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 | Start | When a player connects for the first time | No data is sent in this message. This message is used to tell the server that a user has connected to the sever from the client. | Once the message has been received create a player object in the server that takes the x coordinate of the start of the dungeon and the y coordinate of the start of the dungeon. It then uses the socket.id to create a playerID and the score is set to zero. This new player object is then added to the players array in the server using push(). |
| Server | Player | When the new player has been added to the players array on the server | **Player**  An object containing the following members:   * playerID – the socket.id of the current player * x – the x-coordinate of the dungeon’s starting point * y – the y-coordinate of the dungeon’s starting point * score – the score of the current player set to zero | The player object on the client is replaced with the player object sent by the server. This score is then appended to the score div on the html using the jQuery method .append(). |
| Server | Update | This message is sent every 500 milliseconds using setInterval. | **Players**  An array of objects containing the following members:  **Player:**   * playerID – the socket.id of each player connected to the server * x – the current x-coordinate of all the players connected to the server * y – the current y-coordinate of all the players connected to the server * score – the current score of the player | The players array on the client side is updated with the players array from the server side. |
| Server | Disconnect client | Every time a player disconnects | No data is sent in this message. This message is used to disconnect a player on the server. | A player is removed from the players array. If the playerID equals the socket.id then the player is removed using the splice() method. |
| Client | Player coordinates | When a player is moved using the key, click or tap controls | **player**  An object containing the following members:   * playerID - the socket.id of the client * x – the current x-coordinate of the player character * y – the current y-coordinate of the player character * score – the current score of the player | If the playerID equals the socket.id then that player’s coordinates are updated. |
| Server | Player restart | Once a player has reached the end of the dungeon | **playerRestart**  An object containing the following members:   * x – the x-coordinate of the new dungeons starting position * y – the y-coordinate of the new dungeons starting position | Replace the player coordinates with the player restart coordinates. |
| Server | Update score | Once the player has reached the end of the dungeon | **player.score**  A member of the player object | Replace the player score on the client score with an updated score. Then add this score to the html using the jQuery .html() method. |
| Server | Highscore | Once a player has joined for the first time  **And**  When a player reaches the end of the dungeon | **Results**  An array of objects containing the following members:   * playerid – the id of all the players in the result of the query * score - the id of all the players in the result of the query | When the message is received the results are added the highscore table using the .html() jquery method. |

*Add as many extra rows as you need…*