**1. Use Case Diagram**

**Actors:**

* **Player**: The user who plays the game.
* **Server System**: Handles backend processes like authentication, data storage, and retrieval.
* **Event Coordinator**: Manages event-specific levels and content.

**Use Cases:**

1. **Play Base Level**: Players select and play base levels accessible from any location.
2. **Unlock Location-Specific Level**: Players unlock levels by visiting specific campus locations using geolocation.
3. **Collect Trinket**: Players collect trinkets within levels to add to their inventory.
4. **View Leaderboard**: Players view leaderboards to see top-ranking players.
5. **Multiplayer Interaction**: Players interact and communicate in real-time within the same map.
6. **Account Management**: Players create accounts, log in, and manage their profiles.
7. **Add Friend**: Players add friends and view their progress.
8. **Participate in Event Levels**: Players access time-limited event levels to encourage social interaction.

**Relationships:**

* The **Player** initiates all use cases.
* **Server System** supports use cases by providing necessary backend functionality.
* **Event Coordinator** interacts with **Participate in Event Levels** to manage special events.

**2. Class Diagram**

**Classes and Their Details:**

**Player**

* **Attributes:**
  + playerID: Unique identifier for the player.
  + username: Player's chosen display name.
  + email: Player's email address.
  + password: Encrypted password for authentication.
  + inventory: List of Trinket objects collected.
  + friendsList: List of Player objects added as friends.
  + location: Current geolocation data.
  + rank: Player's rank on the leaderboard.
* **Methods:**
  + register(): Create a new account.
  + login(): Authenticate and access the game.
  + logout(): End the session securely.
  + move(): Control character movement.
  + jump(): Make the character jump.
  + collectTrinket(trinket): Add trinket to inventory.
  + addFriend(player): Send a friend request.
  + viewLeaderboard(): Display the leaderboard.
  + joinMultiplayerSession(session): Enter a multiplayer game.
  + sendMessage(message): Communicate in chat.

**GameLevel**

* **Attributes:**
  + levelID: Unique identifier for the level.
  + levelName: Name of the level.
  + levelType: Type of level (base, location-specific, event).
  + mapData: Data structure containing level design.
  + trinkets: List of Trinket objects within the level.
  + isUnlocked: Boolean indicating if the level is accessible to the player.
* **Methods:**
  + loadLevel(): Initialize the level for play.
  + unlockLevel(playerLocation): Unlock level based on player's location.
  + startLevel(): Begin gameplay.
  + endLevel(): Finish gameplay and record results.

**Trinket**

* **Attributes:**
  + trinketID: Unique identifier for the trinket.
  + name: Name of the trinket.
  + description: Description of the trinket.
  + funFact: Educational or interesting fact related to the trinket.
  + position: Coordinates within the level.
* **Methods:**
  + collect(): Trigger collection by player.
  + displayFunFact(): Show the fun fact to the player.

**Leaderboard**

* **Attributes:**
  + rankings: Sorted list of Player objects based on score or achievements.
* **Methods:**
  + updateRankings(player): Recalculate rankings when a player achieves a new score.
  + display(): Show the leaderboard to the player.

**MultiplayerSession**

* **Attributes:**
  + sessionID: Unique identifier for the multiplayer session.
  + players: List of Player objects in the session.
  + chatLog: Record of messages sent during the session.
* **Methods:**
  + startSession(): Initialize a new multiplayer session.
  + endSession(): Terminate the session.
  + sendMessage(player, message): Handle message sending.
  + receiveMessage(): Display incoming messages to players.

**Server**

* **Attributes:**
  + databaseConnection: Connection to the game's database.
  + activeSessions: List of current multiplayer sessions.
* **Methods:**
  + authenticateUser(email, password): Verify login credentials.
  + savePlayerData(player): Store player data.
  + retrievePlayerData(playerID): Load player data.
  + manageSession(sessionID): Oversee multiplayer sessions.

**GeolocationService**

* **Attributes:**
  + currentLocation: Player's current geolocation data.
* **Methods:**
  + getLocation(): Retrieve geolocation data.
  + isAtLocation(targetLocation): Check if the player is within acceptable range.
  + unlockLevels(player): Unlock levels based on location.

**EventCoordinator**

* **Attributes:**
  + eventLevels: List of special event levels.
  + schedule: Timing details for events.
* **Methods:**
  + createEventLevel(levelData): Design new event levels.
  + scheduleEvent(levelID, timeFrame): Set timing for event levels.
  + notifyPlayers(): Inform players of upcoming events.

**Relationships:**

* **Player** interacts with **GameLevel**, **Trinket**, **Leaderboard**, **MultiplayerSession**, and **GeolocationService**.
* **GameLevel** contains multiple **Trinkets**.
* **MultiplayerSession** includes multiple **Players**.
* **Server** facilitates communication and data exchange among classes.
* **EventCoordinator** manages special **GameLevels** and interacts with the **Server**.

**3. Sequence Diagrams**

**A. Unlock Location-Specific Level**

**Actors and Objects:**

* **Player**
* **Client Application**
* **GeolocationService**
* **Server**
* **GameLevel**

**Flow:**

1. **Player** selects "Location Levels" from the menu.
2. **Client Application** requests geolocation access.
3. **Player** grants permission.
4. **Client Application** calls GeolocationService.getLocation().
5. **GeolocationService** retrieves currentLocation.
6. **Client Application** sends currentLocation to **Server**.
7. **Server** calls GeolocationService.isAtLocation(targetLocation) for available levels.
8. **Server** identifies unlockable levels and updates GameLevel.isUnlocked.
9. **Server** sends list of unlocked levels to **Client Application**.
10. **Client Application** updates the UI to display unlocked levels.
11. **Player** selects a level to play.
12. **Client Application** calls GameLevel.loadLevel().
13. **GameLevel** is loaded and gameplay begins.

**B. Collect Trinket**

**Actors and Objects:**

* **Player**
* **GameLevel**
* **Trinket**
* **Client Application**
* **Server**

**Flow:**

1. **Player** navigates through the **GameLevel**.
2. **Player** approaches a **Trinket**.
3. **Client Application** detects collision with **Trinket**.
4. **Trinket.collect()** is invoked.
5. **Trinket** is added to **Player.inventory**.
6. **Trinket.displayFunFact()** shows fun fact to **Player**.
7. **Client Application** sends updated **Player.inventory** to **Server**.
8. **Server.savePlayerData(player)** stores the updated inventory.
9. **Client Application** updates the on-screen inventory display.

**C. Multiplayer Interaction**

**Actors and Objects:**

* **Player**
* **MultiplayerSession**
* **Other Players**
* **Server**

**Flow:**

1. **Player** selects multiplayer mode.
2. **Client Application** requests to join a **MultiplayerSession**.
3. **Server** adds **Player** to the session and updates session.players.
4. **Player** enters the game environment with other **Players**.
5. **Players** interact within the game (movement, chat).
6. **Player** sends a message via sendMessage(message).
7. **Server** broadcasts the message to other **Players** in the session.
8. **Other Players** receive and display the message via receiveMessage().
9. Gameplay continues until **Player** leaves or session ends.
10. **Server** updates session state and saves any progress.

**4. Package Diagram**

**Packages:**

1. **UserInterface**
   * Classes: MainMenu, GameHUD, InventoryScreen, LeaderboardScreen
   * Responsibilities: Display visuals, capture user input, navigate between screens.
2. **GameLogic**
   * Classes: PlayerCharacter, Enemy, PhysicsEngine
   * Responsibilities: Handle game mechanics, character controls, physics calculations.
3. **LevelManagement**
   * Classes: GameLevel, LevelLoader, LevelEditor
   * Responsibilities: Load and save levels, manage level data, provide tools for level creation.
4. **DataAccess**
   * Classes: DatabaseConnector, PlayerDataDAO, LevelDataDAO
   * Responsibilities: Interact with the database, perform CRUD operations.
5. **Networking**
   * Classes: ServerCommunication, MultiplayerSession, ChatHandler
   * Responsibilities: Manage network connections, handle multiplayer interactions.
6. **Authentication**
   * Classes: LoginManager, RegistrationHandler, PasswordEncryptor
   * Responsibilities: Authenticate users, manage sessions, ensure security.
7. **Geolocation**
   * Classes: GeolocationService, LocationVerifier
   * Responsibilities: Access geolocation data, verify player location.
8. **EventManagement**
   * Classes: EventCoordinator, EventScheduler
   * Responsibilities: Manage event-specific content and timing.
9. **Utilities**
   * Classes: Logger, ConfigManager, Constants
   * Responsibilities: Provide common utilities and configuration management.

**Dependencies:**

* **UserInterface** depends on **GameLogic** for displaying game elements and on **DataAccess** for displaying stored data.
* **GameLogic** interacts with **LevelManagement** to retrieve level data.
* **Networking** relies on **DataAccess** for synchronization of data across clients.
* **Authentication** uses **DataAccess** to verify user credentials.
* **Geolocation** works with **LevelManagement** to unlock location-specific levels.
* **EventManagement** coordinates with **LevelManagement** and **Networking** to schedule events and notify players.

**5. Activity Diagram for "Playing a Location-Specific Level"**

1. **Start**
2. **Player logs in**.
3. **Player selects "Location Levels"**.
4. **System requests geolocation access**.
5. **Player grants access**.
6. **System retrieves current location**.
7. **System checks for levels available at location**.
   * If levels are available:
     + **System unlocks levels**.
     + **Player selects a level**.
     + **System loads the level**.
     + **Gameplay starts**.
     + **Player plays the game**.
     + **Player collects trinkets and gains points**.
     + **Upon completion, system updates leaderboard**.
   * If no levels are available:
     + **System notifies player**.
8. **End**

**6. State Machine Diagram for the Player Character**

**States:**

* **Idle**: Standing still, awaiting player input.
* **Running**: Moving horizontally.
* **Jumping**: In upward motion after a jump command.
* **Falling**: Descending after reaching jump apex or walking off a platform.
* **Collecting**: Interacting with a trinket.
* **Interacting**: Engaging with environment elements or other players.
* **Respawning**: Returning to a starting point after losing a life.

**Transitions:**

* **Idle** to **Running**: When movement input is detected.
* **Running** to **Jumping**: When jump input is received.
* **Jumping** to **Falling**: After upward momentum is exhausted.
* **Falling** to **Idle**: Upon landing on a surface.
* **Any State** to **Collecting**: When colliding with a trinket.
* **Any State** to **Interacting**: When interacting with an object or player.
* **Any State** to **Respawning**: Upon losing a life.

**7. Component Diagram**

**Components:**

* **Client Application**
  + Interfaces: User Interface, Game Engine
  + Dependencies: Requires connection to Server.
* **Server**
  + Services: Authentication Service, Data Service, Multiplayer Service
  + Interfaces: REST API, WebSocket for real-time communication.
* **Database**
  + Contains: User Data, Game Levels, Trinkets, Leaderboards
  + Interfaces: Accessed by Server's Data Service.
* **Geolocation API**
  + Provides: Location data to Client Application
  + Interfaces: Standard geolocation interfaces on the device.

**Connections:**

* **Client Application** communicates with **Server** via HTTP/HTTPS and WebSockets.
* **Server** interacts with **Database** through secure connections.
* **Client Application** accesses **Geolocation API** to retrieve location data.
* **Server** sends updates to **Client Application** for real-time events and multiplayer synchronization.

**8. Deployment Diagram**

**Nodes:**

* **Client Device**
  + Components: Client Application, Geolocation API
  + Devices: Smartphones, Tablets, Computers
* **Server Cluster**
  + Components: Application Server, Database Server
  + Technologies: AWS EC2 Instances, Load Balancers
* **Network**
  + Connections: Internet
  + Protocols: HTTPS for secure communication, WebSockets for real-time data.

**Deployment Configuration:**

* **Client Devices** run the game application locally, requiring internet access for server communication.
* **Server Cluster** handles backend processes, scalable to manage varying loads.
* **Database Server** securely stores persistent data, with backups and redundancy.