BABKA

Design document

Written by: Alexander Kim

Jan 30th

Briefing:

This design encapsulates the entire game into 4 main sections. The goal of the design is set up structure to enable us to work independently for a while. By using dummy classes at critical connections, we can work on fleshing out some internal structure of our code.

> *Directories should be restructured to follow this design encapsulation*

> *Should we use some ID structure or use pointers?*

1: Client interface:

- Focused on UI
- Interprets command actions
- Tracks and updates the relationship (network ID : users)
    - Section communication dummies needed for:
    - Requesting selective data from the world class
- Notable Classes:
    - Commander
    - User_manager

2: World management:

- Focused on managing command-directed changes to the world state
- Changes to the world state (by calling on the managers)
- Tracks and updates the relationship (user: avatar : room)
- Section communication dummies needed for:
    - Requesting selective data from the avatar_m and the area_m
- Notable Classes:
    - World
    - <something to track (avatar:room) >

3: Avatar_manager data

- Responsible for creating, storing, and updating all avatar information
- *Stores a CLONE of objects in avatar inventory
- No dummies needed to communicate with world class
- Notable Classes:
    - Avatar
    - object

4: Area_manager data

- Responsible for creating, storing, and updating all room information
- Reads and creates areas and rooms from JSON
- Internally resets objects
- No dummies needed to communicate with world class
- Notable Classes:
    - <something to Read json>
    - Area
    - Room, Door, Object, NPCs
    - etc.

1: Client interface:

- Focused on UI
- Interprets and identifies command actions
- Tracks and updates the relationship between network ID and users
    - Section communication dummies needed for:
    - Requesting selective data from the world class

MudServer.cpp

Relies on:

- Commander.cpp

For any command:

- 1. Takes input
- 2.1. If (server command) Manages shut-down and disconnects
- 2.2. Else if (game command) Calls on commander to process the command
- // end

Commander.cpp

Relies on:

- **user_manager**.cpp
- **World**.cpp

For login: {create, login!}

- 1. Calls on **user_manager**:
    - SEND network ID and credentials
    - **User_manager**:
        - Creates a new user_class || authenticates login credentials
        - Updates relationship (user ID:network ID)
    - RETURN bool status
- 2. Calls on **world.**cpp: DUMMY FUNCTIONALITY
    - SEND user ID
    - **World.**cpp:
        - Create a new avatar || request a selection from already created avatars
        - Updates internal relationship (avatars:user)
    - RETURN bool status
- 3. Updates the client with success/fail
- // end

For communications {say, yell, tell}

- 1. Calls on **user_manager**:
  - SEND network ID
  - RETURN user ID
- 2. Calls on **world.**cpp: DUMMY FUNCTIONALITY
  - SEND user ID
  - **World.**cpp:
    - Matches the user ID to avatar ID
    - Identifies all other avatar IDs to communicate onto
  - RETURN list of all user IDs to send message to
- 3. Calls on user_manager:
  - SEND list of user IDs
  - RETURN list of network IDs
- 4. Updates the client with message
- //end

For movement [Go {north, south, west, east, etc.}]

- 1. Calls on **user_manager:**
  - SEND network ID
  - RETURN user ID
- 2. Calls on **world.**cpp: DUMMY FUNCTIONALITY
  - SEND network ID
  - **World.**cpp:
    - Matches the user ID to avatar ID
    - Identifies the room the avatar is in
    - Calls on room_manager.cpp to return the new room
  - RETURN string: new room description (or a “can’t move there!” message)
- 4. Updates the client with STRING message
- //end

For other commands like “look” or “attack X”

- 1. Calls on **user_manager**:
  - SEND network ID
  - RETURN user ID
- 2. Calls on **world.**cpp: DUMMY FUNCTIONALITY
  - SEND network ID
  - **World.**cpp:
    - Changes the world as needed
  - RETURN success || response string
- 4. Updates the client with STRING message
- //end

2: World:

- Focused on managing command-directed changes to the world state
- Changes to the world state (by calling on the managers)
- Tracks and updates the relationship (user: avatar : room)
- Section communication dummies

world.cpp

Relies on:

- Avatar_manager.cpp
- Area_manager.cpp
- World_state.cpp

For any command:

- 1. Takes input
- 2. Calls on **world_state**.cpp:
    - SEND user ID
    - RETURN RoomID(s) || Avatar ID(s) || User ID(s)
- 3.1. Calls on **avatar_manager** to change the avatar information: DUMMY FUNC.
- 3.2. Calls on **room_manager** to change the room information: DUMMY FUNC.
- 4. Returns <some> completion response to **commander**.cpp
- //end

world_state.cpp

For any command:

- 1. Takes user ID
- 2. Matches the user ID to avatar ID
- 3. Updates any relationship data as needed
- 4. Returns the Room ID(s) || Avatar ID(s) || User ID(s)
- //end

3: Avatar_manager:

- Responsible for creating, storing, and updating all avatar information

avatar_manager.cpp

Relies on:

- Avatar.cpp

For any command:

- 1. Takes avatar ID
- 2. Identifies the avatar object from avatar ID
- 3. Calls on avatar.cpp:
    - Fetches object information
    - Changes some object information
    - RETURN <some> { status data || success message }
- 4. Return
- //end

avatar.cpp

Relies on:

- object.cpp

For any command:

- 1. Fetches object information || Changes some object information
- 3. Returns <some> { status data || success message }
- //end

4: Area_manager:

- Responsible for creating, storing, and updating all room information
- Reads and creates areas and rooms from JSON
- Internally resets objects

Area_manager.cpp

area.cpp

room_manager.cpp

Room.cpp

...