Simulation Mechanism

December 24, 2023

Lanzhou University Strategy Simulation System

1 Preparation

1.1 *Introduction*

Strategy simulation is an activity that involves simulating historical events. Players take on the role of historical decision makers and navigate the **political**, **economic**, **military**, **diplomatic**, **public opinion**, and **intelligence** aspects of the game.

Diplomacy, public opinion, and intelligence content are subject to a more mature ***Model United Nations rules*** of procedure for reference. This activity involves free consultation and organized core consultation mechanisms, with strict norms for document writing.

For political, economic, military, and other intelligence aspects, we mainly refer to common elements in ***strategy games*** to build relevant mechanisms. The **4X (explore, expand, exploit, exterminate) concept** is specifically addressed, while also incorporating academic research results on measurement methods in related fields to improve the simulation mechanism.

This project focuses on computer simulation of **numerical computation**. The game interface and subsequent visualization are yet to be explored and developed.

1.2 *Framework*

For writing the computation mechanism, we first considered using ***Python*** because of the following properties:

- Simple syntax, easy to read and write
- Rich integration of third-party libraries, easy to call

Of course, its execution speed is still a big problem, especially when the computation mechanism is more complex. Therefore, the subsequent graphical interface design and the complete program development using other languages to write is also to be considered.

We mainly use an ***object-oriented*** approach to **encapsulate** each computer system under a large module in a class, which makes it easy to call and also improves the maintainability of the code.

1.3 *Configuration*

In order for the code to run properly, we need to install some third-party libraries, the relevant configuration is as follows:

[ ]:
```
# Python version: 3.9.13

%pip install matplotlib
# install the library Matplotlib

from functools import reduce
# The reduce() function is invoked to accumulate elements in a sequence.

import matplotlib.pyplot as plt
# The pyplot method is invoked to create image.

from math import ceil
# The ceil function is invoked to round values.

from os import urandom
# The urandom function is invoked to generate real random number.
```

2 Establishment

2.1 *Politics*

The main calculation variables in the political section are:

- Stability
- Government Support
- Political Expenditures
- Military Expenditures
- Labor Expenditures
- Domestic Taxes

[ ]:
```
class Politics(object):
    """
    The Politics module of the strategic simulation system

    Attributes:
        Stability
        Government Support
        Political Expenditures
        Military Expenditures
        Labor Expenditures
        Domestic Taxes
        Number of Incumbents
        Salary of Personnel
        Taxpayers
        Starting Tax Amount
        Personnel Tax Rate
        State Policy Pricing
        Number of National Policies
        Expenditures Buff
        Revenue Buff
        Global Expenditure Effect
        Global Revenue Effect
        National Political Characteristics Revision
        Government Expenditure Effect
        Domestic Taxes Revenue Effect
        Extra Domestic Taxes Revenue Effect
    """
    def __init__(
            self, stability, government, policy, revision, population, soldier,
            soldierpay, salary, tax) -> object:
        """
        This part defines the basic parameters of the politics module.

        :param stability: Degree of stability: -50~100 (float)
        :param government: Level of government support: 0~100 (float)
        :param policy: National policy:
            {
                Type of State policy
                ('Politics', 'Economics', 'Military', 'Intelligence'):
                Policy level (1, 2, 3, 4, 5)), ...
            }
            (dict)
        :param revision: Policy implementation cost revisions (float)
        :param population: The population of the country
            (in 10,000 persons)(float)
        :param soldier: Number of soldiers (in 10,000 persons) (float)
        :param soldierpay: Pay and provisions for soldiers (float)
        :param salary: Salaries of civil servants (float)
        :param tax: Personal income tax rate (float)
        """
        ...
    def implementation(self) -> list:
        """
        This section defines the specific mechanics of policy implementation.

        :return:
            stability, social expenditure (policy, military, and employee),
            Internal revenue, Effects of stability values
        """
        ...
        def add_effect(factor, value):
            """
            This nested function defines the storage operation
            for the effect corresponding to the stabilization level.
            :param factor: Aspects of the effect of stabilization (str)
            :param value: Specific effects value (float)
            """
            ...
        def unrest(level, percent):
            """
            This nested function defines the generation
            of turbulence events and their random effects.

            :param level: The level of the destabilizing event (int)
            :param percent:
                The probability of the occurrence of a destabilizing event
                (float)
            """
            ...
        def stability_production(value):
            """
            This nested function defines the natural output
            of stabilization per turn.

            :param value:
                The value of government support for the current turn (float)
            """
            ...
        ...
```

2.1.1 *Stability*

Let:

- Natural output of ***stability*** in **this round** —> ϑ
- Degree of ***government support*** in the **last round** —> δ

$$\vartheta = \begin{cases} \delta/2 & \text{if } \delta \geq 50 \\ (\delta - 50)/2 & \text{if } 0 < \delta < 50 \end{cases}$$

2.1.2 *Labor/Military Expenditures*

Let:

- ***Expenditures*** —> γ
- **Number** of ***incumbents*** —> n
- ***Salary*** of **personnel** —> s

$$\gamma = n \cdot s$$

2.1.3 *Domestic Taxes*

Let:

- ***Revenue*** —> λ
- ***Taxpayers*** —> t
- **Starting *tax amount*** —> α
- ***Tax rate*** —> r

$$\lambda = t \cdot \alpha \cdot r$$

2.1.4 *Political Expenditures*

Let:

- ***Expenditures*** —> φ
- **Number** of ***national policies*** —> η
- **State policy *pricing*** —> p

$$\varphi = \sum_{i=1}^{\eta} p_i$$

2.1.5 *Buff*

Let:

- ***Expenditures buff*** —> e
- ***Revenue buff*** —> r
- ***Global expenditure* effect** —> ε
- ***Global revenue* effect** —> μ
- National ***political characteristics* revision** —> ω
- ***Government expenditure* effect** —> ε_g
- ***Domestic taxes revenue* effect** —> μ_d
- ***Extra domestic taxes revenue* effect** —> μ_{ed}

$$e_{government} = (1 + \varepsilon_g + \varepsilon) \cdot (1 + \omega) e_{military} = 1 + \varepsilon r = (1 + \mu + \mu_d) \cdot (1 + \mu_{ed})$$

2.2 *Economy*

The main calculation variables in the economic section are:

- Production
- Construction Expenditures
- Restoration Expenditures
- National Product Demand
- Military Training Demand
- Logistic Demand
- Import Tariff

```
[ ]: class Economy(object):
         """
         The Economy module of the strategic simulation system
```

```
    Attributes:
        Production
        Construction Expenditures
        Restoration Expenditures
        Political Expenditures
        National Product Demand
        Military Training Demand
        Logistic Demand
        Import Tariff
        Days
        Yield
        Number of Tasks
        Cost
        Number of Province
        Number of Divisions
        Expenditures Buff
        Revenue Buff
        Production Buff
        Global Expenditure Effect
        Global Revenue Effect
        Global Production Effect
        Fiscal Expenditure Effect
        Extra Trade Revenue Effect
        Factory Production Cost Revisions
    """
    def __init__(
            self, facility, product, equipment, social_expenditure,
            internal_revenue, effects, currency, tax, trade, revision, time
        ) -> object:
        """
        This part defines the basic parameters of the economy module.

        :param facility: Number of facilities of each type in the country:
            {
                'civilian factory': float, 'military factory': float,
                'army fortress': float, 'anti-aircraft gun': float
            }
            (values in block) (dict)
        :param poduct: Quantity of each type of product in the country:
            {
                'agro-pastoral': float, 'synthetic fiber': float,
                'chemicals': float, 'light industrial': float
            }
            (values in kilograms) (dict)
        :param equipment: Quantity of each type of equipment in the country: {
            '7.63mm automatic pistol': int,
            '7.62mm semi-automatic rifle': int,
            '9mm submachine gun': int,
            '7.92mm heavy and light machine gun': int,
            '82mm mortar': int,
            '75mm field artillery': int,
            '115mm howitzer': int,
            '37mm anti-tank gun': int,
            'PzKpfw I light tank': int,
            'T-26 light tank': int,
            'truck': int,
            'fighter aircraft': int
        } (dict)
        :param social_expenditure:
            Policy, military, and employee expenditure (float)
        :param internal_revenue: Domestic taxes (float)
        :param effects:
            Implications of aspects derived from the political module: {
            'global expenditures': float,
            'global production': float,
            'global revenue': float,
            'fiscal expenditure': float,
            'extra trade revenue': float
        }
        :param currency:
            Current volume of currency (in tens of thousands) (float)
        :param tax: Import tariff rate (float)
        :param trade: Transactions of goods and funds through trade: {
            'agro-pastoral': (import, export) (float, float),
            'synthetic fiber': (import, export) (float, float),
            'chemicals': (import, export) (float, float),
            'light industrial': (import, export) (float, float),
            '7.63mm automatic pistol': (import, export) (int, int),
            '7.62mm semi-automatic rifle': (import, export) (int, int),
            '9mm submachine gun': (import, export) (int, int),
            '7.92mm heavy and light machine gun': (import, export) (int, int),
            '82mm mortar': (import, export) (int, int),
            '75mm field artillery': (import, export) (int, int),
            '115mm howitzer': (import, export) (int, int),
            '37mm anti-tank gun': (import, export) (int, int),
            'PzKpfw I light tank': (import, export) (int, int),
            'T-26 light tank': (import, export) (int, int),
            'truck': (import, export) (int, int),
            'fighter aircraft': (import, export) (int, int)
        } (dict)
        :param revision: Factory production cost revisions {
            yield: float,
            cost: float,
            income: float
        } (dict)
        :param time: Number of days the turn lasts (int)
        """
        ...
    def production(self, production) -> list:
        """
        This section defines the specific mechanics of production tasks.

        :param production:
            Storage of goods produced and number of plants allocated
            {product type: factory number, product type: factory number, ...}
        :return: Product, Product expenditure
        """
        ...
    def construction(self, construction) -> list:
        """
        This section defines the specific mechanics of construction tasks.

        :param construction: Type and number of facilities built for storage
            {facility type: number, facility type: number, ...}
        :return: Construction progress, Construction expenditure
        """
        ...
    def restoration(self, restoration) -> list:
        """
        This section defines the specific mechanics of restoration tasks.

        :param restoration: Type and number of facilities restored for storage
            {facility type: number, facility type: number, ...}
        :return: Restoration progress, Restoration expenditure
        """
        ...
    def implementation(self, *args) -> list:
        """
        This section defines the specific mechanics of economic implementation.

        :param args:
            Includes other expenditure:
                product, construction, restoration (tuple)
        :return:
            Total expenditure
                (social, product, construction, restoration, trade),
            Total revenue (tax, tariff), Volume of change in currency
        """
        ...
    def demand(self, province) -> dict:
        """
        This section defines
        the specific mechanics of domestic economic demand.

        :param province: Number of provinces (int)
        :return: National product demand
        """
        ...
    def training(self, training, soldier) -> list:
        """
        This section defines the specific mechanics of military training.

        :param training: Type of army and number for storage
            {military type: number, military type: number, ...} (dict)
        :param soldier: Number of soldiers (in 10,000 persons) (float)
        :return: Military training demand, Solider
        """
        ...
    def logistic(self, logistic) -> list:
        """
        This section defines the specific mechanics of military logistic.

        :param logistic:
            Type of stockpiled military forces and replenishment base:
            [(military type, base), (military type, base), ...] (list)
        """
        ...
    def update(
            self, product, construction_progress, restoration_progress,
            economic_demand, training_demand, logistic_demand
        ) -> list:
        """
        This section defines the specific mechanics of state warehouse.

        :param product: (dict)
        :param construction_progress: (dict)
        :param restoration_progress: (dict)
        :param economic_demand: (dict)
        :param training_demand: (dict)
        :param logistic_demand: (dict)
        :return: facility, product, equipment
        """
        ...
        def value(y_dict, dict, operation_type):
            """
            This nested function defines
            the generation of the value update of the dictionary.
            :param y_dict: Value to be updated (dict)
            :param dict: Amount of change in value (dict)
            :param operation_type: addition, subtraction (str)
            """
            ...
```

2.2.1 *Production*

Let:

- ***Production*** —> ρ
- ***Days*** —> β
- ***Yield*** —> y

$$y = \rho \cdot \beta$$

2.2.2 *Construction/Restoration Expenditures*

Let:

- ***Expenditures*** —> ψ
- ***Cost*** **of product/equipment** —> ζ
- ***Yield*** **of product/equipment** —> ϖ
- **Number of** ***construction/restoration tasks*** —> ς

$$\psi = \sum_{i=1}^{\varsigma} \varpi_i \cdot \zeta_i$$

2.2.3 *National Product Demand*

Let:

- ***Demand*** —> ι
- ***Demand*** **of product** —> d
- ***Type*** **of product** —> t
- **Number of** ***province*** —> τ

$$\iota = \sum_{i=1}^{t} \tau_i \cdot d_i$$

2.2.4 *Military Training/Logistic Demand*

Let:

- ***Demand*** —> χ
- ***Demand*** **of equipment/product** —> e
- ***Type*** **of equipment/product** —> s
- **Number of** ***divisions*** —> δ

$$\chi = \sum_{i=1}^{s} \delta_i \cdot e_i$$

2.2.5 *Import Tariff*

Let:

- ***Revenue*** —> λ
- ***Import value*** —> α
- ***Tax rate*** —> r

$$\lambda = \alpha \cdot r$$

2.2.6 *Buff*

Let:

- ***Expenditures buff*** —> e
- ***Revenue buff*** —> r
- ***Production buff*** —> p
- ***Global expenditure* effect** —> ε
- ***Global revenue* effect** —> μ
- ***Global production* effect** —> ν
- ***Fiscal expenditure* effect** —> ε_{fi}
- ***Extra trade revenue* effect** —> μ_{ed}
- Factory ***production cost* revisions** —> ϑ

$$e = (1 + \varepsilon + \varepsilon_{fi}) \cdot (1 + \vartheta) r = (1 + \mu) \cdot (1 + \mu_{ed}) \cdot (1 + \vartheta) p = (1 + \nu) \cdot (1 + \vartheta)$$

2.3 *Military*

The main calculation variables in the economic section are:

- Strength
- Support Ratio
- Attrition Rate Threshold
- Combat Effectiveness Factor
- Non-Combatant Attrition Factor
- Initial Troops

[ ]:
```
class Military(object):
    """
    The Military module of the strategic simulation system

    Attributes:
        Strength
        Support Ratio
        Attrition Rate Threshold
        Combat Effectiveness Factor
        Non-Combatant Attrition Factor
        Initial Troops
        Division of Troops
        Division of Strength
        Morale of Troops
        Topography Modification
        Experience Modification
        Supply Modification
        Air Power Modification
        Tactic Modification
        Non-Combatant Attrition Baseline
        Attrition Rate Baseline
        National Specificities Effect
    """
    def __init__(
            self, divisions, reinforcement, topography, combatant, experience,
            morale, surround, time, **kwargs
        ) -> object:
        """
        This part defines the basic parameters of the military module.

        :param divisions:
            Components of military units:
                {'Reserve': int, 'Garrison': int, 'Field': int} (dict)
        :param reinforcement: Components of reinforcement units:
            {
                'Reserve': ((distance, speed correction), ...),
                'Garrison': ((distance, speed correction), ...),
                'Field': ((distance, speed correction), ...)
            } (dict)
        :param topography:
            Situation of the terrain where the fighting took place:
                Hilly, Mountain, Forest, Swamp, Desert, City, Fortress (str)
        :param combatant: Battle type: attack, defence (str)
        :param experience: Soldier's experience level: 1, 2, 3, 4, 5 (int)
        :param morale: Warrior morale index: 0.8~1.2 (float)
        :param surround:
            Whether or not the troops are surrounded: True, False (bool)
        :param time: Number of days the turn lasts (int)
        :param kwargs: Includes other factors:
            supply, air power, tactic, national specificities,
            whose values are filled in artificially (dict)
        """
        ...
    def correction(self) -> list:
        """
```

```python
        This section defines
        the specific mechanics of combat power modification.

        :return:
            Initial troops, Combat effectiveness factor,
            Attrition rate threshold, Non-combatant attrition factor
        """
        ...
    def support(self) -> list:
        """
        This section defines the specific mechanics of support ratio.

        :return:
            support ratio:
                [
                    (day 1 troop, day 1 Combat factor),
                    (day 2 troop, day 2 Combat factor), ...
                ]
        """
        ...
def combat(
        attacker, defender, a, d, nona, nond, thra, thrd, u, v, time, title
    ) -> list:
    """
    Conduct a combat assessment

    :param aggressor: Attacker's initial strength
    :param defender: Defender's initial strength
    :param a: Attacker's combat attrition rate
    :param d: Defense's combat attrition rate
    :param nona: Offense's non-combat attrition rate
    :param nond: Defense's non-combat attrition rate
    :param thra: Maximum attrition rate threshold for the offense
    :param thrd: Maximum attrition rate threshold for the defense
    :param u:
        Attacker's reinforcement rate:
            [
                (day 1 troop, day 1 Combat factor),
                (day 2 troop, day 2 Combat factor), ...
            ]
    :param v:
        Defender's reinforcement rate:
        [
            (day 1 troop, day 1 Combat factor),
            (day 2 troop, day 2 Combat factor), ...
        ]
    :param time: Battle duration
    :param title: Image title
    :return: Offensive losses, Defensive losses
    """
```

2.3.1 *Strength*

Let:

- ***Strength*** —> χ
- ***Division*** of **troops** —> e
- ***Division*** of **strength** —> s
- ***Troops*** —> δ

$$\chi = \frac{\sum e \cdot s}{\delta}$$

2.3.2 *Combat Effectiveness Factor*

Let:

- ***Strength*** —> χ
- ***Morale*** of **troops** —> γ
- ***Topography modification*** —> t
- ***Experience modification*** —> e
- ***Supply modification*** —> s
- ***Air power modification*** —> a
- ***Tactic modification*** —> τ
- ***Combat effectiveness factor*** —> φ

$$\varphi = \chi^{\gamma} \cdot (1+t) \cdot (1+e) \cdot (1+s) \cdot (1+\tau) \cdot (1+a)$$

2.3.3 *Non-Combatant Attrition Factor*

Let:

- ***Morale*** of **troops** —> γ
- ***Non-combatant attrition baseline*** —> ω_0
- ***Supply modification*** —> s
- ***Air power modification*** —> a
- ***Tactic modification*** —> τ
- ***Combat effectiveness factor*** —> ω

$$\omega = \omega_0^{\gamma} \cdot (1+s) \cdot (1+a) \cdot (1+\tau)$$

2.3.4 *Attrition Rate Threshold*

Let:

- ***Morale*** of **troops** —> γ
- ***Attrition rate baseline*** —> ρ_0

- ***Topography modification*** —> t
- ***Supply modification*** —> s
- ***Air power modification*** —> a
- ***Tactic modification*** —> τ
- ***Attrition rate threshold*** —> ρ

$$\rho = \begin{cases} \rho_0^{\gamma} \cdot (1+s) \cdot (1+a) \cdot (1+\tau) & \text{if the troops are not surrounded} \\ \rho_0^{\gamma} \cdot (1+t) \cdot (1+s) \cdot (1+a) \cdot (1+\tau) & \text{if the troops are surrounded} \end{cases}$$

2.3.5 *Buff*

Let:

- ***Strength buff*** —> e
- National ***specificities*** **effect** —> ε

$$e = 1 + \varepsilon$$

2.3.6 *Lanchester Equation*

Lanchester's equations are a pair of differential equations used to model the dynamics of military combat. They were developed by Frederick William Lanchester in the early 20th century as a way to describe the behavior of opposing forces in battle.

The equations make several simplifying assumptions, including that the opposing forces are homogeneous and that the rate of loss of combat effectiveness is proportional to the number of casualties. Based on these assumptions, the equations predict how the relative strengths of the opposing forces will change over time as the battle progresses.

There are two versions of Lanchester's equations: the linear version, which assumes that combat effectiveness is proportional to the number of units present; and the square-law version, which assumes that combat effectiveness is proportional to the square of the number of units present. The latter version is generally considered to be more realistic, as it takes into account the fact that larger forces are more difficult to defeat.

Lanchester's equations have been used in a variety of contexts, including military strategy, game theory, and economics. Despite their simplicity, they can provide insights into the dynamics of conflict and the factors that determine the outcome of battles.

- Reference
  - ***Lanchester, Frederick William. Aircraft in warfare: The dawn of the fourth arm. Constable limited, 1916.***
  - ***Kress, Moshe. "Lanchester models for irregular warfare." Mathematics 8.5 (2020): 737.***
  - ***Nan, Jiang, et al. "Warfare command decision making analysis of information support based on Lanchester equation." 2010 Chinese Control and Decision Conference. IEEE, 2010.***

Let:

- ***Troops*** of A and B forces on day t —> $A(t)$, $B(t)$

- ***Initial troops*** of A and B forces —> A_0, B_0
- ***Combat effectiveness factor*** of A and B forces —> a, b
- ***Non-combatant attrition factor*** of A and B forces —> α, β
- ***Support Ratio*** of A and B forces —> $u(t)$, $v(t)$

$$\begin{cases} \frac{\mathrm{d}A}{\mathrm{d}t} = -aB(t) - \frac{\alpha A_0}{\mathrm{d}t} + u(t) \\ \frac{\mathrm{d}B}{\mathrm{d}t} = -bA(t) - \frac{\alpha B_0}{\mathrm{d}t} + v(t) \end{cases}$$

2.4 *Intelligence*

The main calculation variables in the political section are:

- Intelligence Action Verdict

[ ]:
```
class Intelligence(object):
    """
    The Intelligence module of the strategic simulation system

    Attributes:
        Intelligence Network Level
        Intelligence Action
        Target
    """
    def __init__(self, network, action, target) -> object:
        """
        This part defines the basic parameters of the intelligence module.

        :param network:
            Intelligence network level in the target country: 0~4 (float)
        :param action:
            Type of intelligence action: obstruction, sabotage, defense (str)
        :param target:
            The country targeted by intelligence action (str)
        """
        ...
    def verdict(self) -> bool:
        """
        This section defines
        the specific mechanics of intelligence action verdict.

        :return: The output: True, False
        """
        ...
def judgment(resa) -> bool:
    """
    After successful obstruction and sabotage operations
    by the intelligence initiating country,
    a secondary determination is required
    if the target country engages in defense of the intelligence.

    :param resa:
        Results of intelligence operations determinations
        by intelligence attacking state: True, False (bool)
    """
    ...
```