Universidade Federal do ABC

Programação Orientada a Objeto

Prof: Carlos da Silva dos Santos

Projeto - Simulação de jogo de minoria

Nome/RA:

Felipe Augusto dos Santos, 11086610
Gabriel Silvestre Mancini, 11038214
Alex Key Araki, 11019914
Murillo Kerr de Melo, 21017211
Daniel Vital Mansano, 11026413

Agosto/2016

Descrição do Problema

O projeto se baseia na estudo de um jogo de minoria, que tem como objetivo fornecer dados baseados em um sistema similar a um mercado onde, dependendo de ações de compra ou de venda de agentes pertencentes às interações do mercado, o preço do bem negociado oscila e nos são fornecidas informações referentes à essa oscilação, como a relação preço/tempo e a sua variância. O problema gira em torno da definição de modelos de comportamento para os agentes que interagem com o mercado e no impacto que a interação de diferentes modelos causa na oscilação do valor do bem negociado em relação ao tempo.

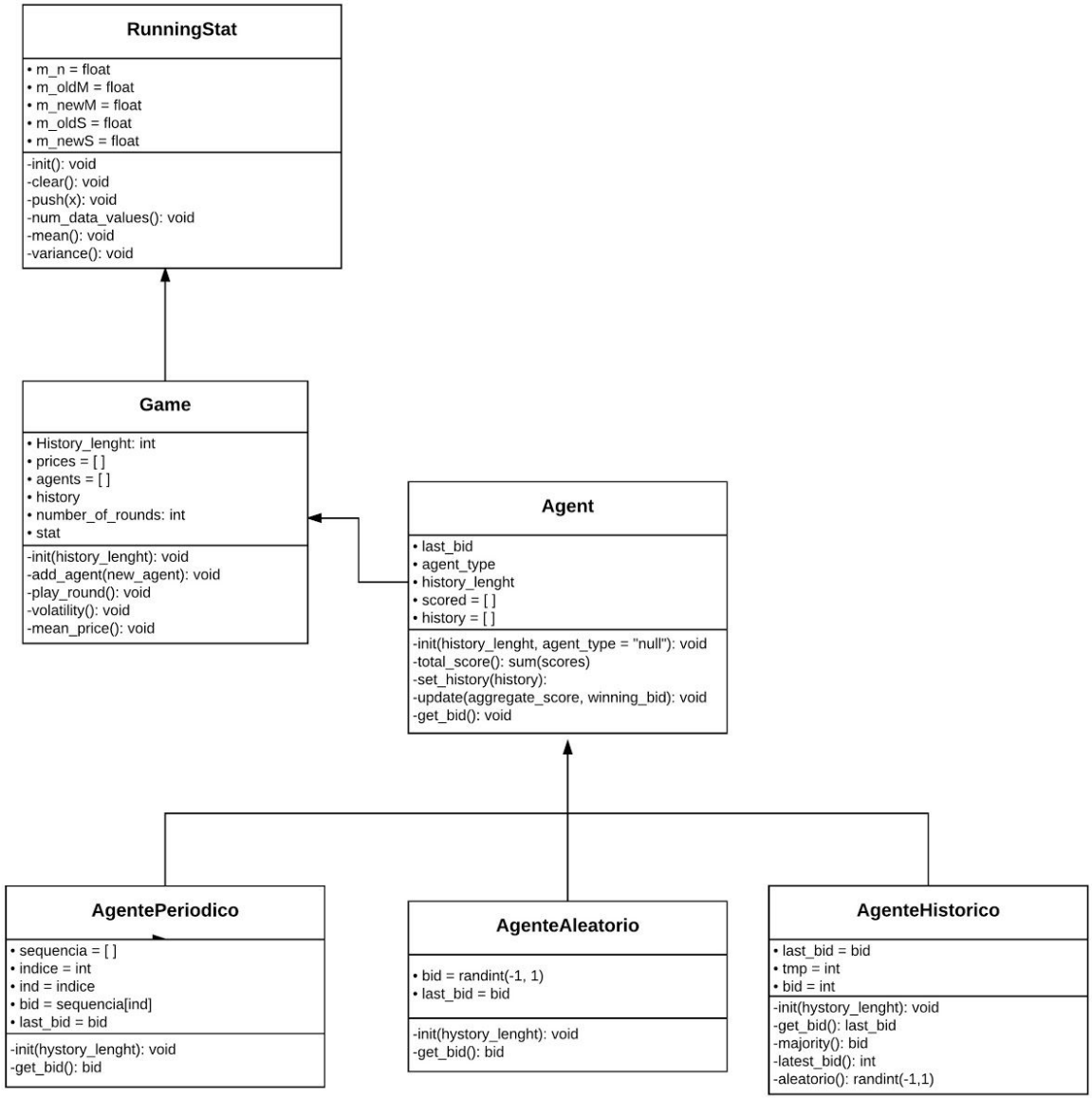
Programa

O objeto Game contém as variáveis que definem o jogo, como o histórico, as listas de preços e de agentes, o número de rounds e o comprimento do histórico e possui métodos que adicionam agentes ao jogo, criam as rodadas da simulação, retornam o valor da variância do preço e do preço médio.

O objeto Agent implementa os agentes que participam do jogo, traz as variáveis que carregam as ordens, o tipo dos agentes, a lista de scores e o histórico de ações; possui métodos que calculam o score, definem e atualizam o histórico de ordens e atualizam a ordem atual.

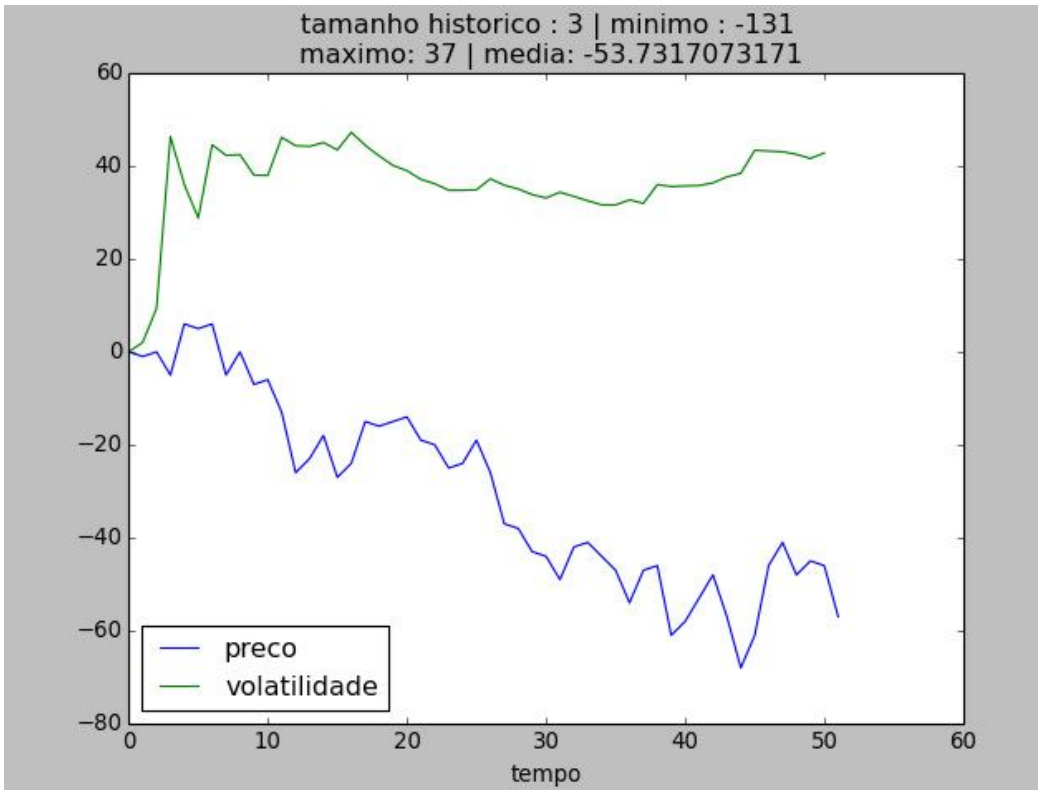
A classe RunningStat é responsável por calcular as médias e variâncias.

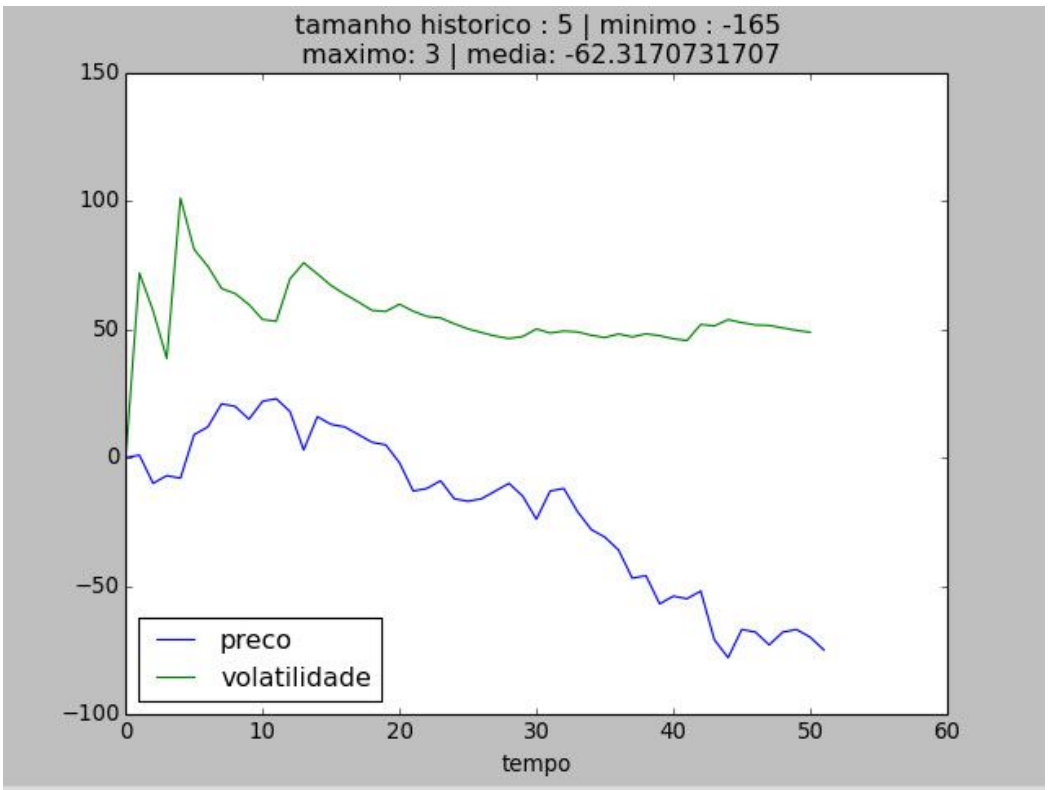
As classes dos Agentes definem o comportamento de cada um, a classe AgenteAleatorio descreve um agente que defina suas ações de compra ou de venda de forma aleatória, a classe AgentePeriodico descreve um agente que tem uma sequencia de ações que se repete, a classe AgenteHistorico descreve um agente que faz suas ações se baseando nas últimas ações realizadas.

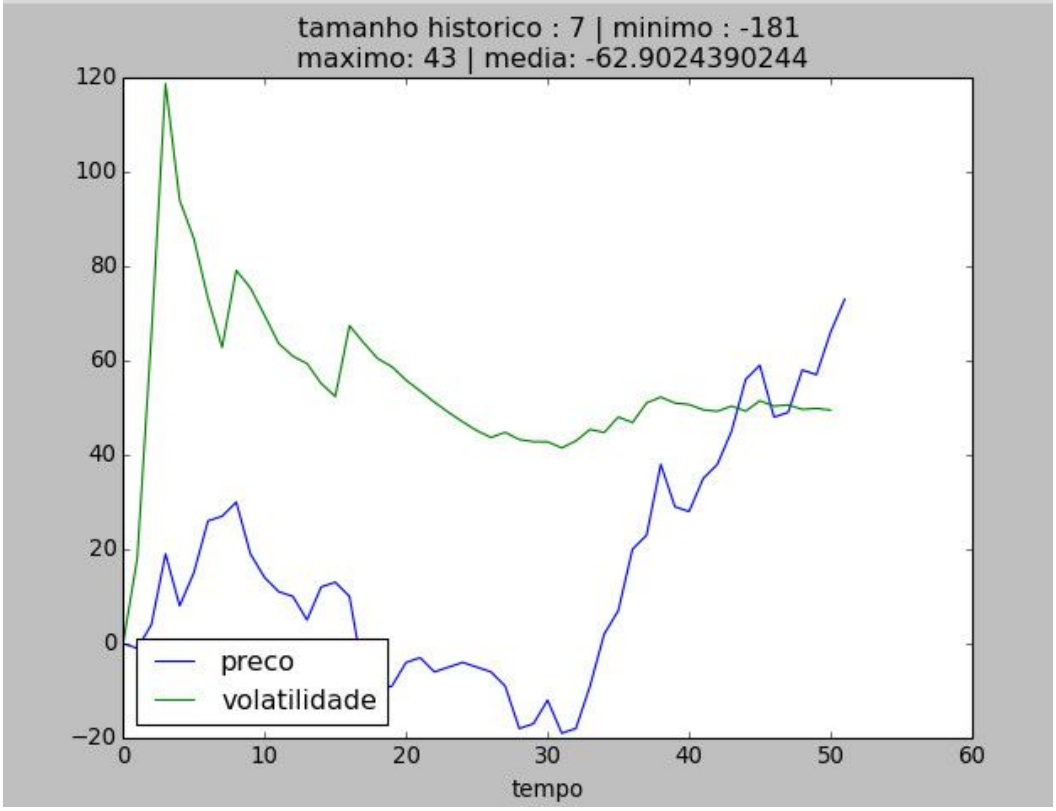


Apresentação dos resultados obtidos

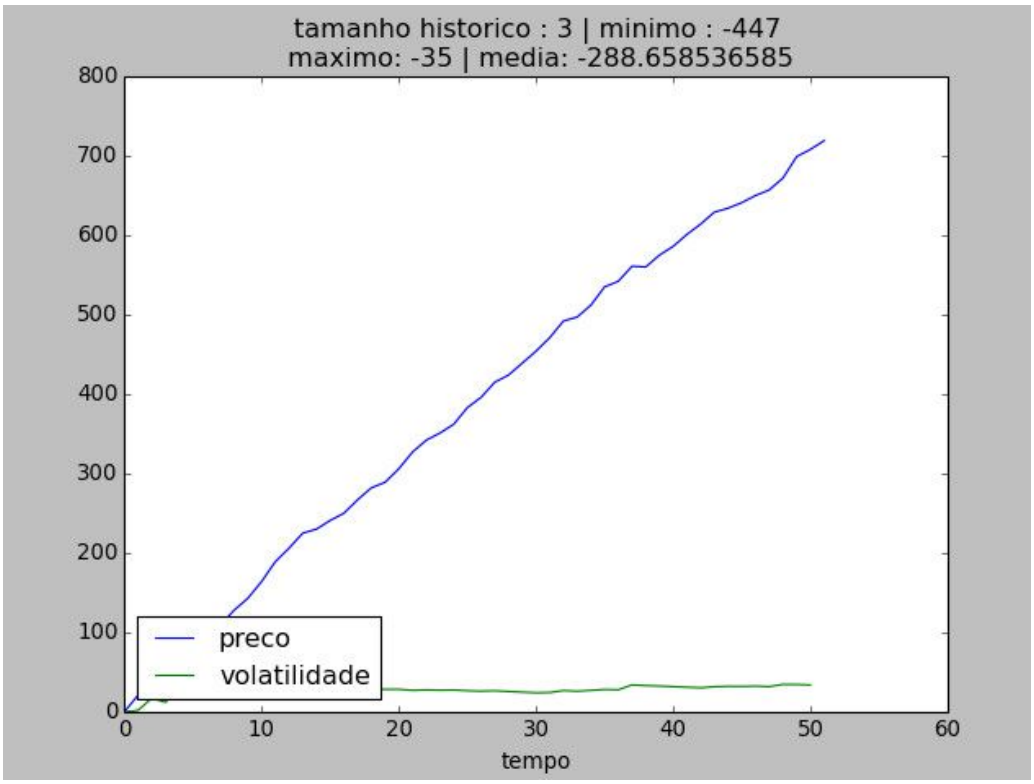
Abaixo estão os resultados das simulações efetuadas no mercado onde apenas agentes aleatórios participam, foram feitos três simulações com valores de históricos distintos. É possível ver no topo do gráfico o valor minimo, médio e máximo da recompensa em cada uma das simulações:

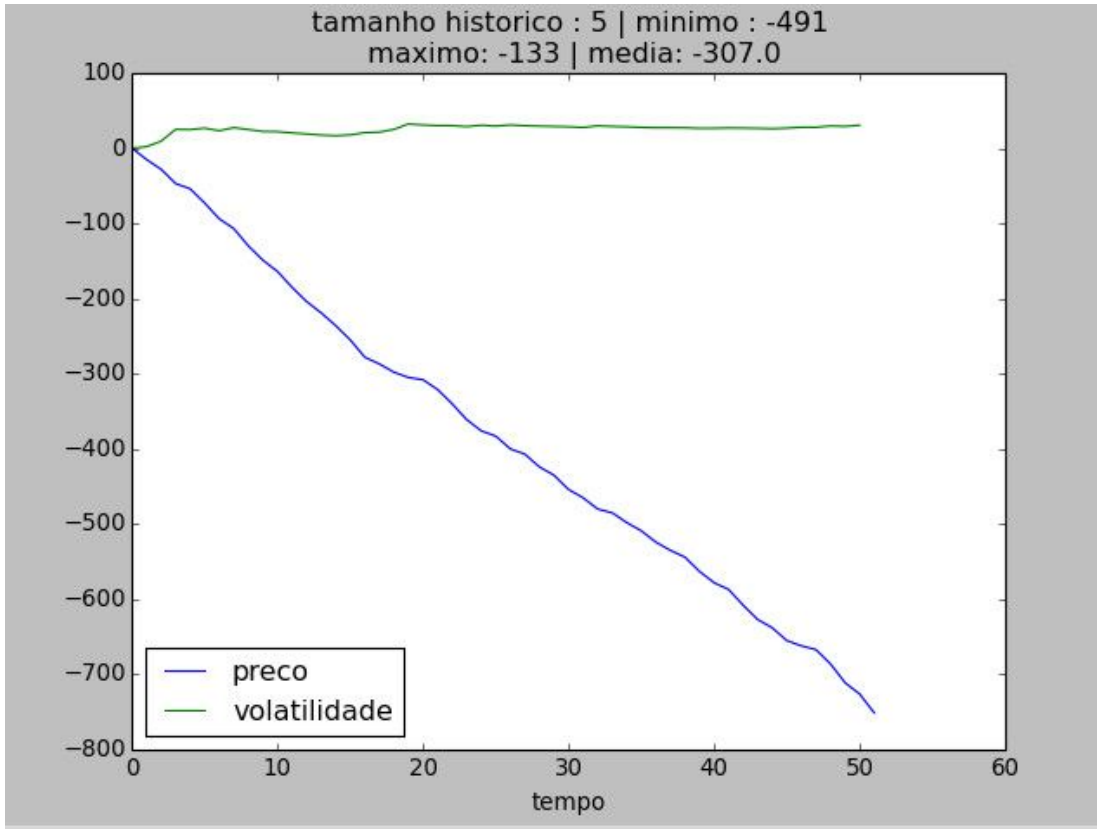

tamanho historico : 3 | minimo : -131
maximo: 37 | media: -53.7317073171
60
40
20
0
−20
−40
−60
−80
0
10
20
30
40
50
60
tempo
preco
volatilidade

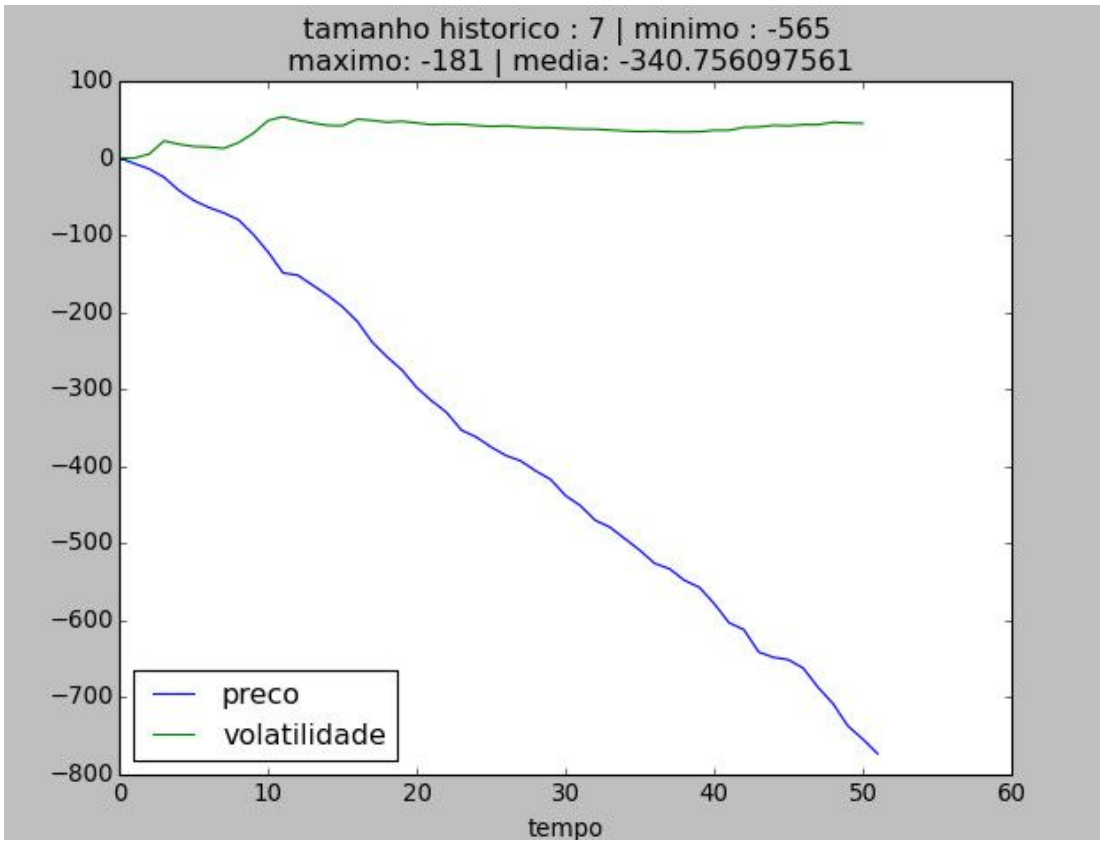

tamanho historico : 5 | minimo : -165
maximo: 3 | media: -62.3170731707
150
100
50
0
−50
−100
0
10
20
30
40
50
60
tempo
preco
volatilidade



O mesmo dito anteriormente vale para os gráficos que se seguem, com a exceção do fato que nesse mercado a simulação feita é com agentes que tomam decisões baseadas no histórico de compras e vendas:




tamanho historico : 5 | minimo : -491
maximo: -133 | media: -307.0
100
0
−100
−200
−300
−400
−500
−600
−700
−800
0
10
20
30
40
50
60
tempo
preco
volatilidade


tamanho historico : 7 | minimo : -565
maximo: -181 | media: -340.756097561
100
0
−100
−200
−300
−400
−500
−600
−700
−800
0
10
20
30
40
50
60
tempo
preco
volatilidade

Conclusões

Podemos concluir que o jogo de minoria é um bom modelo para mostrar como um mercado de ações funciona, seguindo as leis de oferta e demanda. Além disso, é possível perceber que agentes inteligentes, isto é, que empregam alguma estratégia que tomam decisões baseada nos histórico de transações teria uma vantagem competitiva em relação aos demais agentes. Houve uma dificuldade em gerar tal caso nos resultados, pois se todos os agentes empregarem a mesma estratégia, obviamente todos irão perder. Mesmo tentando variar as estratégias dos agentes inteligentes, tivemos pouco sucesso em replicar os resultado teóricos, talvez pela falta de variabilidade das estratégias com relação ao número de agentes.

Apêndice

```
AgentePeriodico(minoria.Agent):
    """Agente que joga periodicamente a mesma sequencia."""

    def get_bid(self):
    """
    Calcula ordem a ser realizada pelo agente na rodada atual.

    Devolve: A ordem baseada na rodada da sequencia fixa estipulada previamente.
    """
```

```
AgenteAleatorio(minoria.Agent):
    """Agente que joga aleatoriamente"""

    def get_bid(self):
    """
    Calcula ordem a ser realizada pelo agentena rodada atual.

    Devolve: Uma ordem aleatoria
    """
```

```
AgenteHistorico(minoria.Agent):
        """Agente que utiliza do historico para tomar decisoes"""

        def get_bid(self):
        """ Varia as estragerias, pois se todos os agentes utilizarem a mesma
estrategia,
        todos no final acabariam perdendo.

        Devolve: Uma ordem calculada por uma estrategia aleatoria
        """

        def majority(self):
        """ Estrategia que verifica se houve mais compras ou vendas na historia,
        se houve mais vendas o agente compra e se houve mais vendes o agente
compra.
        Se o numero for igual a decisao tomada e aleatoriamente
        """

        def latest_bid(self):

        """ Estrategia que verifica o ultimo bid da historia e faz exatamente a mesma
coisa """

        def aleatorio(self):
        """ Estrategia aleatoria """
```