Steven Foltz
CS230
05/26/24

Executive Summary

The Gaming Room has engaged Creative Technology Solutions (CTS) to create an online version of its popular Android game, Draw It or Lose It. This game, inspired by the 1980s television show Win, Lose, or Draw, pits teams against each other to guess a riddle based on stock drawings generated by the application. The web-based version must support various platforms and meet a number of software requirements, such as the ability to have several teams and players, unique game and team names, and only one instance of the game in memory at any given moment. This document describes the design constraints, domain model, and early software architecture for meeting these requirements.

Design Constraints

Developing the game application in a web-based distributed environment imposes several design constraints:

Concurrency Management: The application must handle multiple users simultaneously. This requires careful management of concurrent data access and modifications.

Scalability: The game should be able to scale to accommodate varying numbers of users and teams without performance degradation.

Uniqueness of Identifiers: Unique identifiers for games, teams, and players must be managed efficiently to ensure data integrity and prevent conflicts.

Single Instance Enforcement: Implementing a singleton pattern to ensure that only one instance of the game service exists at any time.

Platform Independence: The application must be accessible across various devices and operating systems, necessitating a web-based solution with responsive design.

User Experience: The application must provide a seamless and intuitive user experience, which requires thoughtful UI/UX design.

These constraints influence the development process by requiring robust backend architecture, efficient data handling mechanisms, and responsive frontend design to provide a consistent and reliable user experience across different platforms.

Domain Model

The UML class diagram for the game application depicts the basic components and their relationships:

Classes and Relationships

Entity: A base class with the common attributes id and name. This class is shared by the Game, Team, and Player classes.

Game: inherits from the Entity. Manages teams and game states.

Team inherits from Entity. manages players.

Player inherits from Entity. Represents individual team members.

GameService is responsible for managing game instances, teams, and players. Uses the singleton technique to assure a single instance in memory and includes methods for adding and retrieving games and teams.

Object-Oriented Programming Principles.

Inheritance: The Entity class serves as a foundation for the Game, Team, and Player classes, allowing for code duplication and simplicity.

Encapsulation: Each class's attributes are private, but they may be accessed and modified using public getter and setter methods, ensuring controlled data access.

The Singleton Pattern ensures that only one instance of the GameService class exists at any given moment, which is crucial for game state management and conflict prevention.

Iterator Pattern: The addGame() and getGame() functions iterate across collections of games, teams, and players to ensure unique names and get specific instances.

The UML diagram (not shown here but referenced) visually depicts these linkages and concepts, serving as a clear blueprint for the application's design.