Social Game System: Project Report

Introduction

Social Game System is a Java-based game for simulation. It places the players in a dynamic situation through an ancient Egyptian social hierarchy. The game simulates social activities, interactions, and life cycles, where the players take up roles from pharaoh down to slave. The game works on rounds whereby players grow, interact, and get life events such as achievements, births, and deaths. This report also provides a detailed description of the mechanics of the game, the primary features, and the functionality of the game classes.

Game Overview and Mechanics

Gameplay Dynamics

In the **Social Game System**,each player is assigned some role in the society; these roles have some features and likelihood of life events. Each round within the game is supposed to simulate a decade in our player's life. Players age, attempt to have children, possibly get promoted, and may die. The latter is based on predetermined probabilities that are associated with one's role and age.

- **Roles and Hierarchies**: The game starts by default with 7 players, one for each role: Pharaoh, Priest, Noble, Scribe, Merchant, Farmer, or Slave. Each role has its color and a corresponding position in the social hierarchy, with Pharaoh being the highest and Slave being the lowest.
- **Aging and Life Events**: As the game progresses, players age by 10 years at the end of each round. Aging influences their likelihood of dying or getting promoted. For example, a player who reaches 90 years automatically dies of old age. The likelihood of promotion and survival varies by role, with lower-ranked roles like Slaves having higher death probabilities and lower chances of promotion.
- **Interactions and Promotions**: Players interact with their neighbors which are players having the same role, their children, or players having 1 higher grade and 1 lower grade. These interactions can result in promotions if certain conditions are met. For example, a Farmer might be promoted to Merchant if they survive and meet the probability threshold.
- **Birth and Death**: Players can have children, which are new players added to the game with the same role as the parent. However, not all attempts at childbirth are successful, and not all players survive each round. Death can occur due to old age or role-specific

probabilities, with the Pharaoh having the lowest probability of death due to role-specific events.

Game Flow

The game goes in cycles, with each cycle following these step:

- **Aging**: Each player's age is increased by 10 years.
- **Reproduction**: Players attempt to reproduce, allowing new players to enter the game.
- **Death Trials**: Players undergo death trials due to growth or possibility based on activity.
- **Promotion**: Eligible players are considered for promotion to a higher social role. Pharaonic Succession: When a Pharaoh dies, a new Pharaoh is chosen based on specific criteria, such as the oldest child or the highest ranking living person.
- **Update neighbors**: Player neighbors are recalculated based on the current game state.
- **Board summary**: All player statuses are printed, showing current social settings and player status.

User Interaction

Players can interact with the game through a menu-driven interface, allowing them to:

1. Continue to the next round.
2. Add new players to the game.
3. Check the player's neighbors and their status, this also provides a board showing the status of all the players still alive.
4. Look at the running threads associated with the game (each player runs on a different thread).
5. Finish the game.

Study Description

Thefunctionalities of the game system are implemented through several interconnected classes, each of which controls different aspects of the game:

Player Class

The `Player` class represents an individual player within the game, encapsulating properties such as `nickname`, `role`, `age`, `alive` status, and a list of `children`. Players are managed as separate threads (`Runnable`) to simulate parallel interactions and events.

- **Attributes**: The class includes attributes like `nickname`, `role`, `age`, `alive`, `children`, and `neighbors`. It also includes constants for ANSI color codes to represent different roles visually in the console output.
- **Methods**: Key methods include `increaseAge()`, `tryMakeChild()`, `tryDeath()`, `tryHigherGrade()`, `promoteToPharaoh()`, and `updateNeighbors()`. These methods control the player's lifecycle, including aging, reproduction, death, and promotion.
- **Thread Management**: The `Player` class implements the `Runnable` interface, allowing each player to run on a separate thread. Methods such as `pause()`, `resume()`, and `run()` manage thread execution.

Game Class

The `Game` class manages the overall flow of the game, including player initialization, round execution, and user interactions.

- **Attributes**: The class maintains a list of players, a list of player threads, the current round number, and a reference to the `Pharaoh` in charge. It also includes a `counterNames` map to track the number of players per role.
- **Methods**: The `initialize()` method sets up the initial game state, including player creation and neighbor updates. The `executeRound()` method progresses the game by aging players, handling births, checking for deaths, and managing promotions. Other methods include `addPlayer()`, `chooseNewPharaoh()`, `updateNeighborsAll()`, `printPlayers()`, and `removeDeadPlayers()`.
- **Menu**: The `menu()` method provides a user interface for interacting with the game, allowing players to start rounds, add new players, and view game statistics.

Role Class

The `Role` class provides utility functions related to player roles, including the social hierarchy, role transitions, and associated colors.

- **Methods**: The `nextGrade()` method returns the next role in the social hierarchy. The `isNext()` method checks if two roles are adjacent in the hierarchy. The `getGradeIndex()` method returns the index of a role in the hierarchy, and the `getColorRole()` method returns the ANSI color code for a specific role.

Main Class

The `Main` class serves as the entry point for the application, instantiating the `Game` object and calling the `menu()` method to start the game.

Conclusion

The social game design is a detailed simulation of ancient Egypt society, providing insights into social dynamics, life spans and generational transitions.

Through round-based design and execution control this game provides an engaging game and tutorials for players interested in the social order of ancient civilizations. The design of the game, from the multi-threaded player interactions to the exploit of role-based life events, showcases the complexity and interconnectivity of societal roles in a historical context