revert UnsufficientEther();
10     _;
11 }
12
13 modifier onlyInPubliPhase() {
14     if (phase != Phase.Public)
15         revert PublicMintNotActive();
16     _;
17 }
18
19 modifier checkMaxPublicMintPerAddress(uint _quantity) {
20     if (_numberMinted(msg.sender) + _quantity > MAX_PUB_MINTS_PER_ADDRESS)
21         revert LimitExceeded();
22     _;
23 }

```

Aunque el despliegue de esta versión es más caro, con un incremento de **3609 unidades de gas**.

### 3.5. *tokensOfOwner()*

En el bucle *for* se hace una llamada al método `_startTokenId()` del fichero ERC721A.sol. Mide el índice de comienzo desde que se inicia el contrato ERC721A en el constructor, el cual no puede ser cambiado post-despliegue. Dado que esta función no se ha modificado y de forma predeterminada devuelve 0, la inicialización de la variable *i* dentro del bucle se puede ahorrar, tal y como se ha visto en la sección 3.1.2:

```

1  // Cost per execution with _startTokenId() when it returns 0 (not overridden):    13357
   ↳ (Cost only applies when called by a contract)
2  // Cost per execution without _startTokenId() when it returns 0 (not overridden): 13322
   ↳ (Cost only applies when called by a contract)
3  function tokensOfOwner(
4      address owner
5  ) external view returns (uint256[] memory) {
6      unchecked {
7          uint256 tokenIdsIdx;
8          address currOwnershipAddr;
9          uint256 tokenIdsLength = balanceOf(owner);
10         uint256[] memory tokenIds = new uint256[](tokenIdsLength);
11         TokenOwnership memory ownership;
12
13         for (uint256 i; tokenIdsIdx != tokenIdsLength; ++i) { // _startTokenId() = 0 in
   ↳ ./ERC721A.sol
14             ownership = _ownershipAt(i);
15             if (ownership.burned) {
16                 continue;
17             }
18             if (ownership.addr != address(0)) {

```

```

19         currOwnershipAddr = ownership.addr;
20     }
21     if (currOwnershipAddr == owner) {
22         tokenIds[tokenIdsIdx++] = i;
23     }
24 }
25 return tokenIds;
26 }
27 }

```

Mencionar que, el ahorro en gas es despreciable y únicamente se aplica el coste cuando se llama a la función desde un contrato. Aún así, por cada llamada a la función se consumen **35 unidades de gas** extra.

### 3.6. *tokenURI()*

Se recomienda la sustitución de las sentencias **require** por condicionales con la sentencia **revert** con errores personalizados:

```

1  (...)
2
3  // Cost with require:           4230849
4  // Cost with revert and custom error:  4207416
5  error TokenDoesNotExist();
6
7  function tokenURI(
8      uint256 tokenId
9  ) public view override returns (string memory) {
10     if (!_exists(tokenId)) {
11         revert TokenDoesNotExist();
12     }
13
14     if (revealed) {
15         return string(abi.encodePacked(baseURI, Strings.toString(tokenId), ".json"));
16     } else {
17         return notRevealedUri;
18     }
19 }
20
21 (...)

```

No seguir esta recomendación implica un gasto adicional de **23433 unidades de gas** en el despliegue del contrato.

### 3.7. *getPhaseMaxValue()*

Al igual que ocurre en la sección 3.2, que la función reciba como parámetro un entero con signo **int** permite introducir datos que pueden ocasionar un **Panic error** con código 0x21 en tiempo de

ejecución. Por ello, se recomienda usar como parámetro un entero sin signo `uint256` para realizar un control de errores que permita revertir la transacción de forma segura:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.8.17;
3
4 import "@openzeppelin/contracts/access/Ownable.sol";
5
6 contract GetPhaseMaxValueTest is Ownable {
7
8     uint256 private MAX_WLA_MINTS_PER_ADDRESS = 1000;
9     uint256 private MAX_WLB_MINTS_PER_ADDRESS = 1000;
10    uint256 private MAX_PUB_MINTS_PER_ADDRESS = 100;
11
12    enum Phase {
13        Before,
14        WhitelistA,
15        WhitelistB,
16        Public,
17        Soldout,
18        Reveal
19    }
20    error InvalidPhase();
21
22    Phase public phase = Phase.Before;
23
24    function getPhaseMaxValue(
25        uint256 _phase
26    ) external view onlyOwner returns (uint256) {
27        // Error handling
28        if(_phase > uint8(type(Phase).max))
29            revert InvalidPhase();
30
31        if (Phase(_phase) == Phase.WhitelistA) {
32            return MAX_WLA_MINTS_PER_ADDRESS;
33        } else if (Phase(_phase) == Phase.WhitelistB) {
34            return MAX_WLB_MINTS_PER_ADDRESS;
35        } else if (Phase(_phase) == Phase.Public) {
36            return MAX_PUB_MINTS_PER_ADDRESS;
37        }
38
39        return 0;
40    }
41 }
```

### 3.8. *setPhaseMaxValue()*

Al igual que ocurre en anteriores secciones, la función permite introducir datos que pueden ocasionar un **Panic error** con código 0x21 en tiempo de ejecución. Por ello, se recomienda usar como parámetro un entero sin signo `uint256` para realizar un control de errores que permita revertir la transacción de forma segura. Por otro lado, la función permite ejecutar una transacción con valores de fase que carecen de sentido en la lógica del contrato: 1) 0 ó “Before”, 2) 4 ó “Soldout” y 3) 5 ó “Reveal”. Aunque no se asignen valores para dichas fases (no tiene efecto ya queda bien asignado bajo condicionales dentro de la función), sí que permite ejecutarla en caso de descuido, lo que conllevaría a pagar **24634 unidades de gas** de promedio.

```
1  (...)
2
3  error PhaseNotMintable();
4
5  function setPhaseMaxValue(uint256 _phase, uint256 _value) external onlyOwner {
6      // Error handling.
7      if(_phase > uint8(type(Phase).max))
8          revert InvalidPhase();
9
10     if (Phase(_phase) == Phase.WhitelistA) {
11         MAX_WLA_MINTS_PER_ADDRESS = _value;
12     } else if (Phase(_phase) == Phase.WhitelistB) {
13         MAX_WLB_MINTS_PER_ADDRESS = _value;
14     } else if (Phase(_phase) == Phase.Public) {
15         MAX_PUB_MINTS_PER_ADDRESS = _value;
16     } else {
17         // Error handling.
18         revert PhaseNotMintable();
19     }
20 }
21
22 (...)
```

Por último, se pueden establecer valores para cada una de las fases mayores al circulante total de tokens `MAX_SUPPLY`. Si bien esto no es un error, un descuido a la hora de establecer el valor puede resultar en valores no deseados para una fase concreta, quedando el resto de fases sin tokens disponibles en el peor de los casos.

## 4. Análisis

Se han aplicado los cambios comentados anteriormente al contrato [16], con el objetivo de comparar resultados de manera global, más allá de los ya comentados. Se han utilizado tanto la librería Hardhat de JavaScript como el editor Remix para los informes de gas. Los resultados son los siguientes:

Solc version: 0.8.17		Optimizer enabled: false		Runs: 200	Block limit: 30000000 gas	
Methods						
Contract	Method	Min	Max	Avg	# calls	usd (avg)
XBettters	approve	-	-	51083	1	-
XBettters	fiatMint	-	-	81885	1	-
XBettters	reveal	-	-	50821	1	-
XBettters	setBaseURI	-	-	33704	1	-
XBettters	setFiatMinter	-	-	29263	1	-
XBettters	setMerkleRootA	-	-	46458	3	-
XBettters	setMintPrice	-	-	29014	1	-
XBettters	setNotRevealedUri	-	-	38164	2	-
XBettters	setPhase	26357	29169	28700	6	-
XBettters	setPhaseMaxValue	-	-	29467	1	-
XBettters	setWhitelistAMintPrice	-	-	29059	1	-
XBettters	transferFrom	85662	87556	86609	2	-
Deployments					% of limit	
XBettters		-	-	5377661	17.9 %	-

Figura 1: Informe de gas para XBettters (Hardhat).

Solc version: 0.8.17		Optimizer enabled: false		Runs: 200	Block limit: 30000000 gas	
Methods						
Contract	Method	Min	Max	Avg	# calls	usd (avg)
XBettters_IP	approve	-	-	51061	1	-
XBettters_IP	fiatMint	-	-	81884	1	-
XBettters_IP	reveal	-	-	50821	1	-
XBettters_IP	setBaseURI	-	-	33687	1	-
XBettters_IP	setFiatMinter	-	-	29263	1	-
XBettters_IP	setMerkleRootA	-	-	46480	3	-
XBettters_IP	setMintPrice	-	-	29036	1	-
XBettters_IP	setNotRevealedURI	-	-	38146	2	-
XBettters_IP	setPhase	26379	29191	28722	6	-
XBettters_IP	setPhaseMaxValue	-	-	29504	1	-
XBettters_IP	setWhitelistAMintPrice	-	-	29037	1	-
XBettters_IP	transferFrom	85662	87556	86609	2	-
Deployments					% of limit	
XBettters_IP		-	-	5178650	17.3 %	-

Figura 2: Informe de gas para XBettters\_IP (Hardhat).

```
✓ [vm] from: 0x5B3...eddC4 to: XBetters.(constructor) value: 0 wei data: 0x608...36f6e logs: 301 hash: 0x23f...9a8c3
status true Transaction mined and execution succeed
transaction hash 0x23f2466d816e08b066b0016cfe23d6567313a3ca9c10082ae2506c860d29a8c3 ⓘ
from 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 ⓘ
to XBetters.(constructor) ⓘ
gas 6184283 gas ⓘ
transaction cost 5377637 gas ⓘ
execution cost 4976689 gas ⓘ
input 0x608...36f6e ⓘ
decoded input {} ⓘ
decoded output - ⓘ
```

Figura 3: Informe de gas para XBetters (Remix).

```
✓ [vm] from: 0x5B3...eddC4 to: XBetters_IP.(constructor) value: 0 wei data: 0x608...36f6e logs: 301 hash: 0x599...3afc8
status true Transaction mined and execution succeed
transaction hash 0x599489c9db2004b1101d747d7d517edb79770cda737ccd37d9c2f92f7a23afc8 ⓘ
from 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 ⓘ
to XBetters_IP.(constructor) ⓘ
gas 5955434 gas ⓘ
transaction cost 5178638 gas ⓘ
execution cost 4793242 gas ⓘ
input 0x608...36f6e ⓘ
decoded input {} ⓘ
decoded output - ⓘ
```

Figura 4: Informe de gas para XBetters.IP (Remix).

Graficando los datos, se observa como **la propuesta de mejora reduce los costes de gas**:



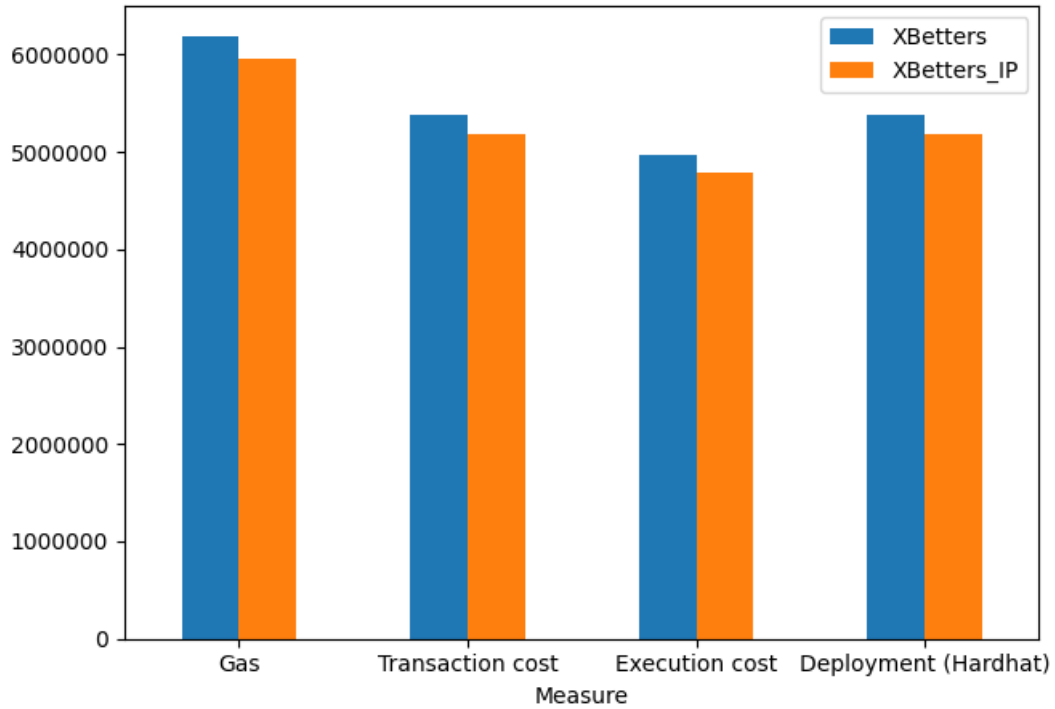


Figura 5: Gráfica de gas.

## 5. Conclusiones

En este informe se ha hecho una revisión del contrato inteligente XBettters. Aparentemente, no se han encontrado vulnerabilidades que puedan afectar de manera severa al contrato. Por otro lado, se han sugerido un conjunto de alternativas que mejoren, tanto la legibilidad como la estructura del contrato como el comportamiento actual del mismo, reduciendo el gasto en unidades de gas. Finalmente, se ha presentado una propuesta de mejora y se ha realizado un análisis global del gasto entre ambos contratos.

## Referencias

- [1] *Solidity style guide: Order of functions*. English. Ver. 0.8.17. Accedido el 7 de marzo de 2023. Ethereum Foundation. URL: <https://docs.soliditylang.org/en/v0.8.17/style-guide.html#order-of-functions>. (2021).
- [2] *Solidity style guide: NatSpec Format*. English. Ver. 0.8.17. Accedido el 7 de marzo de 2023. Ethereum Foundation. URL: <https://docs.soliditylang.org/en/v0.8.17/natspec-format.html>. (2021).
- [3] GitHub: Chiru-Labs. *Chiru-Labs/ERC721A*. Accedido el 3 de marzo de 2023. URL: <https://github.com/chiru-labs/ERC721A>.
- [4] GitHub: Chiru-Labs. *Chiru-Labs/ERC721A (44ab010)*. Accedido el 6 de marzo de 2023. URL: <https://github.com/chiru-labs/ERC721A/tree/44ab0103c2addf9e16bd164422b9a7359eac0a49>.
- [5] GitHub: VenturaLM. *VenturaLM/XBettters-audit/src/test/XBettters.js*. Accedido el 7 de marzo de 2023. URL: <https://github.com/VenturaLM/XBettters-audit/blob/master/src/test/XBettters.js#L104>.
- [6] GitHub: VenturaLM. *VenturaLM/XBettters-audit/src/test/XBettters.js*. Accedido el 7 de marzo de 2023. URL: <https://github.com/VenturaLM/XBettters-audit/blob/master/src/test/XBettters.js#L129>.
- [7] GitHub: ProjectOpenSea. *ProjectOpenSea/operator-filter-registry (419ce28)*. Accedido el 6 de marzo de 2023. URL: <https://github.com/ProjectOpenSea/operator-filter-registry/tree/419ce2835a6f3bd5be4b14ac19278bb6060312fa>.
- [8] GitHub: ProjectOpenSea. *OpenSea Operator Filteer Audit Report*. Accedido el 6 de marzo de 2023. URL: <https://github.com/ProjectOpenSea/operator-filter-registry/blob/main/audit/OpenSea%20Operator%20Filteer%20Audit%20Report.pdf>.
- [9] Ángel Moratilla. *Etherscan: XBettters.sol*. (2022). URL: <https://etherscan.io/address/0xd19056371236ed978aa2e23699bb5efad0bc3566#code#F1#L37>.
- [10] *Solidity style guide: Constants*. English. Ver. 0.8.17. Accedido el 4 de marzo de 2023. Ethereum Foundation. URL: <https://docs.soliditylang.org/en/v0.8.17/style-guide.html#constants>. (2021).
- [11] *Solidity style guide: Local and state variable names*. English. Ver. 0.8.17. Accedido el 4 de marzo de 2023. Ethereum Foundation. URL: <https://docs.soliditylang.org/en/v0.8.17/style-guide.html#local-and-state-variable-names>. (2021).
- [12] Ángel Moratilla. *Etherscan: XBettters.sol*. (2022). URL: <https://etherscan.io/address/0xd19056371236ed978aa2e23699bb5efad0bc3566#code#F1#L45>.
- [13] Ángel Moratilla. *Etherscan: XBettters.sol*. (2022). URL: <https://etherscan.io/address/0xd19056371236ed978aa2e23699bb5efad0bc3566#code#F1#L60>.
- [14] *Solidity style guide: Enums*. English. Ver. 0.8.17. Accedido el 4 de marzo de 2023. Ethereum Foundation. URL: <https://docs.soliditylang.org/en/v0.8.17/types.html#enums>. (2021).
- [15] *Solidity error handling: Panic via assert and Error via require*. English. Ver. 0.8.17. Accedido el 4 de marzo de 2023. Ethereum Foundation. URL: <https://docs.soliditylang.org/en/v0.8.17/control-structures.html#panic-via-assert-and-error-via-require>. (2021).
- [16] GitHub: VenturaLM. *VenturaLM/XBettters-audit/src/contracts/XBettters\_IP.sol*. Accedido el 7 de marzo de 2023. URL: [https://github.com/VenturaLM/XBettters-audit/blob/master/src/contracts/XBettters\\_IP.sol](https://github.com/VenturaLM/XBettters-audit/blob/master/src/contracts/XBettters_IP.sol).
- [17] GitHub: VenturaLM. *VenturaLM/XBettters-audit/src/test/XBettters.js*. Accedido el 6 de marzo de 2023. URL: <https://github.com/VenturaLM/XBettters-audit/blob/master/src/test/XBettters.js#L152>.

## A. Errores en los tests ajenos al contrato

Durante la realización de los tests se han encontrado 4 errores ajenos a la lógica del contrato:

1. **safeTransferFrom()**: Hardhat no detecta la función, pese a tener visibilidad `public`.

```
1 1) XBetters
2     Successful transfer functions
3     FIXME: Runs safeTransferFrom():
4     TypeError: contract.safeTransferFrom is not a function
5     at Context.<anonymous> (test\XBetters.js:108:35)
6     at processTicksAndRejections (node:internal/process/task_queues:96:5)
```

2. **whitelistAMint()**: aunque la transacción se ejecute con una clave privada asociada a una dirección incluida en el *Merkle root*, hardhat revierte la transacción con el error `Not whitelisted`. Se ha comprobado en el editor Remix y la ejecución se ha dado sin defectos.

```
1 2) XBetters
2     whitelistAMint(). This function tests a Merkle Proof
3     FIXME: whitelistAMint(
4         1,
5         [
6             0x1ebaa930b8e9130423c183bf38b0564b0103180b7dad301013b18e59880541ae,
7             0x343750465941b29921f50a28e0e43050e5e1c2611a3ea8d7fe1001090d5e1436
8         ]
9     )
10
11     AssertionError: Expected transaction NOT to be reverted with reason 'Not
↪ whitelisted', but it was
12     at processTicksAndRejections (node:internal/process/task_queues:96:5)
13     at runNextTicks (node:internal/process/task_queues:65:3)
14     at listOnTimeout (node:internal/timers:528:9)
15     at processTimers (node:internal/timers:502:7)
16     at Context.<anonymous> (test\XBetters.js:200:21)
```

3. **publicMint()**: por alguna razón, hardhat no completa la transacción con el ether necesario, devolviendo un error. Esto no pasa en otras funciones de minteo, como `whitelistAMint()`.

```
1 3) XBetters
2     FIXME: publicMint()
3     Runs publicMint():
4     Error: VM Exception while processing transaction: reverted with reason string 'Not
↪ enough ether sent'
5     at XBetters.publicMint (contracts/Xbetters.sol:239)
6     at processTicksAndRejections (node:internal/process/task_queues:96:5)
7     at runNextTicks (node:internal/process/task_queues:65:3)
```

```

8   at listOnTimeout (node:internal/timers:528:9)
9   at processTimers (node:internal/timers:502:7)

```

4. **withdraw()**: dado que se ha utilizado la función **publicMint()** dentro del test para poder enviar ethers<sup>4</sup> y esta no funciona, se obtiene el mismo error que anteriormente.

```

1  4) XBetters
2     FIXME: withdraw()
3     Test the exchange of balance among EOA and the CA:
4     Error: VM Exception while processing transaction: reverted with reason string 'Not
   ↪ enough ether sent'
5     at XBetters.publicMint (contracts/Xbetters.sol:239)
6     at processTicksAndRejections (node:internal/process/task_queues:96:5)
7     at runNextTicks (node:internal/process/task_queues:65:3)
8     at listOnTimeout (node:internal/timers:528:9)
9     at processTimers (node:internal/timers:502:7)
10    at HardhatNode._mineBlockWithPendingTx
   ↪ (node_modules\hardhat\src\internal\hardhat-network\provider\node.ts:1819:23)
11    at HardhatNode.mineBlock
   ↪ (node_modules\hardhat\src\internal\hardhat-network\provider\node.ts:508:16)
12    at EthModule._sendTransactionAndReturnHash
   ↪ (node_modules\hardhat\src\internal\hardhat-network\provider\modules\eth.ts:1522:18)

```

## B. Ejemplo usado en *whitelistAMint()* y *whitelistBMint()*

El ejemplo completo puede ser encontrado en [17].

```

1  WhitelistA Merkle Tree:
2    - 0xb12e5b97c5c34aeb22d4e5f0061100c5072c240346e4d28a1a73659930fe90b2 (Root)
3      - 0x343750465941b29921f50a28e0e43050e5e1c2611a3ea8d7fe1001090d5e1436 (Index 4)
4        - 0x00314e565e0574cb412563df634608d76f5c59d9f817e85966100ec1d48005c0 (Index 0)
5        - 0x8a3552d60a98e0ade765adddad0a2e420ca9b1eef5f326ba7ab860bb4ea72c94 (Index 1)
6      - 0x8393e82ea28dfe71f8a1bfc8bfb8e85da65aa4a7f2ceb7b2e356854fb5983c538 (Index 5)
7        - 0xe9707d0e6171f728f7473c24cc0432a9b07eaa1efed6a137a4a8c12c79552d9 (Index 2)
8        - 0x5b1130ba602a5b64a86675b98193c8989dfff9db8956c9d2eac539828c115523 (Index 3)
9
10   ---
11
12   En whitelistAMint()
13   Verify:
14     Address: 0xf39Fd6e51aad88F6F4ce6aB8827279cFfFb92266
15     Hash: 0xe9707d0e6171f728f7473c24cc0432a9b07eaa1efed6a137a4a8c12c79552d9

```

<sup>4</sup>Dado que el contrato no tiene una función **fallback()**, hardhat muestra un error si se envían ethers en una transacción usando la función **sendTransaction()**.

```
16   Merkle Root: 0xb12e5b97c5c34aeb22d4e5f0061100c5072c240346e4d28a1a73659930fe90b2
17   Proof: ["0x1ebaa930b8e9130423c183bf38b0564b0103180b7dad301013b18e59880541ae",
↪      "0x343750465941b29921f50a28e0e43050e5e1c2611a3ea8d7fe1001090d5e1436"]
```