

# Snek.fun – Milestone 2 Report (Fund 12 – Cardano Catalyst)

This document provides a comprehensive Proof of Achievement for Milestone 2 of the Snek.fun project, funded under Cardano Catalyst Fund 12. It outlines the development and testing of the Bonding Curve Pool and Order Smart Contract systems, along with their deployment on-chain and third-party verification.

## Table of Content:

<b>Snek.fun – Milestone 2 Report (Fund 12 – Cardano Catalyst)</b>	<b>1</b>
Bonding Curve Pool smart contract	2
Testfunctions annotation	2
Test results screenshot	3
On-chain examples	3
Order smart contract	4
Testfunctions annotation	4
On-chain examples	5
Batching (order execution) system	6
Conclusion	6
Annex: Third Proof of Achievement: Link to Explorer	7

In this report we will provide everything for each component of the system strictly according to the Acceptance criteria of this milestone. If you need access to smart contract code to close this milestone, please send us your contacts and how do we share access with you.

# Bonding Curve Pool smart contract

The Aiken implementation of this smart contract as well as the bonding curve formula are closed source due to business reasons so we can provide you only tests.

## Testfunctions annotation

**execution\_of\_correct\_ada\_to\_token\_swap\_should\_succeed Purpose:**

Tests a valid ADA-to-token swap in the bonding curve pool. **Key Logic:** Verifies the swap correctly adjusts ADA and token balances per the bonding curve formula, ensuring the transaction succeeds.

**validation\_of\_incorrect\_ada\_value\_in\_final\_pool\_after\_ada\_to\_token\_swap\_should\_fail**

**Purpose:** Tests rejection of an ADA-to-token swap with incorrect final ADA balance.

**Key Logic:** Ensures the transaction fails if the pool's final ADA value deviates from the expected bonding curve calculation.

**validation\_of\_incorrect\_token\_value\_in\_final\_pool\_after\_ada\_to\_token\_swap\_should\_fail**

**Purpose:** Tests rejection of an ADA-to-token swap with incorrect final token balance.

**Key Logic:** Ensures the transaction fails if the pool's final token value does not match the bonding curve formula.

**execution\_of\_correct\_token\_to\_ada\_swap\_should\_succeed Purpose:**

Tests a valid token-to-ADA swap in the bonding curve pool. **Key Logic:** Confirms the swap correctly adjusts token and ADA balances per the bonding curve formula, ensuring the transaction succeeds.

**validation\_of\_incorrect\_ada\_value\_in\_final\_pool\_after\_token\_to\_ada\_swap\_should\_fail**

**Purpose:** Tests rejection of a token-to-ADA swap with incorrect final ADA balance.

**Key Logic:** Ensures the transaction fails if the pool's final ADA value deviates from the expected bonding curve calculation.

**validation\_of\_incorrect\_token\_value\_in\_final\_pool\_after\_token\_to\_ada\_swap\_should\_fail**

**Purpose:** Tests rejection of a token-to-ADA swap with incorrect final token balance.

**Key Logic:** Ensures the transaction fails if the pool's final token value does not align with the bonding curve formula.

## Test results screenshot

```
bonding_curve_pool_v1
PASS [mem: 541753, cpu: 294388325] execution_of_correct_ada_to_token_swap_should_succeed
PASS [mem: 611542, cpu: 237146383] validation_of_incorrect_ada_value_in_final_pool_after_ada_to_token_swap_should_fail
PASS [mem: 621542, cpu: 237146383] validation_of_incorrect_token_value_in_final_pool_after_ada_to_token_swap_should_fail
PASS [mem: 2681841, cpu: 965138110] execution_of_correct_token_to_ada_swap_should_succeed
PASS [mem: 2681841, cpu: 965138110] validation_of_incorrect_ada_value_in_final_pool_after_token_to_ada_swap_should_fail
PASS [mem: 2681841, cpu: 965138110] validation_of_incorrect_token_value_in_final_pool_after_token_to_ada_swap_should_fail
6 tests | 6 passed | 0 failed
```

## On-chain examples

Launch bonding curve pool:

<https://cardanoscan.io/transaction/b752f988e0bed856b09992bacf39e7f1f34a6cde4e90a5a35500375aa67d66bb?tab=tokenmint>

Examples of trading operations with the bonding curve pool are in the "Order smart contract" section.

# Order smart contract

Order smart contract and fully open sourced. You can check it . All tests [here](#) are included in the file. Below we will explain how each test function ensure proper work of the smart contract.

## Testfunctions annotation

### **test\_reproduce\_preprod\_execution\_one\_to\_one**

**Purpose:** Tests a one-to-one swap execution between two limit orders (ADA → token, token → ADA) in a batched transaction.

**Key Logic:** Validates batched swap of two orders in a single transaction, ensuring correct asset exchange, fees, and authorization.

### **test\_normal\_partial\_swap\_ada\_to\_token**

**Purpose:** Tests partial swap of ADA for a token.

**Key Logic:** Ensures partial execution adjusts input/output amounts, satisfies base price, fees, and marginal output constraints.

### **test\_normal\_partial\_swap\_token\_to\_ada**

**Purpose:** Tests partial swap of a token for ADA.

**Key Logic:** Verifies token-to-ADA swap handles non-ADA input, ADA output, and maintains price and fee constraints.

### **test\_normal\_partial\_swap\_token\_to\_token**

**Purpose:** Tests partial swap between two non-ADA tokens. **Key**

**Logic:** Confirms token-to-token swap manages non-ADA assets, Lovelace for fees, and adheres to price and integrity constraints.

### **test\_normal\_cancellation**

**Purpose:** Tests valid cancellation of a limit order.

**Key Logic:** Ensures cancellation is authorized, returns full value to the correct address.

### **test\_bad\_cancellation\_wrong\_redeemer**

**Purpose:** Tests invalid cancellation due to incorrect redeemer

address. **Key Logic:** Verifies cancellation fails if output address does not match redeemer address.

### **test\_bad\_cancellation\_unauthorized**

**Purpose:** Tests invalid cancellation due to unauthorized signer. **Key**

**Logic:** Ensures cancellation fails if signer does not match cancellation PKH.

### **test\_valid\_batched\_full\_swap\_at\_base\_price**

**Purpose:** Tests batched full swap of multiple identical limit orders at

base price.

**Key Logic:** Confirms batched swaps process multiple orders, maintaining price, fees, and authorization.

#### **test\_valid\_full\_swap\_at\_base\_price**

**Purpose:** Tests single full swap at base price with restricted executors.

**Key Logic:** Verifies single swap enforces executor restrictions, handles asset exchange correctly.

#### **test\_valid\_mixed\_full\_swap\_at\_base\_price**

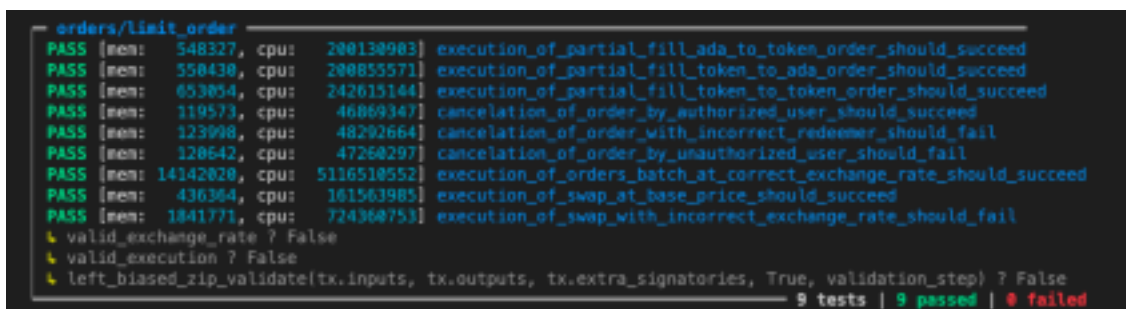
**Purpose:** Tests full swap in a transaction with mixed inputs. **Key Logic:** Ensures full swap validates correctly despite unrelated UTXOs in the transaction.

#### **test\_invalid\_full\_swap\_bad\_exchange\_rate**

**Purpose:** Tests invalid full swap due to insufficient output amount. **Key Logic:** Verifies swap fails if output does not meet base price requirement.

## Test results screenshots

Since the smart contract is open sourced, you can run tests yourself



```
orders/limit_order
PASS [mem: 548327, cpu: 200130903] execution_of_partial_fill_ada_to_token_order_should_succeed
PASS [mem: 558430, cpu: 200855571] execution_of_partial_fill_token_to_ada_order_should_succeed
PASS [mem: 653054, cpu: 242615144] execution_of_partial_fill_token_to_token_order_should_succeed
PASS [mem: 119573, cpu: 46069347] cancelation_of_order_by_authorized_user_should_succeed
PASS [mem: 123998, cpu: 48292664] cancelation_of_order_with_incorrect_redeemer_should_fail
PASS [mem: 120642, cpu: 47260297] cancelation_of_order_by_unauthorized_user_should_fail
PASS [mem: 14142020, cpu: 5116510552] execution_of_orders_batch_at_correct_exchange_rate_should_succeed
PASS [mem: 436364, cpu: 161563985] execution_of_swap_at_base_price_should_succeed
PASS [mem: 1841771, cpu: 724360753] execution_of_swap_with_incorrect_exchange_rate_should_fail
↳ valid_exchange_rate ? False
↳ valid_execution ? False
↳ left_biased_zip_validate(tx.inputs, tx.outputs, tx.extra_signatories, True, validation_step) ? False
9 tests | 9 passed | 0 failed
```

## On-chain examples

Buy token (add ADA, get Token) order placement:

<https://cardanoscan.io/transaction/98d8e1985b72068fc1d3846af39756567a0eef1fdd053e490b7b590d9528c6f5>

Buy token (add ADA, get Token) order execution:

<https://cardanoscan.io/transaction/682a63cb59c93c77a18af4dbb4ea60ac1f6a04152e57aadf67612ae5ff671da8>

Sell token (add Token, get ADA) order placement:

<https://cardanoscan.io/transaction/287615f7243424cbe356da30bb25207088ef691f4d4d9c4033a37b6977efe833>

Sell token (add Token, get ADA) order execution:

<https://cardanoscan.io/transaction/6362e8cf4a935a85a8b033ffc60ac5>

## **Batching (order execution) system**

Snefun is a trading platform and function pretty much as all regular AMM DEXes on Cardano. Saying that we mean that a batching system to execute orders. We mentioned that we will show how the batching system works. You can see the full explanation how the batching system works in the video below:

[https://drive.google.com/file/d/1W6J-V4INw9PPuNJ0fryyCF\\_u\\_jyplgMy/view](https://drive.google.com/file/d/1W6J-V4INw9PPuNJ0fryyCF_u_jyplgMy/view)

## **Conclusion**

In conclusion, all core components required by Milestone 2 have been implemented, tested, and deployed on-chain. The tests are open-source and verifiable, and all transactions are publicly accessible. The batching system is demonstrated in the video linked below.

---

## **Annex: Third Proof of Achievement: Link to Explorer**

### **Bonding curve pool:**

<https://cardanoscan.io/transaction/b752f988e0bed856b09992bacf39e7f1f34a6cde4e90a5a35500375aa67d66bb?tab=tokenmint>

### **Buy Token:**

<https://cardanoscan.io/transaction/98d8e1985b72068fc1d3846af39756567a0eef1fdd053e490b7b590d9528c6f5>

<https://cardanoscan.io/transaction/682a63cb59c93c77a18af4dbb4ea60ac1f6a04152e57aadf67612ae5ff671da8>

### **Sell Token:**

<https://cardanoscan.io/transaction/287615f7243424cbe356da30bb25207088ef691f4d4d9c4033a37b6977efe833>

<https://cardanoscan.io/transaction/6362e8cf4a935a85a8b033ffc60ac5677a4abce8cfa34ae5a9f408ef232f225c>