Programming 2 Project

The social game system

Ahamed Aryan, Fusco Grazia Gabriela, Karabulut Hayat, Romano Gianmaria

A.Y. 2023/2024

Index

1. Project goal

The goal of the Social Game System (SGS) project is to present a structured game model where players interact within a defined universe. We aimed to develop a simulator in Java to formalize and explore this game model where player interactions are governed by structured relationships within finite, manageable subsets of a larger universe.

2. Basic features

The basic features of the Social Game System can be summarized as follows:

2.1 Universe of players:

The game universe consists of a set U of players, which is numerable. Therefore, players can be counted and listed, even if the total number is infinite.

2.2 Active players:

At any point in time, only a finite subset of players from U are actively participating in the game, so we had to implement methods that include deaths and births.

2.3 Acquaintances and relationships:

Each active player is connected to a subset of other active players, called acquaintances. In specific conditions, two acquaintances can become partners, and, only in that case, the birth of a new player can happen.

2.4 Neighborhoods:

A player’s acquaintances are organized into an array of length k, called the player’s neighborhood.

3. Players

The “Player” class is a fundamental component of the SGS, encapsulating the attributes and behaviors of players within the modeled universe.

3.1 Unique Identifier and Position Checking:

Each player is defined by a unique name, that is set upon the instantiation of a Player object and given by “Player” + a random number.

It is ensured that each player is placed at unique position.

3.2 Characteristics and attributes:

- “age”: represents the age of the player. When the game starts, initial players have a random age. This increases at each iteration. The maximum age that can be reached in the simulation before death is 116 years.
- “gender”: represents the gender of the player. Each player can be male (M) of female (F).
- “mood”: the mood of a player is initialized as random, but can increase or decrease, based on the messages he sends/receive from other players
- “intelligence”: each player has a score, indicating his intelligence
- “charisma”: each player has a score, indicating his charisma

3.3 Acquaintance management:

The “Player” class provides several methods to manage acquaintances, add, break and assess relationships:

- Add acquaintance: adds an acquaintance with 20% probability if both players have not reached their maximum number of acquaintances.

- Break acquaintance: removes an acquaintance if the relationship score is negative for either player
- Relationship score: computes the average relationship score for the player, averaging values in the "relationships" map.

4. Events

The "Events" class is designed to manage various significant events that occur within the simulation involving instances of te "Player" class.

4.1 Birth:

The 'createChild' method handles the creation of a new player as a child of two existing players.

Key points we followed:

- **Preconditions**: Both parents must be partners, be of different genders and be at least 25 years old
- **Process**: A new 'Player' instance is created
    - A new 'Player' instance is created.
    - Initial relationships between the child and both parents are established.
    - The child and parents become mutual acquaintances.
    - The child is also introduced to the acquaintances of both parents.
- **Output**: Logs a message indicating the birth of the child

4.2 Becoming partners:

The 'becomePartners' method establishes a partnership between two players if certain conditions are met.

Key points we followed:

- **Preconditions:** Both players must not have existing partners, must have non-negative relationships and must already be acquaintances
- **Process**: if the conditions are satisfied, the players are set as each other's partners
- **Output**: Logs a message indicating the formation of the partnership

4.3 Introduction of new acquaintance:

The 'introduceAcquaintances' method allows a player to introduce two of their acquaintances to each other.

Key points we followed:

- **Preconditions:** the introducer must have at least three acquaintances
- **Process:**
    - Randomly selects two different acquaintances
    - If these two players are not already acquaintances, they are introduced and become acquaintances
- **Output**: Logs a message indicating the introduction.

4.4 Death:

The 'ageAndCheckDeath' method handles the aging process and checks if a player should die.

Key points we followed:

- **Preconditions:** We check the player's age and the number of acquaintances
- **Process:**
    - If the player's age is above certain thresholds or they have very few acquaintances, the player dies.

        - The player is marked as no longer alive, and all acquaintances and relationships are cleared.
- **Output:** Logs a message indicating the player's death

4.5 Depression and suicide:

The 'checkForDepression' method monitors a player's level of disappointment to check for depression. If a player is highly disappointed, the 'commitSuicide' method is called.

Key points we followed:

- **Preconditions**: The player must be alive and his disappointment level must exceed 1000.
- **Process**:
    - If the player is highly disappointed, there is a random chance to commit suicide.
    - When the suicide happens, the player is marked as no longer alive.
    - All acquaintances and relationships are cleared.
- **Output**: Logs a message indicating the player's suicide

5. Exchanging messages

Players communicate with each other through the exchange of messages. These are sent, received and processed, having an impact on player mood and relationships.

5.1 Sending Messages:

The 'sendMessage' method allows a player to send messages to all their acquaintances. The method calculates the positivity scale of each message, which is indluenced by the sender's mood and charisma, as well as recent received messages.

Key points include:

- Calculation of Positivity Scale:

        - Uses a sigmoid function to normalize the player’s mood.
        - Adds a component based on the player’s charisma.
        - Adjusts based on the positivity of the last received message if any.

- Message Creation and Sending:
    - A ‘Message’ object is created for each acquaintance.
    - The message is added to the sender’s list of sent messages.
    - The message counter for sent messages is incremented.
    - The message is sent to each acquaintance via the ‘receiveMessage’ method.

5.2 Receiving Messages:

The ‘receiveMessage’ method handles the reception of a message by a player, It updates the player’s list of received messages, adjusts the relationship score with the sender, and increases the disappointment level if the sender’s mood is negative.

Key points include:

- Updating Received Messages:
    - Adds the received message to the player’s list of received messages.
    - Increments the received messages counter.
- Adjusting relationships:
    - Initializes a relationship score with the sender if not already present.
    - Updates the relationship score based on the message’s positivity scale and the player’s intelligence.

- Handling disappointment: increases the player's disappointment if the sender's mood is negative.

5.3 Average received and sent positivity scale:

The average positivity scales of both sent and received messages is used to assess the overall sentiment of a player's communications.

Key points include:

- Average received positivity scale: calculates the average positivity scale of all received messages.
- Average sent positivity scale: calculates the average positivity scale of all the sent messages.

5.4 Impact on Mood:

The 'changeMood' method adjusts a player's mood based on the difference between the average positivity scales of sent and received messages.

- Mood calculation:
    - Computes the difference between the average positivity scales of sent and received messages.
    - Updates the player's mood by blending the current mood with the calculated mood change.

6. Graphical User Interface

We provided a graphical user interface for managing and visualizing the simulation. The GUI facilitates user interaction, configuration of simulation parameters and display of simulation results.

6.1 Overall Structure

The 'SimulationGUI' class extends 'JFrame', a top-level container in Swing, and consists of three main panels: the simulation panel, the graph panel and the control panel.

The frame uses a 'BorderLayout' to organize its primary components into distinct regions.

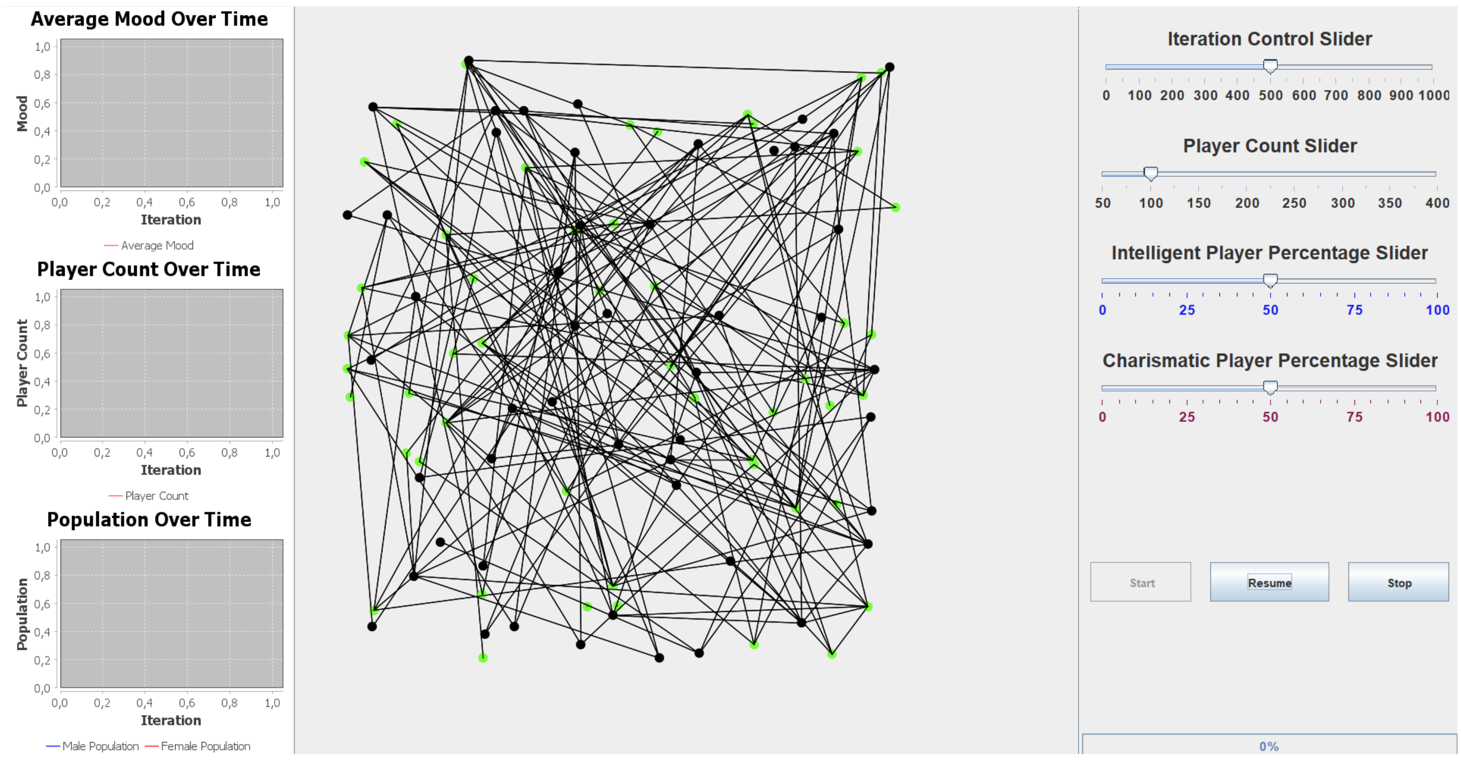


Figure 1: organization of primary components into 3 distinct regions.

6.2 Panels

- **Simulation Panel:** Displays the current state of the simulation, including the players and their interactions.
- **Graph Panel:** Displays various charts related to the simulation data.
- **Control Panel:** Contains controls for adjusting simulation parameters and managing the simulation.

6.3 Control Components

- **Sliders:**
  - Iteration Slider: adjust the number of iterations the simulation will run. It ranges from 0 to 1000, with a default value of 500.

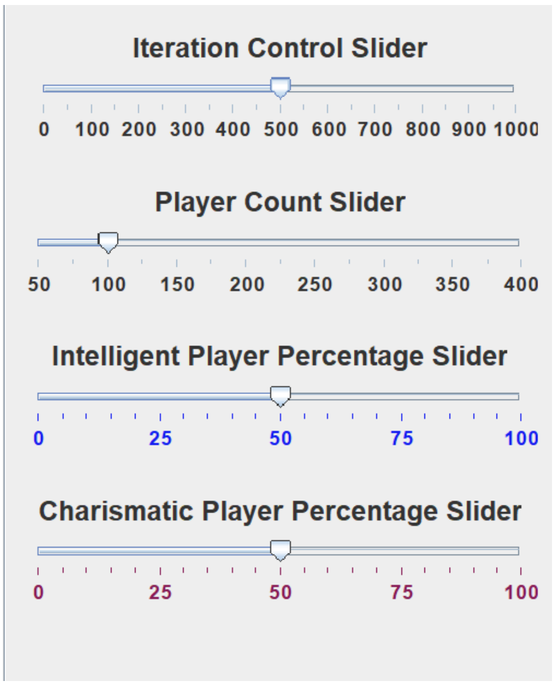


Figure 2: Control sliders.

- Player Count Slider: Controls the number of players in the simulation, ranging from 50 to 400 with a default value of 100.
- Intelligent Player Percentage Slider: sets the percentage of intelligent players ranging from 0% to 100% with a default value of 50%.
- Charismatic Player Percentage Slider: sets the percentage of charismatic players, ranging from 0% to 100% with a default value of 50%.

- **Buttons:**
  - Start Button
  - Stop Button
  - Pause-resume Button

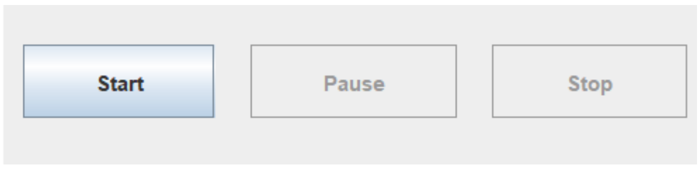


Figure 3: the stop and pause buttons are initially disabled.

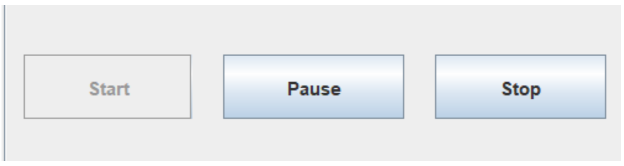
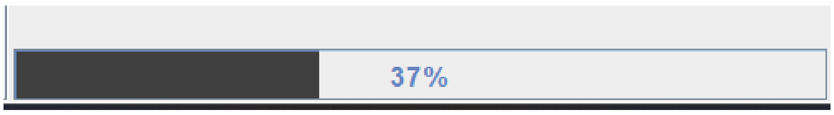


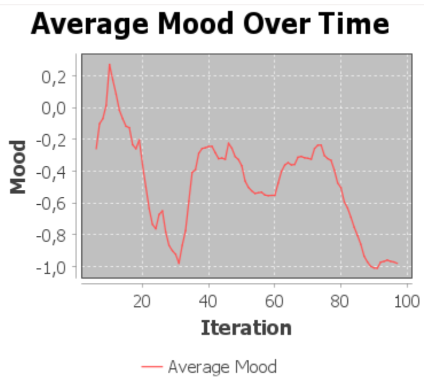
Figure 4: the start button disables itself and enables the stop and pause buttons.

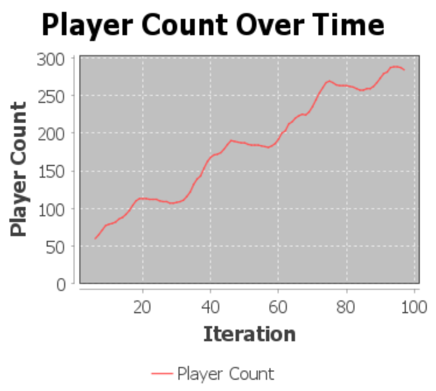
- **Progress Bar**
  - Purpose: displays the progress of the current simulation iteration.
  - Configuration: dynamically adjusts its maximum value based on the iteration slider’s value.

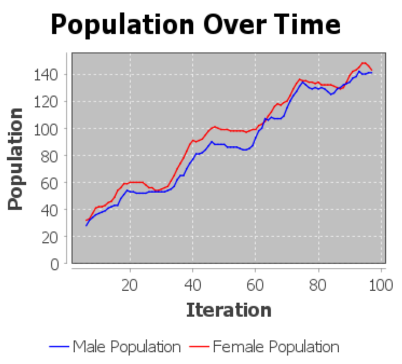


6.4 Chart Integration

- **Chart Panels:** The GUI integrates charts created using JFreeChart into the graph panel. These charts visualize data such as player count and population trends.
- **Dynamic addition:** Charts are added to the graph panel dynamically during runtime using various methods.







6.5 Font and Styles

- **Font**: components use the “Roboto” font for labels and “Helvetica” for sliders, providing a clean appearance
- **Font Size**: it’s used a consistent font size of 20.

6.6 Functionality

- **Start simulation:**
  The start button initializes a new simulation instance with the current slider settings and begins the simulation. It also disables itself to prevent multiple simulations from running simultaneously.
- **Pause/resume simulation:**
  The pause button toggles the simulation state between paused and running. It changes its label to indicate the current action (Pause or Resume).
- **Stop simulation:**

The stop button terminates the simulation and resets the control buttons to their initial state.

- **Update mechanism:**
  The ‘updateData’ method updates the ‘SimulationPanel’ with the latest player data from the simulation. It also updates the progress bar to reflect the current iteration.
- **Thread safety:**
  The ‘updateData’ method uses “SwingUtilities.invokeLater” to ensure updates are performed on the Event Dispatch Thread, maintaining thread safety.

7. Results and particular cases

7.1 Real-World Simulation

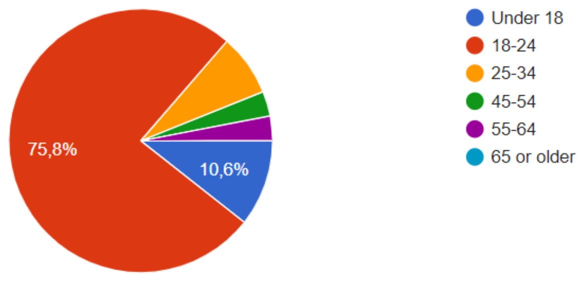
The simulation is designed to be an alias of real-world dynamics, capturing complex interactions and emergent behaviors. It provides insights into how certain parameters affect the overall system, akin to observing real-life social and behavioral trends.

7.2 Survey to validate simulation results.

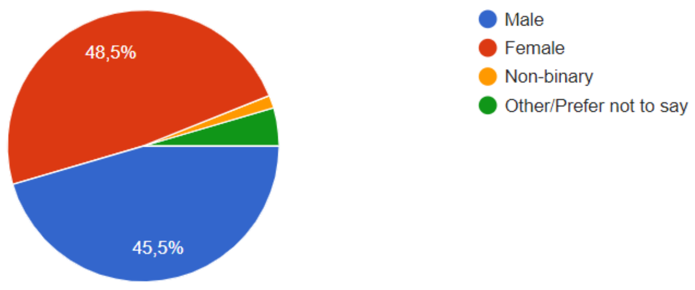
To determine whether the results of our simulation align with real-world statistics, we decided to conduct a survey. The primary aim was to compare the simulated data with actual human experiences and moods, thereby assessing the accuracy and validity of our simulation.

- Demographic information: To ensure that the survey results were representative, we collected demographic data such as age, gender, occupation and country of residence.

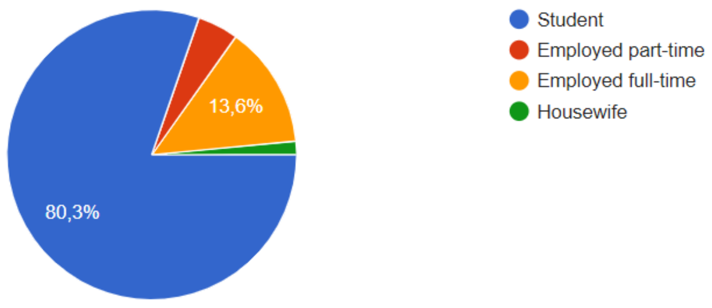
What is your age?



What is your gender?



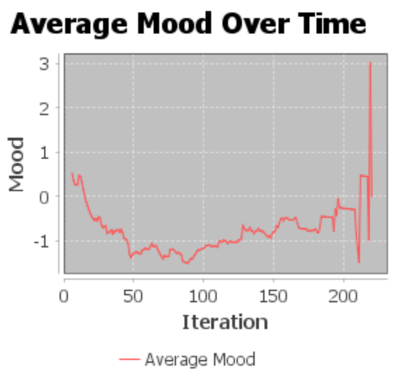
What is your current occupation?



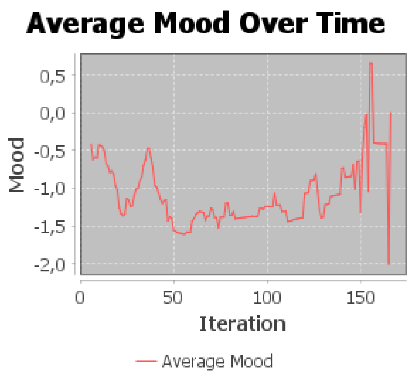
- Results: the average mood score from the survey is consistent with the neutral mood predicted by our simulation. The frequency distribution of mood reported a neutral mood, closely matching our simulation results.

7.3 Mood balance

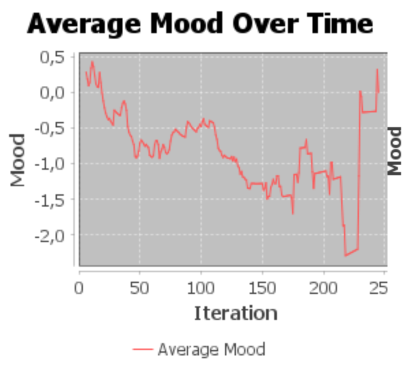
One of the interesting observations from the simulation is the balance of mood among players. The final mood of the players ranges from -1 to 0, simulating a realistic spectrum of negative to neutral emotions. Despite the randomness in interactions and initial conditions, the simulation tends to maintain a balance, reflecting the inherent stability observed in real-world populations.



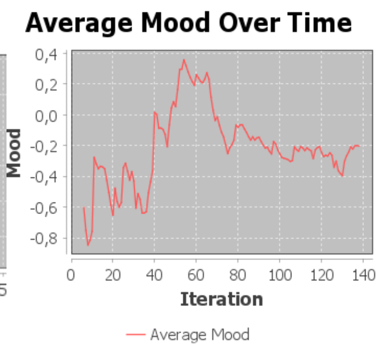
1st simulation’s average mood



2nd simulation's average mood

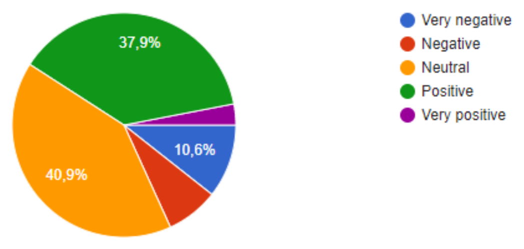


3rd simulation's average mood



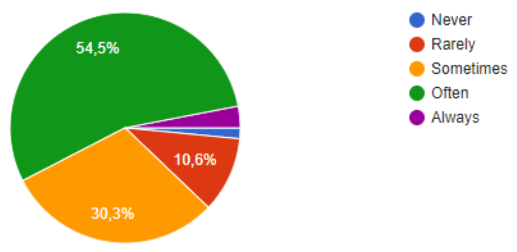
4th simulation's average mood

How would you describe your current mood?



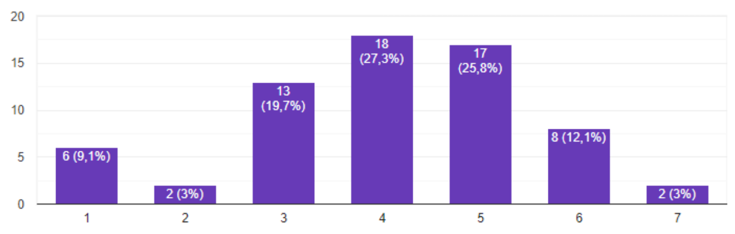
Real-world's statistics

How often do you feel in a neutral mood in a typical week?



Real-world's statistics

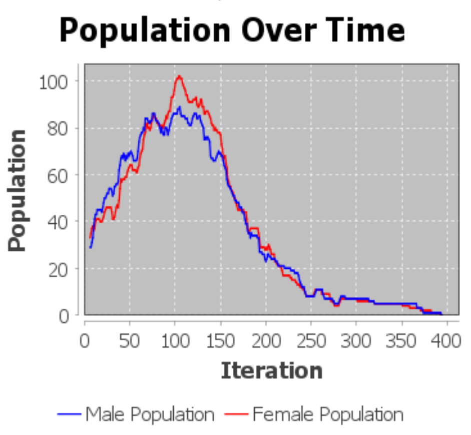
Over the past week, how have you generally felt? (1=Extremely negative; 7=Extremely



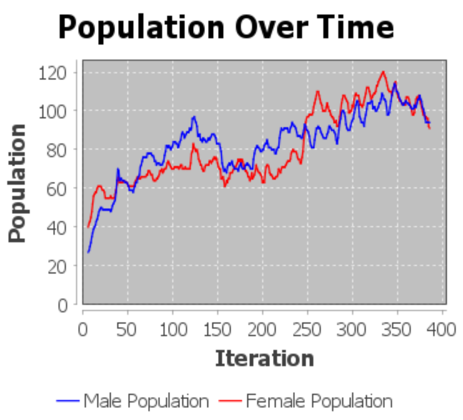
Real-world's statistics

7.4 Gender balance

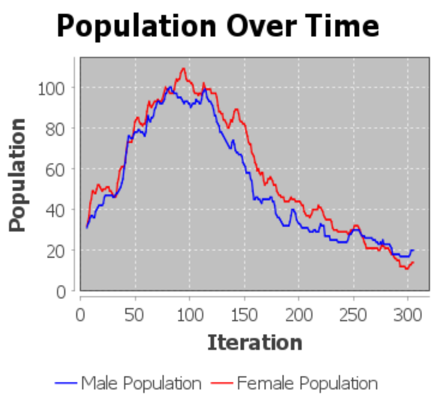
Another notable result is the balance between the number of male and female players in the simulation. Despite the random assignment of genders, the system exhibits a natural equilibrium. This mirrors real-world demographics where, over large populations and extended periods, gender ratios tend to stabilize around an equilibrium point.



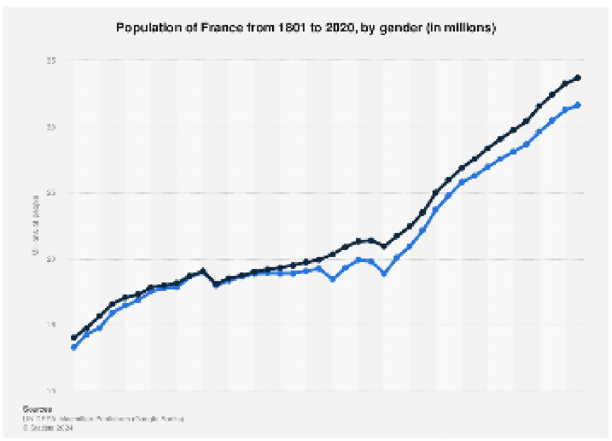
Simulation's graph



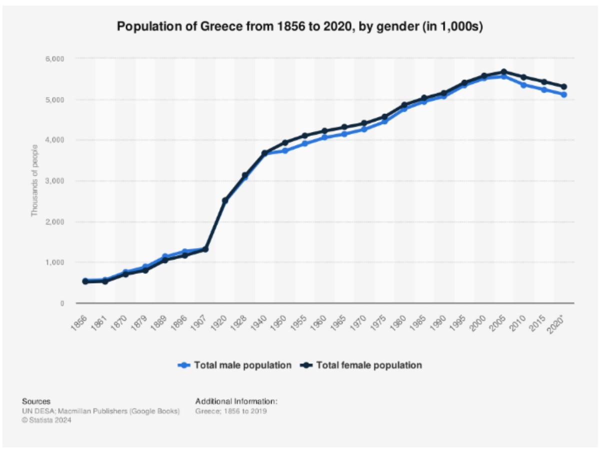
Simulation's graph



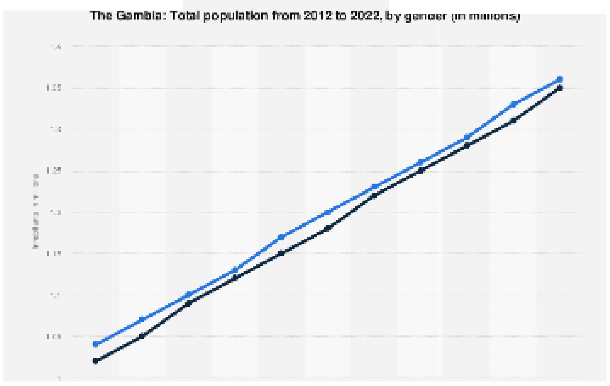
Simulation's graph



Real world's graph



Real world's graph



Real world's graph

7.5 Randomization and emergent patterns

Although the simulation parameters and initial conditions are randomized, emergent patterns and trends can be observed. These patterns are not explicitly programmed but arise from the interactions of individual agents, highlighting the power of simulations to reveal complex dynamics from simple rules.

8. Conclusion

We have been able to provide a comprehensive and user-friendly interface for controlling and visualizing this simulation, that offers valuable insights into real-world dynamics, maintaining a balance of mood and gender distribution despite randomization, showcasing the robustness and reliability of the system in mimicking real-life scenarios.

Thank you for reading.

This project was made by:

Ahamed Aryan

ahamed.2126202@studenti.uniroma1.it

Fusco Grazia Gabriela

fusco.2043561@studenti.uniroma1.it

Karabulut Hayat

karabulut.2110120@studenti.uniroma1.it

Romano Gianmaria

romano.2105539@studenti.uniroma1.it