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*** |
| client | move | When a client attempts to move in a direction | Direction   * A string that will represent the direction of the attempted movement, either left/right/up/down | The server works out if the move is possible (or blocked by walls) and if so updates the position data of the player that made the move request inside the server’s Players array, then sends a player data message to update all the clients |
| server | player data | Every time a move request is made by any client  AND  When a new dungeon is generated due to a dungeon being cleared  AND  When a new player connects  AND  When a player disconnects | Players   * An array consisting of Player objects. Each Player object contains   + X – the x-coordinate of the player   + Y – the y-coordinate of the player   + Facing – the direction the player is facing as a string (either left/right/up/down)   + ID – the socket ID of each player   + Name – the name of each player | Each client updates their internal Players array with the newly sent one, which they use to draw the players in the right positions |
| client | username update | Whenever a client submits the form to change their username | Username   * A string representing the new username to assign to the player | The server will update the name variable of the correct player |
| server | hiscore data | Every time a client connects  AND  Whenever a dungeon is cleared and a new one generated | Hiscores   * An array consisting of hiscore objects. Each hiscore object contains:   + Username: the name of the player   + Time Taken: the time taken to clear the dungeon | The client will update their local version of hiscores, and update the table displayed on the browser |
|  |  |  |  |  |

*Add as many extra rows as you need…*