Warped Games
Escazu San Rafael
Del Centro Comercial La Paco
Trescientos Metros Norte
Plaza Florencia
Local Diez
San Jose
Costa Rica

Warped Token with Dynamic Tax

Audit Techspec

1.Project Overview

Warped Games is a Web3 gaming initiative dedicated to delivering captivating, interconnected gaming environments to a broader spectrum of gamers. Our mission to bridge the divide between traditional gaming and blockchain technology involves the creation of proprietary tools and services, facilitating the seamless incorporation of blockchain principles within a gaming ecosystem. Warped Universe, our flagship project, will utilize the ERC20 token, WARPED, within its expansive digital economy. This enables users to access advanced features, trade assets and influence design decisions.

2.Functional Requirements

2.1.Roles

The Warped contract project has the following roles:

- **Admin**: The control of the warped contracts is managed by an admin wallet. Initially, a deployer wallet has this control, and later it will be transferred to a multisig wallet. The admin wallet has several abilities, such as adding liquidity, adding or removing exchange pools, and setting the primary pool in the token manager. Additionally, it can add or remove NFTs, reset tax rate points, and pause or resume tax in the tax handler. The admin wallet is also capable of setting parameters like liquidity basis point, price impact basis point, tax swap value, treasury wallet, and can withdraw any tokens or ethers from the treasury handler.
- **Normal user**: A regular user can buy or sell the warped token at the standard tax rate.
- **NFT owned user**: Users who own NFTs specified in the WarpedTaxHandler buy or sell the warped token at a lower tax rate based on the NFTs they possess.

2.2.Features

The Warped contract project has the following features:

- The administrator has the ability to perform administrative tasks as necessary as well as the following actions:
    - Add liquidity to the Warped token.
    - Add or remove exchange pools for the Warped token.
    - Set the primary pool for the Warped token.
    - Add or remove NFTs and their corresponding levels.
    - Reset the tax rate points.
    - Pause or resume the tax mechanism.
    - Set the liquidity basis point, price impact basis point, tax swap value, and treasury wallet.
    - Can withdraw any tokens or ethers in the treasury handler.
- The user has the ability to perform the following actions:
    - buy or sell warped tokens with lower tax rate if they owned NFTs minted by us

Warped Games
Escazu San Rafael
Del Centro Comercial La Paco
Trescientos Metros Norte
Plaza Florencia
Local Diez
San Jose
Costa Rica

2.3.Use case

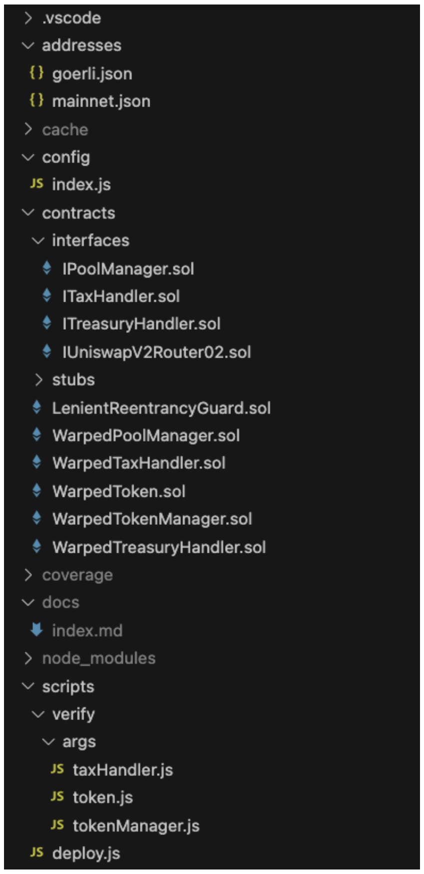
The Warped token will serve as the blockchain-based utility token of the Warped Universe ecosystem. Our vision for Warped is multifaceted—it will facilitate cross-chain accessibility, reward gamers, enable purchases of marketplace digital assets & Pass Points, and empower participation in game development and design decisions through a Governance-style voting mechanism.

3.Technical Requirements

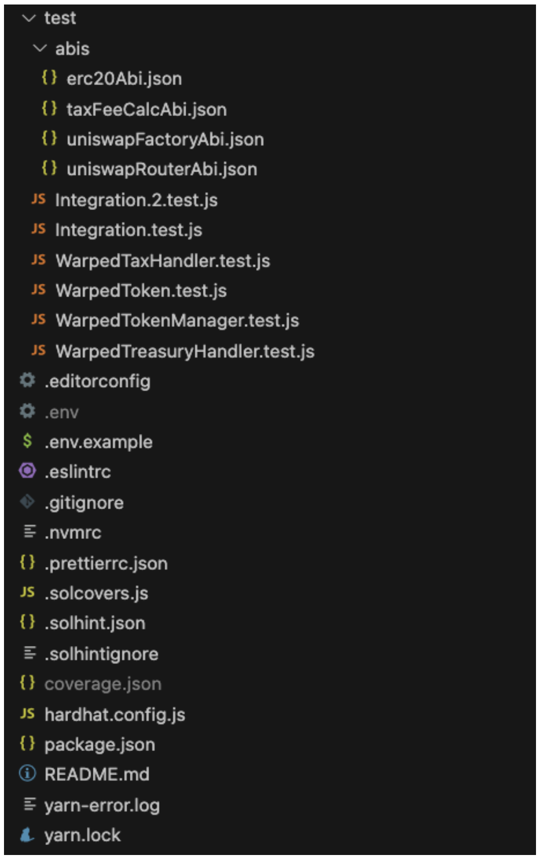
This project has been developed with Solidity language, using Hardhat as a development environment. JavaScript is the selected language for testing and scripting. The project is fully linted with Solhint, Eslint and Prettier following best practices. Slither static analysis was performed to ensure security best practices were adhered to.

In addition, OpenZeppelin's libraries are used in the project. All information about the contracts library and how to install it can be found in their GitHub. Finally, Uniswap v2 API is used in the contract to add liquidity and swap tokens to ETH.

In the project folder, the following structure is found:



```
> .vscode
v addresses
  goerli.json
  mainnet.json
> cache
v config
  index.js
v contracts
  v interfaces
    IPoolManager.sol
    ITaxHandler.sol
    ITreasuryHandler.sol
    IUniswapV2Router02.sol
  > stubs
  LenientReentrancyGuard.sol
  WarpedPoolManager.sol
  WarpedTaxHandler.sol
  WarpedToken.sol
  WarpedTokenManager.sol
  WarpedTreasuryHandler.sol
> coverage
v docs
  index.md
> node_modules
v scripts
  v verify
    v args
      taxHandler.js
      token.js
      tokenManager.js
  deploy.js
```



```
v test
  v abis
    erc20Abi.json
    taxFeeCalcAbi.json
    uniswapFactoryAbi.json
    uniswapRouterAbi.json
  Integration.2.test.js
  Integration.test.js
  WarpedTaxHandler.test.js
  WarpedToken.test.js
  WarpedTokenManager.test.js
  WarpedTreasuryHandler.test.js
.editorconfig
.env
.env.example
.eslintrc
.gitignore
.nvmrc
.prettierrc.json
.solcovers.js
.solhint.json
.solhintignore
coverage.json
hardhat.config.js
package.json
README.md
yarn-error.log
yarn.lock
```

Start with README.md to find all basic information about the project structure and scripts that are required to test and deploy the contracts. Node v18.16.0 and Solidity v0.8.18 are required to compile the project.

3.1. Architecture Overview

The following chart provides a general view of the Warped contract structures and interactions between different contracts / functions.

Graphviz visualisations for each contract have been added in the graphviz folder in the warped-contracts repository.

3.2. Contract Information

This section contains detailed information (their purpose, assets, functions, and events) about the contracts used in the project.

3.2.1. WarpedPoolManager.sol

The **WarpedPoolManager** contract is an implementation of the **IPoolManager** interface, extending OpenZeppelin's **Ownable** contract. It manages the interaction with multiple exchange pool addresses, maintaining an enumerable set of such addresses.

The contract has the ability to designate one of these addresses as the primary exchange pool and also ensures that the primaryPool and _exchangePools values are consistent.

3.2.1.1. Assets

The contract should include the following state variables:

EnumerableSet.AddressSet _exchangePools: An instance of EnumerableSet.AddressSet, used to keep track of all registered exchange pool addresses.
address primaryPool: A public variable storing the address of the primary exchange pool.

3.2.1.2. Events

The contract should emit the following types of events:

- **ExchangePoolAdded:** Emitted when a new exchange pool address is added to _exchangePools.
- **ExchangePoolRemoved**: Emitted when an exchange pool address is removed from **_exchangePools**.
- **PrimaryPoolUpdated**: Emitted when the **primaryPool** address is updated.

3.2.1.3.Functions

The contract has the following function:

- **isPoolAddress**: This method accepts an Ethereum address as input and returns a boolean indicating whether the address is present in _exchangePools.
- **addExchangePool**: This method, restricted to the owner of the contract, allows the addition of a new exchange pool address to _exchangePools. If successful, it emits the ExchangePoolAdded event.
- **removeExchangePool**: Also restricted to the contract owner, this method allows the removal of an exchange pool address from _exchangePools. If successful, it emits the ExchangePoolRemoved event.
    - *reverts* when given poolAddress and current primaryPool address are the same .
    - *succeeds* when given poolAddress and current primaryPool address are not the same.
- **setPrimaryPool**: This owner-restricted method updates the primaryPool address. It has two requirements: the address must already be part of **_exchangePools**, and it should not be the same as the current **primaryPool**. If successful, it emits the **PrimaryPoolUpdated** event.: This method accepts an Ethereum address as input and returns a boolean indicating whether the address is present in _exchangePools.

3.2.1.4.Error Messages

The **setPrimaryPool** method returns two error messages when the requirements are not met:

- "**Not registered as exchange pool**": Thrown when the provided address is not present in **_exchangePools**.
- "**Already primary pool address**": Thrown when the provided address is already the **primaryPool**.
- "**Primary pool cannot be removed**": Thrown when the given pool address is the same as the current **primaryPool**.

The **addExchangePool** method returns an error message when the requirements are not met:

- "**Zero address passed**": Thrown when the provided address is zero address..

3.2.2.WarpedTaxHandler.sol

The WarpedTaxHandler contract serves as a tax handler system for an NFT based platform, wherein tax rates are determined by a user's ownership level in the system's NFTs. It's built on Ethereum blockchain using the Solidity programming language and follows the ERC721 token standard. It should include the OpenZeppelin's Ownable contract to limit certain contract interactions to the contract's owner.

1. It should allow for setting, getting, and disabling taxes.
2. It should provide functionality for managing different levels of NFTs (ERC721).
3. It should interact with a pool manager contract and retrieve information on whether a specific address is a pool address or not.
4. It should prevent high gas costs by limiting the number of NFT contracts and tax rates that can be added for tax purposes.

5. Should check that NFT collections are unique.
6. Checks that the tax thresholds are in ascending order

3.2.2.1.Assets

The contract should include the following state variables:

- uint8 **NFT_CONTRACTS_LIMIT**: Limit for max number of contracts. Constant variable setted as 10.
- uint8 **TAX_RATES_LIMIT**: Limit for max number of tax rate points. Constant variable setted as 10.
- IERC721[] **nftContracts**: An array of type IERC721 to store the NFTs used for determining the tax level of a user.
- mapping(IERC721 => uint8) **nftLevels**: A mapping of type IERC721 to uint8 to represent the levels of each NFT.
- TaxRatePoint[] **taxRates**: An array of struct **TaxRatePoint** to store the different tax rates. This array should be sorted by threshold in ascending order. To satisfy this, setTaxRates which sets this array should pass an array where threshold values are sorted by descending order. This is not required but should be sorted to work correctly. TaxRatePoint struct has 2 values:
  - uint256 threshold: If user's level which is bit-or of all level values of NFTs user owned is equal or bigger than this value, tax rate of this user will be following rate
  - uint256 rate: tax rate for this threshold. This value has 2 decimals.
- uint256 **basisTaxRate**: A uint256 to store the basis tax rate. This value has 2 decimals.
- uint256 **maxTaxRate**: A uint256 to store the maximum allowed tax rate. This value has 2 decimals.
- bool **taxDisabled**: A boolean to toggle tax applicability.
- IPoolManager **poolManager**: An instance of **IPoolManager** to interact with the pool manager contract.

3.2.2.2.Events

The contract should emit the following types of events:

- **TaxRatesUpdated:** Emitted when tax rates are updated.
- **NFTsAdded**: Emitted when new NFTs are added.
- **NFTsRemoved**: Emitted when NFts are removed.
- **TaxPaused**: Emitted when tax is paused.
- **TaxResumed**: Emitted when tax is resumed.

3.2.2.3.Functions

The contract has the following function:

- **constructor()**: Initializes the contract with a pool manager, an array of NFT contracts, and an array of their corresponding levels. It also sets the default tax rates.
- **getTax()**: Given a benefactor, beneficiary, and transfer amount, it calculates and returns the tax to be paid.

- **setTaxRates():** Allows the contract owner to set new tax rates. It should validate that the new rates meet certain conditions before applying them, such as order of rate thresholds and limit of max number of tax rates.
- **addNFTs()**: Allows the contract owner to add new NFTs and their levels to the contract.
- **removeNFTs**(): Allows the contract owner to safely remove NFTs from the contract.
- **pauseTax()**: Allows the contract owner to pause the application of taxes.
- **resumeTax()**: Allows the contract owner to resume the application of taxes.
- **_getTaxBasisPoints**(): An internal function to calculate the tax rate based on the user's NFT holdings.
- **_addNFTs()**: An internal function to validate and add NFTs to the contract.

3.2.2.4.Modifiers

The contract should include the following modifiers:

- **onlyOwner**: A modifier that restricts certain function calls to the contract's owner.

3.2.2.5.Error Messages

The **setTaxRates** method returns 4 error messages when the requirements are not met:

- "**Invalid level points**": Thrown when the provided thresholds and rates haven't the same length.
- "**Invalid base rate**": Thrown when the provided **_basisTaxRate** is zero.
- "**Base rate must be <= than max**": Thrown when the provided **_basisTaxRate** is larger than **maxTaxRate**.
- "**Rate must be less than max rate**": Thrown when any value in provided rates is larger than **maxTaxRate**.
- "**Tax rates limit exceeded**": Thrown when number of given tax rates exceed the limit defined by **NFT_CONTRACTS_LIMIT** constant variable.
- "**Thresholds not descending order**": Thrown when the given threshold array is not in descending order.

The **addNFTs** method returns 2 error messages when the requirements are not met:

- "**Invalid parameters**": Thrown when one of the provided contracts and levels has zero length or their length is not the same.
- "**IERC721 not implemented**": Thrown when one of the provided contracts is not implementation of IERC721 in openzeppelin.
- "**Duplicate NFT contract**": Thrown when the given NTF contract has already been added.
- "**Invalid NFT level**": Thrown when the given NFT level is **zero**.
- "**No. of NFT contracts over limit**": Thrown when number of given nft contracts exceeds the limit defined by **TAX_RATES_LIMIT** constant variable.
- "contract address is zero address": Thrown when the given nft contract address is zero address.

The **removeNFTs** method returns 1 error message when the requirements are not met:

- "**Invalid parameters**": Thrown when the length of provided contracts is zero.

The **pauseTax** method returns 1 error message when the requirements are not met:

- "**Already paused**": Thrown when tax is already disabled.

The **resumeTax** method returns 1 error message when the requirements are not met:

- "**Not paused**": Thrown when tax is not disabled.

3.2.3.WarpedToken.sol

The WarpedTaxHandler contract serves as a contract to handle the Warped Token, The contract should override the ERC20 token transfer functions to implement custom logic for before and after token transfers.

The contract should use a reentrancy guard to prevent reentrancy attacks.

- All sensitive actions, such as updating handlers, should only be callable by the contract's owner.
- The contract should validate all user inputs to prevent malicious attacks.
- Only the contract's owner should be able to call these functions.
- The functions should validate the input to ensure it is a non-zero address and different from the current handler's address.

3.2.3.1.Assets

The contract should include the following state variables:

- The contract should define the total supply of tokens as **10 billion**.
- The contract should define the number of decimal places for the token as **18**.
- The contract should define the name of the token as "**WARPED**".
- The contract should define the symbol of the token as "**WARPED**".

3.2.3.2.Events

The contract should emit the following types of events:

- **TaxHandlerUpdated**: Emitted when tax handler is updated.
- **TreasuryHandlerUpdated**: Emitted when treasury handler is updated.

3.2.3.3.Functions

The contract has the following function:

- **Constructor**: The constructor should initialize the token's name, symbol, total supply, tax handler, and treasury handler. The constructor should mint the total supply of tokens to the deployer's address.

- **_beforeTokenTransfer** function should use the treasury handler's processTreasury function to process the treasury part of a transfer.
- **_afterTokenTransfer** function should use the tax handler's getTax function to calculate the tax for a transfer, and then transfer the tax amount to the treasury handler's address.
- **updateTaxHandler()**: function should update the tax handler address.
- **updateTreasuryHandler()**: function should update the treasury handler address.
- The treasury handler should process the treasury part of each transaction.

3.2.3.4.Modifiers

The contract should include the following modifiers:

- **onlyOwner**: A modifier that restricts certain function calls to the contract's owner.
- **skipWhenTaxProcessing**: A modifier that skips running of the function if tax is in processing.

3.2.3.5.Handlers

- The contract should implement **interfaces** for tax and treasury handlers.
- The contract should define **public getters** for tax and treasury handlers.
- The tax handler should **calculate the tax** for each transaction.
- The treasury handler should **process the treasury part** of each transaction.

3.2.3.6.Error Messages

The **constructor** returns 3 error messages when the requirements are not met:

- "**Deployer is zero address**": Thrown when the provided deployer address is zero address.
- "**taxHandler is zero address**": Thrown when the provided tax handler is zero address.
- "**treasuryHandler is zero address**": Thrown when the provided treasury handler is zero address.

The **updateTaxHandler** method returns two error messages when the requirements are not met:

- "**Zero tax handler address**": Thrown when the provided address is zero address.
- "**Same tax handler address**": Thrown when the provided address is the same as the current handler address.

The **updateTreasuryHandler** method returns two error messages when the requirements are not met:

- "**Zero treasury handler address**": Thrown when the provided address is zero address.
- "**Same treasury handler address**": Thrown when the provided address is the same as the current handler address.

3.2.4.WarpedTokenManager.sol

- The contract must be able to interact with the Uniswap V2 Router.
- The contract must use and integrate with other contracts including WarpedToken, WarpedTaxHandler, WarpedTreasuryHandler, and WarpedPoolManager.
- The contract must use the SafeERC20, EnumerableSet, and IERC20 interfaces from the OpenZeppelin library.
- The contract must be able to interact with the Uniswap V2 Router.
- The contract must use and integrate with other contracts including WarpedToken, WarpedTaxHandler, WarpedTreasuryHandler, and WarpedPoolManager.

3.2.4.1.Functions

The contract has the following function:

- **Constructor**: The constructor must instantiate a **WarpedTreasuryHandler** contract, a **WarpedTaxHandler** contract, and a **WarpedToken** contract. The constructor must initialize the **treasury handler** with a provided **treasury address** and the **token contract's address**. The constructor must transfer the ownership of the **TaxHandler**, **TreasuryHandler**, and **token contracts** to the **contract's owner**. The constructor must set the **warpedToken state variable** to the newly created **token contract**.
- **addLiquidity()**: The contract must expose a payable function addLiquidity which accepts an amountToLiquidity argument. The addLiquidity function must be callable only by the contract owner. The addLiquidity function must transfer the specified amount of tokens from the owner's wallet to the contract. The addLiquidity function must approve the Uniswap router to use the transferred tokens. The addLiquidity function must create a new Uniswap pair between the token and WETH. The addLiquidity function must add liquidity to the Uniswap pair, using all available ETH in the contract. The addLiquidity function must add the Uniswap pair to the exchange pools and set it as the primary pool. The addLiquidity function must only be accessible to the contract owner.

3.2.4.2.Events

The contract should emit the following types of events:

- **LiquidityAdded**: Emitted when primary pool is created and liquidity created on that pool.

3.2.4.3.Error Messages

The **constructor** returns an error message when the requirements are not met:
- "**treasury is zero address**": Thrown when the provided treasury address is zero address.

3.2.5.WarpedTreasuryHandler.sol

The WarpedTreasuryHandler contract is an Ethereum smart contract developed to manage the operations related to the treasury of an ERC20 token system. The contract mainly processes token transactions that have accumulated through taxes and sells the tokens to obtain ETH, which is then forwarded to the treasury.

- The contract should allow initialization only once.
- The contract should prevent actions on transfers other than sell.
- The contract should prevent selling more tokens than the maximum price impact.
- The contract should not allow liquidity basis points to exceed 10,000 (i.e., 100%).
- The contract should not allow a zero treasury address.
- The contract should allow the contract to accept ETH.
- The contract should ensure that the Uniswap router can perform the swap or transfer for the designated number of tokens.
- The contract should ensure that the price impact is within reasonable bounds.
- The contract should prevent high price impact basis points value (should be less than 1500).
- The contract should allow withdrawal of the current ETH balance or any token balance and token transfer should use safeTransfer from SafeERC20.
- The contract should not allow zero
- The contract should protect against Front-Running attacks by using **getAmountsOut**

The contract depends on the following external contracts and libraries:

- Ownable from the OpenZeppelin library, to provide basic authorization control functions.
- IERC20 from the OpenZeppelin library, for the interface of the ERC20 standard.
- Address from the OpenZeppelin library, for functions related to address type.
- IUniswapV2Router02, for the Uniswap router that handles the sell and liquidity operations.
- IPoolManager for managing the primary exchange pool.
- ITreasuryHandler for the interface of the treasury handling functions.

3.2.5.1.Assets

- **IPoolManager poolManager**: The instance of the IPoolManager contract.
- **address payable treasury**: The Treasury address.
- IERC20 token: The token that accumulates through taxes. This will be sold for ETH.
- **uint256 liquidityBasisPoints**: The basis points of tokens to sell and add as liquidity to the pool.
- **uint256 priceImpactBasisPoints**: The maximum price impact the sell (initiated from this contract) may have.
- **uint256 _taxSwap**: A private variable for managing the swapping of tokens due to taxes.
- **bool _isInitialized**: A flag to indicate whether the contract is initialized.
- **IUniswapV2Router02 UNISWAP_V2_ROUTER**: The Uniswap router that handles the sell and liquidity operations.

3.2.5.2.Events

The contract should emit the following types of events:

- **LiquidityBasisPointsUpdated**: Emitted when the basis points value of tokens to add as liquidity is updated.
- **PriceImpactBasisPointsUpdated**: Emitted when the maximum price impact basis points value is updated.
- **TreasuryAddressUpdated**: Emitted when the treasury address is updated.
- **TaxSwapUpdated**: Emitted when _taxSwap is updated.
- **LiquidityAdded**: Emitted when liquidity is added to the uniswap v2 pool.

3.2.5.3.Functions

The contract has the following functions:

- **initialize**: Initializes the contract with a treasury address and the token address.
- **processTreasury**: Handles the treasury processing before a sell or liquidity addition action is executed.
- **setLiquidityBasisPoints**: Sets a new value for the liquidity basis points. When this value is 0, we don’t collect for liquidity, but when it is above 0 then it collects a percentage from the tax and adds to liquidity.
- **setPriceImpactBasisPoints**: Sets a new value for the price impact basis points.
- **setTreasury**: Sets a new treasury address.
- **withdraw**: Allows the withdrawal of any tokens or ETH in the treasury handler.
- **receive**: Allows contract to accept ETH
- **updateTaxSwap**: Update _taxSwap value.
- **_swapTokensForEth**: Swap accumulated tokens for ETH.
- **_addLiquidity**: Add liquidity to the primary pool.

3.2.5.4.Modifiers

The contract should include the following modifiers:

- **onlyOwner**: The contract uses the onlyOwner modifier from the Ownable contract to limit access to certain functions.

3.2.5.5.Error Messages

The **initialize** method returns 2 error messages when the requirements are not met:

- "**treasury is zero address**": Thrown when the provided treasury is zero address.
- "**token is zero address**": Thrown when the provided token is zero address.

The **setLiquidityBasisPoints** method returns 1 error message when the requirements are not met:

- "**Max is 10000**": Thrown when the provided newBasisPoints is bigger than 10000.

The **setPriceImpactBasisPoints** method returns 1 error message when the requirements are not met:

- "**Too high value**": Thrown when the provided newBasisPoints is equal or bigger than 1500.

The **setTreasury** method returns 1 error message when the requirements are not met:

- "**Zero address**": Thrown when the provided address is zero.

The **updateTaxSwap** method returns 1 error message when the requirements are not met:

- "**Zero taxSwap**": Thrown when the provided value is zero.