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 | positions | When client moves the player using both keyboard control and mouse control | An object called movement is being sent to the server, inside this object, there are 3 Booleans that determine which way the client is intending to go. when the client presses a key to move or the mouse is clicked, the way the client wants to move is set to true. Then the movement object described, is sent and the server listens for this using socket.on("positions", function (movement). After it has been sent, the Boolean that had been set to true has then been set back to false.  X position which determines the position the player has chosen to move to  Y position which determines the position the player has chosen to move to | The server then checks to see if the client is within the bounds of the rooms, if so then allows the movement to take place, if not. The movement request is rejected and player does not move.  The position of the player should be updated to the position where the client chose to move to. As well as this, the position needs to be updated on all client’s. |
| Server | totalPlayers | Every time there is a connection or disconnect made to the server | A variable containing the number of players has been declared in the server.  Each time a client connects to the server, the playerCount variable is incremented, thus, giving us the number of players that are playing the Dungeon Escape game.  Also, when a client disconnects, the playerCount variable is deducted by one to keep the number of players currently playing accurate. | The client is listening to the data being sent from the server in the animate function. This is because a player could disconnect at any moment and therefore, as the animate function is continuously being ran, the number of players is constantly being checked. Once the playerCount variable has been received, it is displayed on the client using innerHTML. |
| Server | level | Once a connection is established and also in code for checking if player has reached dungeonEnd. | Initially, the level variable is declared and set to 1, then, once a connection has been made, the variable is sent out to all clients.  There is an if statement that checks to see if any one of the players have reached the endpoint of the dungeon. Within this if statement is where the level variable is incremented by one each time as well as a new dungeon that is generated each time a player reaches the end of the maze.  This variable is also used for database information storage. | The client listens for the message that is sent from the server, indicating that the player has reached the end of the maze. The level variable being sent is displayed using get element by id. |
| Server | state | Continuously being sent using a setInterval function that emits multiple times a seconds. | The whole players object is being sent over along with all its members.   * ID – used for giving each client a unique identifier which is important for distinguishing between clients and in finding out which player has reached the end of the maze * playerCharacter – used for assigning a visual character to each client that has connected. * x – position of the client’s player across the x axis – used for player movement. * y – position of the client’s player across the y axis – used for player movement. * timeElapsed – used to determine how long the player has took to complete a level. All players’ timeElapsed resets to 0 once a player has reached the end of the maze. * goingUp – Boolean to decide which image will be displayed, display up image if true. * goingLeft – Boolean to decide which image is displayed, display left image if true. * goingRight – Boolean to decide which image is displayed, display right image if true. | The client stores the players object in a new object called player. This object stores all the information about that particular client which is used to determine, where the client’s player is displayed using the x and y members, which character is displayed using playerCharacter and which image is displayed using goingUp,goingLeft and goingRight are all used. |
| Server | timeMessage | Sent continuously along with state as the time is counted in seconds and the client needs the time elapsed to be as accurate as possible. | timeMessage is sent as a variable. As time Message is incrementing each second, this variable is used to count the seconds that have elapsed since the player has connected. | Client side, once the message is received, the time elapsed is displayed onto the page and this gives players an incentive to complete the maze as fast as they can, alongside levels, this makes the game more enjoyable and competitive. |
|  |  |  |  |  |

*Add as many extra rows as you need…*