Colyseus

Extension for Colyseus server : https://colyseus.io/

This extension can be used to create multiplayer games in Stencyl.

I highly recommend you adjust / re-work / study the example games for the game type you want to create.

Example files: http://photoquesting.com/colyseus/
The example files use my personal server. For testing purposes you can use this server as well, but I could decide to stop it at any time. So be careful not to do any publishing with this server in the configuration. You should really host your own server when you want to publish.

Multiplayer logic will be hard and you need to think and realise about multiplayer data flow.
If you just make a connection to the server and assume that your game will be automatically multi play you are wrong!

In general : There will be always a lag between the server and the client.
The more clients there are to the server and the slower the bandwidth between the client and the server; the higher this lag will become.
Think about a mobile device on a cellular data plan versus a local area network.
The mobile device will not have the same speed to the server as the same game on the local network.
For instance when you develop your game it could connect to a virtual machine or docker on your own system. That connection is always faster in regards to the same game on a web server that is running on another location.

The Colyseus Extension is meant to be for lightweight multiplayer games. To get a feeling and understanding of multiplayer games in general.
The Colyseus Extension provides SOME authoritative mechanism.
To get actual authoritative servers doing client side prediction you need to write GAME-SPECIFIC server.. (See https://www.gamedev.net/forums/topic/697159-client-side-prediction-and-server-reconciliation/)
That is out of scope of this extension.
.
For more and general information on what online multiplayer is:
https://www.youtube.com/watch?v=IoOrY-MPVzw

The extension works on these platform publications:

- HTML5
- Windows
- Android

Older version worked on IOS simulator and IOS publication but I don't have a system capable of creating iOS games anymore so I can't verify with Stencyl 4.5

Linux was tested and should work but didn't test with Stencyl 4.5

Flash will not work!

GENERAL LAYOUT of the extension

The analogy the server uses is kind of like a **Hotel**.
There is a **lobby** where you register yourself with the game.
You book a (conference) **room** for your party.
When you have players **join**ed this room there are **seats** available in the room. A player is assigned a seat.

This is the flow

> Connects to server (Enter Hotel)
>
> A Lobby is created for the ApplicationID.
> And you register your player at the check-in desk.
>
> When there are no rooms available for the ApplicationID you need to create one for your game-session.
>
> If there are rooms available you can have the player join that game party.
>
> When joined a player is assigned a seat to take part of the game

Of course you can do checks before joining or even avoid showing the list of rooms based on some criteria.
For example if you want only 2 players in one room you could hide the rooms that have already 2 players in them.

SERVER

It is highly recommended that you run your own server.
This can be from your home as long as you have control over the router from your service provider. You need to be able to allow global internet users to access your computer.
If you cannot do that or you don't want to have your computer on at all times you need to have a server in the cloud.
Virtual Private Server is usually what you want.
(https://en.wikipedia.org/wiki/Virtual_private_server)

The issue is that you need to start an application on a server that listens to a specified port.
The serverplan you can use must include the ability to run your own software.
This can be a docker if the provider allows uploading dockers.

There are instructions on how to run the server from a virtual machine you create.
That can be a docker on a virtual machine (where most of the installation is already done) or you can run the code directly on that virtual machine. In that case the machine needs to be a Linux virtual machine.

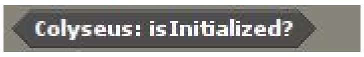
Stencyl Blocks

CONNECTION BLOCKS.

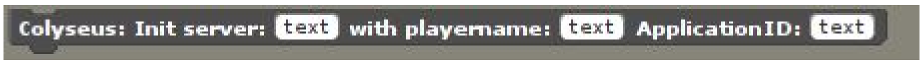
The following blocks are for connecting to the server. Like entering the hotel.
They need to be used when starting the game.

Probably you have these blocks followed after you request or generate a playername.
Typically you ask a player for their name or use another extension to have a social media login.

When you have the playername you initiate a connection to the server.



Returns true if connection has been made to the server and there
is a connection with the Lobby. You check if the player has entered the hotel.
When the server is down or you lost connection this will return false.


Colyseus: Init server: text with playername: text ApplicationID: text

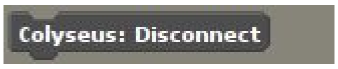
The **SERVER** text is in the form of: server.com:port
Or you can use a global IP address: 154.116.12.80:port
Port is where the server process is listening on.
You may know about port 80 which is a web request to http.
Port 443 is for https. Port 22 is for secure shell service.
You can create your own application or service on a port that is larger than 1024 and smaller than 49151.
As long as there is no other application/service using a port you can have the server listening on that port.
34321 is a port I randomly selected for my server.
The port is where the connection is made with.
On the server you start the Colyseus Server on that port you select.

When the server is listening you can initialize a connection using this block.

Provide a **PLAYERNAME** that is associated with the connection to Colyseus. This playername must be a text.
It is best if the player name is unique for a player.
But in theory this playername can occur multiple times on the server.
Usually you ask a player to fill in a form where you ask for email address and a nickname.
Nickname is most likely the name your player will want to have displayed in a game. Not the email address!
Each connection to the server will lead to a unique player ID.
This is to make sure that communication is always done to the same player.
Each player ID is unique for the connection and should not be stored on a device and attempt to re-use.
If you have multiple sessions this player ID can change for instance. So if you don't use the player name across your game and would use the player id the game logic will not work.
In theory you can have very long playernames but you have to realize that all data to and from the server consumes band-width.
So if there is no reason to have a name with more than 80 characters you shouldn't allow it.
Also this would make displaying the player name to other players somewhat difficult :)

APPLICATIONID:

The Colyseus server is designed to host different games.
In your case it would not make sense if you had only one multiplayer game using Colyseus.
But if you want to have multiple games on the same server it makes sure that each game you create has a unique connection to the server.
Data from two games will be separated by this ApplicationID.



This block will disconnect = leave Hotel.

Usually this block is used at the start of your scene. (!)
You can leave it out, but when you for instance switch to another scene and switch back you want to have this block so that the connection is closed before attempting to open a new connection.
This can also be necessary when there was a bad connection or connection loss in your game.

If you leave this block out and you create two connections to the server it will generate problems for your game. (like entering the hotel twice)

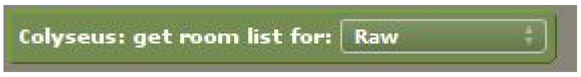
Typically I use a scene behavior where I put DISCONNECT at the create event of the scene.

Then I use a [every second time] event to test for connection to the server. If there is no connection made (yet) I initialize the server.
Usually the connection is done within a second. So every second attempt is a best practice.
If the connection is NOT done within a second you should realize that even when you get a connection after a second the speed between the client and the server will not be fast enough to even attempt other multiplayer sessions.
A TurnBased game wouldn't necessarily have a faster need, but you should realize this fact.

When a connection is made (checking with isInitialized) I usually switch to another scene where I get a list of rooms.
Or have the player create a room.

ROOM CREATION

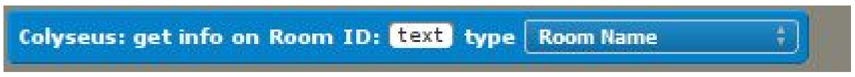
The following blocks are to communicate with the Lobby registrator.


Colyseus: get room list for: Raw

The return list contains sessions open for rooms of the RoomType (Raw,Turn,Lock)

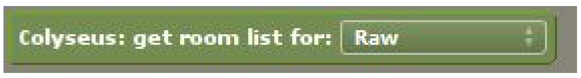
You could use this list to check if there are rooms. If there are no rooms you could create a room for the player without asking a player to create a room.

When the list contains rooms it will return the IDs of the rooms.
With that ID you can ask the Lobby registration for information about the players in the room.


Colyseus: get info on Room ID: text type Room Name

Depending on the type requested you get a text, number or list.

Provide the Room ID that you

obtained from the 
Colyseus: get room list for: Raw

block.

ROOM NAME
Returns the room name you provided with the create room block.

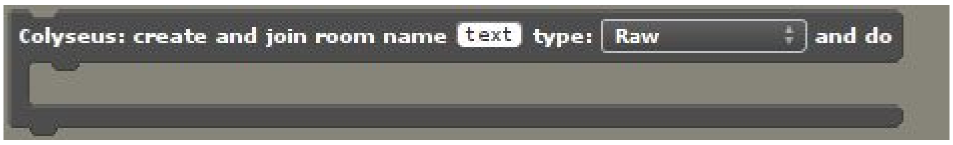
NUMBER OF PLAYERS
The number of registered players in the room. It will not give you information about the state of the player.

PLAYER LIST
The list will contain the Player Names that are inside the room.

Note: Room Name. It is possible to have more games run on a server where there are multiple games for each room type.
It is like running TicTacToe and a Card playing game. The roomlist will contain both the sessions for TicTacToe and the Card Playing game.
(This is the case with the server that I run with the example games !!!)
Again, this might not be the case for you. And you only have one game running on the server.
But in case you are connecting to a server that potentially runs many games it can be necessary to filter out the rooms that you get from the roomlist.
For instance:

- TICTAC_1324
- CARD_5512
- TICTAC_5231
- CARD_1661

Usually I use an index block to check if the roomname contains part of the name and I use that filtered list to show to the player.



When you want to create a room it will automatically join, the player who requested the creation, to the room.

Make sure that the **room** you create has a unique **name**.
If you for instance create two rooms with the
same name there will be conflicts in later game-logic.

I use a random creation which you could combine with getting the roomlist to make sure that the name is unique.

Room Type

There are currently three types of rooms. The room type indicates the logic that is on the server side.

TURN

When you for instance create a turn based game; there is logic that checks for the turns.
So what is the active player and how to find the next available player in the room.
Even getting a list between the 'human' active player and the last 'human' active player.
So that you can have an A.I. that makes a turn for a player that has left or when you have only 2 players in a game where you need four in your game.

LOCK
This room type has server logic that is like a database. So only one player at a time can change some value.
For instance a grid where every position can only be altered by one player.
The flow is that the game requests a lock with a given set of data points (from a Map-attribute). When no other player has changed the data point it will lock the positions and change the value(s).
When all the data points are locked the data from the player is changed.
Then the locks on the data points are released so that other players can change the value.

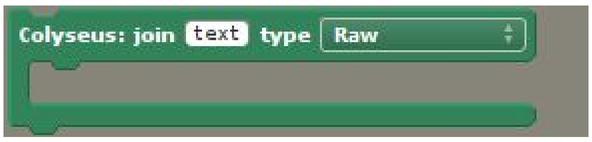
Typically this is used for board type of games but you can also use the mechanism for a RPG kind of game.
Like there is a CHEST and it is closed. When two players 'at the same time' open the chest only one of them will get the data point lock and prevent the other player from accessing the chest.

The logic on the second player should incorporate a flow of events when the lock is denied or when there was an out of sync.

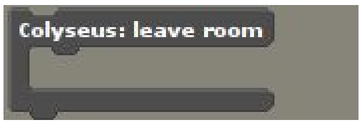
Out of sync can be caused when the data that was available on the client is older than the data on the server. Like that the last state of the CHEST that it was open and a request was sent to the server to open the chest.
The server could have received another request from another player to open the chest so the chest is open on the server and not on the client who requested the lock.
Usually the logic of the game would refresh the state from the server on the client indicating that another player has opened the chest in the meantime.

RAW
There is no logic on the server and you need to do all logic of the game on the client.
This type of communication is used for the ' Client - As - A - Server' approach.
Also good for games that do not have collisions between players. For instance a split screen game where you see the state of other players but do not interact with them. When you have an athletics game where each player is in their own lane.
Or an endless runner where you see the other players ' as ghosts '. So you can see the state of the other players but they don't affect the run of the other.

When a room is created it will appear in the list of rooms of the given room-type. The player that created the room will be automatically registered in the room and you do not need to use the [join] block.

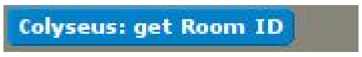


When you have the player select a roomid from the roomlist you can use that room-id to join the player to the room.
You need to specify the room type that created the room.

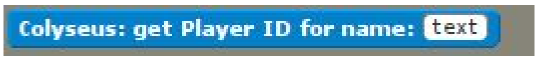


This block will remove the player from the room.
This will not disconnect the player to the server.
Like the analogy that the player goes to the lobby to join another room or create a new room.
When you want to have the player leave the game entirely you should use disconnect which automatically removes the player from the room.

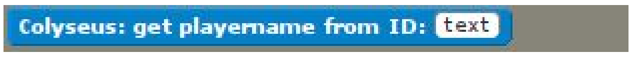
BE CAREFULL
When you use [leave room] or [disconnect] the game logic should not attempt to use stencyl blocks. Usually this is done by setting a boolean attribute. That is why there is a wrapper block. So you can set those attributes in the wrapper block.
In your other blocks (for instance update,draw and time blocks) you need to check for that attribute in order to avoid doing anything in the room where the player has in fact left.
Leaving a room with that stencyl block does not change the behavior of stencyl. So other stencyl actions can still be running. When your intent is to switch scenes
after the [leave room] there could still be code running BEFORE the switch scene is actually completed and create errors or inflict game-data-corruption.


Colyseus: get Room ID

The room name is not like the room id. RoomID is unique and is different each time a room is created. You cannot create a room name with a specific roomid. The roomID is generated on the server.
This ID is used to communicate uniquely to the room.


Colyseus: get Player ID for name: text

The list of players returns names of players in the room. You can get the ID from that name using this block.
The playerid is unique and generated on the server for each connection.


Colyseus: get playername from ID: text

Used when you have received information on the Player ID; for instance from the [getplayerid in seat] block.

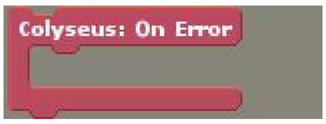
Error routines

Before diving into the specific room blocks I would like to discuss the Error routines.

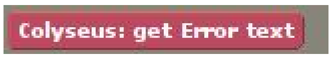
While there is a connection from the client to the server there can be issues.
Like (temporary) loss of internet connection. Bad internet. Too slow connection.
When those issues arise your game should react to them. Usually disconnecting and initializing again.

I have not yet made a system where the game could re-connect to the room that was joined.

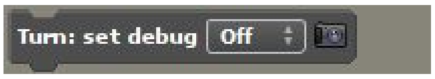
You might want to store the RoomID on the device using loading/saving the attributes kind of mechanism.
By automatically reinitializing and obtaining the roomlist you could join the player automatically.
Again: not implemented in any demo game.



Wrapper block where inside you can do action when errors occur. You need to disconnect the player.
Leaving the room could not be enough since that would not reconnect to the server.
You should assume that when an error occurs that all next colyseus blocks will have problems.
So best is to show the user that there was an error and do a reconnect or the mechanism with reconnecting with a stored roomid.



Unfortunately there were instances during development of the extension that the Colyseus server reported an empty error : NO TEXT !
So this block can return an empty string. You would normally use this text during debugging. It is not meaningful to the player.
When an error occurs it is best you make a general error text for your players.



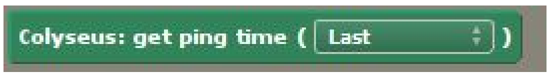
This block can be used to get a flow of all debug messages in console during testing of the game. The debug messages will not appear in a published game even though you set this to ON.
Typically you would use this during development to get information on state of the network.
Usually turned off when publishing your game.

Room blocks.

Depending on the room type you have a selection of blocks you can use.

There are however a few blocks that are implemented on all room types.

GENERAL ROOM Blocks

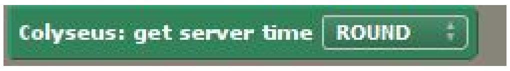


When a client has sent something to the server it will receive some data back.
The time that it took from the client to the server AND BACK will be reported with this block.
Depending on the speed of the connections between client and server this time can vary.
Usually between 10 and 300ms.
On local area networks 10ms could be received but usually it is much higher.
These ping times are from a server running in the Netherlands:

- The Netherlands : 30 - 50
- U.S.A. West Coast : 200 - 300

When your game needs a fast connection you could use this number to inform the player that the connection is too slow for the game. Advising them to use WIFI instead of cellular for instance.

When you host a game from another continent (for instance in Europe) and there are connections from Australia or United States the ping time can increase dramatically.
Collision based games usually require less than 300ms ping.

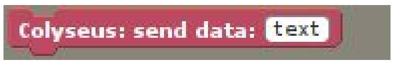


You can use this block to obtain the server time. When you have multiple devices and computers running your game they possibly have all different times. There is only one time that is the same on all : servertime.
But when you request the server time the actual request could take some time.
The ping time is used to calculate that difference. That can lead to non-accurate time differences. During the usage of the colyseus block this is recalculated but you never should rely on it!

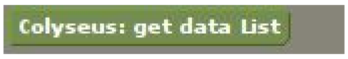
Do not use the server time to base client action on. An attempt was made and at first it looks okay but after running the game for over a period of one hour there will be differences. Even on local area networks. Also there is a general note about floating point differences between hardware platforms.

So in general use this server time to base non-fast-games. For instance by having a countdown clock (On seconds) to start a game session.

ROUND and RAW: Since Stencyl can't handle large floating point numbers in calculations there is a ROUND function inside the block.



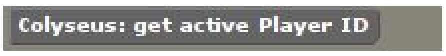
For all roomtypes : this block sends data to all connected players (including the sender!)
You can use the Lock and Turn Map attribute blocks to ease data manipulation. But you can do all data manipulation with the send and receive list blocks. Usually with some kind of selfmade protocol. Comma-separated data is typically some way of doing this.



To catch the sending data that is done by other players (and yourself)

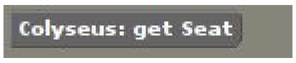
The order of the received data is based for the list. When you need a different order you need to include some data in the send-text to accommodate. For instance using the

ServerTime block or some kind of calculation of your own. Some mulitplayer documentation you could find online about online multiplayer use ‘FRAME’ time to communicate.

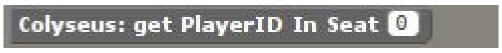

Colyseus: get active Player ID

All roomtypes (Raw,Turn,Lock) have this block enabled so that you can get information of the active PlayerID.
At room creation time this is always the player that started the room.

Use this block to get the state of the active player. When you are not on Turn Based type you could still use this block to make sure only one player in the room is handling some room data. Especially when you want to initialize the data.


Colyseus: get Seat

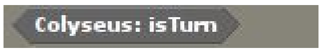
Returns the seat number from the player. (0-based)


Colyseus: get PlayerID In Seat 0

Number ranging from 0 to <number of players>

If requesting ID from out of bounds it will return empty text

---

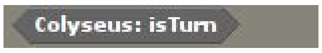

Colyseus: isTurn

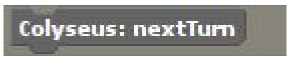
You could think that this block only works on TurnBased but in fact it combines
[get playerid in seat] with [get seat] and [get active playerid]

Specific Room Type Blocks

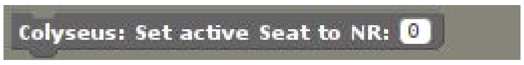
The following blocks have only effect when you have a game created for the room type.

Turn

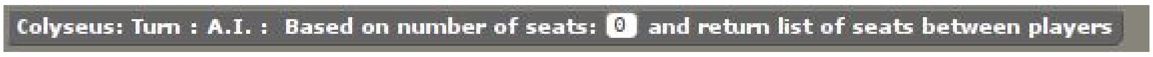
Already discussed : the 
Colyseus: isTurn
block


Colyseus: nextTurn

This block will request a turn switch on the server. When the server has acknowledge this the blocks for get
ActivePlayer and isTurn will be altered accordingly.

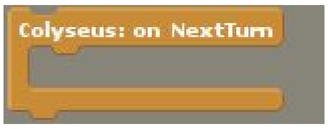


You can use [next turn] block to let the server pick the next player in the order.
This block can be used to alter that order.
Suppose you have a turn based game where the first player to act is different each time the game plays a round. Typically card based games have a turning player.
Default behavior is that the next player action is always started from the creator of the room.

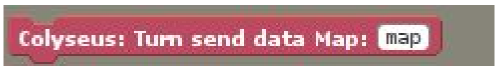

Colyseus: Turn : A.I. : Based on number of seats: 0 and return list of seats between players

The block returns a LIST of seat numbers between the last player and the current player.
The **Seats** Given [0] is the number of seats the game needs to check. Typically for a tictactoe/chess game this value is 2 and for most card-playing games this is 4 or 6.

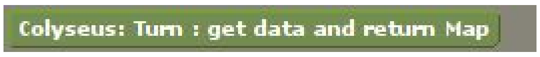
Used for A.I. When you don't have the required humans in the game session.
Or when one of the players leaves or has a bad connection. When you have an A.I. it can use this block to find out what seats need to be calculated.



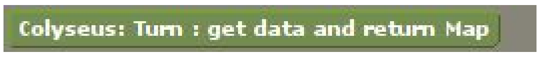
Use this wrapper block to run client-side-logic when a change of a player is detected.
Usually this is the part where you want to identify to the game to show new state and act only once on the data.
For instance doing change animation. Slide actors on board and stuff like that.
When you would do all that on each data receive it could lead to more calls to the stencyl blocks.



This is not like the Lock data mechanism. The data Map attribute that is used here makes sure that the Map is converted into text to send to the server.

In combination with  this will make life easier.

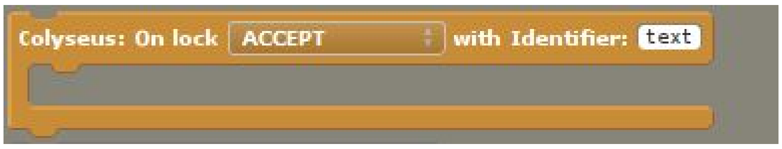
You can still use the general get data list and send data blocks but when you already have a Map attribute for your data it can be helpful.



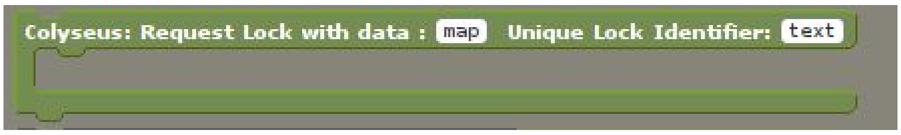
Returns a Map with all the data that is received from the server.

LOCK

The following blocks are only effective in a Lock-based room type.



This block is used inside this block


Colyseus: Request Lock with data : map Unique Lock Identifier: text

The identifier is used to distinguish between different actions.
For instance you could have more send data map locations in your code. Depending on different actions you could use the **identifier**.
In case you only have one data send mechanism the identifier can be non-unique.

When you receive **ACCEPT** it means that the server has accepted all changes and unlocked the data.
This is the desired state.

DENY

This occurs when the data on the client is in sync with the server and that two clients are requesting the lock exactly at the same time.
(occurs very rarely)

OUT-OF-SYNC

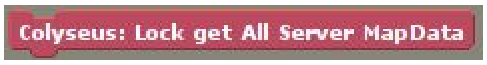
The most important block with the locking mechanism is to react to out-of-sync.
This means that the client has requested a change on data that was changed on the server.
Suppose you have a Grid board.

The Client has Map Key: CHEST with value CLOSED
The Client sends change CHEST=OPEN. The extension will send the current client state in kind of this form: CHEST=CLOSED,OPEN

The server has Map Key CHEST with value OPEN due to some other player that requested the open
CHEST=OPEN
The server receives CHEST=CLOSED,OPEN from the client and detects that the client request contains an
old-value that is not equal to the current value.

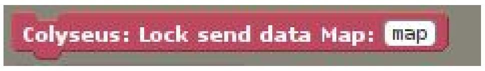
The server sends ' out-of-sync'

Usually you would request all lock map data when you receive this out-of-sync.



When starting your game and it is not the creator of the room or when there was an error you want to request all the data on the server.

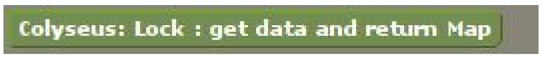
This block requests the action and the [lock get data and return map] will receive the data requested from the client.



The data in the form of a Map Attribute is sent to the server. This should only be used on INITializatin of the room data. Check for instance upon creation of the game if the player is the active player in the room : The one who created the room.

You should not use the send data text and get data list blocks with the Lock mechanism!!

Use the Request Lock wrapper block when you send client-data.



Returns a Map with all the data that is received from the server.

Have Fun!