Prediction Market Contract

The Prediction Market contract sets up a scenario to determine the outcome of a football game between two teams. The contract uses the Equivalence Principle to ensure accurate and consistent decision-making based on the game's resolution data.

PredictionMarket

```python
import json
from genvm.base.equivalence_principle import Equivalen
from genvm.base.icontract import IContract

class PredictionMarket(IContract):
    def __init__(self, game_date: str, team1: str, tea
        """
        Initializes a new instance of the prediction m

        Args:
            game_date (str): The date of the game in t
            team1 (str): The name of the first team.
            team2 (str): The name of the second team.

        Attributes:
            has_resolved (bool): Indicates whether the
            game_date (str): The date of the game.
            resolution_url (str): The URL to the game'
            team1 (str): The name of the first team.
            team2 (str): The name of the second team.
        """
        self.has_resolved = False
        self.game_date = game_date
        self.resolution_url = 'https://www.bbc.com/spo
        self.team1 = team1
        self.team2 = team2

    async def resolve(self) -> None:

        if self.has_resolved:
            return "Already resolved"

        final_result = {}
        async with EquivalencePrinciple(
                result=final_result,
                principle="The score and the winner ha
                comparative=True,
            ) as eq:
            web_data = await eq.get_webpage(self.resol
            print(web_data)

            task = f"""In the following web page, find
            Team 1: {self.team1}
            Team 2: {self.team2}

            Web page content:
            {web_data}
            End of web page data.

            If it says "Kick off [time]" between the n
            If you fail to extract the score, assume t

            Respond with the following JSON format:
```

```
            {{
                "score": str, // The score with number
                "winner": int, // The number of the wi
            }}
            """
            result = await eq.call_llm(task)
            print(result)
            eq.set(result)

        result_json = json.loads(final_result['output'

        if result_json['winner'] > -1:
            self.has_resolved = True
            self.winner = result_json['winner']
            self.score = result_json['score']

        return result_json
```

You can check out this code on our GitHub

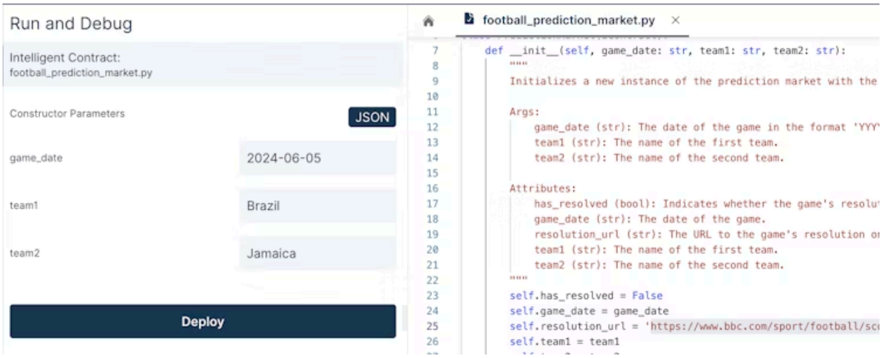
Deploying the Contract

To deploy the Prediction Market contract, you'll need to initialize the contract state correctly. This will impact how the contract will respond to the game's resolution.

1. Provide the game date and the names of the two teams. The `game_date` , `team1` , and `team2` constructor

parameters are automatically detected from the code. For example, you might set `game_date` to "2024-06-05", `team1` to "Brazil", and `team2` to "Jamaica".

2. Once the game details are set, deploy the contract to make it ready to interact and resolve the game results.

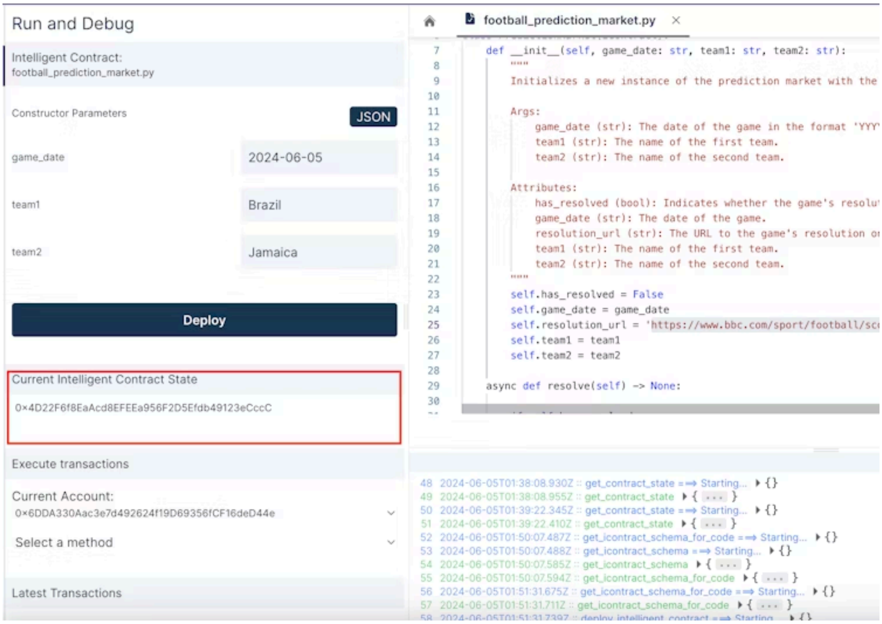


game_date	2024-06-05
team1	Brazil
team2	Jamaica

```
    def __init__(self, game_date: str, team1: str, team2: str):
        """
        Initializes a new instance of the prediction market with the

        Args:
            game_date (str): The date of the game in the format 'YYY
            team1 (str): The name of the first team.
            team2 (str): The name of the second team.

        Attributes:
            has_resolved (bool): Indicates whether the game's resolu
            game_date (str): The date of the game.
            resolution_url (str): The URL to the game's resolution o
            team1 (str): The name of the first team.
            team2 (str): The name of the second team.
        """
        self.has_resolved = False
        self.game_date = game_date
        self.resolution_url = 'https://www.bbc.com/sport/football/sc
        self.team1 = team1
```

Checking the Contract State

Once the contract is deployed, you can check its state in the **Current Intelligent Contract State** section. This section displays the contract address and the current account.

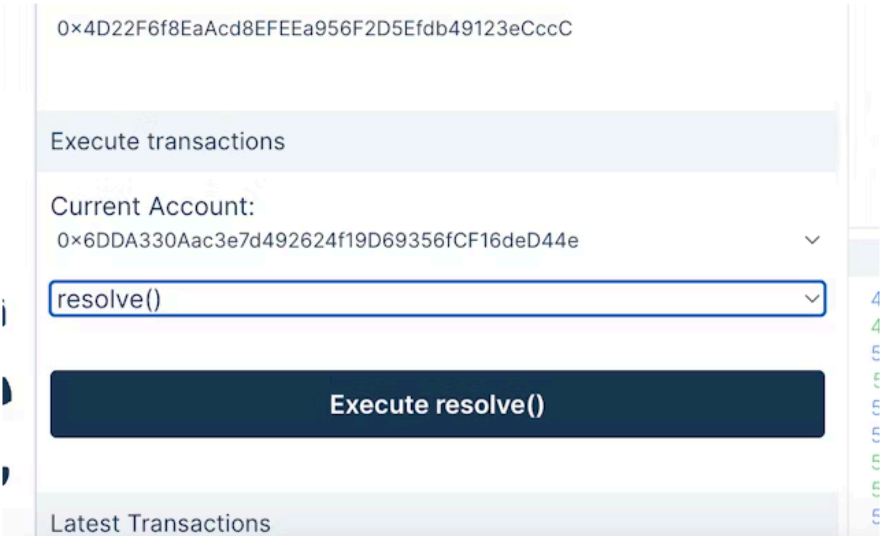


```
    def __init__(self, game_date: str, team1: str, team2: str):
        """
        Initializes a new instance of the prediction market with the

        Args:
            game_date (str): The date of the game in the format 'YYY
            team1 (str): The name of the first team.
            team2 (str): The name of the second team.

        Attributes:
            has_resolved (bool): Indicates whether the game's resolu
            game_date (str): The date of the game.
            resolution_url (str): The URL to the game's resolution o
            team1 (str): The name of the first team.
            team2 (str): The name of the second team.
        """
        self.has_resolved = False
        self.game_date = game_date
        self.resolution_url = 'https://www.bbc.com/sport/football/sc
        self.team1 = team1
        self.team2 = team2

    async def resolve(self) -> None:
```

```
2024-06-05T01:38:08.930Z :: get_contract_state ==> Starting... ▸ {}
2024-06-05T01:38:08.955Z :: get_contract_state ▸ { ... }
2024-06-05T01:39:22.345Z :: get_contract_state ==> Starting... ▸ {}
2024-06-05T01:39:22.410Z :: get_contract_state ▸ { ... }
2024-06-05T01:50:07.487Z :: get_icontract_schema_for_code ==> Starting... ▸ {}
2024-06-05T01:50:07.488Z :: get_icontract_schema ==> Starting... ▸ {}
2024-06-05T01:50:07.585Z :: get_icontract_schema ▸ { ... }
2024-06-05T01:50:07.594Z :: get_icontract_schema_for_code ▸ { ... }
2024-06-05T01:51:31.675Z :: get_icontract_schema_for_code ==> Starting... ▸ {}
2024-06-05T01:51:31.711Z :: get_icontract_schema_for_code ▸ { ... }
2024-06-05T01:51:31.739Z :: deploy_intelligent_contract ==> Starting... ▸ {}
```

Executing Transactions

To interact with the deployed contract, go to the **Execute Transactions** section. Here, you can call the `resolve` method to process the game's result. This triggers the contract's logic to retrieve the game's data and determine the outcome based on the Equivalence Principle criteria defined.


0x4D22F6f8EaAcd8EFEEa956F2D5Efdb49123eCccC

Execute transactions

Current Account:
0x6DDA330Aac3e7d492624f19D69356fCF16deD44e

Analyzing the Contract's Decisions

Analyzing the Contract's Decisions

When the `resolve` method is executed:

- The LLM retrieves the game data from the specified URL.
- It validates the game's outcome according to the Equivalence Principle defined in the code.

- Finally, it returns a JSON response that includes the game's score and the winner.

Handling Different Scenarios

- If the game has started but not finished, the JSON response will indicate the game is not resolved yet.
- If the game has finished, the JSON response will include the final score and the winning team.
- If the game hasn't started, the JSON response will indicate this status.

You can view the logs to see detailed information about the contract interaction.

Last updated on June 6, 2024