Name: EPA1324 2023-04-11A p01

Nachiket Kondhalkar

Student number: 5833884

ANSWER FORM

Exam for *Introduction to TPM modelling* (EPA1324)

Tuesday, 11 April 2023, 13:30

This form comprises **3** pages. Only text inside the designated areas will be considered for grading.

PLEASE DON'T ALTER THE SIZE OF ANY OF THE GREY BOXES

*Please mark each digit like this: ■, **not** like this: ✉*

(2023-04-11 13:30:40)

(signature)

0. Honours
YES

```
function step():
  for each election cycle:
    for each party in the model:
      calculate and update party fitness
      if party fitness < party.fitness_threshold:
        remove the party from the model (party death)
    for each population member:
      calculate and update party_frustration
      if party_frustration > party_frustration_threshold:
        create a new party centered on the population member's policy position (party birth)
    for each district in the model:
      for each population member in the district:
        calculate and update district_frustration
        if district_frustration < district_frustration_threshold:
          mark the population member as an eligible voter

      hold elections and determine the winning candidate
      update the policy positions of the political parties based on the winning candidates
```

1b. Implement.

```
  def step(self):
    # Party death: remove parties with low fitness
    for party in self.parties:
      if party.fitness < Party.fitness_threshold:
        self.party_scheduler.remove(party)

    # Party birth: create new parties centered on population members with high frustration
    for population in self.district_populations:
      for i, frustration in enumerate(population.party_frustration):
        if frustration > Population.party_frustration_threshold:
          new_party_position = population.population[i]
          new_party = Party(count(), self, new_party_position, self.n_districts)
          self.party_scheduler.add(new_party)
    self.district_scheduler.step()
    self.party_scheduler.step()
    self.population_scheduler.step()
    self.datacollector.collect(self)
```

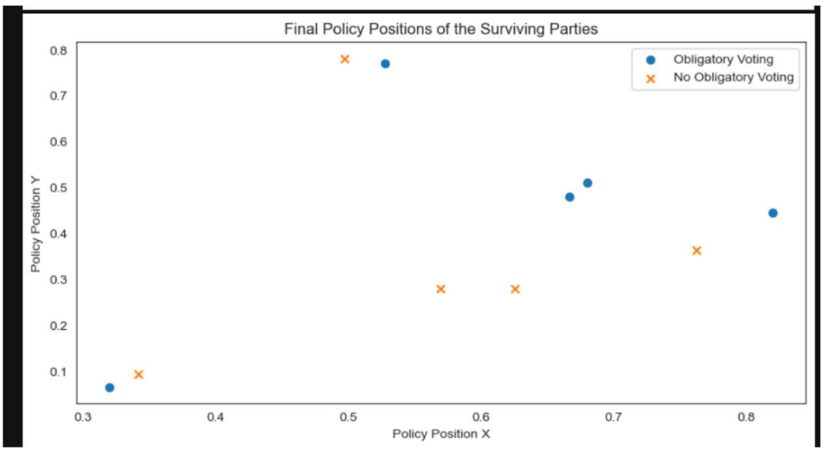
2. Modify.
In order to modify the model, we need to break the District class into smaller units called counties or neighbourhoods. They will have a fixed population and policy preference for a set number of election cycles. There will be a Redistricting function called every 10 years that is controlled by the ruling party at the time. This function will redraw the district boundaries according to one of two strategies. Packing or Cracking. These strategies can be chosen from a Gerrymander class. Packing will concentrate the opposing party's supporters into a few districts, causing them to win by large margins in those districts, but lose everywhere else. Cracking on the other hand will distribute the opposing party's supporters across multiple districts, diluting their voting power and causing them to lose by narrow margins. This may also involve recalculating the population's policy preferences and party frustrations based on the new district compositions.

The step method will need to be adjusted to conduct the redistricting process at regular intervals. ContinuousSpace used in the model will also need adjusting. The data collector may also need to change to keep track of the margin by which a party has won in a given district as this may have an impact on choosing which strategy the ruling party uses to gerrymander the given set of districts.

3a. implement experiments

```
def run_experiment(n_districts, n_parties, n_steps, obligatory_voting, seed=None):
    model = VotingModel(n_districts=n_districts,   n_parties=n_parties,   obligatory_voting=obligatory_voting,   seed=seed,)
    for _ in range(n_steps):
        model.step()
    return model.datacollector.get_model_vars_dataframe(), model.datacollector.get_agent_vars_dataframe()
n_districts = 10
n_parties = 5
n_steps = 50
seed = 42
model_data_obligatory, agent_data_obligatory = run_experiment( n_districts, n_parties, n_steps, True, seed)
model_data_no_obligatory, agent_data_no_obligatory = run_experiment( n_districts, n_parties, n_steps, False, seed)
```

3b. visualize



3c. changes in behaviour
With Obligatory voting, parties seemed to move further along their spectrum and try to move away form each other. On the other hand, without Obligatory voting, the political parties seem more spread out evenly. This could be because with absolutely everyone voting, parties may find it easier to secure their positions in certain niche districts and change their political ideology to build their voter base in these districts. This in a way is characteristic of the polarisation we see in modern politics with increased voter turn outs.