Sanctuary City Project Final Summary

Group 11 - Michael Rosas, Zeehan Sharif, Fernando Caudillo Tafoya

Sanctuary City is a web-based simulation/management game where players manage a sanctuary that cares for fantasy creatures. While the main gameplay focuses on caring for creatures, managing habitats, and completing story-driven quests, the system is designed to be modular, scalable, and responsive. Building on the foundations of player profiles, species management, and resource tracking, the design uses these concepts for an architecture that is maintainable and capable of supporting future expansions.

At the center of the system is the SimulationEngine, which drives a fixed one hertz tick loop. Each tick updates creature needs, resource consumption, environmental conditions, and quest progress, which ensures the simulation behaves consistently and predictably. The TickClock component manages the timing of each tick, preventing drift, supporting debugging, and allowing controlled adjustments to the simulation speed. WorldState stores a complete snapshot of the player's sanctuary at any given time. Focusing on mutable game state in this way ensures consistency across subsystems, and also makes persistence/autosave logic much more simple, plus it enables reliable crash recovery using snapshot restoration. This structure also makes it easier to implement undo and redo operations, which can be useful for both players and developers during testing.

The domain layer models the core entities within the sanctuary ecosystem. Creatures are represented as objects that contain their needs, traits, and a reference to the habitat they live in. Species behaviors such as feeding, healing, and training patterns, are managed using the strategy pattern. This allows for new behaviors or species to be added without changing the core engine. Habitats manage environmental resources, ward strength, and conditions that directly influence creature well-being. Resources and inventories are represented as structured objects with controlled mutation to maintain invariants and ensure simulation that is deterministic. Quests are tracked as separate objects that keep track of objectives, progress, and completion state. These quest objects interact with domain entities through event signals instead of direct modification, allowing new content to be added more conveniently.

A critical part of the design is the separation of concerns between simulation, service, and interface layers. Services such as PersistenceService, AuthService, NotificationService, LocalizationService, and QuestEngine handle communication between the simulation core and external systems like cloud storage and user interface. Persistence uses a repository pattern with write-behind caching, batching updates and synchronizing with the server asynchronously. This approach supports offline play while making sure that conflicts between client and server states are resolved easily. Cosmetic or UI changes are distinguished from significant data, so that integrity is maintained. Event-driven notifications implement an observer pattern, which allows user interface components to react in real time to changes with creature needs, habitat events, or quest triggers.

The user interface is designed to be reactive and loosely coupled from the simulation. Components such as SanctuaryView, CreaturePanel, HabitatBuilder, QuestLog, and HUD

subscribe to events and update automatically when changes occur in the underlying state. Controllers and lightweight view models mediate user interactions, ensuring that the gameplay logic remains within the domain and service layers. This separation allows the interface to remain responsive, supports accessibility adjustments and localization, and prevents inadvertent mutations of core game state from user interactions. The interface also provides immediate feedback for player actions, such as visual alerts when creatures become hungry or when habitat wards weaken.

Additional considerations include startup and shutdown sequences, crash recovery, and hardware mapping. When the game starts, it loads authenticated player profiles, warms caches, and initializes the tick loop. During shutdown, pending operations are flushed and the current state is saved automatically. Crash recovery relies on frequent snapshots of WorldState to allow seamless continuation after unexpected interruptions. On lower-end systems, the simulation can degrade gracefully by capping active creatures or simplifying visual updates, while maintaining deterministic simulation behavior. Security measures, including authentication, rate-limiting, and encrypted storage of player data, are implemented even though sensitive personal data is minimal. This ensures that both player progress and game integrity are protected.

Design patterns play an important role in maintainability and extensibility. The state pattern underpins WorldState for managing mutable game state, while strategy patterns allow species-specific behaviors to be plugged into the Creature.tick function. Factory methods are used to make creation of domain entities abstract. Pub-sub architecture keeps the simulation and user interface decoupled, while repository persistence abstracts the details of local or cloud storage. The system is organized into five main subsystems: the simulation core with tick and timing mechanisms, domain entities with behaviors and relationships, quests and derived state, supporting services including persistence and authentication, and the user interface layer.

Overall, Sanctuary City’s final design focuses on deterministic simulation, modular domain logic, reactive user interfaces, and flexible services. By implementing these elements with certain strategies, event notifications, and content expansion mechanics, the system supports future addition of species, habitats, quests, and more advanced gameplay features. The architecture balances technical aspects with the educational and entertainment goals of the game which provides a solid foundation for maintainable development, consistent game state, and a good player experience that is also fun for the people that play it.