FINAL PROJECT
Team members: Kairkulova Dilnaz, Myrzakhan Mereke
PredictionChain – Decentralized Crowdfunding for Prediction Events

1.Project description:

1. Overview of the Application Architecture

Our project is called PredictionChain, and the main idea is to build a decentralized prediction platform where users can bet on different outcomes using test ETH. In this system, an event is created with a title, a deadline, and several possible outcomes such as A/B/C. Users connect their MetaMask wallets, choose an outcome they believe will happen, and send test ETH to the smart contract. All contributions are stored on-chain, and the contract tracks exactly how much each user placed on each outcome.

A campaign becomes finalizable once the deadline passes. After that, the event creator calls the function to finalize the campaign and choose the correct outcome. If a user predicted the right outcome, they are allowed to claim a proportional share of the entire ETH pool. This creates a transparent and fair distribution mechanism where everyone's reward depends on how much they contributed.

The system also includes an ERC-20 token called PredictionChain Participation Token (PRED), which was deployed on the Sepolia test network. This token is not used as real money but works as a participation or reputation token. For every 1 wei contributed to any event, the contract automatically mints 1 PRED token to the user.

The main smart contracts — PredictionToken and PredictionChain — were deployed through Remix using MetaMask. I manually tested all core functions: creating events, placing bets with ETH value, minting PRED tokens, and checking updated user balances. The architecture focuses on simplicity, transparency, and safe prediction categories such as game releases, movie ratings, IT updates, or true/false events. Overall, PredictionChain demonstrates a complete decentralized prediction workflow with real blockchain interaction on the Sepolia testnet.

Since the project is developed for educational purposes, the event finalization process is performed manually by the contract owner. This approach allows the system to function without external data sources.

In a production-ready prediction market, event outcomes would typically be resolved using decentralized oracles or governance mechanisms. However, such solutions were intentionally excluded from this project to keep the focus on smart contract fundamentals, frontend interaction, and secure on-chain logic.

2. Explanation of Design and Implementation Decisions

The system was designed with clarity and educational value in mind. A prediction-based model was chosen instead of regular crowdfunding because it clearly demonstrates smart contract features: deadlines, reward logic, outcome selection, and proportional distribution.

We separated the logic into two contracts:

- **PredictionToken handles ERC-20 reward token minting.**
- **PredictionChain handles events, bets, deadlines, and payouts.**

The decision to mint 1 PRED per 1 wei makes the reward formula simple, transparent, and easy for students to verify. Deployment through Remix and MetaMask was also chosen because this method provides the clearest visualization of testnet interactions.

3. Description of Smart Contract Logic

The PredictionChain contract enables users to create prediction events by specifying a title, deadline, and number of outcomes. Each event stores its own ETH pool and outcome pools.

The placeBet() function accepts test ETH and records the contribution amount inside mappings. The function also triggers ERC-20 minting using the PredictionToken contract.

Once the deadline has passed, the creator finalizes the event by selecting the correct outcome. Winning participants can then claim their proportional share of the pool, based on how much they contributed to the correct outcome.

The PredictionToken contract follows the ERC-20 standard and includes setMinter() to restrict minting rights only to the PredictionChain contract.

4. Explanation of Frontend-to-Blockchain Interaction

Although the contracts were tested directly through Remix, the same logic applies to any JavaScript frontend. MetaMask is used to request account access, check the selected network, and sign all blockchain transactions.

The interaction flow is:

1. **MetaMask connects the user's address.**

2. **The user selects an action (create event, bet, finalize).**
3. **The frontend sends a transaction to the smart contract.**
4. **MetaMask asks the user to confirm.**
5. **The transaction is mined and reflected on-chain.**

Remix fully supports MetaMask integration, so all interactions were performed and verified through Sepolia.

5. Deployment and Execution Instructions

1. **Install MetaMask and switch to the Sepolia test network.**
2. **Obtain test ETH using a faucet.**
3. **Open Remix IDE and import both Solidity files.**
4. **Compile contracts using Solidity 0.8.28.**
5. **Deploy PredictionToken with Injected Provider – MetaMask.**
6. **Deploy PredictionChain, passing the token's address.**
7. **Call setMinter() on PredictionToken to set PredictionChain as the minter.**
8. **Test the system by creating events, placing bets, and checking token balances.**

All deployments and tests were completed successfully in Sepolia.

6. Description of the Process for Obtaining Test ETH

To obtain test ETH for deployment and testing, I used a public Sepolia faucet. The steps were:

1. **Copy my MetaMask wallet address.**
2. **Visit the faucet (sepolia-faucet.pk910.de).**
3. **Paste the address and request ETH.**

4. **The faucet sent 0.25 SepoliaETH within seconds.**

5. **The balance appeared in MetaMask and was used for smart contract deployment and transactions.**

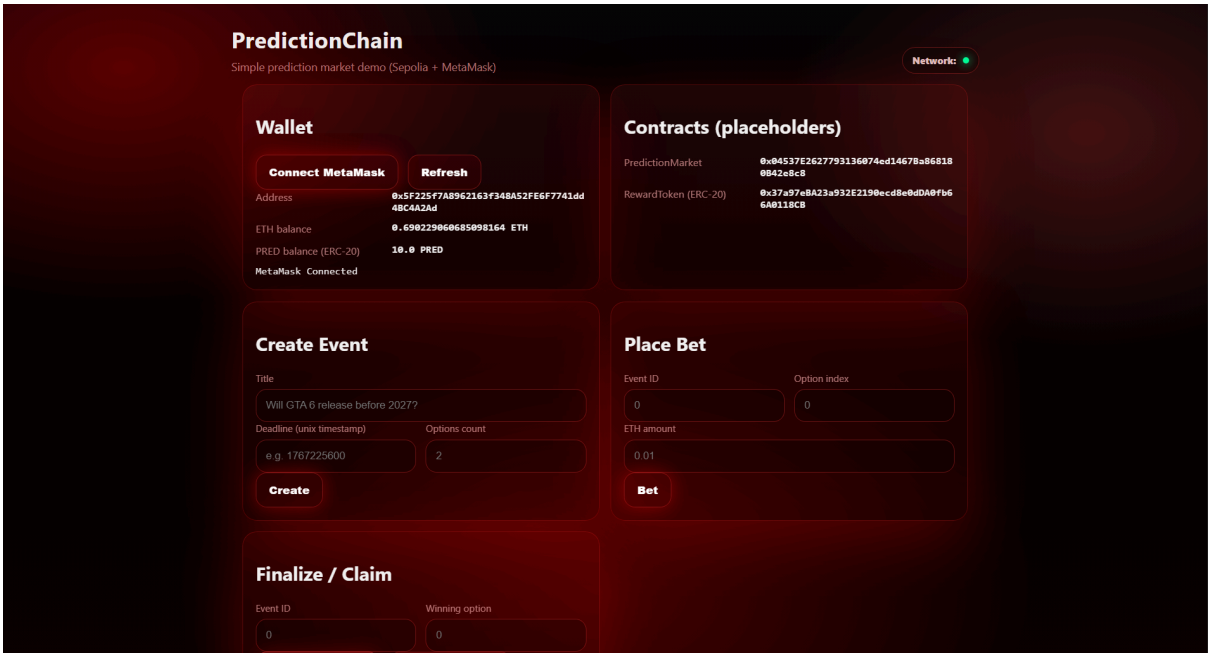
This ETH is only for testing and has no real monetary value.

7. Screenshots / Experimental Results

This section presents screenshots that demonstrate successful deployment and execution of the PredictionChain decentralized application on the Sepolia test network.

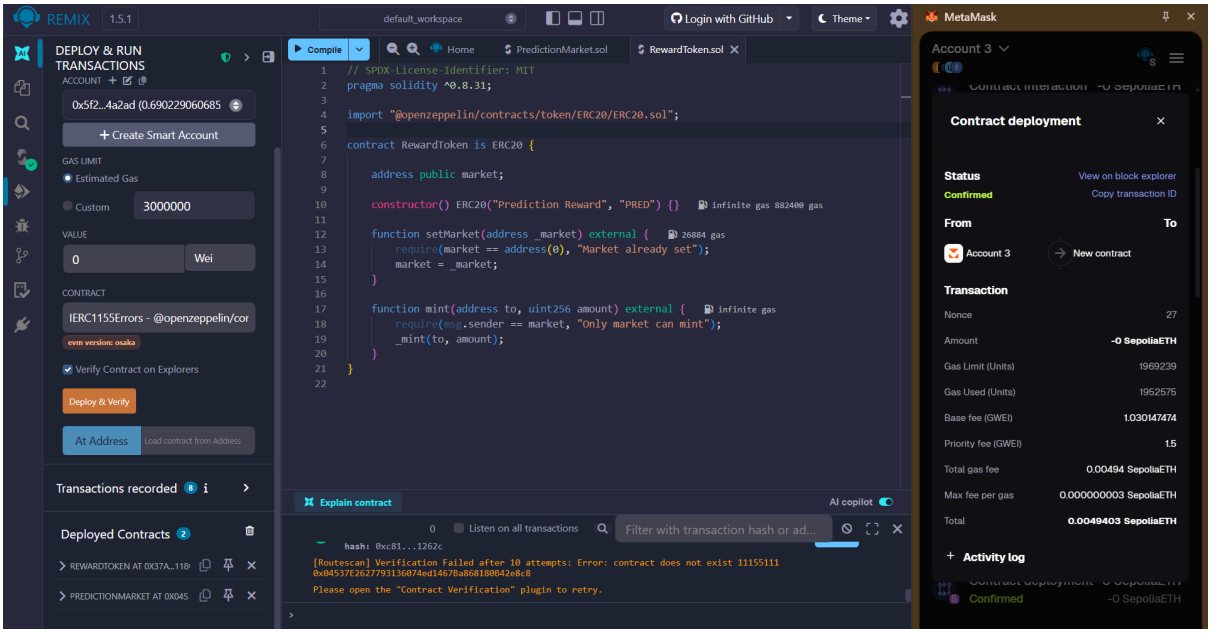
Screenshot 1 – MetaMask Connected

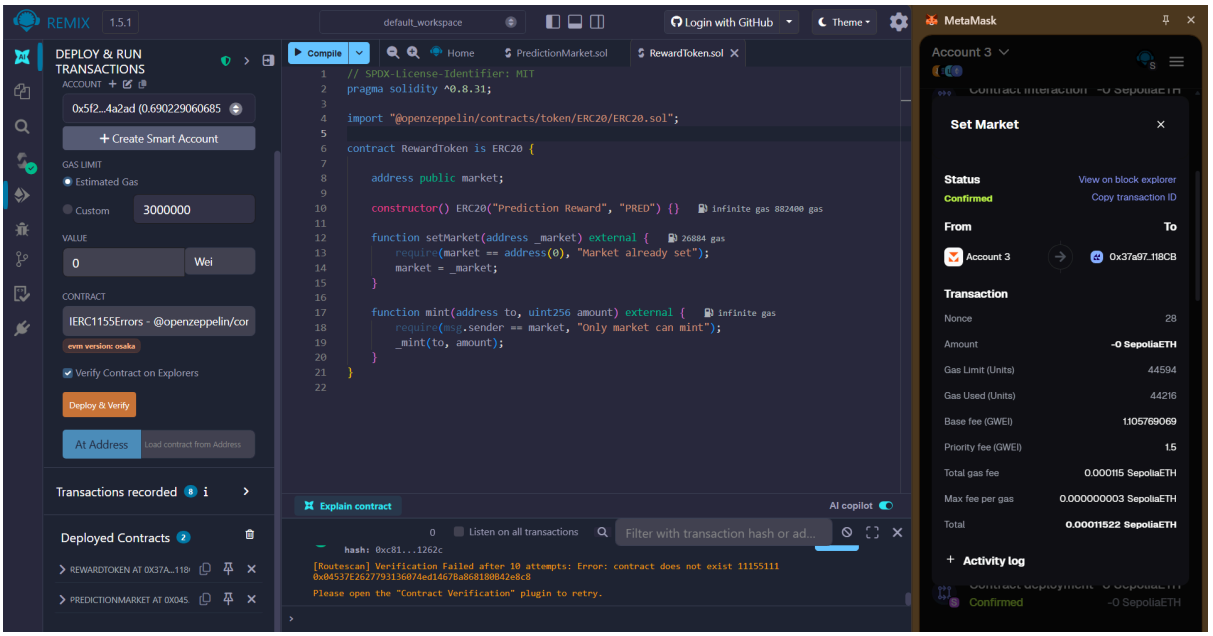
Description: **MetaMask wallet connected to the Sepolia test network with the active account displayed.**



Screenshot 2 – Contract Deployment

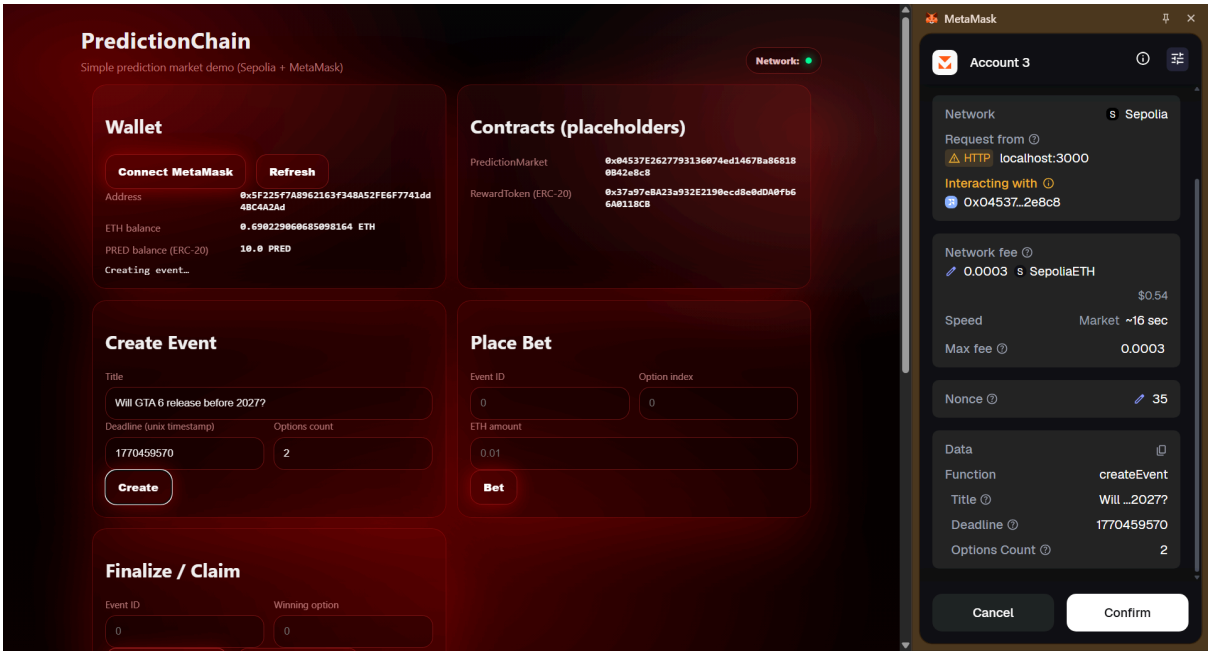
Description: **Deployment of the PredictionToken and PredictionChain smart contracts using Remix IDE and MetaMask.**





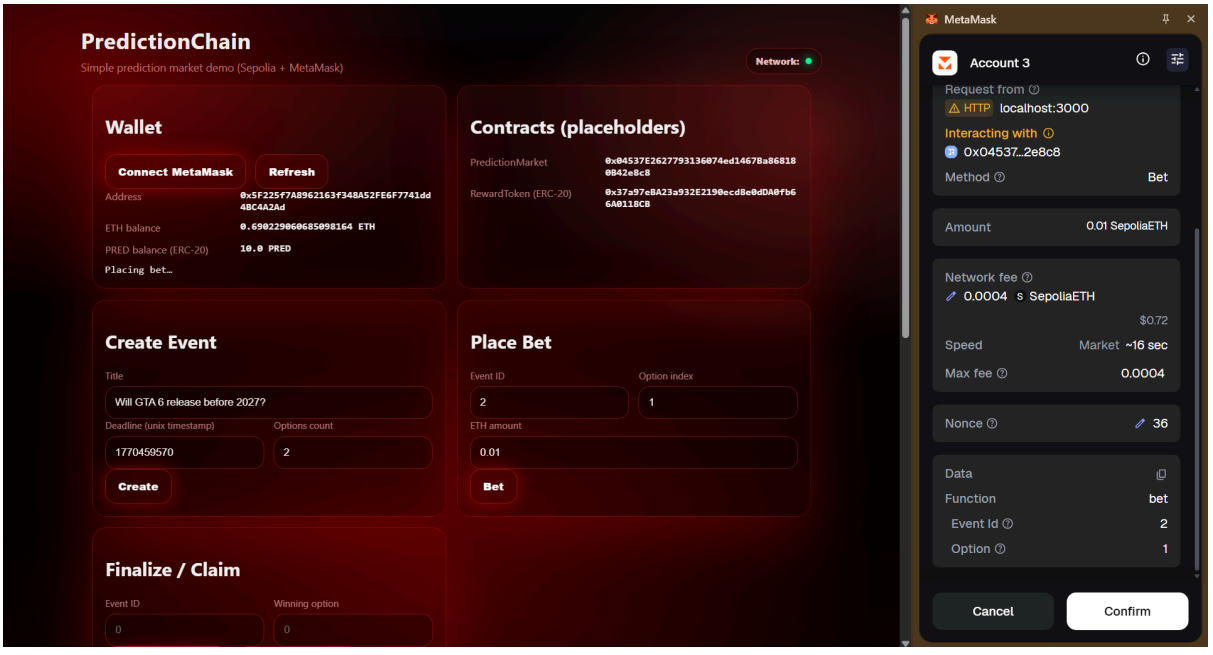
Screenshot 3 – Create Event Transaction

Description: Successful execution of the createEvent() function confirmed in MetaMask.



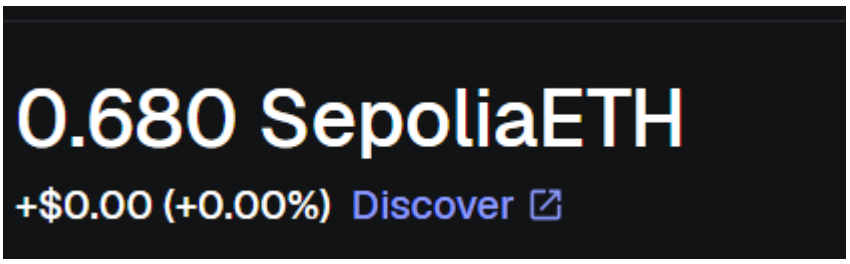
Screenshot 4 – Place Bet with ETH

Description: **User placing a bet by sending test ETH to the PredictionChain contract.**



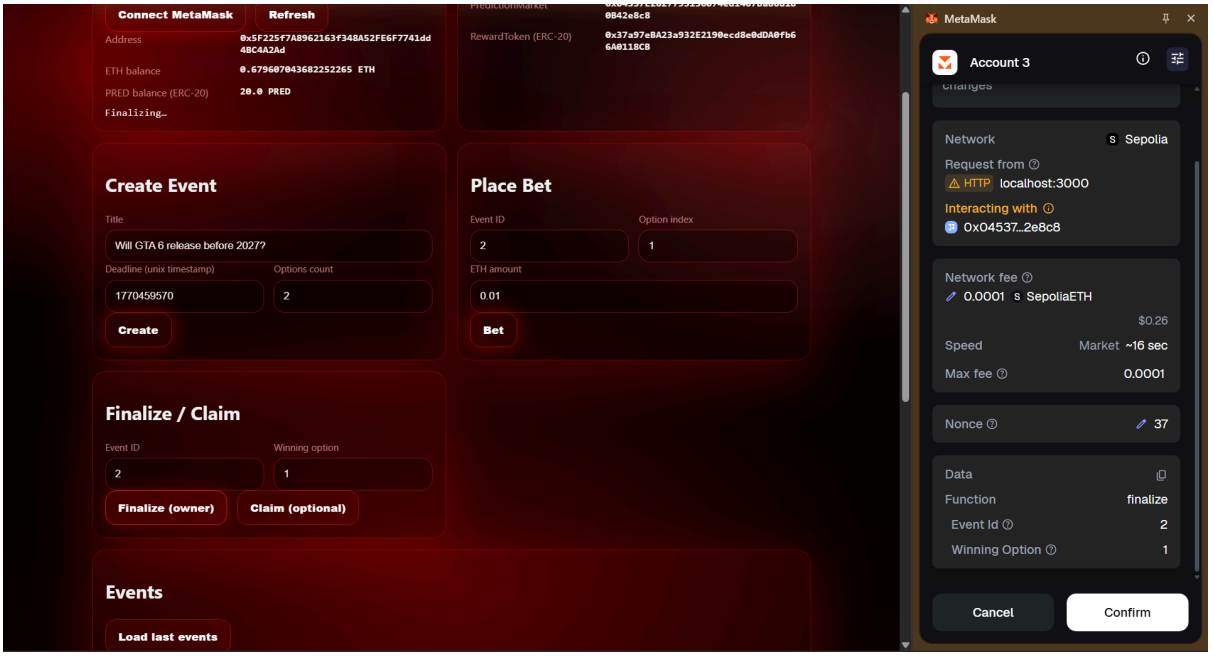
Screenshot 5 – ERC-20 Token Balance

Description: **Updated PRED token balance after participating in a prediction event.**



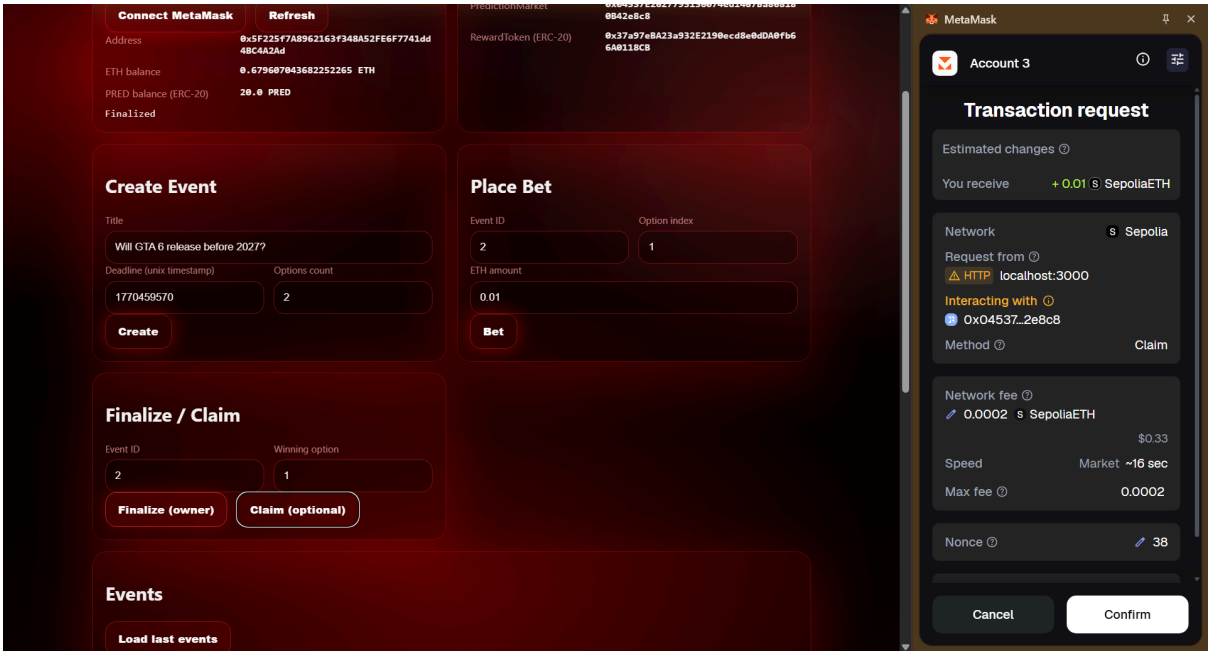
Screenshot 6 – Finalize Event

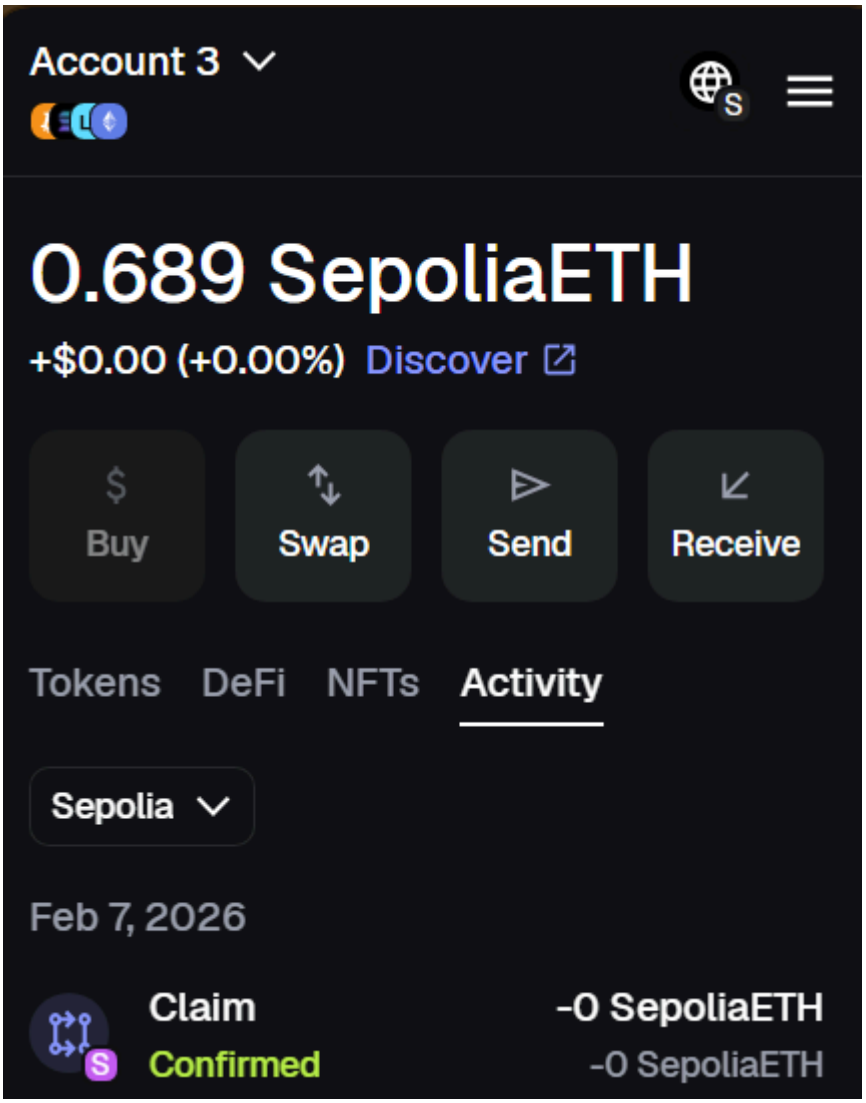
Description: **Event finalization after the deadline with a selected winning outcome.**



Screenshot 7 – Claim Reward

Description: **Winning participant successfully claiming ETH reward from the contract.**


Connect MetaMask
Refresh
Address
0x5F225f7A8962163f348A52FE6F7741dd4BC4A2Ad
ETH balance
0.679607043682252265 ETH
PRED balance (ERC-20)
20.0 PRED
Finalized
RewardToken (ERC-20)
0x37a97eBA23a932E2190ecd8e0dDA0fb66A0118CB
Create Event
Title
Will GTA 6 release before 2027?
Deadline (unix timestamp)
1770459570
Options count
2
Create
Place Bet
Event ID
2
Option index
1
ETH amount
0.01
Bet
Finalize / Claim
Event ID
2
Winning option
1
Finalize (owner)
Claim (optional)
Events
Load last events
MetaMask
Account 3
Transaction request
Estimated changes
You receive
+ 0.01 SepoliaETH
Network
Sepolia
Request from
HTTP localhost:3000
Interacting with
0x04537...2e8c8
Method
Claim
Network fee
0.0002 SepoliaETH
$0.33
Speed
Market ~16 sec
Max fee
0.0002
Nonce
38
Cancel
Confirm

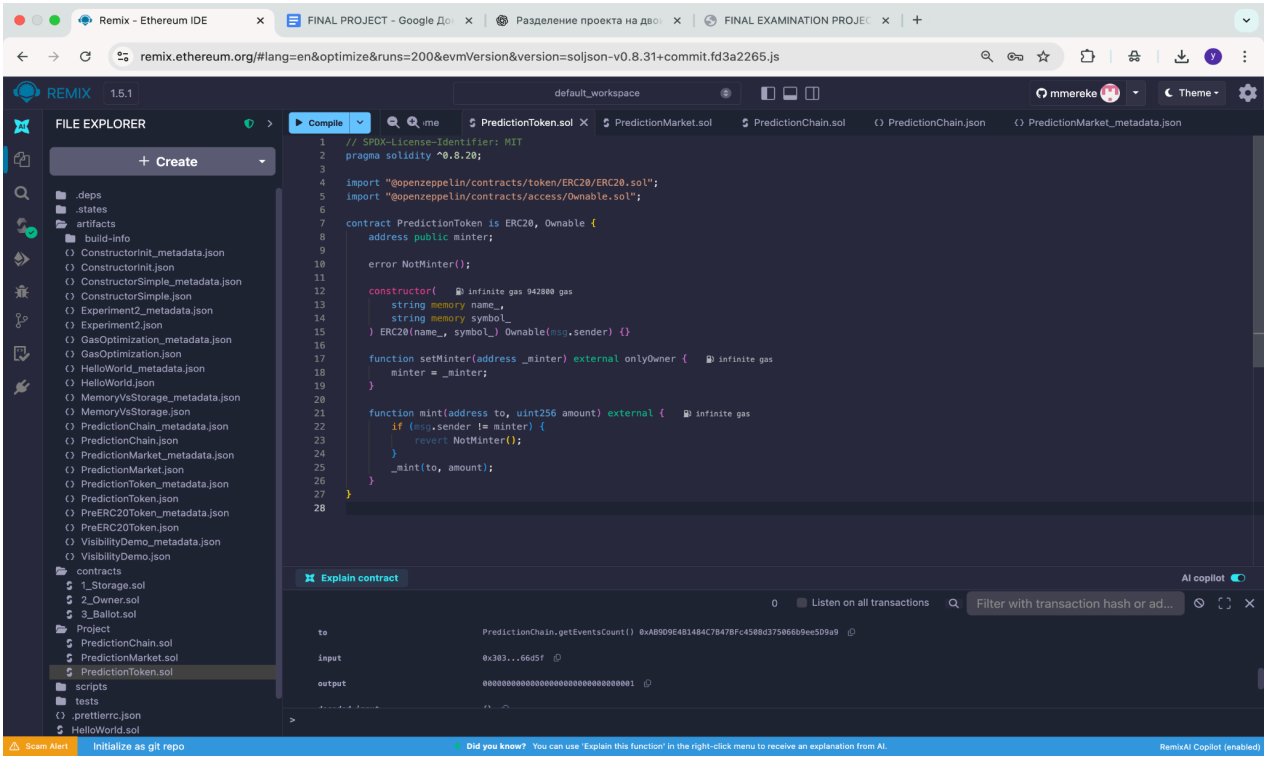

Account 3
0.689 SepoliaETH
+$0.00 (+0.00%) Discover
Buy
Swap
Send
Receive
Tokens
DeFi
NFTs
Activity
Sepolia
Feb 7, 2026
Claim
-0 SepoliaETH
Confirmed
-0 SepoliaETH

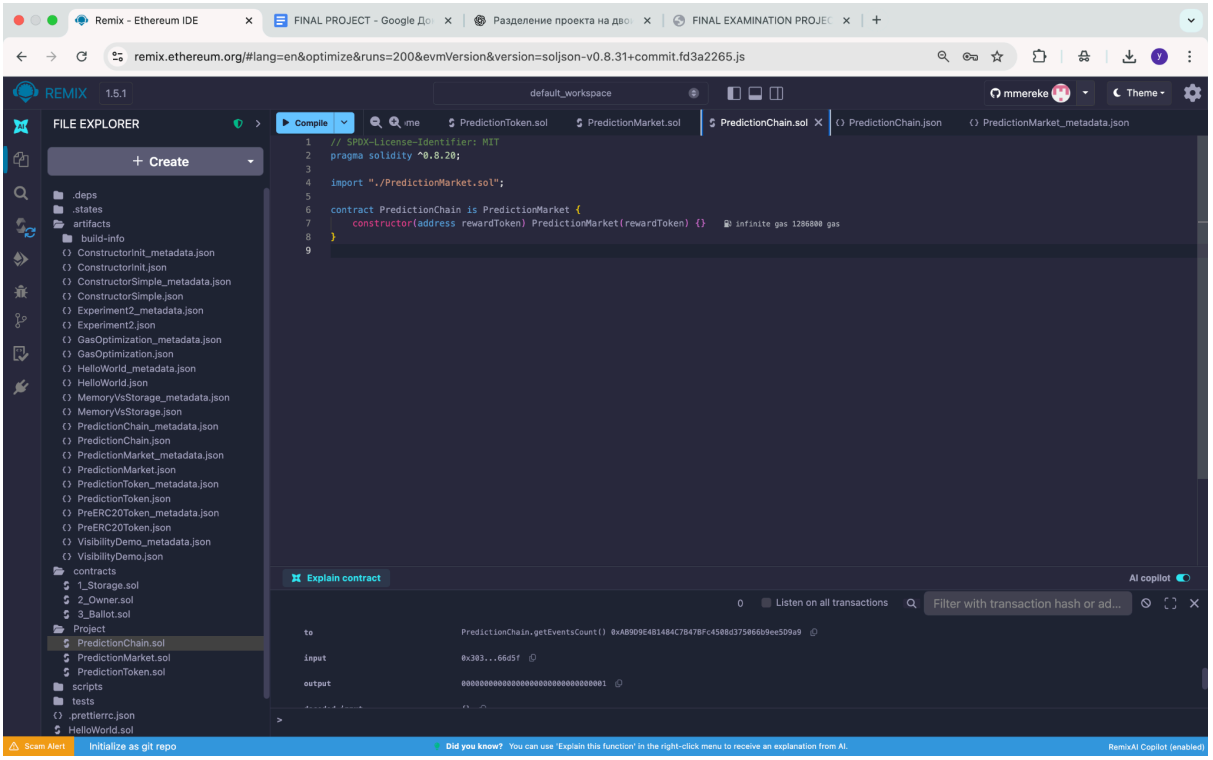
2.1. Smart Contract Implementation

Smart contract implementation in this project includes two main components:

1. **RewardToken (ERC-20 participation token)**

2. **PredictionMarket (main prediction market and crowdfunding logic)**





2.2. ERC-20 Reward Token

The project includes an ERC-20 token called PredictionChain Participation Token (PRED), implemented in the RewardToken smart contract.

This token is automatically minted whenever a user participates in a prediction event. The token is not used as a currency and has no real monetary value. Instead, it serves as a participation and reputation token, demonstrating ERC-20 integration within a decentralized application.

The token follows the ERC-20 standard.
Minting rights are restricted to the PredictionMarket smart contract using a dedicated setter function.
No ownership-based control is used in order to keep the token logic simple, transparent, and suitable for educational purposes.

2.3. Token Minting Logic

Token minting is triggered automatically inside the placeBet() function of the PredictionMarket contract.
For every 1 wei contributed to an event, the user receives 1 PRED token.

This simple reward formula was chosen to make the system easy to understand and to clearly demonstrate the relationship between ETH contributions and ERC-20 token issuance.

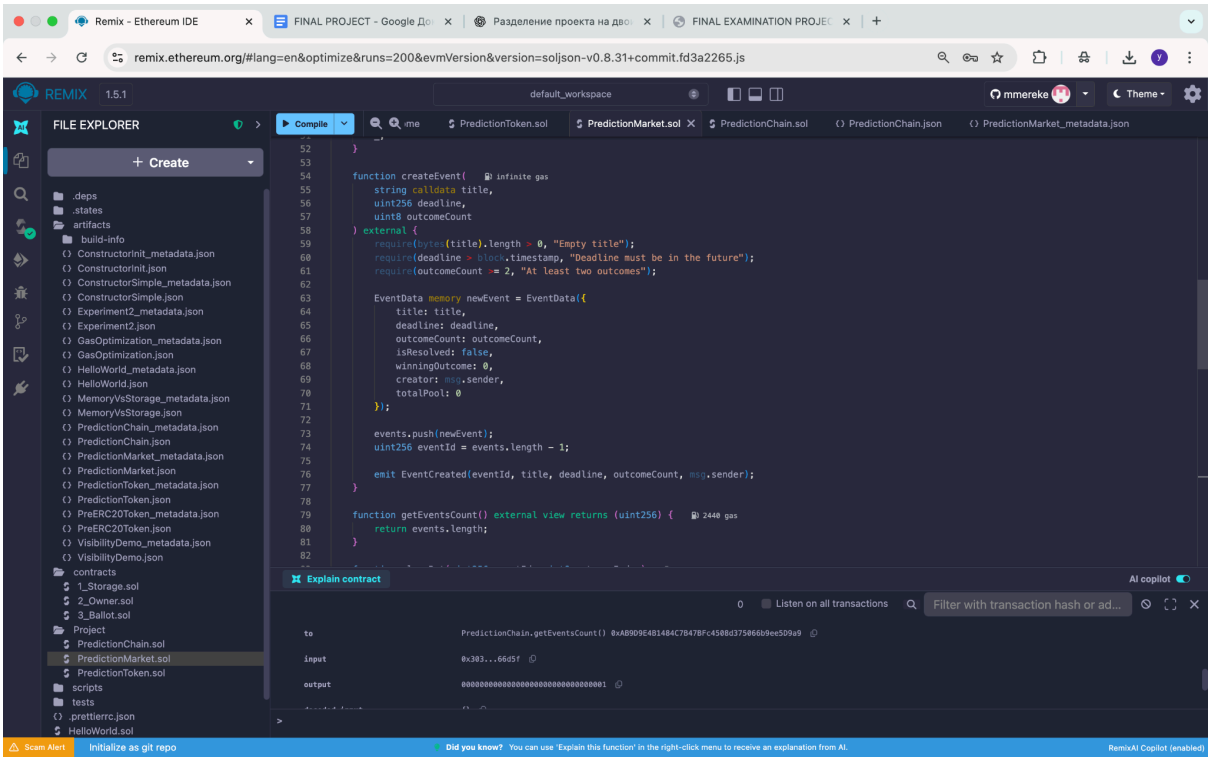
2.4. Event/Campaign Creation

Users can create events (campaigns) defining:

– title

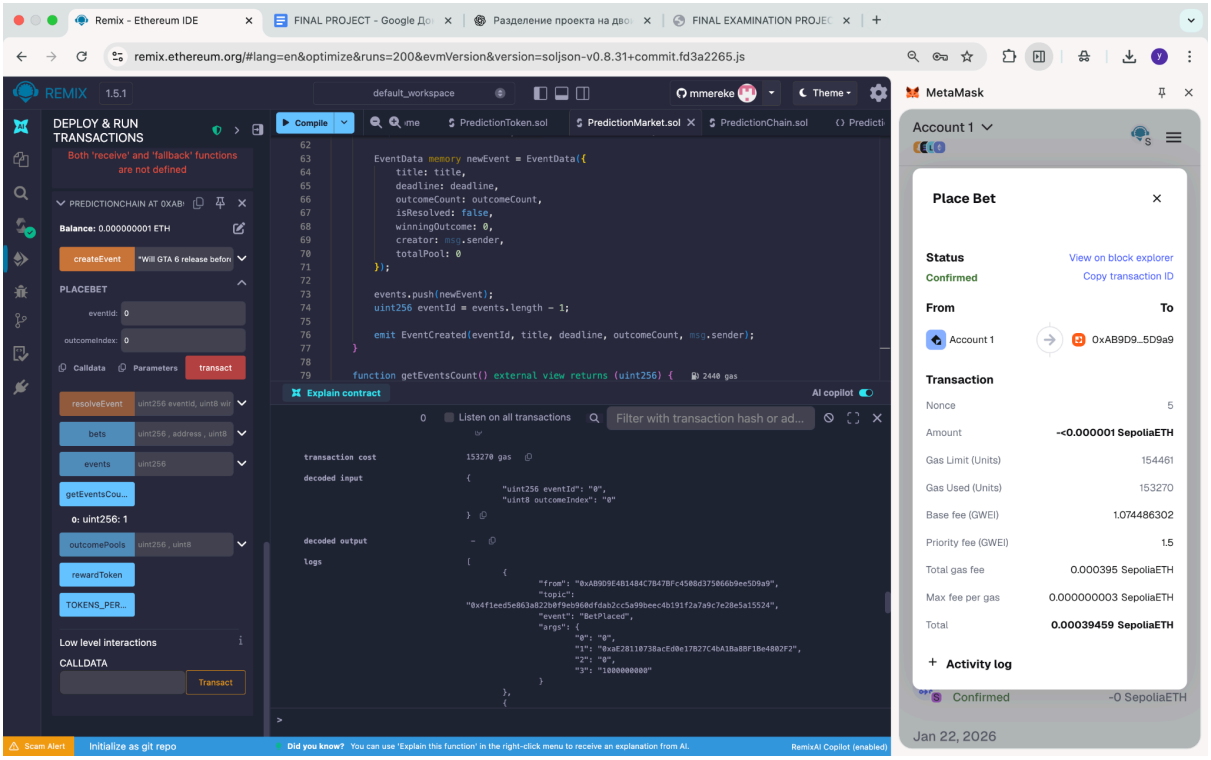
– deadline

– number of outcomes



2.5. Contributions (placeBet)

Users participate by sending ETH to the contract using placeBet().
The contract records the contribution amount and outcome choice.



2.6. Event Finalization and Reward Claiming

After an event reaches its deadline, it can be finalized by the event creator using the finalize() function.
During finalization, the correct outcome is selected.

A participant is considered a winner if the outcome selected during placeBet() matches the winning outcome defined during finalization.

Winning participants can call the claim() function to withdraw their share of the ETH pool.
The payout is calculated proportionally, based on the amount of ETH contributed by the user relative to the total ETH contributed to the winning outcome.

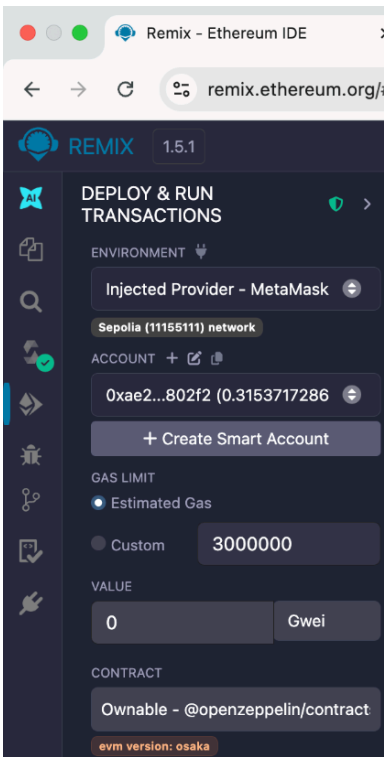
To ensure security:

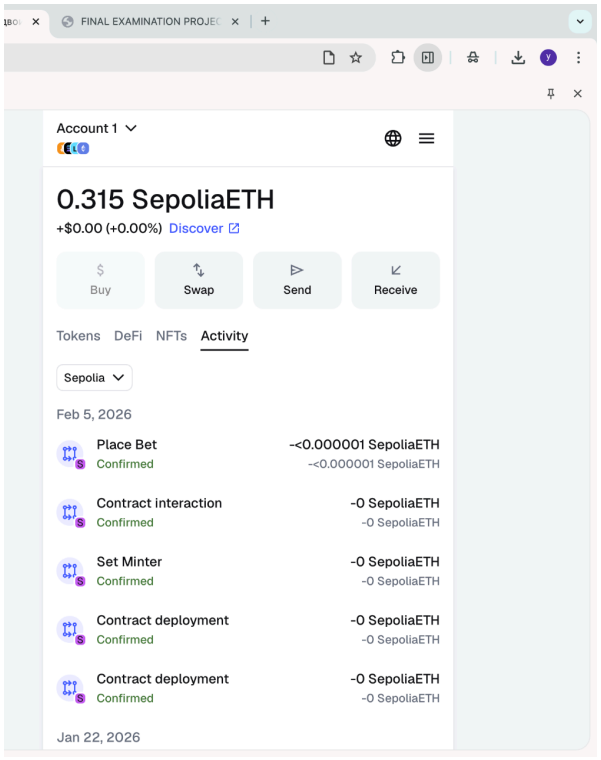
- each participant can claim rewards only once,
- repeated claims are rejected by the smart contract,
- participants who selected an incorrect outcome cannot withdraw ETH.

This logic ensures a fair and transparent reward distribution mechanism enforced entirely on-chain.

2.7. MetaMask Integration

The project fully integrates MetaMask:
– account connection
– Sepolia network validation
– signing all transactions (deploy, createEvent, placeBet)

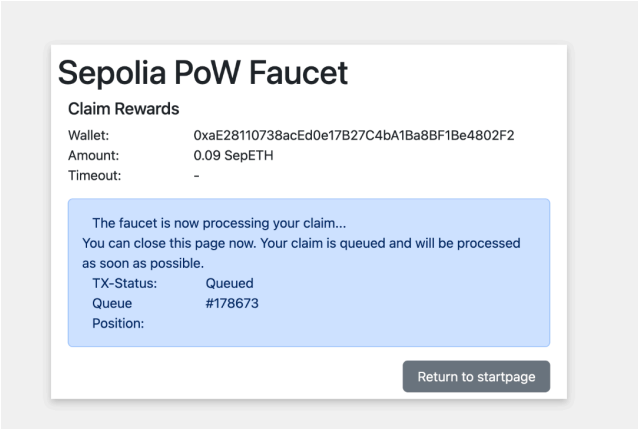




2.8. Sepolia Testnet Deploy

Both contracts were deployed to the Sepolia test network using Remix and MetaMask.
PredictionToken deployed: 0x2ac98aab672a13F69FA5e939F366dd6daCF09f56
PredictionChain deployed: 0xAB9D9E4B1484C7B47BFc4508d375066b9ee5D9a9

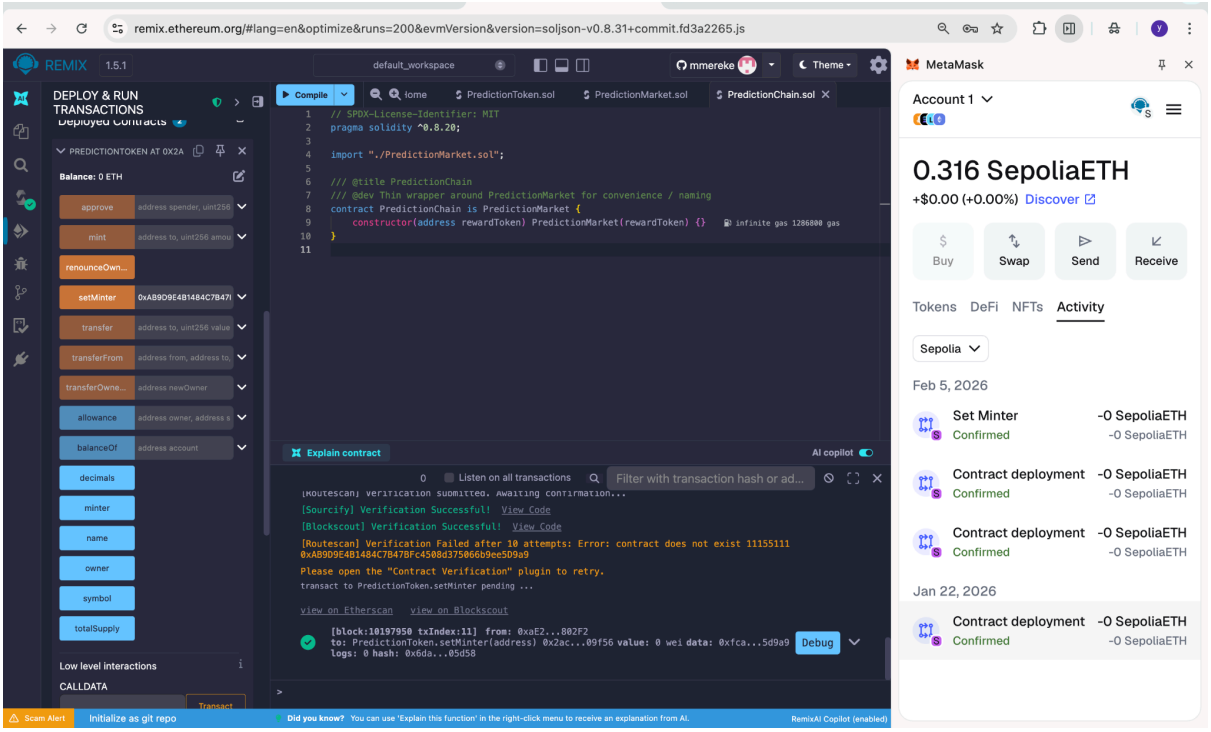
2.9. Test ETH Receipt

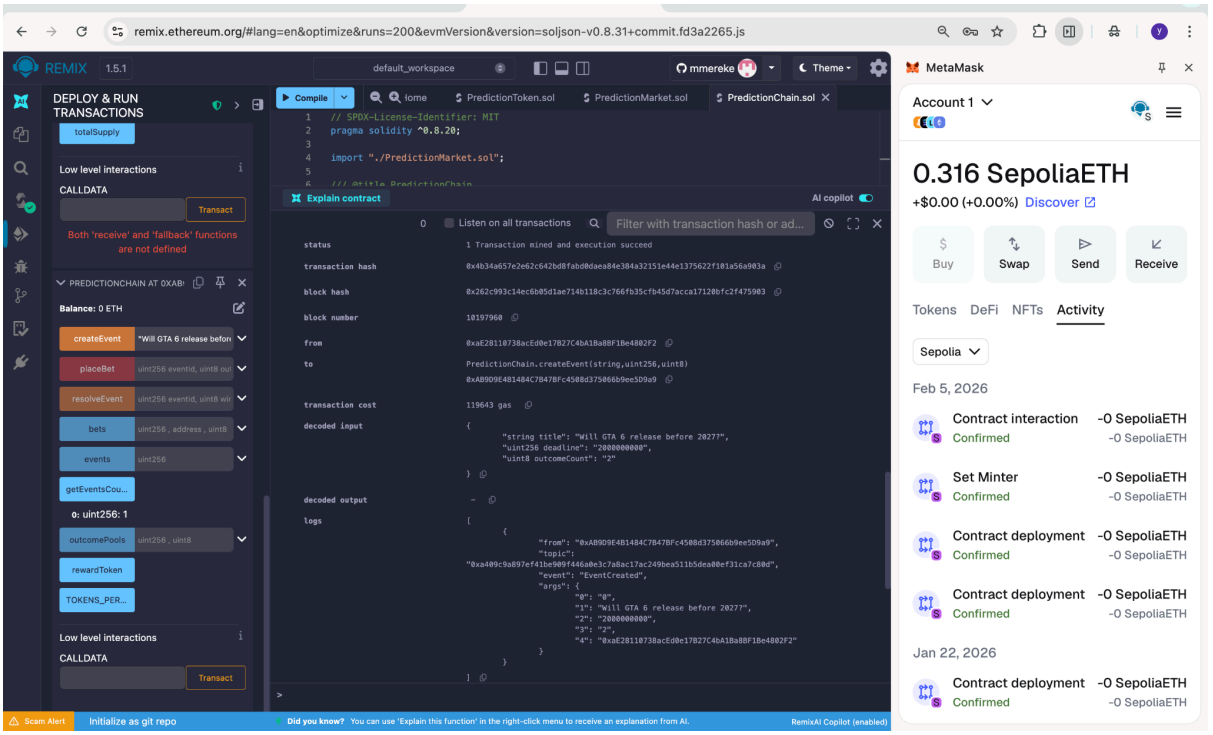


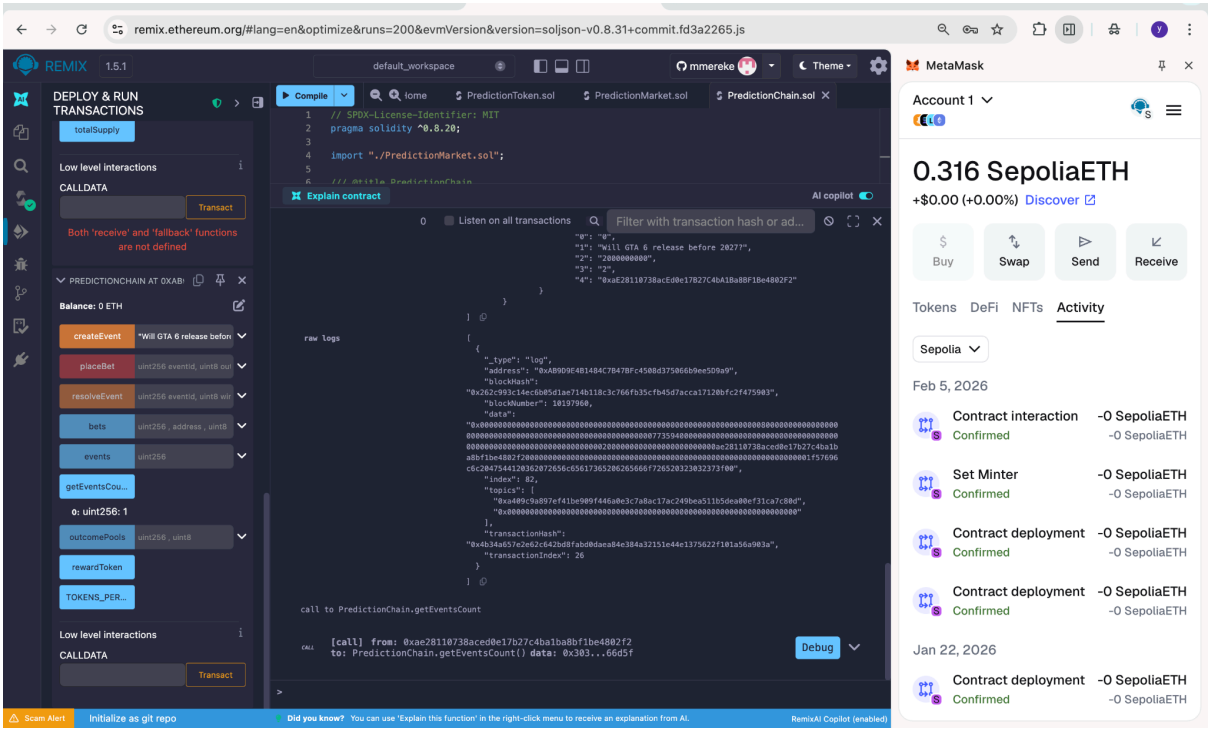
3.Manual Testing

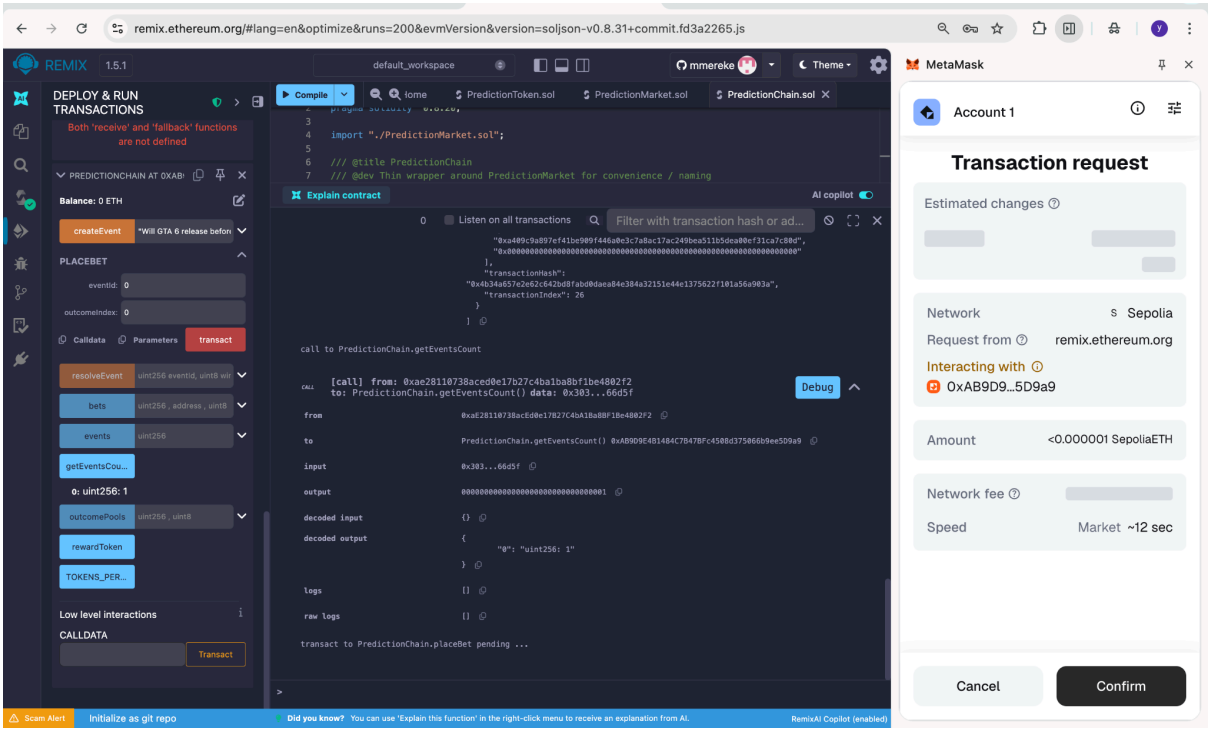
Manual contract testing was performed via Remix:

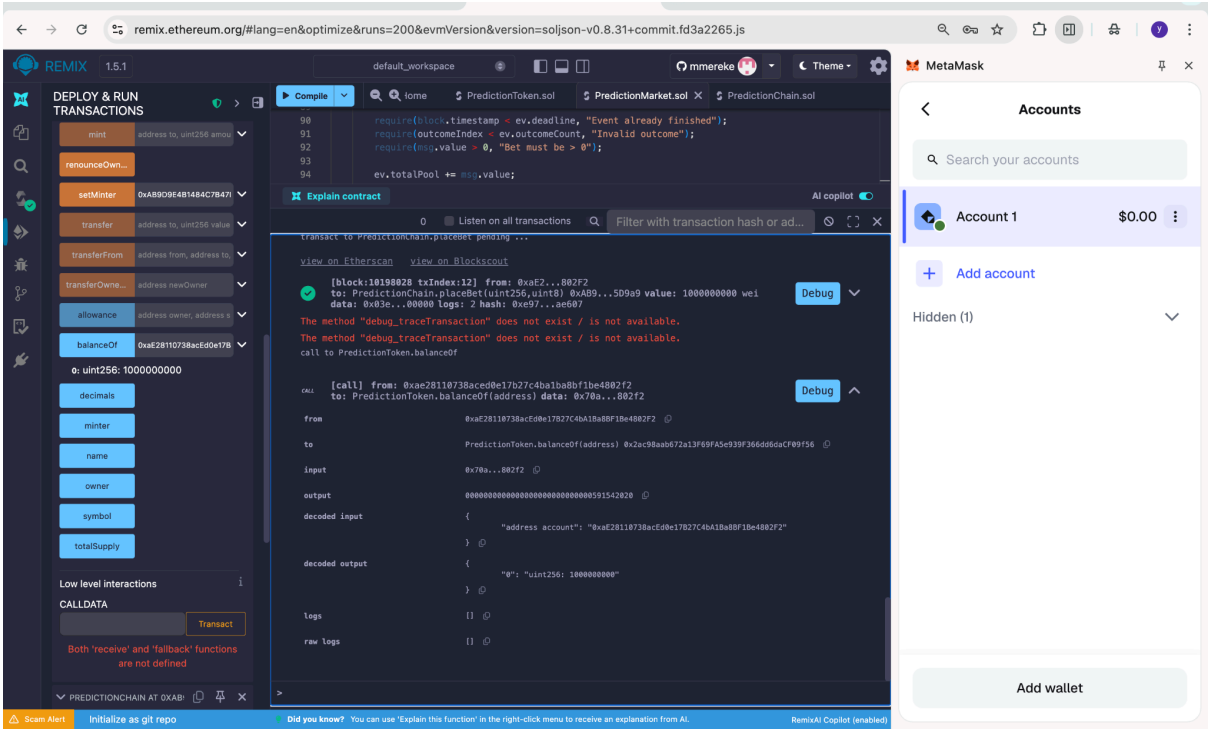
1. **createEvent eventCount = 1**
2. **placeBet with value transaction confirmed**
3. **Check pools updated correctly**
4. **Check PRED balance tokens minted**
5. **MetaMask showed all TX confirmations**




remix.ethereum.org/#lang=en&optimize&runs=200&evmVersion&version=soljson-v0.8.31+commit.fd3a2265.js
REMIX 1.5.1
DEPLOY & RUN TRANSACTIONS
totalSupply
Low level interactions
CALLDATA
Transact
Both 'receive' and 'fallback' functions are not defined
PREDICTIONCHAIN AT 0XAB
Balance: 0 ETH
createEvent
placeBet
resolveEvent
bets
events
getEventsCou...
0: uint256: 1
outcomePools
rewardToken
TOKENS_PER...
Compile
PredictionToken.sol
PredictionMarket.sol
PredictionChain.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;
import "./PredictionMarket.sol";
Explain contract
AI copilot
Listen on all transactions
Filter with transaction hash or ad...
status
1 Transaction mined and execution succeed
transaction hash
0x4b34a657e2e62c642bd8fabd0daea84e384a32151e44e1375622f101a56a903a
block hash
0x262c993c14ec6b05d1ae714b118c3c766fb35cfb45d7acca17120bfc2f475903
block number
10197960
from
0xaE28110738acEd0e17B27C4bA1Ba8BF1Be4802F2
to
PredictionChain.createEvent(string,uint256,uint8)
0xAB9D9E4B1484C7B47BFc4508d375066b9ee5D9a9
transaction cost
119643 gas
decoded input
"string title": "Will GTA 6 release before 2027?",
"uint256 deadline": "2000000000",
"uint8 outcomeCount": "2"
decoded output
logs
"from": "0xAB9D9E4B1484C7B47BFc4508d375066b9ee5D9a9",
"topic": "0xa409c9a897ef41be909f446a0e3c7a8ac17ac249bea511b5dea00ef31ca7c80d",
"event": "EventCreated",
"args": {
"0": "0",
"1": "Will GTA 6 release before 2027?",
"2": "2000000000",
"3": "2",
"4": "0xaE28110738acEd0e17B27C4bA1Ba8BF1Be4802F2"
Scam Alert
Initialize as git repo
Did you know? You can use 'Explain this function' in the right-click menu to receive an explanation from AI.
RemixAI Copilot (enabled)
MetaMask
Account 1
0.316 SepoliaETH
+$0.00 (+0.00%) Discover
Buy
Swap
Send
Receive
Tokens
DeFi
NFTs
Activity
Sepolia
Feb 5, 2026
Contract interaction -0 SepoliaETH
Confirmed -0 SepoliaETH
Set Minter -0 SepoliaETH
Confirmed -0 SepoliaETH
Contract deployment -0 SepoliaETH
Confirmed -0 SepoliaETH
Contract deployment -0 SepoliaETH
Confirmed -0 SepoliaETH
Jan 22, 2026
Contract deployment -0 SepoliaETH
Confirmed -0 SepoliaETH


remix.ethereum.org/#lang=en&optimize&runs=200&evmVersion&version=soljson-v0.8.31+commit.fd3a2265.js
REMIX 1.5.1
DEPLOY & RUN TRANSACTIONS
totalSupply
Low level interactions
CALLDATA
Transact
Both 'receive' and 'fallback' functions are not defined
PREDICTIONCHAIN AT 0XAB
Balance: 0 ETH
createEvent
placeBet
resolveEvent
bets
events
getEventsCou...
0: uint256: 1
outcomePools
rewardToken
TOKENS_PER...
Compile
PredictionToken.sol
PredictionMarket.sol
PredictionChain.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;
import "./PredictionMarket.sol";
Explain contract
AI copilot
Listen on all transactions
Filter with transaction hash or ad...
raw logs
call to PredictionChain.getEventsCount
[call] from: 0xae2811073Baced0e17b27c4ba1ba8bf1be4802f2
to: PredictionChain.getEventsCount() data: 0x303...66d5f
Debug
Scam Alert
Initialize as git repo
Did you know? You can use 'Explain this function' in the right-click menu to receive an explanation from AI.
RemixAI Copilot (enabled)
MetaMask
Account 1
0.316 SepoliaETH
+$0.00 (+0.00%) Discover
Buy
Swap
Send
Receive
Tokens
DeFi
NFTs
Activity
Sepolia
Feb 5, 2026
Contract interaction -0 SepoliaETH
Confirmed -0 SepoliaETH
Set Minter -0 SepoliaETH
Confirmed -0 SepoliaETH
Contract deployment -0 SepoliaETH
Confirmed -0 SepoliaETH
Contract deployment -0 SepoliaETH
Confirmed -0 SepoliaETH
Jan 22, 2026
Contract deployment -0 SepoliaETH
Confirmed -0 SepoliaETH


remix.ethereum.org/#lang=en&optimize&runs=200&evmVersion&version=soljson-v0.8.31+commit.fd3a2265.js
REMIX 1.5.1
DEPLOY & RUN TRANSACTIONS
Both 'receive' and 'fallback' functions are not defined
createEvent
PLACEBET
transact
resolveEvent
bets
events
getEventsCou...
0: uint256: 1
outcomePools
rewardToken
TOKENS_PER...
Low level interactions
CALLDATA
Transact
PredictionToken.sol
PredictionMarket.sol
PredictionChain.sol
Explain contract
AI copilot
Listen on all transactions
Filter with transaction hash or ad...
call to PredictionChain.getEventsCount
[call] from: 0xae28110738aced0e17b27c4ba1ba8bf1be4802f2 to: PredictionChain.getEventsCount() data: 0x303...66d5f
Debug
transact to PredictionChain.placeBet pending ...
Initialize as git repo
Did you know? You can use 'Explain this function' in the right-click menu to receive an explanation from AI.
RemixAI Copilot (enabled)
MetaMask
Account 1
Transaction request
Estimated changes
Network
Sepolia
Request from
remix.ethereum.org
Interacting with
0xAB9D9...5D9a9
Amount
<0.000001 SepoliaETH
Network fee
Speed
Market ~12 sec
Cancel
Confirm


remix.ethereum.org/#lang=en&optimize&runs=200&evmVersion&version=soljson-v0.8.31+commit.fd3a2265.js
REMIX 1.5.1
DEPLOY & RUN TRANSACTIONS
mint
renounceOwn...
setMinter
transfer
transferFrom
transferOwne...
allowance
balanceOf
0: uint256: 1000000000
decimals
minter
name
owner
symbol
totalSupply
Low level interactions
CALLDATA
Transact
Both 'receive' and 'fallback' functions are not defined
PredictionToken.sol
PredictionMarket.sol
PredictionChain.sol
require(block.timestamp < ev.deadline, "Event already finished");
require(outcomeIndex < ev.outcomeCount, "Invalid outcome");
require(msg.value > 0, "Bet must be > 0");
ev.totalPool += msg.value;
Explain contract
AI copilot
Listen on all transactions
Filter with transaction hash or ad...
view on Etherscan
view on Blockscout
[block:10198028 txIndex:12] from: 0xaE2...802F2 to: PredictionChain.placeBet(uint256,uint8) 0xAB9...5D9a9 value: 1000000000 wei data: 0x03e...00000 logs: 2 hash: 0xe97...ae607
Debug
The method "debug_traceTransaction" does not exist / is not available.
The method "debug_traceTransaction" does not exist / is not available.
call to PredictionToken.balanceOf
[call] from: 0xae28110738aced0e17b27c4ba1ba8bf1be4802f2 to: PredictionToken.balanceOf(address) data: 0x70a...802f2
Initialize as git repo
Did you know? You can use 'Explain this function' in the right-click menu to receive an explanation from AI.
RemixAI Copilot (enabled)
MetaMask
Accounts
Search your accounts
Account 1
$0.00
Add account
Hidden (1)
Add wallet