توجه: این گزارش بهتر است به صورت آنلاین در آدرس زیر مشاهده گردد.

https://ahbanavi.github.io/blockchain-fuzzy-voting

پیاده‌سازی سیستم تصمیم‌گیری غیرمتمرکز مبتنی بر بلاکچین و منطق فازی

امیرحسین بانوی

دانشجوی کارشناسی ارشد،نرم‌افزار،کامپیوتر،دانشکده مهندسی،دانشگاه آزاد اسلامی،مشهد،ایران

فهرست مطالب

۱. چکیده

دنیای امروز همواره دارد به سمت یک محیط به هم پیوسته مهاجرت می‌کند و در این دنیا، نیاز به سیستم‌های توزیع شده بسیار مهم شده است. اکثر سیستم‌های توزیع شده به دلیل اینکه برای تصمیم‌گیری متکی به یک سیستم مرکزی هستند دارای مشکلاتی همچون single point of failure می‌باشند. همچنین کمبود امنیت اطلاعات و شفاف نبودن فرایند تصمیم‌گیری در این سیستم‌های مدیریتی متمرکز باعث شده است رویکردهای تصمیم‌گیری توزیع شده و غیرمتمرکز بر روی بلاکچین مورد استفاده قرار گیرند. در این پیاده‌سازی سعی شده است یک قرارداد هوشمند رای‌گیری بر روی بلاکچین اتریوم نوشته شده و در فرایند رای‌گیری و اولویت دهی تصمیمات از منطق فازی برای پوشش عدم قطعیت‌ها استفاده گردد.

۲. مقدمه

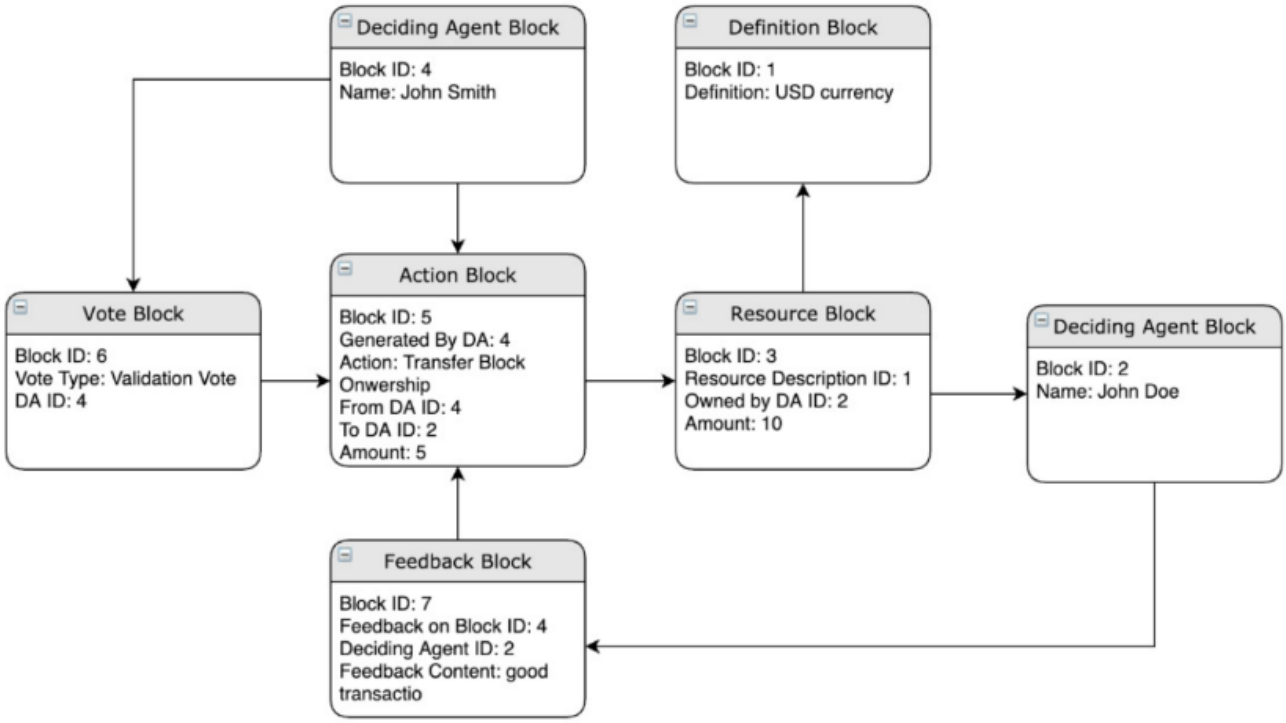
بیشتر سیستم‌های تصمیم‌گیری امروزه، نیاز به یک سیستم مدیریتی مرکزی دارند و تصمیم‌گیری ها عموما بر عهده‌ی یک موجودیت واحد و متمرکز است. این رویکرد مشکلاتی را مانند خرابی سایت متمرکز، سر بار اضافی، فساد پذیری موجودیت متمرکز در روند تصمیم‌گیری و همچنین امنیت پایین اطلاعات را دارد. قدم اول برای حل این مشکلات این است که فرآیند تصمیم‌گیری را به جای یک موجودیت واحد، بر عهده‌ی چندین عامل تصمیم‌گیرنده (Decision Agent) گذاشت و تصمیم اصلی را از طریق رای‌گیری میان این عوامل دریافت کرد. برای این‌که این سیستم مشکلاتی مانند SPOF، فساد پذیری و امنیت را نداشته باشد، می‌توان آن را بر روی یک بلاکچین BFT (Byzantine Fault Tolerance))، مانند اتریوم، پیاده سازی کرد. بلاکچین اتریوم از مفهومی به نام قراردادهای هوشمند برای پیاده‌سازی کاربردهای مختلف استفاده می‌کند، در این گزارش، از این مفاهیم استفاده و یک قرارداد هوشمند رای‌گیری پیاده سازی شده است.
ما نمی‌توانیم از رای‌گیری صحبت کرده و از عدم قطعیت‌ها در سیستم‌های رای‌گیری صحبت نکنیم، برای پوشش این عدم قطعیت‌ها یک سیستم فازی ممدانی پیاده‌سازی کرده و بر اساس آن از روی ورودی‌های ثبت شده در بلاکچین توسط عوامل تصمیم‌گیرنده، به هر تصمیم یک امتیاز اولویت می‌دهیم.
در این گزارش سعی شده است با پیاده‌سازی یک سیستم تصمیم‌گیری غیرمتمرکز بر روی بلاکچین بر مشکلات مربوط به مدیریت متمرکز غلبه کرد و همچنین با ترکیب این سیستم با منطق فازی، عدم قطعیت‌ها را پوشش داد.

۳. پیاده‌سازی

این پیاده سازی به سه بخش کلی تقسیم می‌گردد:
بخش اول: ابتدا باید قرارداد هوشمند رای گیری نوشته شود، برای نوشتن قرار داد هوشمند از زبان Solidity نسخه‌ی 0.8.15 استفاده شده است.
بخش دوم: در این بخش برای پیاده‌سازی منطق فازی از زبان پایتون استفاده شده است. در اینجا با استفاده از کتاب‌خانه‌ی SciKit-Fuzzy یک موتور استنتاج ممدانی نوشته شده و بر اساس آن تصمیمات اولویت بندی می‌گردند.
بخش سوم: ورودی‌های این سیستم باید از روی بلاکچین و از طریق قرارداد هوشمند گرفته شود، برای این کار از کتاب‌خانه‌ی Web3.py استفاده شده و اطلاعات به صورت مستقیم از قرارداد هوشمندی که در بخش اول پیاده‌سازی شده است گرفته شده، و به عنوان ورودی به سیستم فازی پیاده‌سازی شده در بخش دوم داده می‌شود.

۳.۱. قرارداد هوشمند

قرارداد هوشمند سیستم رای‌گیری بر اساس بلاک‌های معرفی شده در مقاله‌ی پایه پیاده‌سازی شده است:



قرارداد هوشمند به زبان سالیدیتی نوشته شده و در فایل Voting.sol ذخیره شده است.
به دلیل طولانی بودن کد قرارداد، از اضافه کردن آن به این گزارش چشم پوشی شده است، برای مشاهده‌ی کد این قرار داد به صورت کامل، اینجا را کلیک کنید.
اما به طور کلی این قرارداد شامل ویژگی‌های زیر می‌باشد:

- بلاک رای یا `VoteBlock` - ارجا به کد
  - شناسه‌ی رای
  - زمان ایجاد رای
  - مقدار رای: بین -۱ و ۱
  - آدرس Agent رای دهنده
- بلاک بازخورد یا `FeedbackBlock` - ارجا به کد
  - شناسه‌ی بازخورد
  - زمان ایجاد بازخورد
  - متن بازخورد
  - آدرس Agent بازخورد دهنده
- بلاک تصمیم یا `Decision` - ارجا به کد
  - شناسه‌ی تصمیم
  - زمان ایجاد تصمیم

- آدرس Agent ایجاد کننده‌ی تصمیم
- محتوای تصمیم
- لیستی از رای‌ها - `VoteBlock`
- لیستی از بازخوردها - `FeedbackBlock`

- بلاک عامل یا `Agent` - ارجا به کد

- نام عامل
- آدرس عامل - به صورت مپ - ارجا به کد

۳.۲. سیستم فازی

برای پیاده‌سازی سیستم فازی، از کتاب‌خانه‌ی SciKit-Fuzzy استفاده شده است.
مراحل کار به‌صورت زیر است:
ابتدا کتاب‌خانه‌های مورد نیاز در کد import می‌شود:

In [1]:
```
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt
import numpy as np


def Diagram(data):
    data.view()
```

۳.۲.۱. ورودی‌ها

سپس باید ورودی‌های سیستم فازی را تعیین کنیم
سیستم ما به تعداد عوامل تصمیم‌گیرنده، متغیر دارد. در این سیستم، هر چه جایگاه عامل بالاتر باشد، رای آن تاثیر بیشتری بر روی خروجی دارد.
مثلا اگر سیستم ۳ عامل داشته باشد، ما سه ورودی و متغیر زیر را داریم:

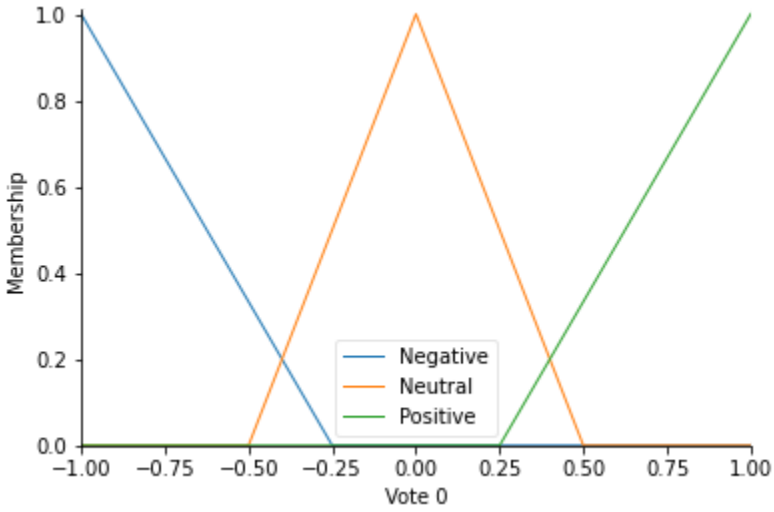
- ورودی اول: رای عامل ۱ - دارای بیشترین تاثیر
- ورودی دوم: رای عامل ۲ - تاثیر متوسط
- ورودی سوم: رای عامل ۳ - کمترین تاثیر

مقادیر زبانی تمامی ورودی‌ها به صورت زیر است:

- رای منفی
- رای ممتنع
- رای مثبت

قطعه کد زیر، به صورت پویا بر اساس تعداد عوامل تصمیم‌گیرنده در سیستم، ورودی‌ها را با مقادیر زبانی مشخص ایجاد می‌کند:
توابع و نمودار تعلق در کد و خروجی قابل مشاهده است.

In [2]:
```
TOTAL_AGENTS = 3

# create fuzzy Votes input variables for the number of agents
votes = {}
for i in range(TOTAL_AGENTS):
    votes[i]  = ctrl.Antecedent(np.arange(-1, 1.25, 0.25), 'Vote ' + str(i))
    votes[i]['Negative'] = fuzz.trimf(votes[i].universe, [-1, -1, -0.25])
    votes[i]['Neutral'] = fuzz.trimf(votes[i].universe, [-0.5, 0, 0.5])
    votes[i]['Positive'] = fuzz.trimf(votes[i].universe, [0.25, 1, 1])

Diagram(votes[0])
```

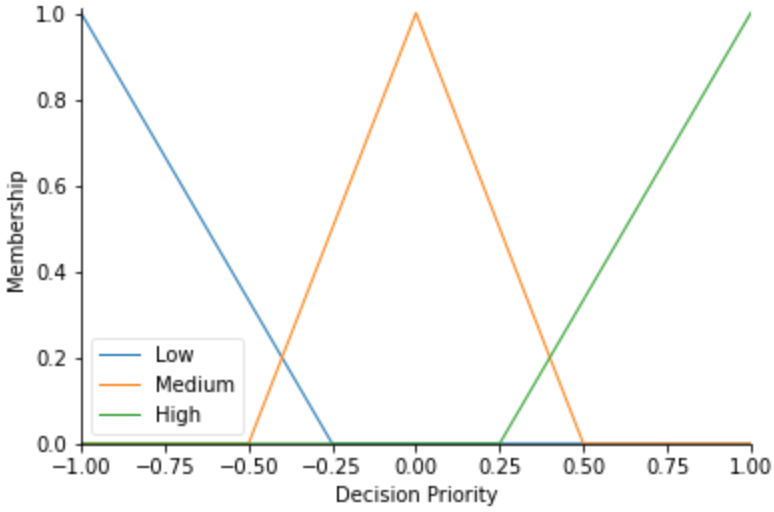


۳.۲.۲. خروجی

خروجی سیستم، میزان اولویت تصمیم می‌باشد. این میزان بین -1 و 1 بوده و به صورت زیر تعریف می‌گردد:

- اولویت پایین
- اولویت متوسط
- اولویت بالا

In [3]:
```
# output for decision priority
decision_priority = ctrl.Consequent(np.arange(-1, 1.25, 0.25), 'Decision Priority')
decision_priority['Low'] = fuzz.trimf(decision_priority.universe, [-1, -1, -0.25])
decision_priority['Medium'] = fuzz.trimf(decision_priority.universe, [-0.5, 0, 0.5])
decision_priority['High'] = fuzz.trimf(decision_priority.universe, [0.25, 1, 1])

Diagram(decision_priority)
```



۳.۲.۳. قوانین

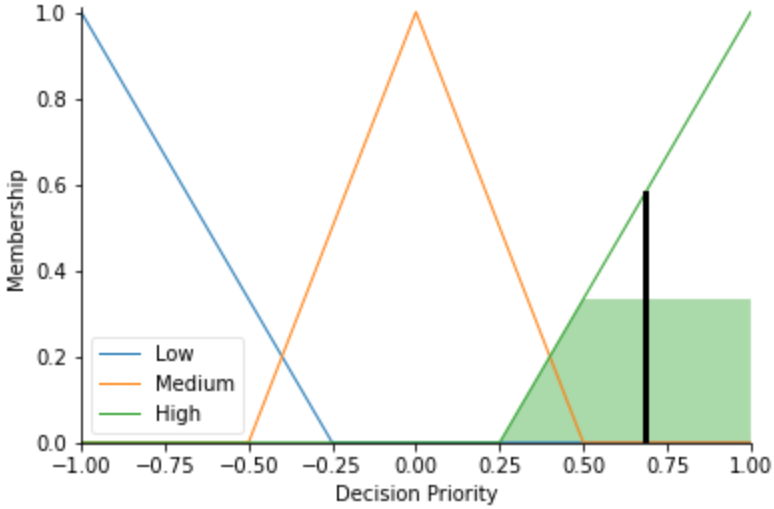
این سیستم دارای ۳ ورودی است که هر ورودی ۳ مقدار زبانی دارد، در نتیجه تعداد قوانین سیستم ۲۷ می‌باشد.
قوانین سیستم به صورت زیر است:

In [5]:
```
# create fuzzy rule base for the decision priority
# first votes has higher impact on decision priority

# lower the vote number, the higher the impact
rules = []
rules.append(ctrl.Rule(votes[0]['Negative']  & votes[1]['Negative']  & votes[2]['Negativ
rules.append(ctrl.Rule(votes[0]['Negative']  & votes[1]['Negative']  & votes[2]['Neutral
rules.append(ctrl.Rule(votes[0]['Negative']  & votes[1]['Negative']  & votes[2]['Positiv
rules.append(ctrl.Rule(votes[0]['Negative']  & votes[1]['Neutral']   & votes[2]['Negativ
rules.append(ctrl.Rule(votes[0]['Negative']  & votes[1]['Neutral']   & votes[2]['Neutral
rules.append(ctrl.Rule(votes[0]['Negative']  & votes[1]['Neutral']   & votes[2]['Positiv
rules.append(ctrl.Rule(votes[0]['Negative']  & votes[1]['Positive']  & votes[2]['Negativ
rules.append(ctrl.Rule(votes[0]['Negative']  & votes[1]['Positive']  & votes[2]['Neutral
rules.append(ctrl.Rule(votes[0]['Negative']  & votes[1]['Positive']  & votes[2]['Positiv
rules.append(ctrl.Rule(votes[0]['Neutral']   & votes[1]['Negative']  & votes[2]['Negativ
rules.append(ctrl.Rule(votes[0]['Neutral']   & votes[1]['Negative']  & votes[2]['Neutral
rules.append(ctrl.Rule(votes[0]['Neutral']   & votes[1]['Negative']  & votes[2]['Positiv
rules.append(ctrl.Rule(votes[0]['Neutral']   & votes[1]['Neutral']   & votes[2]['Negativ
rules.append(ctrl.Rule(votes[0]['Neutral']   & votes[1]['Neutral']   & votes[2]['Neutral
rules.append(ctrl.Rule(votes[0]['Neutral']   & votes[1]['Neutral']   & votes[2]['Positiv
rules.append(ctrl.Rule(votes[0]['Neutral']   & votes[1]['Positive']  & votes[2]['Negativ
rules.append(ctrl.Rule(votes[0]['Neutral']   & votes[1]['Positive']  & votes[2]['Neutral
rules.append(ctrl.Rule(votes[0]['Neutral']   & votes[1]['Positive']  & votes[2]['Positiv
rules.append(ctrl.Rule(votes[0]['Positive']  & votes[1]['Negative']  & votes[2]['Negativ
rules.append(ctrl.Rule(votes[0]['Positive']  & votes[1]['Negative']  & votes[2]['Neutral
rules.append(ctrl.Rule(votes[0]['Positive']  & votes[1]['Negative']  & votes[2]['Positiv
rules.append(ctrl.Rule(votes[0]['Positive']  & votes[1]['Neutral']   & votes[2]['Negativ
rules.append(ctrl.Rule(votes[0]['Positive']  & votes[1]['Neutral']   & votes[2]['Neutral
rules.append(ctrl.Rule(votes[0]['Positive']  & votes[1]['Neutral']   & votes[2]['Positiv
rules.append(ctrl.Rule(votes[0]['Positive']  & votes[1]['Positive']  & votes[2]['Negativ
rules.append(ctrl.Rule(votes[0]['Positive']  & votes[1]['Positive']  & votes[2]['Neutral
rules.append(ctrl.Rule(votes[0]['Positive']  & votes[1]['Positive']  & votes[2]['Positiv


# create fuzzy control system
decision_priority_ctrl = ctrl.ControlSystem(rules)
fuzzy_compute = ctrl.ControlSystemSimulation(decision_priority_ctrl)
```

پیش از آن‌که سراغ دریافت ورودی از بلاکچین برویم، بهتر است سیستم فازی را تست کنیم:

In [68]:
```
def compute_decision_priority(votes_):
    print('Votes: ', votes_)
    # set input values
    fuzzy_compute.input['Vote 0'] = votes_[0]
    fuzzy_compute.input['Vote 1'] = votes_[1]
    fuzzy_compute.input['Vote 2'] = votes_[2]

    # compute output
    fuzzy_compute.compute()

    # print output
    print('Decision Priority: ' + str(fuzzy_compute.output['Decision Priority']))
    decision_priority.view(sim=fuzzy_compute)

compute_decision_priority([0.5, 0.5, 0.5])
```

```
Votes:  [0.5, 0.5, 0.5]
Decision Priority: 0.6833333333333333
```



۳.۳. وب ۳ - اتصال قرارداد هوشمند به سیستم فازی

در این مرحله با استفاده از کتاب‌خانه‌ی Web3.py ما قرارداد هوشمند را بر روی شبکه دیپلوی کرده و با آن ارتباط برقرار می‌کنیم.

In [51]:
```
import web3
import os
import datetime
import random
import time
from dotenv import load_dotenv

load_dotenv()
```

Out[51]:
```
True
```

به منظور شبیه‌سازی شبکه‌ی اتریوم، از بلاکچین محلی Ganache استفاده می‌کنیم.

In [11]:
```
ganache-cli
```

```
Ganache CLI v6.12.2 (ganache-core: 2.13.2)
```

```
Available Accounts
==================
(0) 0x24ea6b8435F103DD873E9a48841a8dA5456f96e3 (100 ETH)
(1) 0x39a206c90a57f9477EF7d3c2cda6599b2D96CC19 (100 ETH)
(2) 0xd7184128E6E2fd8Baa10A6a781bb77FCc7af41dD (100 ETH)
(3) 0x9DF21B3c4bcA71f9c4063A4104Cc7a1c8d12A0A4 (100 ETH)
(4) 0xc1c523971477a5302ecbBf172cbA12ab8332BC56 (100 ETH)
(5) 0x79686BA36E05E7029D446DE1706FA86B9935Af53 (100 ETH)
(6) 0x7Ed2fdCAE69355391B38f214e113FabF18b9B510 (100 ETH)
(7) 0xd76ef98a52BEb57b29CBff26cbD7A806f985CB18 (100 ETH)
(8) 0xBc2a3C97744879AD43f35D99Fe1039c52DeDEc5d (100 ETH)
(9) 0xbe5ee610E578084532658D126d7689F88574A067 (100 ETH)

Private Keys
==================
(0) 0x193cb00808fdf28a4975cb2c608dbaf5ab6241ace72e335d3e1e1cb38081b838
(1) 0xdfa9c2507beeb2f9a4ca1f477c0fb77a5d2c9219bcd3d840ef34a6ef666dade4
(2) 0x62148e3fe4436a02cf386650abc3e3ce699486f077741eb2cf70501575782256
(3) 0x3fe1bc8dc3993869b335044351aae3e397e22217d4899d04a0d008aa0a6e9620
(4) 0xb937cbea5932204ed234992a691d9cf78af4affb58fea5439588aedc8f726a4c
(5) 0xd79c5158121810494ab6d0bb95baf3bc75edd1ab50a3b543ac3fc1622b60f233
(6) 0xb23a17b78fb88de0b6716cada6b3c0f057dbf038887d91e1ba69041c02e2274c
(7) 0xae35e6b3858f083619c82a84f8978836bee7c4b749872bbb6e65b5111d9c9ded
(8) 0x097b9470fdefd2d194a2e79d1dbdacc5855b484b6700924fd267e7de2565b794
(9) 0x2904923336435b16598415a0e08f4978fdb94a2cac2cd9905ffeeb447636cdf2

HD Wallet
==================
Mnemonic:      chicken become ripple love buyer sell shed universe drill solar stick goa
t
Base HD Path:  m/44'/60'/0'/0/{account_index}

Gas Price
==================
20000000000

Gas Limit
==================
6721975

Call Gas Limit
==================
9007199254740991

Listening on 127.0.0.1:8545
^C
```

همانطور که از خروجی بالا پیداست، بلاکچین Ganache به صورت پیش‌فرض شامل ۱۰ اکانت است که ما جلوتر از سه اکانت اول آن استفاده می‌کنیم.
سپس از طریق زیر کد را به شبکه متصل می‌کنیم:

In [13]:
```
w3 = web3.Web3(web3.HTTPProvider(os.getenv("WEB3_PROVIDER", "http://127.0.0.1:8545")))

# check if connected
print(w3.isConnected())
```

```
True
```

همچنین برای تعامل با قرارداد هوشمند، ما به سه عامل نیاز داریم که این سه Agent را از طریق زیر از شبکه دریافت می‌کنیم:

In [16]:
```
# load first 3 accounts from provider
accounts = w3.eth.accounts[:3]
print(accounts)
```

```
['0x95a661E5749B9333984D77b57B93eF03521FB7ED', '0x57CdaED8DAcB9081D67a1D598eae5269Be5e76
7B', '0x002f4562E0AbB80e2870317ab111b43677204453']
```

به جهت اینکه فرآیند دیپلوی کردن بر روی شبکه از طریق کد طولانی و توضیح آن خارج از بحث اصلی است، قرارداد هوشمند را در این مرحله با استفاده از Remix - Ethereum IDE دیپلوی کرده و آدرس آن را در کد وارد می‌کنیم.
همچنین برای تعامل با قرارداد هوشمند ما به ABI نیز نیاز داریم که از طریق کامپایلر آن را ایجاد و در کد بارگزاری می‌کنیم.

In [14]:
```
solc Voting.sol --abi -o build --overwrite
```

```
Compiler run successful. Artifact(s) can be found in directory "build".
```

In [22]:
```
# load abi from abi file
with open("build/Voting.abi", "r") as abi_file:
    abi = abi_file.read()

# get the contract
contract = w3.eth.contract(address='0xBF19e8bBF60f55051Cbb90E21bAe1e4dBEbF2182', abi=abi

# check if contract is connected
print("The name of first acc: " + contract.functions.agents(accounts[0]).call())
```

```
The name of first acc: First Agent
```

همانطور که از خروجی بالا پیداست، قرارداد هوشمند در کد ما لود شده و ما توانستیم نام اولین عامل را `First Agent` از آن دریافت کنیم.
اولین Agent هنگام دیپلوی کردن قرارداد هوشمند بر روی شبکه ایجاد می‌گردد.
در نتیجه ما به ۲ عامل دیگر نیاز داریم که از طریق تابع `registerAgent` به قرارداد هوشمند اضافه می‌شوند:

In [23]:
```
# add 2 more agents
contract.functions.registerAgent(accounts[1], 'Secound Agent').transact({'from': account
contract.functions.registerAgent(accounts[2], 'Third Agent').transact({'from': accounts[

# check if the agents are added
print("The name of second acc: " + contract.functions.agents(accounts[1]).call())
print("The name of third acc: " + contract.functions.agents(accounts[2]).call())
```

```
The name of second acc: Secound Agent
The name of third acc: Third Agent
```

حال ما تمامی ۳ عامل مورد نیاز خود را داریم، سپس برای ایجاد اولین decision باید به صورت زیر اقدام کنیم:

In [52]:
```
# create decision
tx_hash = contract.functions.createDecision('First Decision Test Context').transact({'fr
# w8 for the decision to be minted
w3.eth.wait_for_transaction_receipt(tx_hash)
```

```
# get decision id
decision_id = contract.functions.currentDecisionID().call()
print("The decision id is: " + str(decision_id))
```

```
The decision id is: 6
```

برای گرفتن اطلاعات decision ایجاد شده از تابع زیر استفاده می‌کنیم:

In [53]:
```
# get first decision
decision = contract.functions.decisions(decision_id).call()
print({
    'context': decision[0],
    'agent': decision[1],
    'createdAt': str(datetime.datetime.fromtimestamp(decision[2])),
})
```

```
{'context': 'First Decision Test Context', 'agent': '0x95a661E5749B9333984D77b57B93eF035
21FB7ED', 'createdAt': '2022-07-13 12:55:42'}
```

سپس ۳ عامل ما، باید به این decision ایجاد شده رای بدهند:

In [54]:
```
def vote_to_decision(decision_id_, vote_, account_id):
    print("Agent " + str(account_id) + " votes for decision " + str(decision_id_) + " wi
    tx_hash = contract.functions.vote(decision_id_, vote_).transact({'from': accounts[ac
    w3.eth.wait_for_transaction_receipt(tx_hash)

for i in range(3):
    random_vote = random.randint(-100, 100)
    vote_to_decision(decision_id, random_vote, i)
    time.sleep(1)
```

```
Agent 0 votes for decision 6 with vote -1
Agent 1 votes for decision 6 with vote -61
Agent 2 votes for decision 6 with vote 45
```

در این مرحله ما بایستی آرای مرحله‌ی قبل را از قرارداد هوشمند بر روی بلاکچین دریافت کنیم:

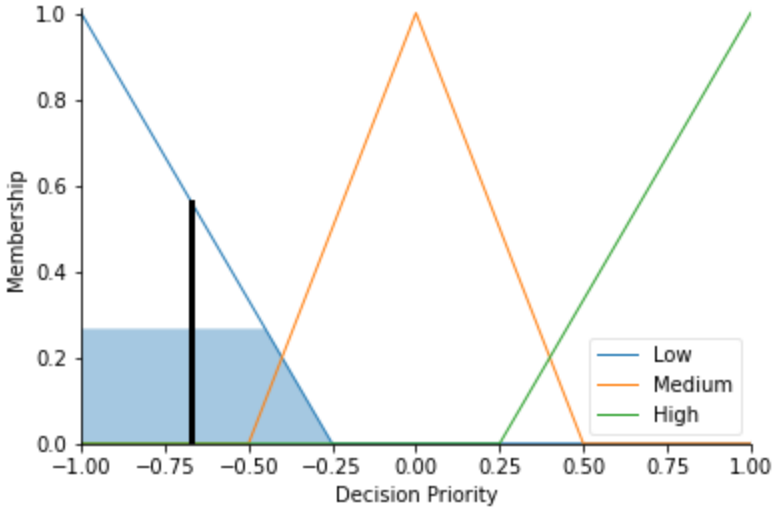
In [76]:
```
# get votes of first decision
votes_info = contract.functions.getVotes(decision_id).call()

def normialize_vote(vote_):
    v = []
    for i in range(len(vote_)):
        v.append(vote_[i][2] / 100)

    return v

votes_input = normialize_vote(votes_info)

print("Votes of first decision: " + str(votes_input))
```

```
Votes of first decision: [-0.01, -0.61, 0.45]
```

حال آرای دریافت شده بالا را به ورودی سیستم فازی ایجاد شده در مرحله قبل داده و نتایج را مشاهده می‌کنیم:

In [62]:
```
compute_decision_priority(votes_input)
```

```
Votes:  [-0.01, -0.61, 0.45]
Decision Priority: -0.6724358974358974
```



۴. نتایج

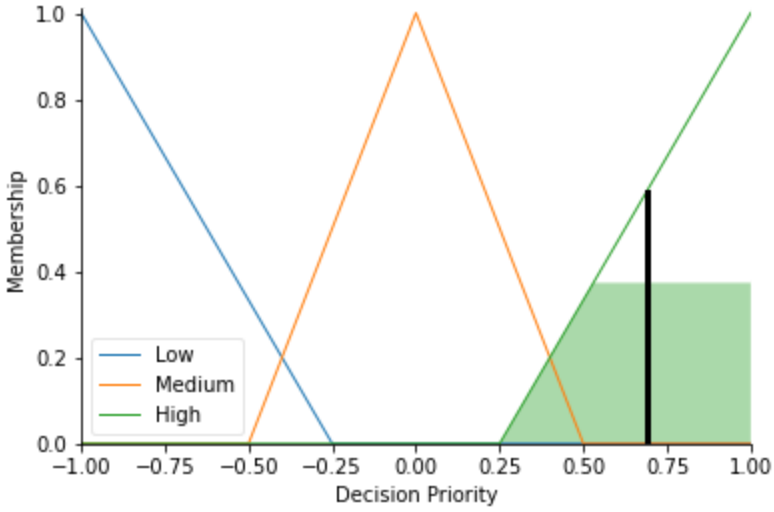
به منظور مشاهده‌ی نتایج بیشتر، مراحل قبل را در یک تابع خلاصه می‌کنیم:
این مراحل شامل گام‌های زیر است:

- ایجاد یک تصمیم
- دادن رای تصادفی به تصمیم توسط هر یک از عوامل
- دریافت آرا از قرارداد هوشمند بر روی بلاکچین
- دادن آرا به سیستم فازی و مشاهده‌ی نتایج

In [80]:
```
def full_steps():
    # create decision
    tx_hash = contract.functions.createDecision('Decision Context Test').transact({'from
    # w8 for the decision to be minted
    w3.eth.wait_for_transaction_receipt(tx_hash)

    # get decision id
    decision_id_ = contract.functions.currentDecisionID().call()
    print("The decision id is: " + str(decision_id_))

    # vote to decision
    for i in range(3):
        random_vote = random.randint(-100, 100)
        vote_to_decision(decision_id_, random_vote, i)
        time.sleep(1)

    # get votes of first decision
    votes_info_ = contract.functions.getVotes(decision_id_).call()
    votes_input_ = normialize_vote(votes_info_)

    compute_decision_priority(votes_input_)
```

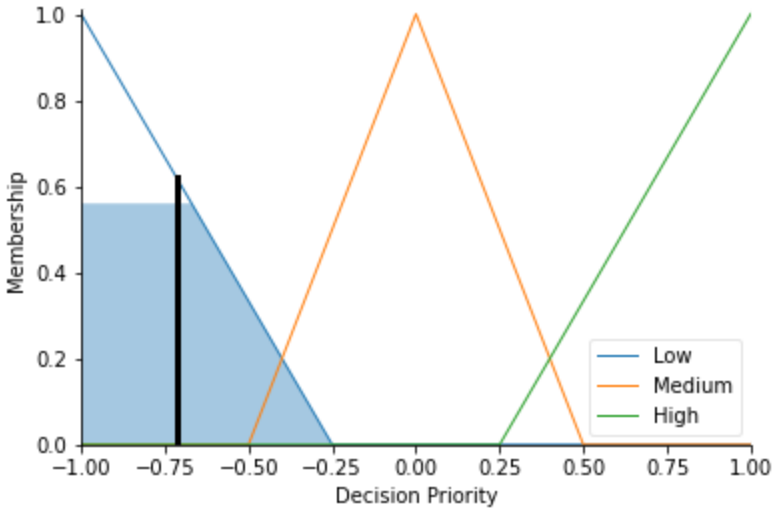
In [81]:
```
full_steps()
```

```
The decision id is: 15
Agent 0 votes for decision 15 with vote 61
Agent 1 votes for decision 15 with vote 88
Agent 2 votes for decision 15 with vote -53
Votes:  [0.61, 0.88, -0.53]
Decision Priority: 0.6896448087431692
```



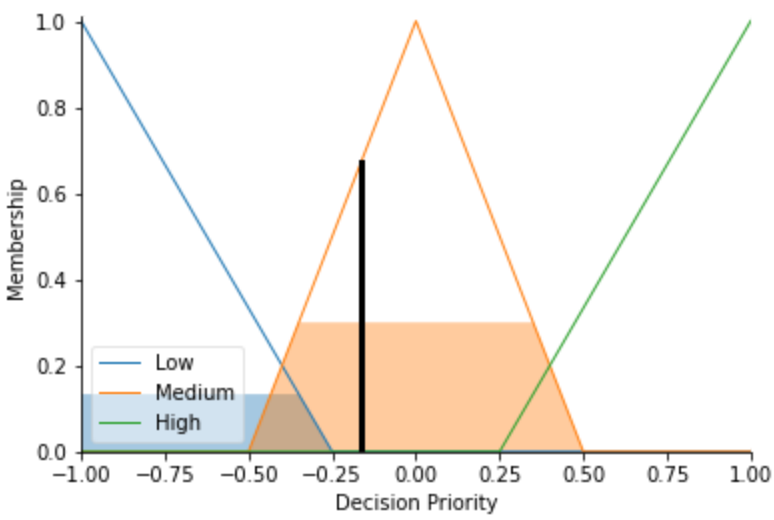
In [83]:
```
full_steps()
```

```
The decision id is: 17
Agent 0 votes for decision 17 with vote -12
Agent 1 votes for decision 17 with vote -67
Agent 2 votes for decision 17 with vote 10
Votes:  [-0.12, -0.67, 0.1]
Decision Priority: -0.7163888888888887
```



In [84]:
```
full_steps()
```

```
The decision id is: 18
Agent 0 votes for decision 18 with vote -35
Agent 1 votes for decision 18 with vote 67
Agent 2 votes for decision 18 with vote 20
Votes:  [-0.35, 0.67, 0.2]
Decision Priority: -0.16432160804020093
```


1.0
0.8
0.6
0.4
0.2
0.0
Membership
Low
Medium
High
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Decision Priority