Simulating an isolated microgrid powered economy

Creating a peer-to-peer economy by agent based modelling

By Spurrya Jaggi

“We're leading a fundamental shift from centralized energy to distributed energy. Energy will go in that direction, just like mainframe computers went to client servers, then to the Internet.” – Lynn Jurich

Table of Contents

1. Problem

The global energy crisis has severe implications on densely populated nations such as India. 3 billion people globally still rely on kerosene and solid fuels for cooking and heating, and over 1 billion lack access to electricity (Foundation, 2018). People living here lack access to clean, affordable, and reliable energy. Since many of these communities are not recognized by the Indian government, they do not receive any support or services and are left to fend for themselves.

Households lacking access to electricity are forced to rely on biomass or kerosene for cooking and lighting. Their toxic combustion puts families, especially women and children, at a much higher risk of deadly respiratory diseases, leading to 4.3 million premature deaths every year. (WHO, 2018)

The inability to regularly use electrical appliances prevents member of these communities from adopting a modern lifestyle, thus putting them at an economic disadvantage. Time and money spent on biomass and kerosene also deprives people of educational and employment opportunities. In off-grid communities where the average income is 100 Rupees a day, a liter of kerosene costs 75 Rupees (Sen, 2016). This is a significant financial burden to those who do not have electricity access. This limits community members upward mobility and puts them in a perpetual debt cycle.

2. Microgrid as a solution

It is costly for the government to provide grid extensions that provide electricity to these off grid communities. In such situations, microgrids can aid in providing electricity to these communities while allowing an opportunity for producers of electricity in these microgrids to have an alternative source of income and reducing the health-related problems caused by alternatives fuels such as kerosene. Despite massive electrification plans, India still has 54,000 of its isolated communities electrified. (Sen, 2016)

A DC microgrid (capacity less than 1500 W) is low voltage and can be useful to power electric bulbs and can charge phone without requiring a DC-to-AC convertor. Since the solar panel produces electricity in DC, it makes it cheap for the consumer to buy a basic solar panel and share electricity with their neighbours in close vicinity. It is environmentally friend, provides a reliable source of electricity which is economical and self-sustaining.

3. Simulation

An agent based model was created to simulate how the economy of an isolated un-electrified community will be developing by creating a micro-grid.

Purpose of report**:** It is to find how micro-grids behave, their development and, how the behavior and saving of the producers and consumers change based on the cost at which producers sell their electricity to the consumers. The aim of the simulation is to maximize the number of people who have electricity as soon as possible and minimize the standard deviation within the economy.

Input Parameters**:**

The following table shows parameters used in developing the simulation based of a case study conducted in a rural village of India described in the paper "Solar DC Microgrid for rural electrification - a case study" (Sen, 2016). The price of the solar panel and estimated output was estimated by consulting a business in India, Pico Energy.

Parameters	Units
Hourly electricity needs per villager	6 W-hr
Hours of electricity required	5 hours
Daily electricity needs per villager	30 W-hr/daily
Daily Income	100 Rupees
Cost of 1 W-hr of electricity by producer	5 Rupees
Cost of alternative fuels such as Kerosene	75 Rupees
Cost of 1 unit of solar panel and battery storage sold by Pico Energy	1500 Rupees
Output of 1 unit of solar panel	60 W- hr

Based on the above parameters, the parameters for the simulations were taken. They were scaled down by a factor of 10.

Following are additional parameters in the model for a more realistic model:

Randomness in the model

Many of them allowed the simulation to have variance in the data as all the above statistics are the average. Different agents have different income, different energy requirements, different expenditures and lastly, since the micro-grid is being developed on solar energy, the energy produced daily will not be the same. Hence, a small variance is all the parameters above.

Probability in the model

Incentivizing an agent to be a producer by increasing the returns on 1 unit of electricity (= 1 W-hr). They are more likely to want to be a producer as there is a significant upfront cost on the solar panel.

The probability of buying kerosene if their daily needs are not satisfied is lowered to allow the agent from saving in order to potentially buy a solar panel. If the probability of buying kerosene to fulfill daily energy requirements is not lowered, the agent will be in a perpetual downward cycle.

The agents are more incentivized to produce more electricity if their own electricity needs or their neighbors are not being met. The producer has an option to buy more solar panels or the neighbors can choose the become a producer.

Simulation Parameters:

Based on the above background research, the following parameters were picked while simulating.

```
    "width":10,
    "height":10,
    "consumption_of_energy_mean": 3,
    "consumption_of_energy_std":0.1,
    "daily_production_of_energy_mean": 6,
    "daily_production_of_energy_std": 1,
    "daily_outcome_mean": 2,
    "daily_outcome_std":0.1,
    "daily_income_mean":10,
    "daily_income_std":0.5,
    "price_of_alternative_fuels":7.5,
    "price_of_solar_panel":150,
    "price_of_electricity_from_producer": 0.5,
    "probability_of_converting_into_producer":0.15,
    "probability_of_neighbour_converting_into_producer":0.05
```

Since the probability of converting into a producer depends on the amount of money the producer receives, it is taken into account as following:

```
parameters["probability_of_converting_into_producer"] =
parameters["probability_of_converting_into_producer"] *
parameters["price_of_electricity_from_producer"]/ 100
parameters["probability_of_neighbour_converting_into_producer"] =
parameters["probability_of_neighbour_converting_into_producer"] *
parameters["price_of_electricity_from_producer"] / 100
```

Behavior of each agent:

1. All the agents are provided an income based on the parameter: "daily_income_mean" and to for deviation in the model the parameter: "daily_income_std" is used.
2. After the income, the net savings for the day is calculated after taking out other expenses (does not include electricity cost).
3. All the producers presented on the grid produce electricity based on the day. A day can be cloudy which will reduce the electricity produced or sunny which will increase it.
4. The producer then consumes the energy for himself before selling to his neighbour.
5. The producer trade electricity with all the neighbour nearby (radius of 2) till there is no more electricity needs or the neighbour does not have enough money to buy more or, the neighbours energy needs have been reached.
6. If an agent's daily energy needs were not met and they have enough money for today and savings, the agent can purchase kerosene. However, since the cost of kerosene is high and the agents are incentivized to save in the simulation, the agent may not be able to get all their energy needs fulfilled for the day.
7. The savings of all the agents is updated based on the day's savings
8. All agents are given a chance to convert into producer given they have enough savings to afford a solar panel and enough incentives, they will convert into a producer making all their neighbours consumers.

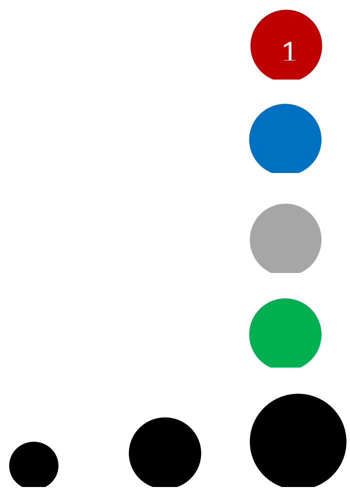
Incentive Based Programs:

Incentive-based programs try to incentive a neighbour to come a producer to contribute towards the microgrid. This can usually be in form of a payment or reward. For the purpose of this project, the price of alternative fuels such as Kerosene was high while the cost of consuming from a producer was low.

Incentives Provided: If an agent owns enough money to buy a PV solar cell and, there is at least one neighbor who does not have their daily energy needs, they are incentives to become a producer. However, if everyone's energy needs in the surrounding were satisfied, the probability of everyone wanting to converting to producer decreases.

4. Results

Understanding the simulation



A producer who owns 1 solar panel

A consumer

An alternative fuels user such as Kerosene

Agents whose energy needs for the day have been fulfilled.

The size of the agent represent agent's savings

In the beginning, all the agents are randomly generated each with the parameters specified above as shown below in Figure 1

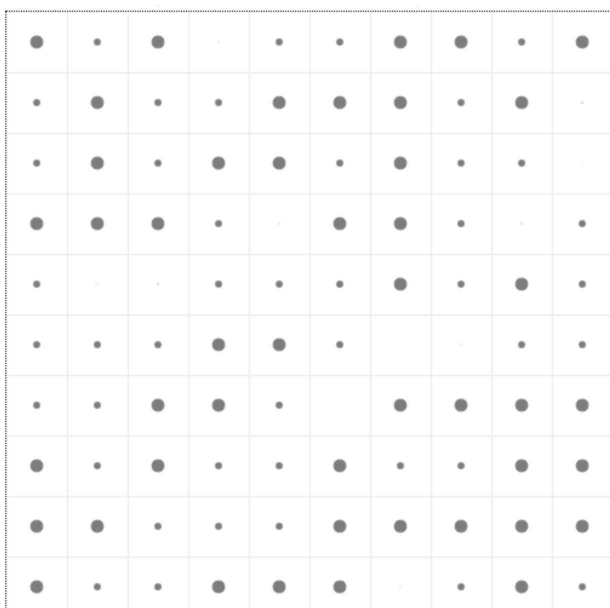


Figure 1: Random initiation of agents with different savings

Some of the agents used their savings for a temporary solution and decided to buy kerosene as shown below in Figure 2.

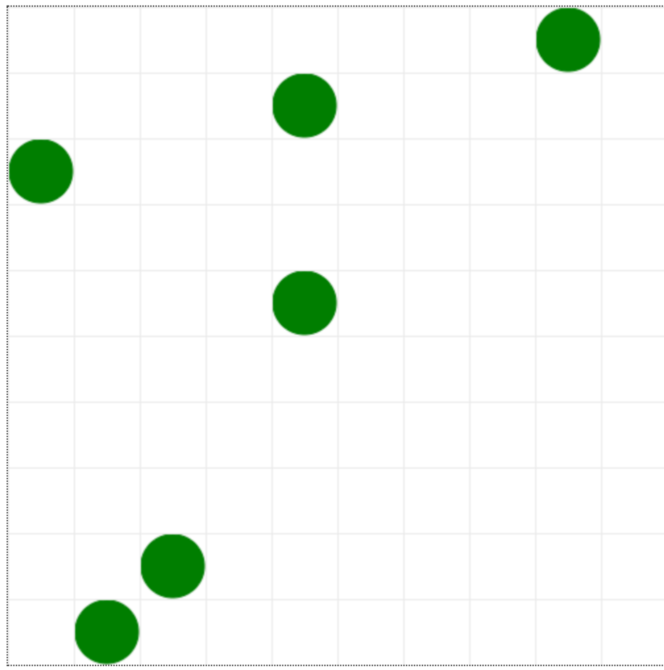
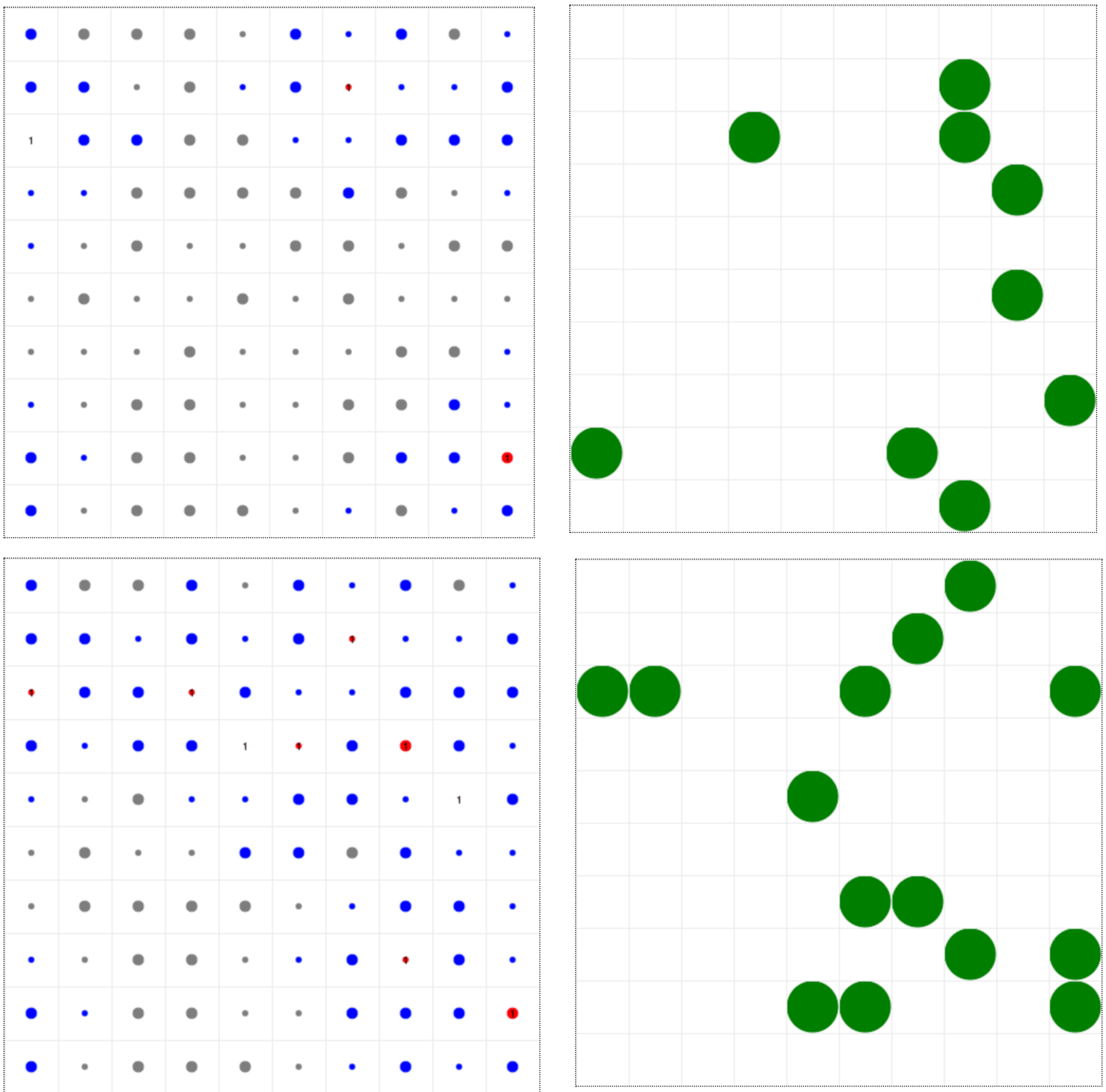


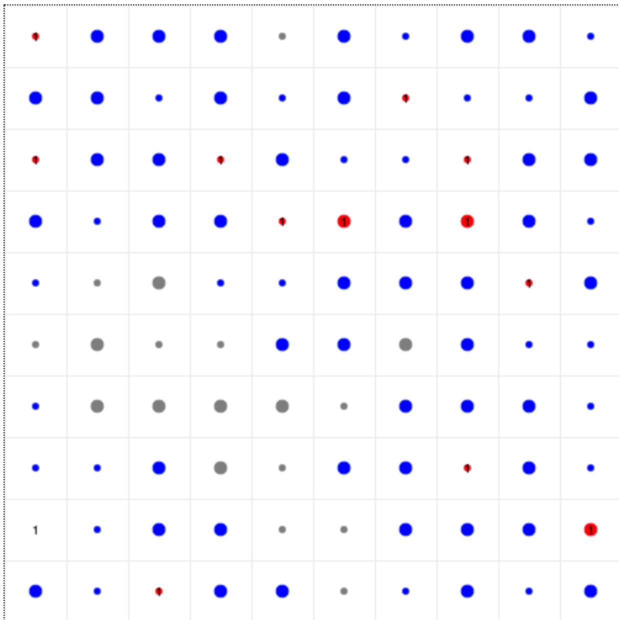
Figure 2: Some agents brought kerosene and satisfied their energy requirements for the day

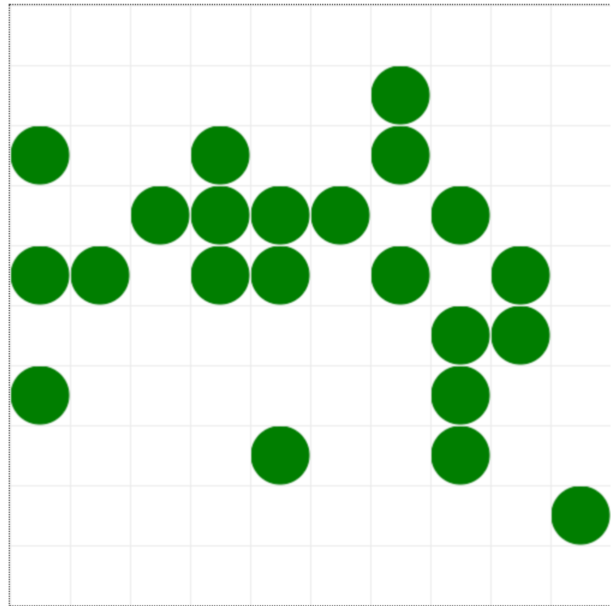
When the production of electricity is low, all the neighbours of the producer are not guaranteed all units of electricity as shown in the next sequence of iterations. Furthermore,

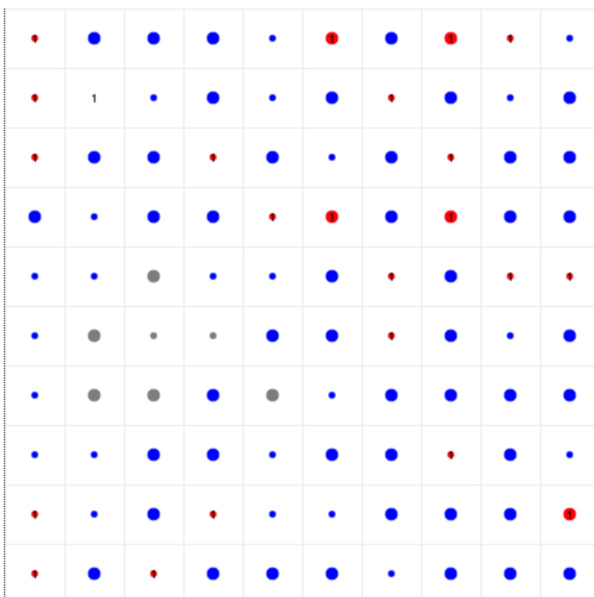
Price Point: 0.5 units

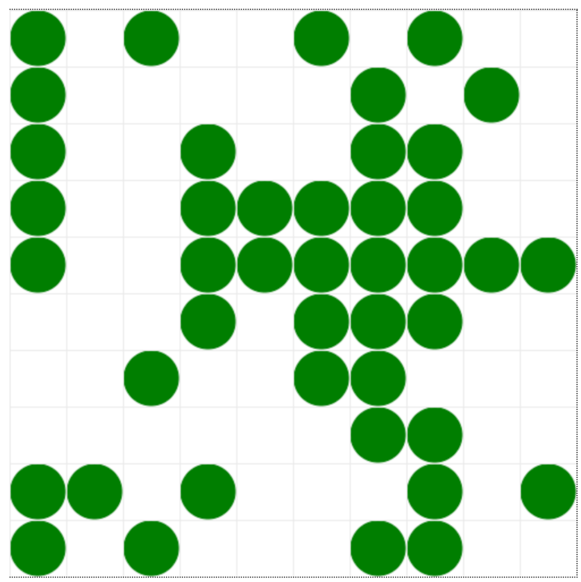
The following are the next few steps of the simulation when price of electricity to consume is **0.5 units** or 5 rupees:

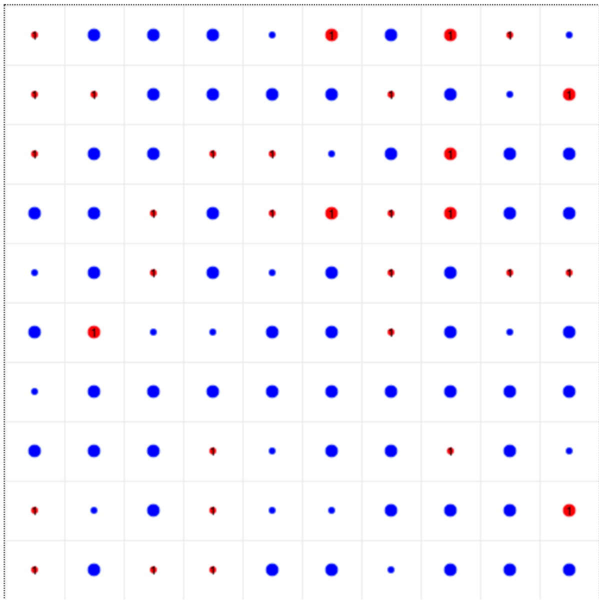


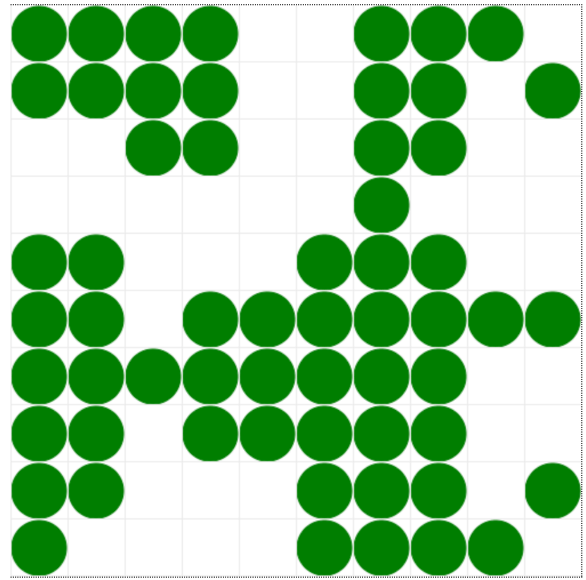


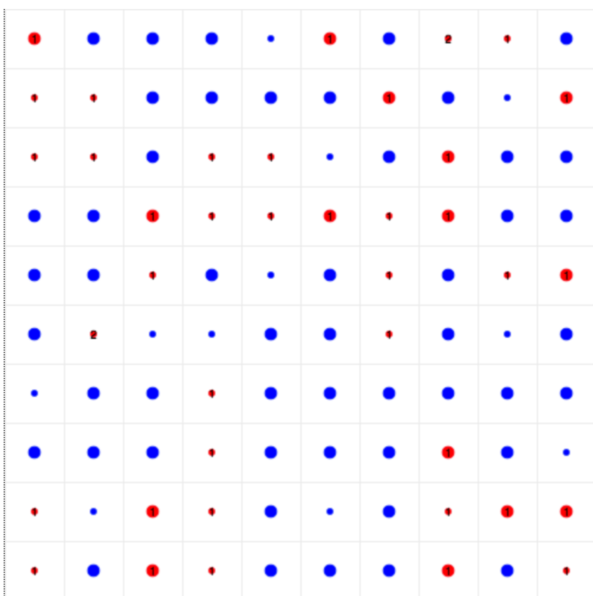


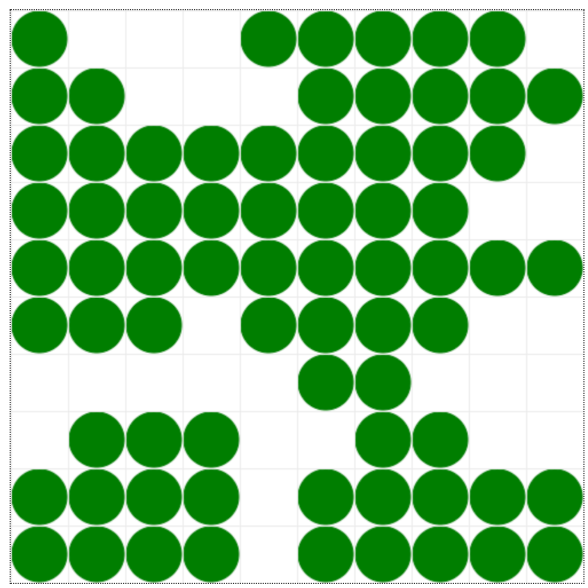


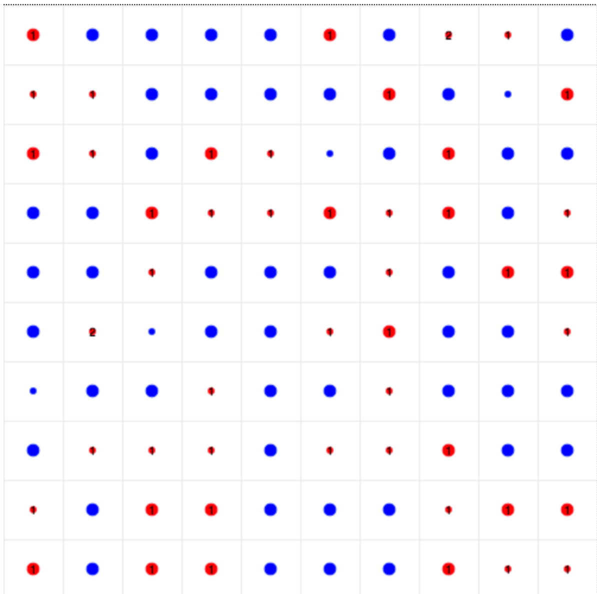


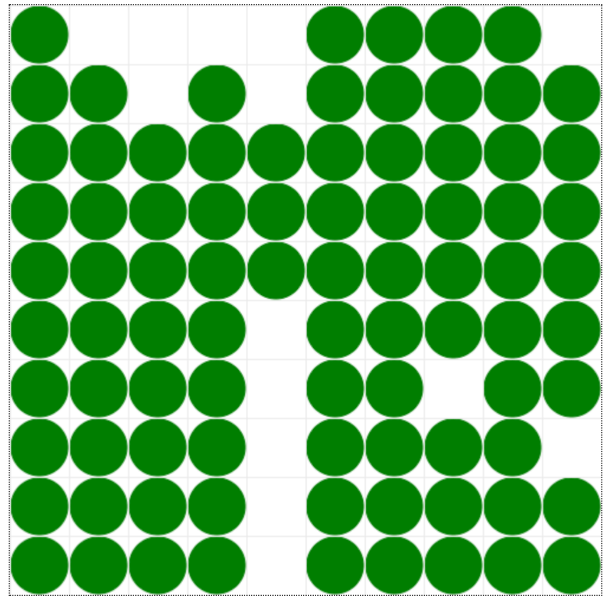










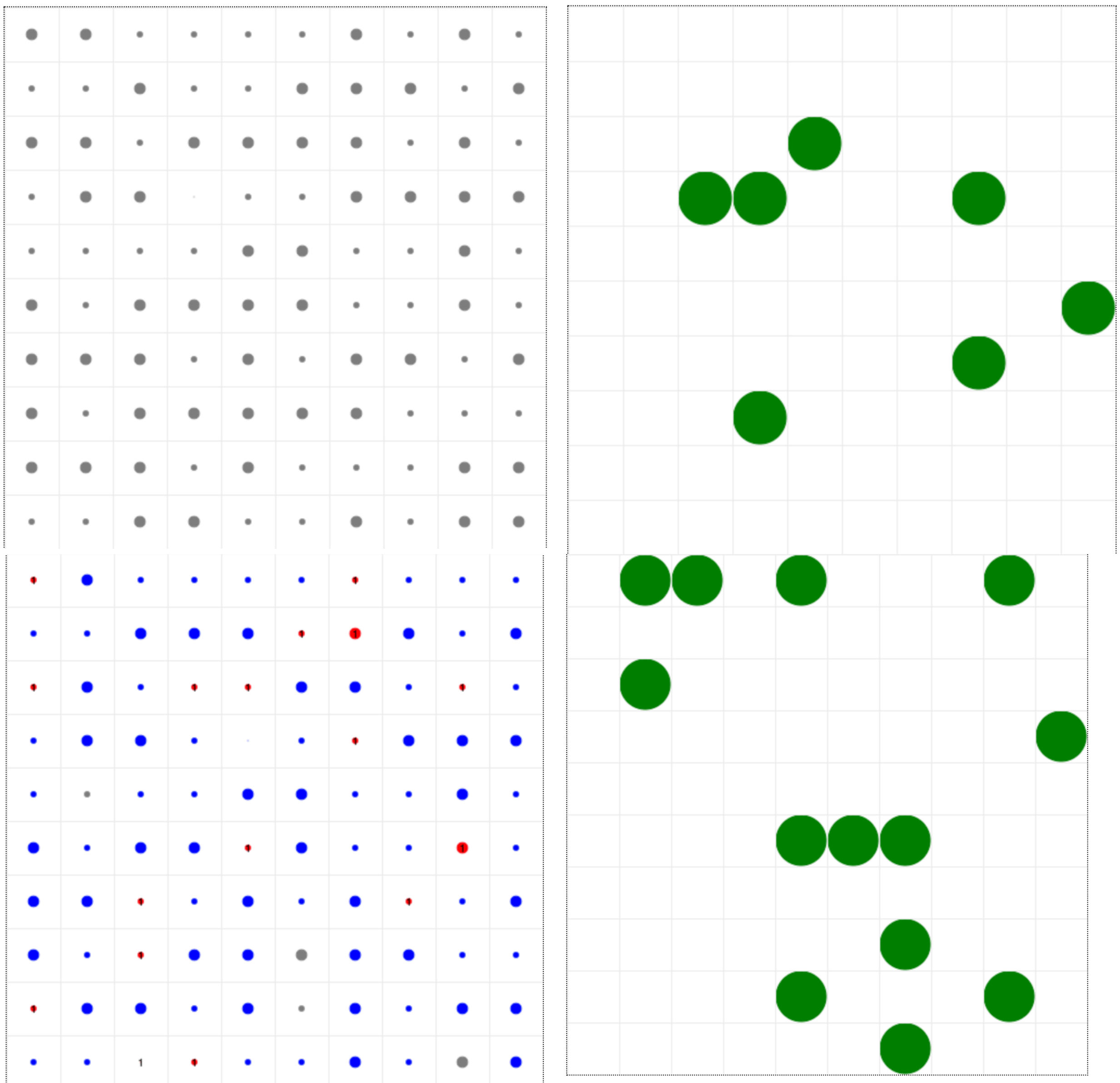


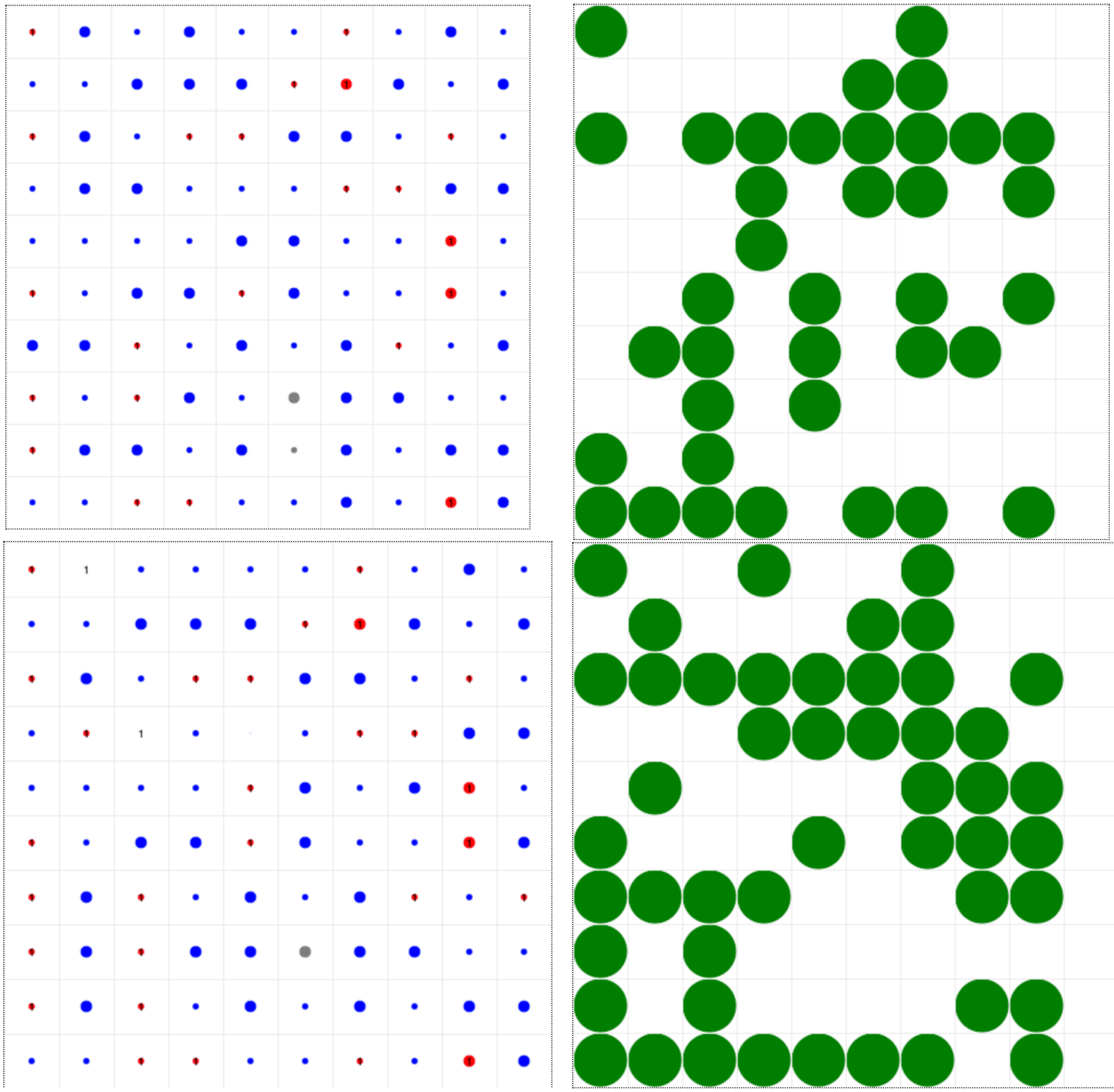
80% of the population has access to their daily needs of electricity **within 14 iterations of the simulation** and **standard deviation of 67 units or 670 rupees**.

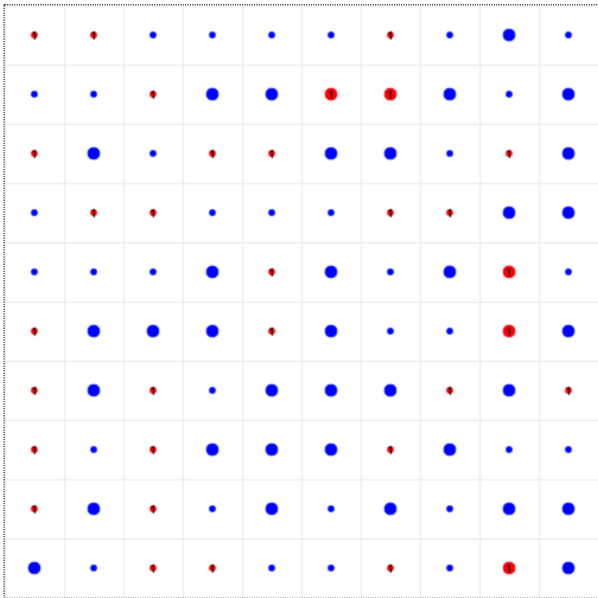
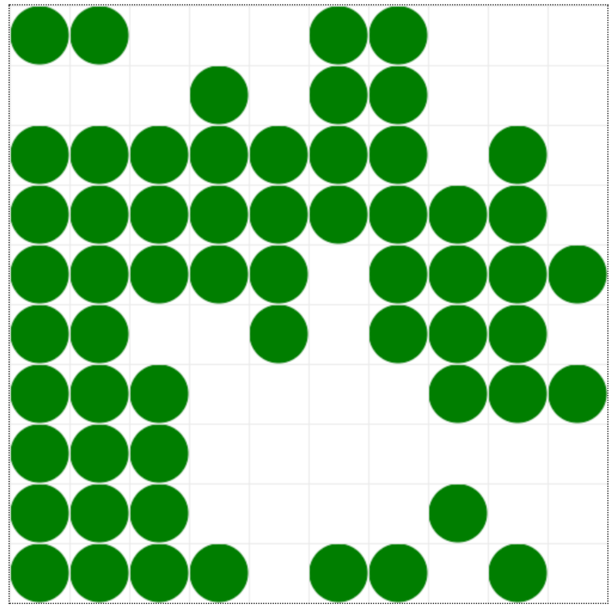
To see how quickly the microgrid grows and how the savings of the agents change.

Price of electricity: 2 units

The value of price at which electricity is sold is change from 0.5 units (5 rupees) to 2 units (20 rupees):

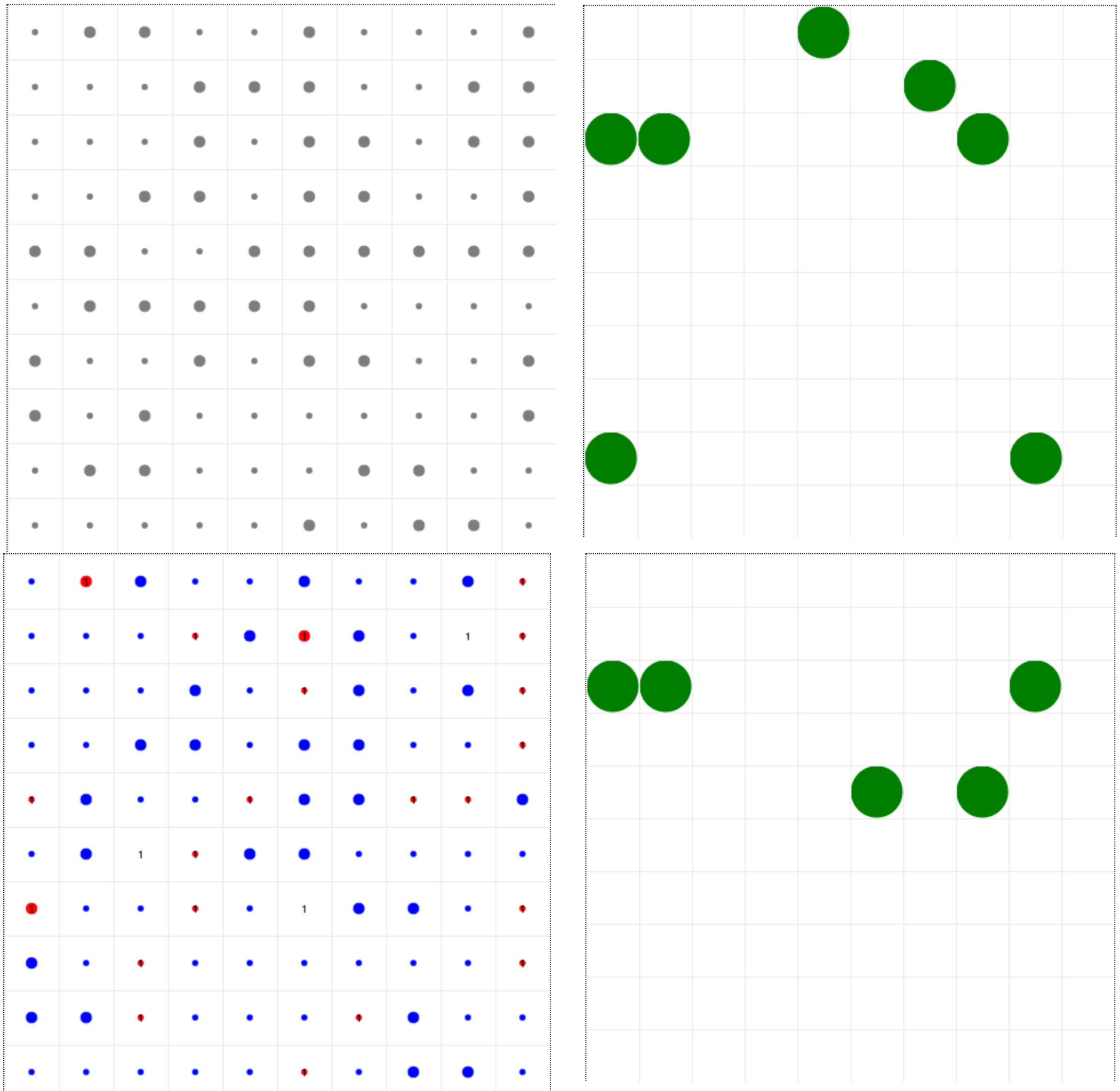


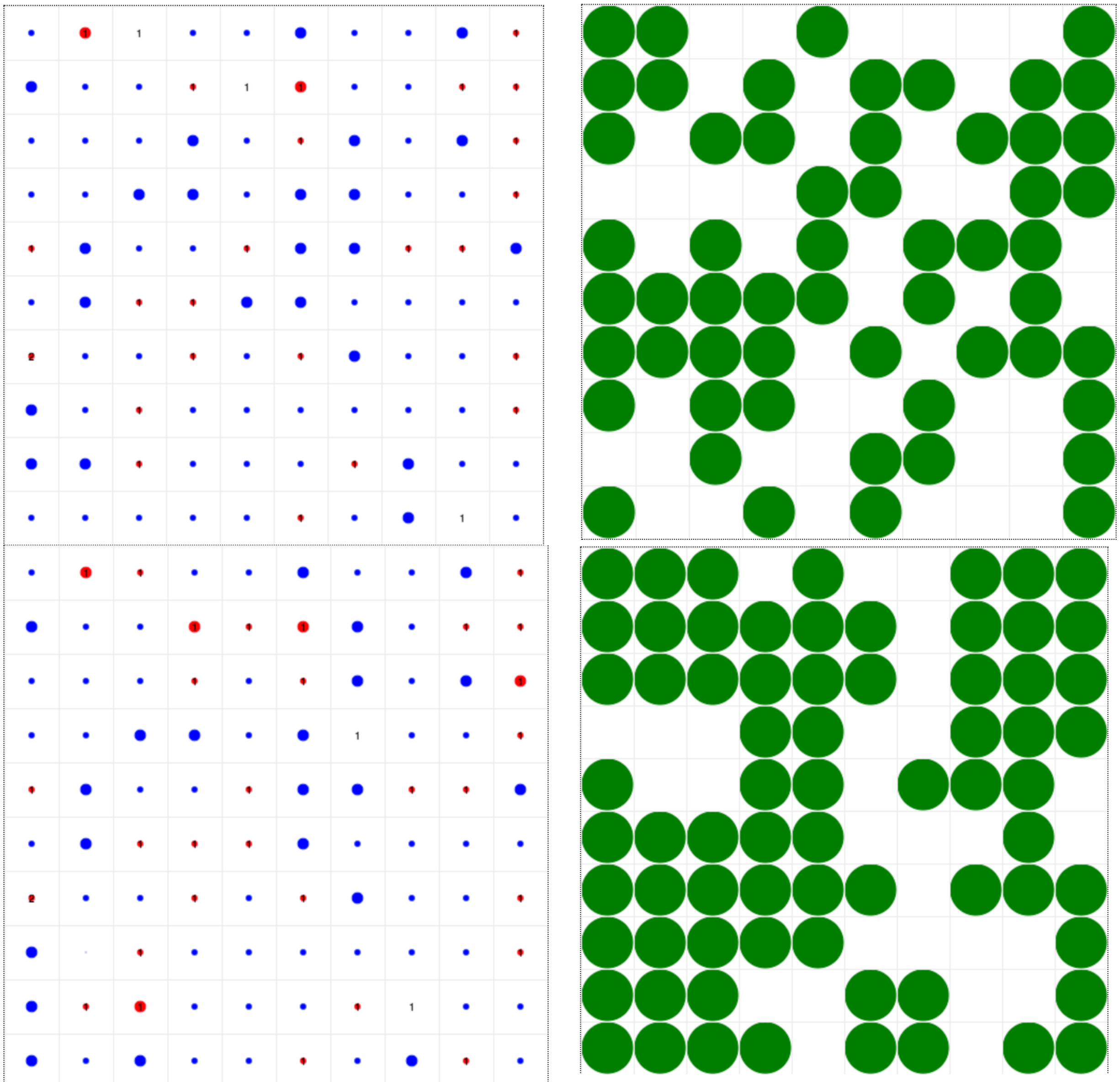


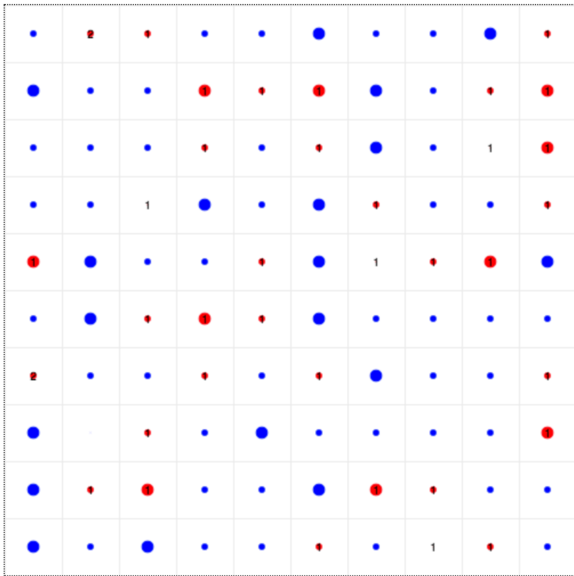



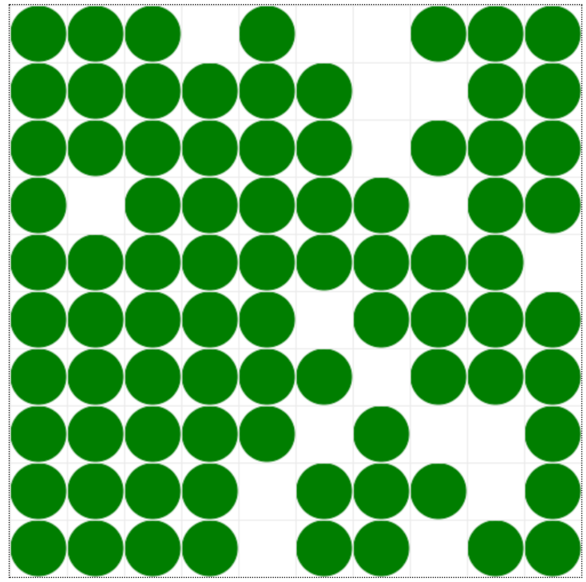
80% of the population has access to their daily needs of electricity within **10 iterations of the simulation** with a standard deviation of **42 money units or 420 Rupees**

Price of electricity: 4 units









80% of the population had access to their electrical needs by 7 iterations and the standard deviation at step 7 is 42 units. However, the rate of increase of standard deviation in this model is much higher than previous two. The model achieves a standard deviation in economy of 44 units at 10th iteration.

Increasing the price to 6 units or 60 rupees, eliminates most of the agents savings and agents start “disappearing” off the charts as shown in figure 3 and the model didn’t reach the 80% point within the first 50 iterations.

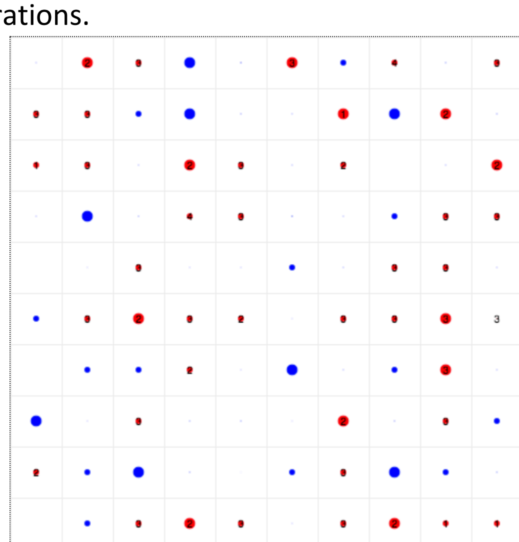


Figure 3: Agents spending all their wealth and yet not meeting their electricity needs.
For data collection purposes, the model was also repeated with price of electricity as 1 unit.

Observations

The ideal price point in this model is 1-2 unit of money or 10-20 rupees (per unit). This cost an average agent about 30-60 rupees daily. It has the lowest standard deviation in the model and reaches 80% of the agent population within 10 iterations.

When the price of the electricity traded is 0.5 units of money or 5 rupees per unit (or 15 rupees daily), the following observations are seen:

1) Agents are not as incentivized to become producers are they have low returns.
2) The price point 0.5 units should theoretically have a low standard deviation point at iteration 10 but since the incentive of the agents to become producers was low, they were relying on kerosene and preferred not meeting their energy needs till an agent decided to be a producer.
3) Consumers are more interested in buying electricity than investing long term for a solar panel for their own as they are not that incentivized which means the producer can benefit in the long run as there won't be a lot of competition while keeping the consumers content.

As the price of the electricity traded increases, the following observations are seen:

1) A lot of agents are incentivized to become produces and producers are introduced very quickly to the model
2) However, most consumers are not able to afford the price of electricity and often experienced a downward cycle.
3) If consumers can afford, they will prefer buying their solar panel as soon as possible as they are incentivized to be a producer over being a consumer.

Pricing it higher than kerosene means the consumers prefer remaining kerosene users.

5. Limitations and Recommendations:

This simulation has several limitations as it does not represent a real-world. It follows rules of what would be a good choice to make but that necessarily is not how the real-world behaves. There are few areas of improvements identified such as:

Simulation does not take debt into account: In the simulation, the agents are provided income daily with a little saving. However, in real-world scenarios, people have debt with interest, which often induces perpetual down cycle.

Micro-grids success depends on community engagement**:** One of the failure points for micro-grid is lack of community engagement. Since micro-grids are designed for a community, it is necessary to involve the stakeholders of the community to achieve sustainable energy planning and development.

Lack of revenue generation**:** A good micro-grid system is self-sustaining in terms of income and revenue generation, arising from the energy trades by the intended users in the community.

High initial cost of installation**:** A micro-grid consists of the PV solar cells that can be expensive as opposed to a "quick fix" that alternative fuels such as Kerosene offer. There needs to be an incentive for a Kerosene user to convert into a producer and save for the long term. Saving behavior is not easy to quantify and model.

Energy production is uncertain**:** Since a lot of micro-grid system rely on solar energy, the production of electricity is not certain. There can be times when there is more electricity produced than required and times when basic energy needs cannot be fulfilled.

Block-chain to verify transactions: The use of blockchain during the implementation of microgrid solution will be beneficial to provide transparency and increase accountability. It will promote the growth of microgrids as the producers can expect the neighbours who used their electricity to pay them for it

Producer's neighbor converting into a producer: To improve the model, the simulation can take into account the probability of an agent converting into a producer given the fact their neighbor already produces electricity for them. The probability of a producer upgrading solar cells to serve his neighbors is more due to the gained experience and responsibility.

6. Conclusion

Around the world, there are over 1 billion people who do not have access to electricity to perform their basic activities. In India, there are over 54,000 un-electrified isolated grid communities. Often grid extension is not feasible to power them and hence, microgrids should be developed in these communities. It provides an alternative source for the members of the community while reducing their usage of alternatives fuels such as kerosene. This reduces economic stress and improves living conditions. An agent based model was developed that aimed to simulate how agents will behave in an economy if microgrids are introduced. The cost at which the producer sells their electricity to their neighbours was varied and the results were studied. It was seen the optimal price for selling 6 W-hr of electricity of 5 hours is 30 to 60 rupees. Observations on why high price points and low price points can be negative for the community development. Since the model was simulated and does not depict how people in real life behave, it is limited in its accuracy. Finally, the report recommends to take more parameters into account to improve accuracy and use blockchain to verify transactions in real world scenarios.

7. Next Steps

This model was created to aid the author, one of the co-founders of a peer to peer energy transaction company, Prava by seeing the behaviors of how agents will expect to behave during the creation of microgrids. The price of the electricity will be chosen between 30 rupees to 60 rupees daily. The co-founder aim to produce the device that allows this transaction by end of August 2018 and compete for a spot in HULT accelerator to help grow their business. Prava aims to be in Bangalore, India working directly with the users.

Citation

Foundation, T. (2018). India, Connected: How the Rockefeller Foundation is bringing electricity to thousands of villages across India (Paid Content by The Rockefeller Foundation). [online] Mashable. Available at: https://mashable.com/2017/09/17/how-rockefeller-foundation-is-bringing-electricity-to-villages-in-india/#y6I3j3Lm5sq8 [Accessed 6 Apr. 2018].

World Health Organization, (2018). [online] Available at: http://www.who.int/mediacentre/factsheets/fs292/en/] [Accessed 6 Apr. 2018].

Sen, D. (2016). Solar DC Microgrid for Rural Electrification-A Case Study. Indonesian Journal of Electrical Engineering and Informatics (IJEEI), 4(1).

Appendix

For access to the software, visit https://github.com/Spurrya/Simulating-peer-to-peer-economy

The code:

Viz_MoneyModel.py

```
from mesa.visualization.modules import CanvasGrid
from mesa.visualization.modules import ChartModule
from mesa.visualization.ModularVisualization import ModularServer

from MoneyModel import EnergyModel
from MoneyModel import EnergyUsageType

DEBUG = 0

def agent_portrayal1(agent):


    portrayal = {"Shape": "circle",
                 "Filled": "true"}


    if(agent.type == EnergyUsageType.CONSUMER):
      portrayal["Color"] = "blue"
      portrayal["r"] = agent.savings/1000
      portrayal["Layer"] = 0
      # portrayal["text"] = str(agent.savings)
      # portrayal["text_color"] = "#000"

    elif(agent.type == EnergyUsageType.KEROSENE):
      portrayal["Color"] = "grey"
      portrayal["r"] = agent.savings/1000
      portrayal["Layer"] = 0
      # portrayal["text"] = str(agent.savings)
      # portrayal["text_color"] = "#000"

    elif(agent.type == EnergyUsageType.PRODUCER):
      portrayal["Color"] = "red"
      portrayal["r"] = agent.savings/1000
      portrayal["Layer"] = 0
      portrayal["text"] =  str(agent.level_solar)
      portrayal["text_color"] = "#000"
```

```
    if(agent.savings == 0):
      portrayal['r'] = 0
    elif(agent.savings > 2000 ):
      portrayal['r'] = 0.9
    elif(agent.savings > 1500):
      portrayal['r'] = 0.8
    elif(agent.savings>1000):
      portrayal['r'] = 0.6
    elif(agent.savings > 500):
      portrayal['r'] = 0.5
    elif(agent.savings > 100):
      portrayal['r'] = 0.2
    elif(agent.savings > 10):
      portrayal['r'] = 0.1

    return portrayal

def agent_portrayal2(agent):
  portrayal = {}


  if(agent.today_energy_needs == 0):
    portrayal["Shape"] = "circle"
    portrayal["Filled"] = "true"
    portrayal["Color"] = "green"
    portrayal["r"] = 1
    portrayal["Layer"] = 0



    return portrayal

grid1 = CanvasGrid(agent_portrayal1, 10, 10, 500, 500)
grid2 = CanvasGrid(agent_portrayal2, 10, 10, 500, 500)
chart = ChartModule([
    {"Label": "Number of people satisfied with energy needs", "Color":
"Black"}],
    data_collector_name='datacollector'
)


parameters =  { "width":10,
   "height":10,
   "consumption_of_energy_mean": 3,
```

```
    "consumption_of_energy_std":0.1,
    "daily_production_of_energy_mean": 6,
    "daily_production_of_energy_std": 1,
    "daily_outcome_mean": 2,
    "daily_outcome_std":0.1,
    "daily_income_mean":10,
    "daily_income_std":0.5,
    "price_of_alternative_fuels":7.5,
    "price_of_solar_panel":150,
    "price_of_electricity_from_producer": 0.5,
    "probability_of_converting_into_producer":0.25,
    "probability_of_neighbour_converting_into_producer":0.05
}


parameters["probability_of_converting_into_producer"] =
parameters["probability_of_converting_into_producer"] *
parameters["price_of_electricity_from_producer"]
parameters["probability_of_neighbour_converting_into_producer"] =
parameters["probability_of_neighbour_converting_into_producer"] *
parameters["price_of_electricity_from_producer"]

server = ModularServer(EnergyModel, [grid1, grid2], "Money Model",
parameters)
server.port = 5000

server.launch()
```

MoneyModel.py

```
import random


from mesa import Agent, Model

from mesa.time import SimultaneousActivation

from mesa.space import MultiGrid

from mesa.datacollection import DataCollector

import random

import enum

import numpy as np
```

```
energy_produced_today = 0

def number_of_people_whose_energy_requirement_are_fulfilled(model):
    i = 0
    for agent in model.schedule.agents:
        if(agent.today_energy_needs == 0):
            i = i + 1
    return i

def standard_deviation_of_savings(model):
    return np.std([agent.savings for agent in model.schedule.agents])

class EnergyModel(Model):
    """A model with some number of agents."""

    def __init__(self,
        width,
        height,
        consumption_of_energy_mean,
        consumption_of_energy_std,
        daily_production_of_energy_mean,
        daily_production_of_energy_std,
        daily_outcome_mean,
        daily_outcome_std,
        daily_income_mean,
        daily_income_std,
        price_of_alternative_fuels,
        price_of_solar_panel,
        price_of_electricity_from_producer,
        probability_of_converting_into_producer,
```

```
        probability_of_neighbour_converting_into_producer
        ):
        self.num_agents = (width * height)
        self.running = True
        self.grid = MultiGrid(height, width, True)
        self.schedule = SimultaneousActivation(self)

        self.price_of_alternative_fuels = price_of_alternative_fuels
        self.price_of_solar_panel = price_of_solar_panel
        self.price_of_electricity_from_producer =
price_of_electricity_from_producer
        self.probability_of_converting_into_producer =
probability_of_converting_into_producer
        self.probability_of_neighbour_converting_into_producer =
probability_of_neighbour_converting_into_producer
        self.daily_production_of_energy_mean =
daily_production_of_energy_mean
        self.daily_production_of_energy_std =
daily_production_of_energy_std

        self.datacollector = DataCollector(
            model_reporters={"Number of people who have access to
energy needs":
number_of_people_whose_energy_requirement_are_fulfilled},
            agent_reporters={"Wealth": lambda a: a.savings}
        )
        i = 0

        print("initing")
        for x in range(width):
            for y in range(height):
```

```
                consumption_of_energy =
abs(int(np.random.normal(consumption_of_energy_mean,
consumption_of_energy_std, 1)))
                daily_income =
abs(int(np.random.normal(daily_income_mean, daily_income_std, 1)))
                daily_outcome =
abs(int(np.random.normal(daily_outcome_mean, daily_outcome_std, 1)))
                savings = abs(int(np.random.normal(100, 80, 1)))

                a = EnergyAgent(x,y, savings, i,
consumption_of_energy, daily_income, daily_outcome, self)
                self.schedule.add(a)
                self.grid.place_agent(a, (x, y))
                i = i +1
    k = 0
    def step(self):
        self.schedule.step()
        self.datacollector.collect(self)
        self.k = self.k + 1

if(number_of_people_whose_energy_requirement_are_fulfilled(self) >=
self.num_agents*0.80):
            print("=====")
            print("The selling price:" +
str(self.price_of_electricity_from_producer))
            print("Time it took:" + str(self.k) )
            print("Std:" + str(standard_deviation_of_savings(self)))
    def run_model(self, n):
        for i in range(n):
            self.step()
```

```
class EnergyUsageType(enum.IntEnum):
    PRODUCER = 0
    CONSUMER = 1
    KEROSENE = 2

class EnergyAgent(Agent):
    def __init__(self, x,y, savings, unique_id,daily_energy_needs,
daily_income, daily_outcome, model):
        self.x = x
        self.y = y
        self.unique_id = unique_id
        self.savings = savings
        self.type = EnergyUsageType.KEROSENE
        self.daily_energy_needs = daily_energy_needs
        self.daily_income = daily_income
        self.daily_outcome = daily_outcome
        self.level_solar = 0 # Kerosene users and consumers are level
0
        self.energy_owned = 0
        self.today_money = 0
        self.today_energy_needs = self.daily_energy_needs

        self.price_of_alternative_fuels =
model.price_of_alternative_fuels
        self.price_of_solar_panel = model.price_of_solar_panel
        self.price_of_electricity_from_producer =
model.price_of_electricity_from_producer
        self.probability_of_converting_into_producer =
model.probability_of_converting_into_producer
```

```
        self.probability_of_convertion = 0
        self.probability_of_neighbour_converting_into_producer =
model.probability_of_neighbour_converting_into_producer
        self.model = model

    def probability_of_buying_kerosene(self):
        return random.random() < 0.1

    def update_savings(self):
        self.savings = self.savings + self.today_money

    def convert_to_producer(self):
        if(self.savings > self.price_of_solar_panel and
random.random()< self.probability_of_convertion):
            if(self.type != EnergyUsageType.PRODUCER ):
                self.level_solar = 1
                self.type = EnergyUsageType.PRODUCER
                self.savings = self.savings -
self.price_of_solar_panel

                neighbours =
self.model.grid.iter_neighbors([self.x,self.y],include_center=False,
moore=True, radius=2)
                for neighbour in neighbours:
                    if(neighbour.type == EnergyUsageType.KEROSENE and
self.price_of_alternative_fuels >
self.price_of_electricity_from_producer):
                        neighbour.type = EnergyUsageType.CONSUMER

            else:
                self.level_solar = self.level_solar + 1
```

```
                self.savings = self.savings -
self.price_of_solar_panel


    def net_saving_for_today(self):
        self.today_money = self.daily_income - self.daily_outcome

        if(self.today_money < 0):
            self.today_money = 1


    def provide_income(self):
        #self.savings = self.daily_income + self.savings
        self.today_money = self.daily_income


    def produce_electricity(self, energy_produced_today):
        if(self.type == EnergyUsageType.PRODUCER):
            self.energy_owned = energy_produced_today *
self.level_solar


    def buy_kerosene(self):
        while(self.today_energy_needs > 0 and (self.today_money >
self.price_of_alternative_fuels or self.savings >
self.price_of_alternative_fuels)):

            # Buy electricity
            self.energy_owned = self.energy_owned + 1

            # Exchange Money
```

```
            if(self.today_money > self.price_of_alternative_fuels ):
                self.today_money = self.today_money - 
self.price_of_alternative_fuels
            elif(self.savings > self.price_of_alternative_fuels):
                self.savings = self.savings - 
self.price_of_alternative_fuels



            #self.today_money = self.today_money - 
self.price_of_alternative_fuels

            # Consume Electricity
            self.today_energy_needs = self.today_energy_needs - 1
            self.energy_owned = self.energy_owned - 1



    def trade_electricity(self):
        if(self.type == EnergyUsageType.PRODUCER):
            neighbours = 
self.model.grid.get_neighbors([self.x,self.y],include_center=False, 
moore=True, radius=2)

            while(self.energy_owned > self.today_energy_needs):
                i = 0
                j = 0
                for neighbour in neighbours:
                    j = j + 1
                    if(neighbour.today_energy_needs > 0 and 
(neighbour.savings > neighbour.price_of_electricity_from_producer or 
neighbour.today_money > 
neighbour.price_of_electricity_from_producer)):
```

```
                        # Buy electricity
                        self.energy_owned = self.energy_owned - 1;
                        neighbour.energy_owned =
neighbour.energy_owned + 1;

                        # Exchange Money
                        self.today_money = neighbour.today_money +
neighbour.price_of_electricity_from_producer
                        if(neighbour.today_money >
neighbour.price_of_electricity_from_producer ):
                            neighbour.today_money =
neighbour.today_money - neighbour.price_of_electricity_from_producer
                        elif(neighbour.savings >
neighbour.price_of_electricity_from_producer):
                            neighbour.savings = neighbour.savings -
neighbour.price_of_electricity_from_producer

                        #Consume
                        neighbour.today_energy_needs =
neighbour.today_energy_needs - 1
                        neighbour.energy_owned =
neighbour.energy_owned - 1
                    else:
                        i = i + 1
                if(i == j):
                    break;

            # print("===")
            # for n in neighbours:
            #     print("Today Money," + str(n.today_money))
            #     print("Today Needs," + str(n.today_energy_needs))
```

```
            #     print("Today Owned," + str(n.energy_owned))

    def consume_own_needs_producer(self):
        if(self.type == EnergyUsageType.PRODUCER):
            while(self.energy_owned > 0 and self.today_energy_needs > 
0):
                self.energy_owned = self.energy_owned -1;
                self.today_energy_needs = self.today_energy_needs - 1;



    def update_prob_of_converting(self):
        neighbours = 
self.model.grid.iter_neighbors([self.x,self.y],include_center=True, 
moore=True, radius=2)
        i = 0
        for neighbour in neighbours:
            if(neighbour.today_energy_needs > 0):
                i = i+ 1

        if(i > 0):
            self.probability_of_convertion = 
self.probability_of_converting_into_producer
        else:
            self.probability_of_convertion = 
self.probability_of_neighbour_converting_into_producer



    def advance(self):
        self.today_energy_needs = self.daily_energy_needs
        self.energy_owned = 0
```

```
        energy_produced_today =
abs(int(np.random.normal(self.model.daily_production_of_energy_mean,
self.model.daily_production_of_energy_std, 1)))

        self.provide_income()
        self.net_saving_for_today()
        self.produce_electricity(energy_produced_today)

        self.consume_own_needs_producer()



        self.trade_electricity()
        if(self.probability_of_buying_kerosene()):
            self.buy_kerosene()



        if(self.today_money < 0):
            self.today_money = 0



        # if(self.today_energy_needs != 0):
        #     print("======")
        #     print("I am a ," + str(self.type))
        #     print("I am at," + str(self.x) + ","+str(self.y))
        #     print("I made today, $" + str(self.today_money))
        #     print("I own energy," + str(self.energy_owned))
        #     print("For today, I need, this much energy " +
str(self.today_energy_needs))



        self.update_savings()
```

```
        self.convert_to_producer()
        self.update_prob_of_converting()




    # def move(self):
    #     possible_steps = self.model.grid.get_neighborhood(
    #         self.pos, moore=True, include_center=False
    #     )
    #     new_position = random.choice(possible_steps)
    #     self.model.grid.move_agent(self, new_position)

    # def give_money(self):
    #     cellmates =
self.model.grid.get_cell_list_contents([self.pos])
    #     if len(cellmates) > 1:
    #         other = random.choice(cellmates)
    #         other.wealth += 1
    #         self.wealth -= 1

    # def step(self):
    #     self.move()
    #     if self.wealth > 0:
    #         self.give_money()
```

[redacted]