Capítulo 1

T6 - Aprimoramento de uma Simulação: Laboratório e Experimento sobre Modelo de Simulação do Dilema do Prisioneiro, por João Antonio Desiderio de Moraes (joaoadm94)

1.1 Introdução

Este capítulo apresenta a construção e uso do laboratório de simulações do Dilema do Prisioneiro usando o framework MESA para a realização de experimentos. O objetivo é investigar a seguinte hipótese causal: "O prêmio de deserção influencia o estado de estabilidade de uma rede de agentes". Investiga-se o relacionamento entre as variáveis independentes na determinação da estratégia dominante na rede após um certo tempo. A inspiração para essa hipótese deriva diretamente dos estudos bibliométricos realizados no prévio estudo de bibliometria sobre o tema.

1.2 O Fenômeno do Mundo Real

De acordo com o verbete em (PRISONER'S. . . , 2022), o dilema do prisioneiro é um problema bastante conhecido no campo de teoria dos jogos e originalmente discutido por Merrill Flood e Melvin Dresher enquanto trabalhavam na RAND Corporation. O dilema do prisioneiro tem uma apresentação narrativa clássica de onde deriva seu nome. Aqui optou-se por uma abstração dos detalhes narrativos em favor do modelo de interação mais central. Nele, dois participantes devem optar entre cooperar ou não para definir o resultado de sua interação. Os cenários resultantes possíveis são:

1. Ambos resolvem cooperar;

2. Ambos escolhem divergir;

3. As estratégias diferem entre si - um decide cooperar e o outro não.

O formato do problema visa investigar o comportamento de participantes guiados pelos próprios interesses em situações de conflito. Devido a sua temática de cooperação, foi aplicado na modelagem de diversos fenômenos nos campos da política e economia. A própria RAND Corporation, iniciativa onde o experimento foi desenvolvido, estudava teoria de jogos visando possíveis aplicações a estratégia nuclear global. Segundo (AMADAE, 2016) algumas problemáticas já foram estudadas com apoio do modelo de relações do dilema do prisioneiro: controle de armas, trocas de mercado, utilização de bens públicos, mudança climática, vacinação e várias outras.

1.3 O Laboratório

Neste laboratório queremos entender o comportamento de agentes simulados em um ambiente de conflito. Os experimentos envolvem a alteração controlada dos diversos condicionantes dos agentes para observar as tendências de cooperação.

Segundo a página GitHub da simulação, esse exemplo demonstra como a cooperação se torna dominante mesmo em cenários com vários agentes inicialmente não cooperantes e com um prêmio significativo pela inércia desses agentes. De fato, a simulação executada com as configurações padrão tende a um cenário de breve dominância da estratégia não cooperativa seguida de uma transição gradual até surgir uma predominância sólida da estratégia cooperativa.

Interessa entender a relação entre as recompensas programadas para as interações com o comportamento total do sistema. O laboratório propõe tornar essa característica facilmente manipulável para observamos os resultados de cada cenário possível.

1.3.1 O Simulador

O framework MESA é um sistema útil para modelagem e simulação do comportamento de agentes. É baseado em python e dotado de uma interface visual servida via navegador web. Seu pacote básico possui uma demonstração nativa do dilema do prisioneiro chamada PD_Grid.

A simulação consiste originalmente de uma matriz de agentes 50x50. A configuração inicial de cada agente pode ser de cooperação ou de traição. Os agente interagem entre si em rodadas. As interações se dão na ordem realizada pelo escalonador nos seguintes modos: Sequencial, Randômico e Simultâneo. A pontuação de cada agente em uma rodada depende das estratégias de seus vizinhos. Após cada rodada, o agente adotará a estratégia do seu

vizinho com maior pontuação a cada iteração. Essa configuração deverá nos permitir estudar qual estratégia rende mais pontuação aos sistema como um todo.

Neste laboratório, adicionamos dois elementos a mais no simulador. Um deles é um seletor de números controlando o valor do prêmio por deserção. Podemos definir um valor de 0 a 20; este será dividido por 10 para representar o novo prêmio.

O segundo elemento adicionado é uma visualização de dados que mostra os totais de agentes cooperativos e não cooperativos em forma de gráfico.

1.3.1.1 Variáveis Independentes ou de Controle

A simulação apresenta as seguintes variáveis independentes:

- as dimensões do campo de agentes em termos de largura e altura;
- o modo de escalonamento dos agentes;
- os pesos para cada tipo de interação.

São dependentes deles as variáveis:

- a pontuação de cada agente;
- a última estratégia adotada pelo agente;
- a pontuação conjunta do sistema;
- a a porcentagem de agentes cooperativos;

1.3.2 A Hipótese Causal Inicial

Em um primeiro momento será desafiada a hipótese "O prêmio de deserção influencia o estado de estabilidade de uma rede de agentes". O intuito aqui é observar a influência de apenas um fator no resultado total do sistema. Espera-se que com um menor prêmio para a deserção tenhamos um cenário de cooperação predominante desenvolvido rapidamente. Da mesma forma, um maior prêmio deve estimular mais rapidamente a deserção como estratégia predominantes.

1.3.3 O Código do Simulador

O código abaixo implementa as mudanças mencionadas em 1.3.1.

O arquivo contendo as definições do agente não foi alterado de forma significativa nesta etapa. Serviu apenas para informar o uso dos outros módulos. Ele contém parâmetros como sua posição na rede, sua pontuação e estratégia atual. Possui funções para retornar a estratégia atual e também para decidir a nova estratégia adotada.

Listagem de Código 1.1: Código do agente.

```
import mesa


class PDAgent(mesa.Agent):
    """Agent member of the iterated, spatial prisoner's dilemma model."""

    def __init__(self, pos, model, starting_move=None):
        """
        Create a new Prisoner's Dilemma agent.

        Args:
            pos: (x, y) tuple of the agent's position.
            model: model instance
            starting_move: If provided, determines the agent's initial state:
                           C(ooperating) or D(efecting). Otherwise, random.
        """
        super().__init__(pos, model)
        self.pos = pos
        self.score = 0
        if starting_move:
            self.move = starting_move
        else:
            self.move = self.random.choice(["C", "D"])
        self.next_move = None

    @property
    def isCooperating(self):
        return self.move == "C"

    def step(self):
        """Get the best neighbor's move, and change own move accordingly if better than own score."""
        neighbors = self.model.grid.get_neighbors(self.pos, True, include_center=True)
        best_neighbor = max(neighbors, key=lambda a: a.score)
        self.next_move = best_neighbor.move

        if self.model.schedule_type != "Simultaneous":
            self.advance()

    def advance(self):
        self.move = self.next_move
        self.score += self.increment_score()

    def increment_score(self):
        neighbors = self.model.grid.get_neighbors(self.pos, True)
        if self.model.schedule_type == "Simultaneous":
            moves = [neighbor.next_move for neighbor in neighbors]
        else:
            moves = [neighbor.move for neighbor in neighbors]
        return sum(self.model.payoff[(self.move, move)] for move in moves)
```

O código do servidor contém as configurações da camada de apresentação via web. O arquivo foi alterado para conter a interface dos novos elementos adicionados. São eles: o novo parâmetro "defect_multiplier"e os elementos "chart_population"e "chart_score".

Listagem de Código 1.2: Código do servidor MESA.

```
import mesa

from .portrayal import portrayPDAgent
from .model import PdGrid


# Make a world that is 50x50, on a 500x500 display.
canvas_element = mesa.visualization.CanvasGrid(portrayPDAgent, 50, 50, 500, 500)

model_params = {
    "height": 50,
    "width": 50,
    "schedule_type": mesa.visualization.Choice(
        "Scheduler type",
        value="Random",
        choices=list(PdGrid.schedule_types.keys()),
    ),
    "defect_multiplier": mesa.visualization.Slider(
        "Defect multiplier",
        16,
        0,
        20,
        description="Multiplier of the defection payoff"
    ),
```

```
    #"cooperation_start": mesa.visualization.Slider(
    #    "Cooperating at start",
    #    50,
    #    0,
    #    100,
    #    description="Percent of agents cooperating at start"),
}

# map data to chart in the ChartModule
chart_population = mesa.visualization.ChartModule(
    [
        {"Label": "Cooperating", "Color": "blue"},
        {"Label": "Defecting", "Color": "red"},
    ]
)

# map data to chart in the ChartModule
chart_score = mesa.visualization.ChartModule(
    [
        {"Label": "Score", "Color": "black"},
    ]
)

server = mesa.visualization.ModularServer(
    PdGrid, [canvas_element, chart_population, chart_score], "Prisoner's Dilemma", model_params
)
```

Por último, apresentamos o modelo de dados com as alterações necessárias para interagir com as novas funcionalidades. Inclui-se nova variável, "defect_multiplier", para receber o valor definido na interface é aplicá-lo ao sistema. Coletores de dados foram adicionados para saber o comportamento de alguns valores ao longo dos passos. Os coletores rastreiam o total de agentes cooperantes, o total de não cooperantes, assim como a pontuação total do sistema.

Listagem de Código 1.3: Código geral do modelo.

```
import mesa

from .agent import PDAgent

def get_num_coop_agents(model):
    """return number of cooperating agents"""

    coop_agents = [a for a in model.schedule.agents if a.isCooperating(a)]
    return len(coop_agents)


def get_num_def_agents(model):
    """return number of defecting agents"""

    def_agents = [a for a in model.schedule.agents if not a.isCooperating(a)]
    return len(def_agents)

def get_total_grid_score(model):
    """return sum of scores from every agent on the grid"""

    total_score = 0
    for a in model.schedule.agents:
        total_score = total_score + a.score
    return total_score


class PdGrid(mesa.Model):
    """Model class for iterated, spatial prisoner's dilemma model."""

    schedule_types = {
        "Sequential": mesa.time.BaseScheduler,
        "Random": mesa.time.RandomActivation,
        "Simultaneous": mesa.time.SimultaneousActivation,
    }

    # This dictionary holds the payoff for this agent,
    # keyed on: (my_move, other_move)

    payoff = {("C", "C"): 1, ("C", "D"): 0, ("D", "C"): 1.6, ("D", "D"): 0}

    starting_payoff_dc = payoff[('D', 'C')]

    def __init__(
        self, defect_multiplier, width=50, height=50, schedule_type="Random", payoffs=None, seed=None
```

```
    ):
        """
        Create a new Spatial Prisoners' Dilemma Model.

        Args:
            width, height: Grid size. There will be one agent per grid cell.
            schedule_type: Can be "Sequential", "Random", or "Simultaneous".
                           Determines the agent activation regime.
            payoffs: (optional) Dictionary of (move, neighbor_move) payoffs.
        """
        self.grid = mesa.space.SingleGrid(width, height, torus=True)
        self.schedule_type = schedule_type
        self.schedule = self.schedule_types[self.schedule_type](self)
        self.payoff[('D', 'C')] = defect_multiplier/10

        # Create agents
        for x in range(width):
            for y in range(height):
                agent = PDAgent((x, y), self)
                self.grid.place_agent(agent, (x, y))
                self.schedule.add(agent)

        self.datacollector = mesa.DataCollector(
            {
                "Cooperating": lambda m: len(
                    [a for a in m.schedule.agents if a.move == "C"]
                ),
                "Defecting": lambda m: len(
                    [a for a in m.schedule.agents if a.move == "D"]
                ),
                "Score": get_total_grid_score,
            }
        )

        self.running = True
        self.datacollector.collect(self)

    def step(self):
        self.schedule.step()
        # collect data
        self.datacollector.collect(self)

    def run(self, n):
        """Run the model for n steps."""
        for _ in range(n):
            self.step()
```

1.4 Os Experimentos Realizados

Foram necessários cerca de 100 experimentos curtos para depurar o código e obter uma alteração satisfatória no sistema. O objetivo era conferir se as alterações feitas na interface web realmente impactavam o sistema, e começar a perceber a dinâmica emergente das alterações.

A trajetória das simulações usando configurações originais serviu como um grupo de controle. Ela é caracterizada por uma dominância dos não cooperantes no primeiros 5 a 10 passos, após os quais a estratégia cooperativa passa a predominar.

A depuração foi encerrada quando percebemos uma proporcionalidade entre a variação no número de passos e a variação no prêmio por deserção.

A simulação demonstrou uma predominância da estratégia competitiva quando aumentado o prêmio, assim como uma tomada pela estratégia cooperativa quando o prêmio era diminuído. Os experimentos deram origem a uma intuição de que a hipótese proposta reflete uma real relação causal entre as variáveis independentes e dependentes. Entretanto, a simplicidade da hipótese realçou a possibilidade de explorarmos questões mais complexas com o simulador.

Abaixo, veja três momentos da simulação funcionando com o prêmio para deserção em seu valor padrão.

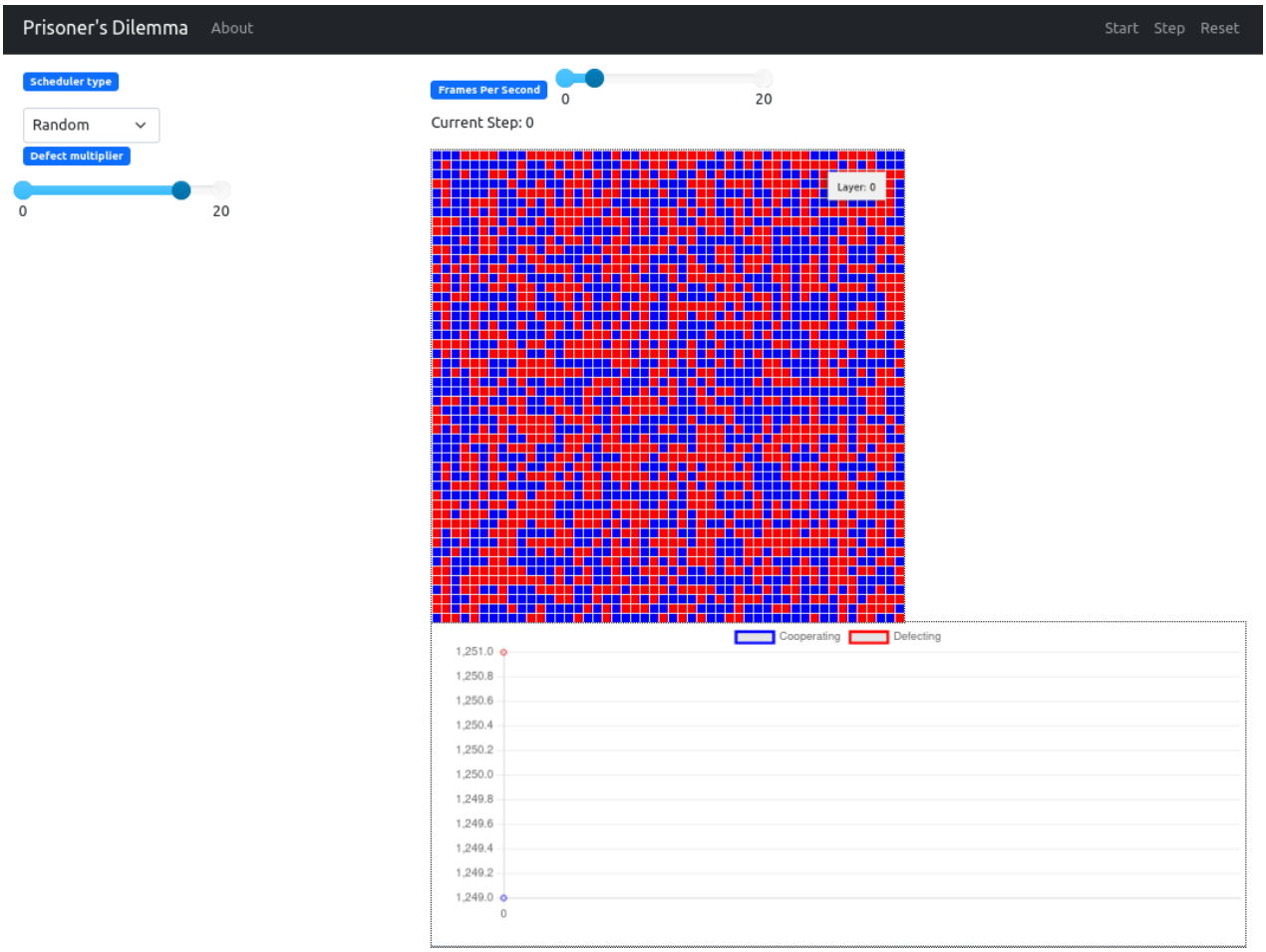

Figura 1.1: Simulação no passo 0.

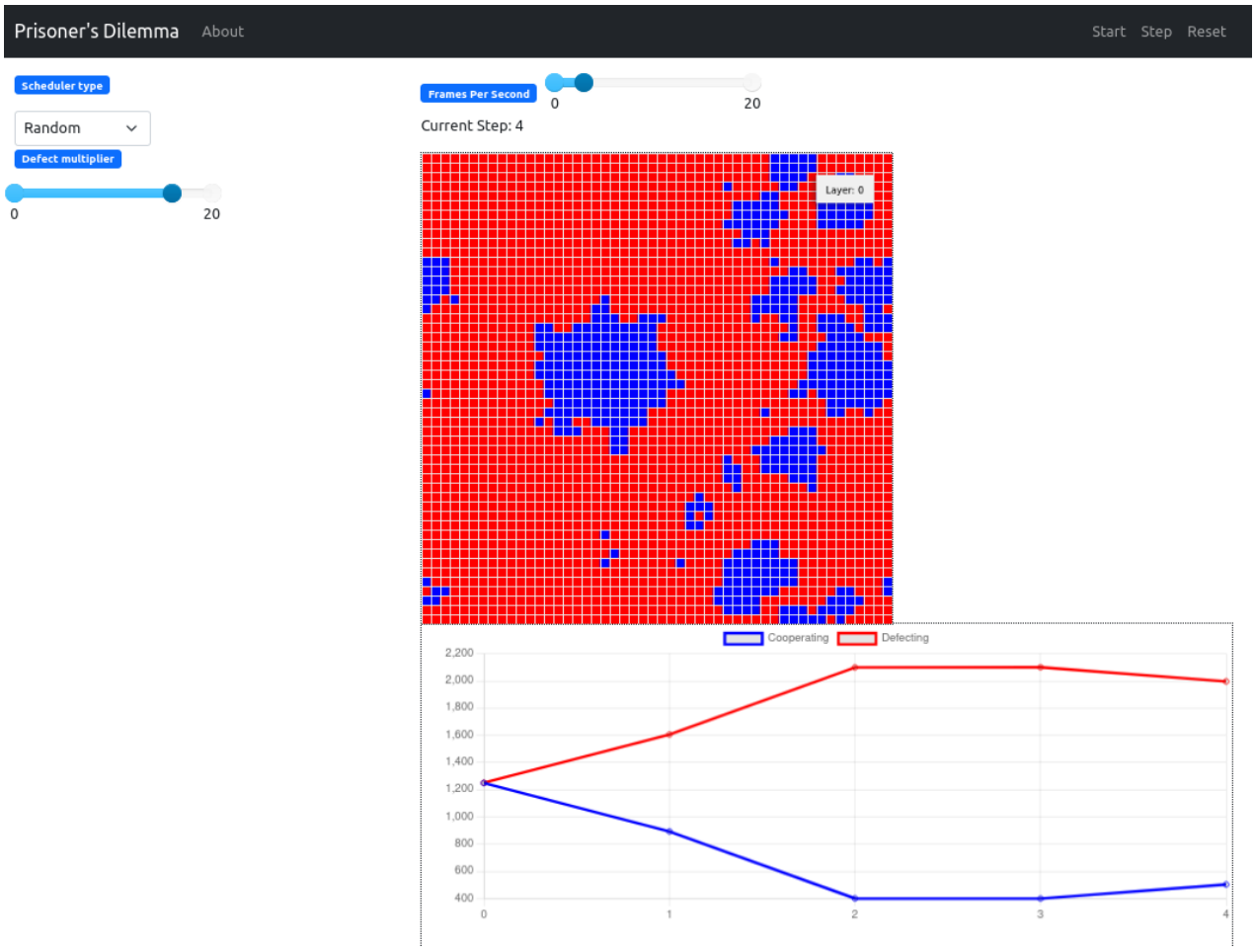

Figura 1.2: Simulação no passo 4.

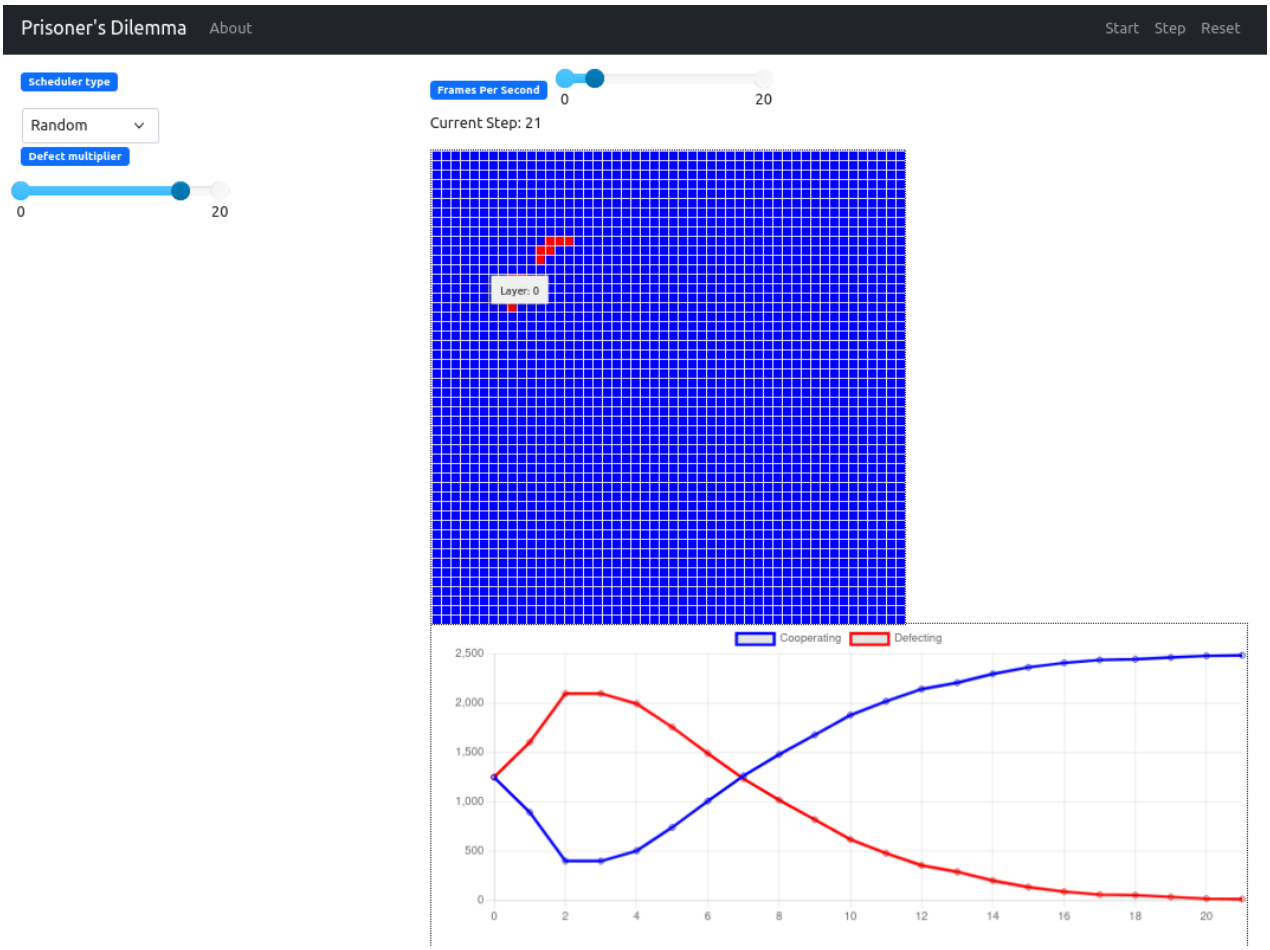

Figura 1.3: Simulação no passo 21

1.5 Conclusão

O exercício foi bastante proveitoso para melhorar o entendimento sobre as diversas partes que compõem o simulador: o framework MESA, o código do modelo em si, as implementações específicas das variáveis existentes e suas interfaces. Afinal, foi necessário codificar corretamente novas funcionalidades no modelo já existente.

Sobre a hipótese causal, intuimos no sentido de confirmá-la porém também despertou interesse na busca de questões mais profundas sobre o sistema. Superadas as dificuldades com a codificação em si, notou-se a simplicidade ou quase obviedade da hipótese causal. Para tarefas futuras fica o desafio de elaborar hipóteses mais desafiadoras que permitam elucidar relações mais sutis entre as variáveis.

Nesse primeiro momento, os testes foram bastante rudimentares, sendo realizados à mão ou por meio de impressões do código. Nenhuma capacidade mais poderosa de análise de dados foi utilizada ainda. Certamente será possível obter conhecimentos mais interessantes a partir de futuras análises estatísticas. Para obter esse resultado, o simulador será utilizado para produzir dados em grande quantidade via execução de simulações em batch e os dados resultantes poderão ser analisados com auxílio de outras ferramentas.

Bibliografia

AMADAE, S. M. *Prisoners of reason: game theory and neoliberal political economy*. New York, NY: Cambridge University Press, 2016. ISBN 978-1-107-06403-4 978-1-107-67119-5. Citado na p. 2.

PRISONER'S Dilemma. Disponível em: <https://en.wikipedia.org/wiki/Prisoner%27s_dilemma>. Acesso em: 3 dez. 2022. Citado na p. 1.