Multiplayer Design Plan

Think about what messages you need to send from the client to the server, and from the server to the client, to achieve your tasks. You should write your design plan **before** you attempt to start any coding, and then adjust it as necessary.

For each message you think you need to achieve working multiplayer, please complete the table with the following information:

* **Message is Sent from (client/server)**
  + Write ‘server’ if this is a message sent from the server to the client
  + Write ‘client’ if this is a message sent from the client to the server
* **Message Name**
  + What is your message called? Try to give your messages sensible, but short, names – like variables.
* **When the Message is Sent**
  + What triggers your message to be sent? Is it when a specific event happens? Is it when something happens in your game? Is your message sent when several different things happen? Describe them all (briefly) in this column.
* **Data Sent & Description**
  + What data/information might you need to send with this message, and what format does it take. Are you sending an object? What variables are included in that object? Do you only need to send a variable? What will these be called? Where does the information come from? What assumptions have you made, or what do they represent? What types of data can they carry?
* **What Happens when the Message is Received**
  + What do you need to do in the event handler for this message? If your message is sent from the server to the client, remember that your event handler will be on the client (and vice-versa). Does your message make something happen? Does it mean that you need to do something with some of your variables? Briefly describe that here.

An example message, already included in the skeleton code, has been completed for you on the next page. Study this carefully (and compare it to the skeleton code) and use it to help you complete the rest of the table.

Think carefully about the messages you will need, and refer to ‘*Real-time Servers V - Server Design: From Start to Finish*’ for more guidance on planning the design of client-server architecture.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Message is Sent From (client/server)*** | ***Message Name*** | ***When the Message is Sent*** | ***Data Sent & Description*** | ***What Happens when the Message is Received*** |
| server | dungeon data | When a client connects for the first time  **AND**  When a new dungeon is generated | **dungeon**  An object containing the following members:   * maze – a 2D array of integers representing the dungeon layout. 0s represent impassable spaces (walls), 1s represent corridors, and numbers 2 or greater represent rooms. * h – the height of the dungeon (size in the y dimension) * w – the width of the dungeon (size in the x dimension) * rooms – an array of objects, describing the rooms in the dungeon. Each object in this array contains the following members:   + id – an integer representing this room in the dungeon, numbered by order of creation   + h – the height of the room (size in the y dimension)   + w – the width of the room (size in the x dimension)   + x – the x-coordinate of the top-left corner of the room   + y – the y-coordinate of the top-left corner of the room   + cx – the x-coordinate of the centre of the room   + cy – the y-coordinate of the centre of the room * roomsize – the average size of the rooms, used when making the dungeon * \_lastRoomId – the id of the next room to be generated   **startingPoint**  An object containing the following members:   * x – the x-coordinate at which players should start in this dungeon * y – the y-coordinate at which players should start in this dungeon   **endingPoint**  An object containing the following members:   * x – the x-coordinate at which players can escape this dungeon * y – the y-coordinate at which players can escape this dungeon | The client should replace its existing maze information:  dungeon replaces the *dungeon* variable, *startingPoint* replaces the *dungeonStart* variable and endingPoint replaces the *dungeonEnd* variable. |
| ***Message is Sent From (client/server)*** | ***Message Name*** | ***When the Message is Sent*** | ***Data Sent & Description*** | ***What Happens when the Message is Received*** |
| Server | ConnectionID | When a new Player connects to the server for the first time | **ConnectionNo**  A variable containing the connection number is sent to the client to uniquely identify the player when dealing with the client’s player. | The client should store the connection number into a variable called ConnectionID. |
| Server | allPlayers | When a new ‘Player connects for the first time  **AND**  When a player disconnects  **AND**  When one of the players moves | **allPlayers**  An array of players object with following members in each player object.   * connectionNo – each player in the array consists of a connectionNo which is randomly given to each player. * x – contains the x position of the player on the dungeon maze * y – contains the y position of the player on the dungeon maze * shifty – contains the y position to decide where the image should start clipping to draw the correct frame. | The client will check the connectionNo of each player in the array. If the connectionNo matches the connectionID sent earlier, then the client’s player is drawn uniquely compared to others.  If the connectionNo does not match the connectionID sent earlier, then the image drawn is the skeleton, the image used to identify all the other players.  Once identified if the player is local or other players, each players x and y positions are used to draw the image on a specific point on dungeon and shiftY value is used to clip the image frame according to the direction the player is moving. |
|  |  |  |  |  |
| Client | Movement | When a client requests the server to move the player based on key, mouse click and tap on the touchscreen. | **Movement**  An object containing the following Boolean type members which are false by default:   * up – a Boolean value which if set true and sent to the server moves the player one block up. * down – a Boolean value which if set true and sent to the server moves the player one block down. * right – a Boolean value which if set true and sent to the server moves the player one block to the right. * left – a Boolean value which if set true and sent to the server moves the player one block to the left. | When the server receives the message, first the connection number is identified using the socket id to identify the player in the array.  Once the user is identified, it checks which member of the movement object is set to true, based on the member identified as true, specific lines of codes are processed, which moves the player’s position to the direction which was triggered by the client. |
| Client | Disconnect | When a client disconnects from the server. |  | When the message is received by the user server, the clients player is identified using the socket id and using the connection number. The connection number is used to delete the player from the array and the message is sent to all clients which consists of allPlayers array. |

*Add as many extra rows as you need…*