Centria’s interactive campus map documentation

Written by: Juhani Paananen

This documentation is about Centria’s interactive campus map; its main functions and purposes.

What features interactive campus map has currently (2.2.2026):

1. Users have free roam mobility controls to inspect all floors (0-2) and rooms on campus.
2. Accurate room search.
3. Navigation path to any room step by step.
4. Show all reservations to any room.
5. Reservations and rooms’ status react to the current time. (Finnish time specifically.)

What knowledge you need to understand this project:

- Unity engine and it’s logics
- C# coding language
- Blender, but only at the basic level.

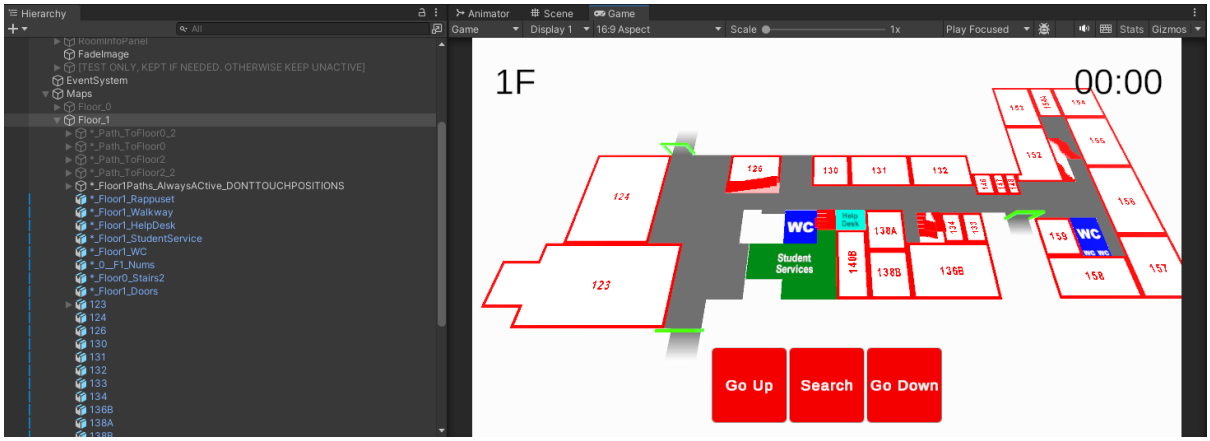
This documentation is created in mind that you have some experience in some of the listed requirements above. If you are new to Unity, but know the C# language, please visit Unity documentation for further information about Unity’s logic and functionalities.

You can use this documentation on the side of working on this project, but most documentation is still inside the project itself in code form. This is only meant to help you understand the project itself.

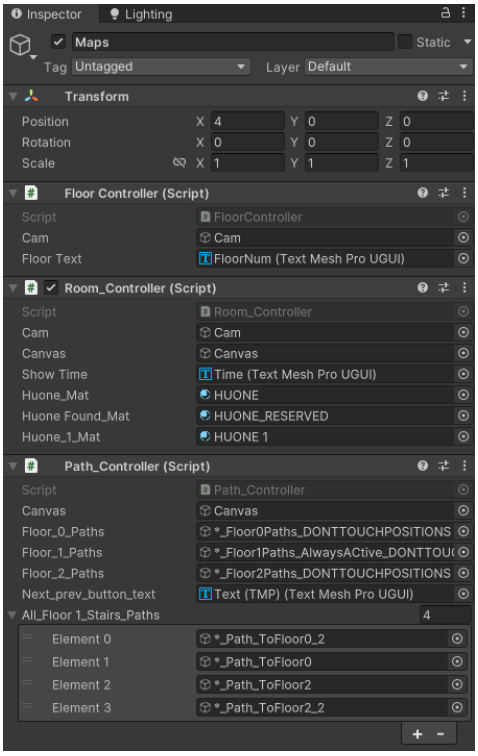
Objects in the map

All objects in the 3D space are separated by floors objects, that are child objects inside "Maps" parent object like seen in picture below. For example: All paths, rooms, stairs and any that should be inside floor 1 are inside that floor parent object.

There are automated processes for getting different objects, so future management would be faster and easier for such large amounts of objects.



When it comes to controlling them, all objects are controlled mainly by these three components:



It is important to know that moving or removing any of the assigned objects to different places will fail the system, if script is not modified in some way.

Floor Controller

This controller controls floor visibility, so only one floor is visible to the user at once. This controller is clearly the simplest of all and only has three main functions, that are in use:

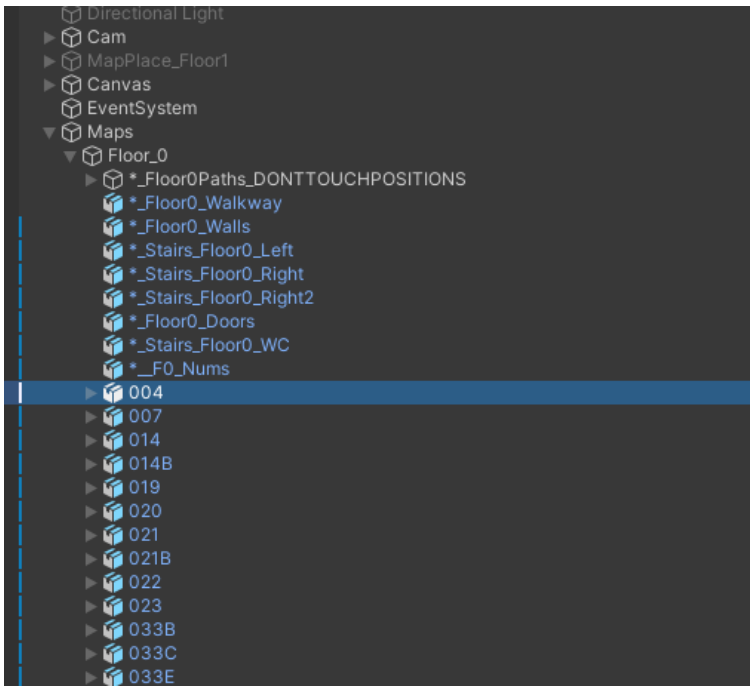
FloorUp(), is used to move floor one up.

FloorDown(), same as above, but down.

SetFloor>ToBe(int floor), is used to direct user to specific floor. Unlike the function above, this allows bigger jumps such as changing the current floor from 0 to 2.

Room controller

This controller controls rooms and reservations for them. Component also contains unused real reservation loader from lukkari.fi as reference to achieving a working loader.



Rooms are taken from floor objects, that hold them as parents, such as seen above. You can see that some child objects on the floor parents have stars (*) and some don't, and it is important.

```
Unity Message | 0 references
void Awake()
{
    RoomData.LoadMats();


    for (int i = 0; i < transform.childCount; i++)
    {
        for(int j = 0; j < transform.GetChild(i).transform.childCount; j++)
        {
            if(!transform.GetChild(i).GetChild(j).gameObject.name.Contains("*"))
            {
                GameObject room = transform.GetChild(i).GetChild(j).gameObject;
                RoomData roomData = new RoomData(room.name, room, 0);
                rooms.Add(roomData);
                roomLookupByName.Add(room.name, roomData);
                if (room.transform.childCount > 0)
                {
                    transform.GetChild(i).GetChild(j).GetChild(0).GetComponent<TextMeshPro>().text = room.name; //Rooms have text on top of them. If text missing or it is wrong, this fixes it.
                    transform.GetChild(i).GetChild(j).GetChild(0).gameObject.SetActive(false); //There is a option to turn these on/off
                }
            }
        }
    }
```

Above Awake () function is used to assign these rooms for room controllers to use. As you can see, foreach only adds those to the list that don't contain, that star (*).

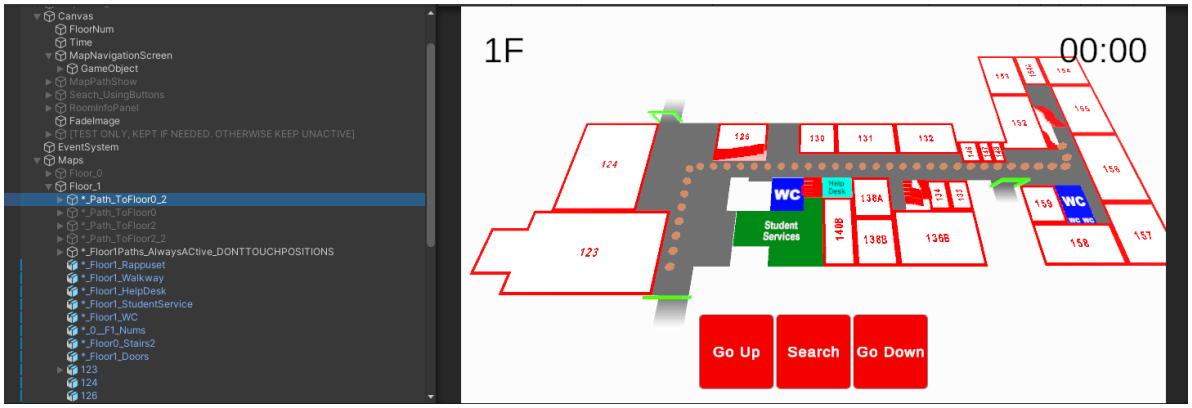
So, sign star (*) is used to ignore other objects to be added to the room list.

Room objects are named after, what room should it be like: "130". If adding more rooms in case some room is accidentally missing or wanted to be replaced with a newer version: Just rename old room to something else (Like adding that * sign in front of it) and then name new object with that room name.

Path Controller

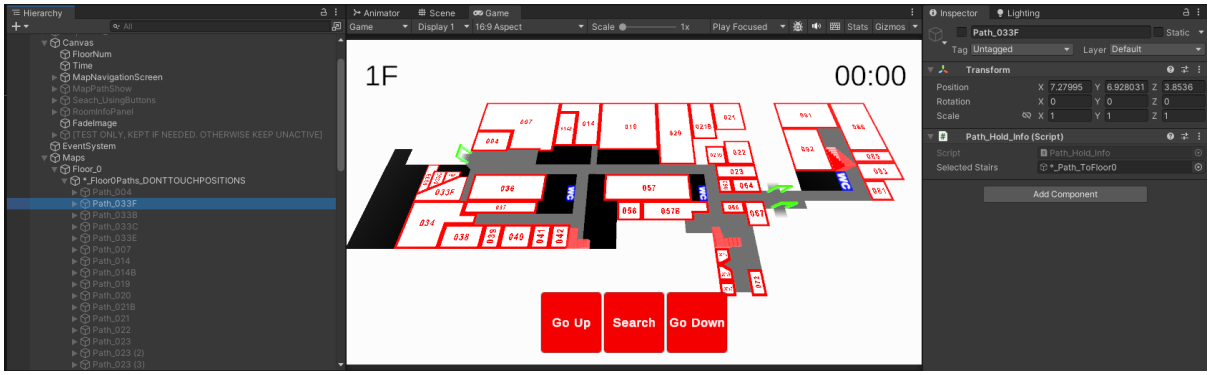
Pathing happens similarly to how floor switching works. All paths already exist as objects, and this controller only activates what the user needs to see.

Example from pathing you can see below. Pathing visuals are done using Unity's particle system, so direction is shown as moving balls. Pathing always starts from the main door, where the user is expected to start the navigation.



There are two types of pathing: those that direct to room and others that direct to stairs.

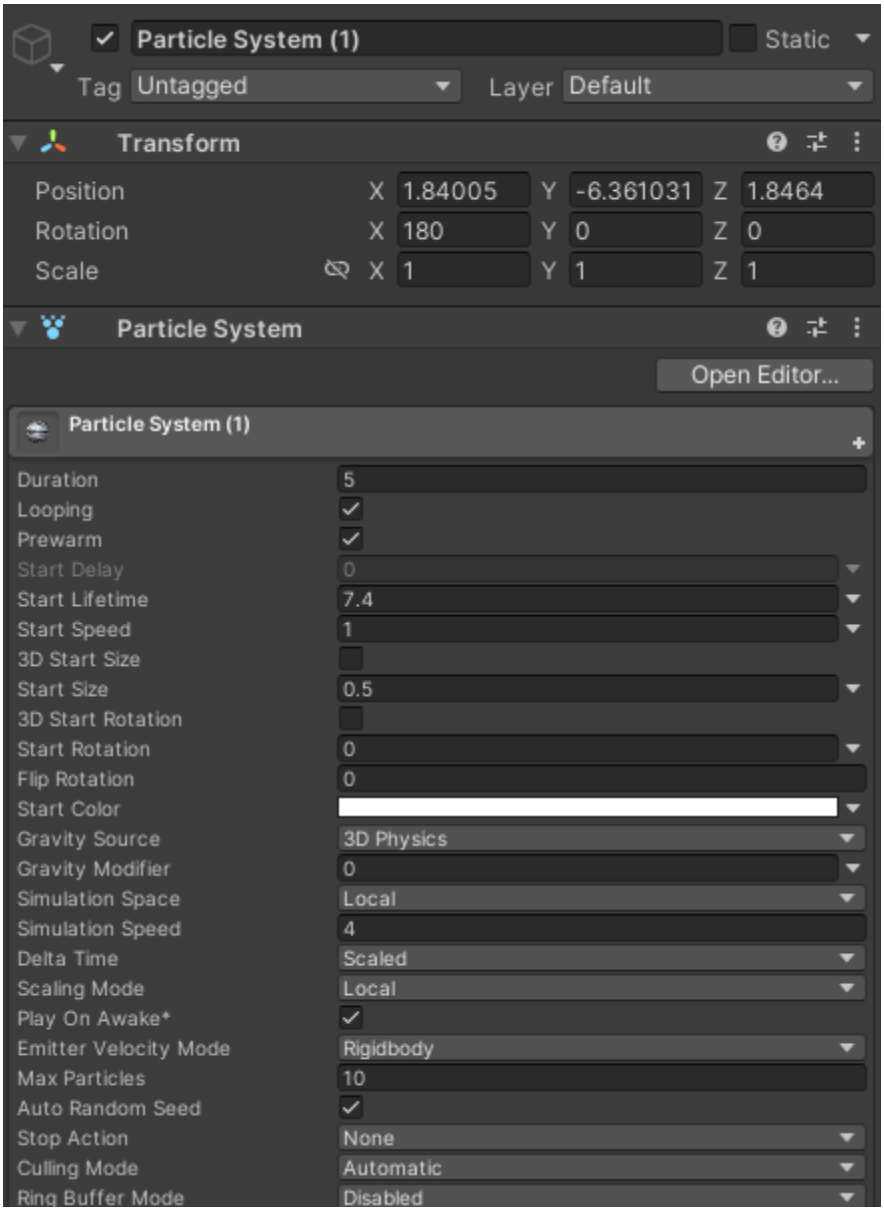
The room's directing path is floor specific. Every room path is contained as child in an object named: “FloorPaths” on each floor. This is important and what is also warned in object’s name, that the parent should be always active and should not be moved so locations stay the same. Floors 0 and 2 have special component: “Path_Hold_Info”, that is needed to contain the path to stairs so the system can activate the right stairs path. Variable: “Selected Stairs” is where the stairs path must be placed.



Paths to stairs are only located on floor 1 and nowhere else. Those are stored individually without parent expect floor parent itself. These also cannot be activated normally, but only when user searches room from floor 0 or 2 it activates like seen below:

```
1 reference
private void ActivatePath_Floor2(string room)
{
    bool Success = false;
    for (int i = 0; i < Floor_2_Paths.transform.childCount; i++)
    {
        GameObject path = Floor_2_Paths.transform.GetChild(i).gameObject;

        if (path.name.Contains(room))
        {
            path.SetActive(true);
            Success = true;
            selectedPath = path;
        }
        else
        {
            path.SetActive(false);
        }
    }

    if (!Success) { Debug.Log("Activating path was: Fail"); }
    else
    {
        Debug.Log("Activating path was: Success");
        selectedPath.GetComponent<Path_Hold_Info>().GetStairs().SetActive(true);
    }
}
```

Creating new paths



Like said, paths are created using a particle system. To create new paths, copy any existing path and name it to whatever you want it to be. Using a move tool, you can move individual particle system in 3D space. There are only a couple of settings, what you need to know: Rotation, Start Lifetime, Simulation speed and max particles. Also **prewarm** should be always on making paths always appear as formed.

Rotation is direction, where balls will travel. You only must move the x rotation to change direction unless you want to direct path up towards the sky.

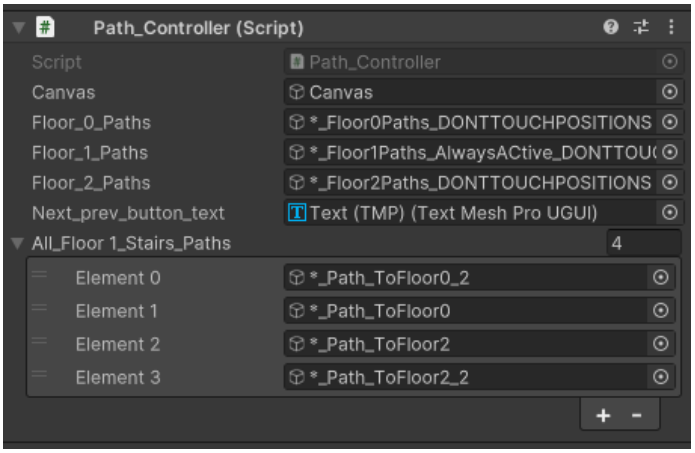
Start Lifetime is a time when spawned particles exist until it is destroyed. The bigger the value, the further the path will be.

Simulation speed is how fast you want balls to move. 4 is a really good number, but when testing longer paths it is advice to turn speed to 10, because whenever you change something, it reset making you want it to form a path again. So, it simply speeds up things.

Max particles mean how many balls can be on screen at once. Well, this can be anything, but for longer paths this value should be increased so there will be no caps between traveling balls.

```
1 reference
private void ActivatePath_Floor1(string room)
{
    bool Success = false;

    for (int i = 0; i < Floor_1_Paths.transform.childCount; i++)
    {
        GameObject path = Floor_1_Paths.transform.GetChild(i).gameObject;
        string[] name = path.name.Split('_');

        if (name[1] == room)//if (path.name.Contains(room))
        {
            path.SetActive(true);
            Success = true;
            break;
        }
        else
        {
            path.SetActive(false);
        }
    }

    if(!Success) { Debug.Log("Activating path was: Fail" ); }
    else { Debug.Log("Activating path was: Success"); }
}
```

Paths are activated from the controller functions: ActivatePath_Floor1, ActivatePath_Floor2 and ActivatePath_Floor0. These are used with a String variable called room. It goes set parent's (Floor_1_Paths) all child objects and activates the correct path.



When it comes to stairs paths, those are assigned to All_Floor 1_Stairs_Paths. To this list all stairs' paths must be contained for controller to use them such as deactivating them.

Navigation in steps

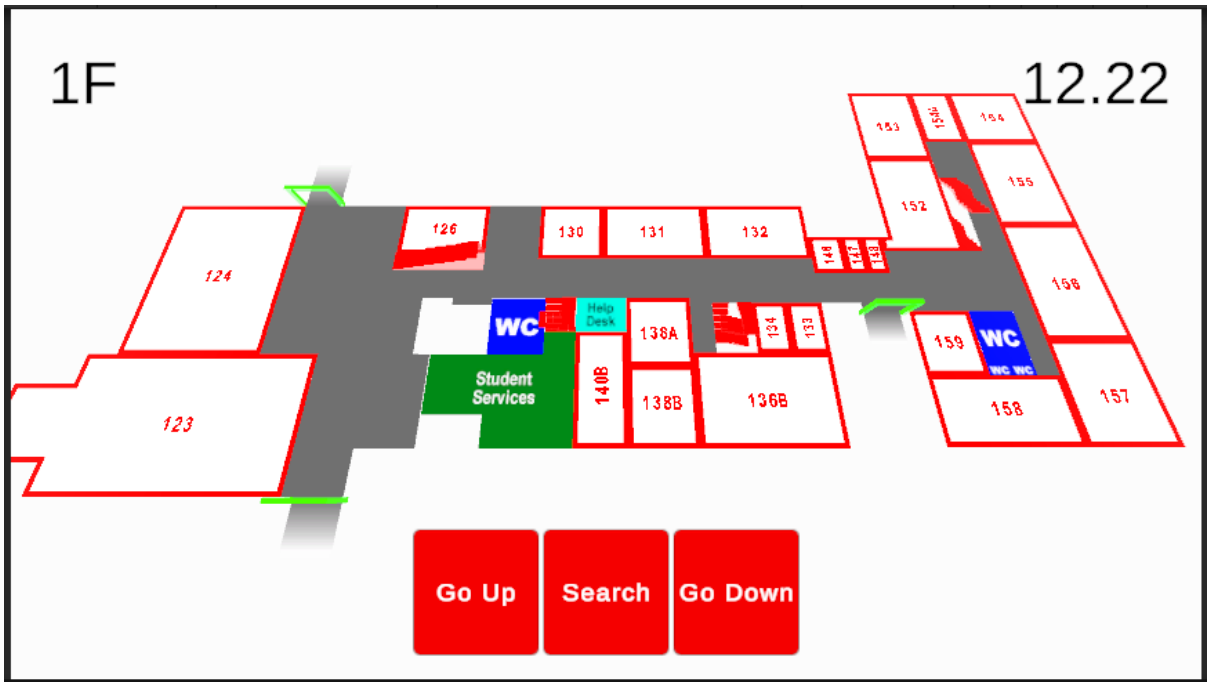
Steps are only used during navigation only if trying to get room on floor 0 or 2. Currently the button which moves user to the next or previous step is visible during any navigation but is not usable in navigation; that has only one step. You can see the process function here:

```
0 references
public void GoToNextStep() //Floor0 = 4, Floor1 = 5, Floor2 = 6
{
    if(NextStep != 1)
    {
        if (IsAtNextStep) { Canvas.GetComponent<Canvas_Controller>().PlayFade(5); next_prev_button_text.text = "Next step"; }
        else { Canvas.GetComponent<Canvas_Controller>().PlayFade(NextStep + 4); next_prev_button_text.text = "Previous step"; }
        IsAtNextStep = !IsAtNextStep;
    }
}

1 reference
public void Operation(string room) //Will Activate path. Always founded room expected.
{
    int floor = room[0] - '0';
    Debug.Log("This room " + room  + " is located on floor " + floor + ".");
    Deactivate_StairsPaths();
    if (floor == 0)
    {
        Debug.Log("Activating path on floor 0");
        ActivatePath_Floor0(room);
    }
    else if (floor == 1)
    {
        Debug.Log("Activating path on floor 1");
        ActivatePath_Floor1(room);
    }
    else if(floor == 2)
    {
        Debug.Log("Activating path on floor 2");
        ActivatePath_Floor2(room);
    }
    NextStep = floor;
}
```

Function does not contain any room detection; that checks if that room exists, because those are already in the search controller before this one. So, look from the search component chapter for better information on that.

Operation function is simply used for activating paths for specific rooms. Activating happens by using the name of that room like: “033F”. You can see it is also parsing the floor from the name of the room for further use. Floor number in Centria campus is presented always as first character.

Map mobility for users



User can move left and right by holding the right mouse button or touching touchscreen and then moving towards the desired way.

Floor changes can be done from those two buttons in the picture above. Function is found from the inside button component.

```
[SerializeField] public float sens;
[SerializeField] public float MaxX; //Maximum X value where cam is able to move.
[SerializeField] public float MinX; //Minimum X value where cam is able to move.

private float h; //Used for getting input from mouse/touch.

Unity Message | 0 references
void Update()
{
    CamControl();
}

1 reference
void CamControl()
{
    if (Input.GetKey(KeyCode.Mouse0))
    {
        h = (Input.GetAxis("Mouse X") * sens);
        //transform.Rotate(Vector3.up * h * 1); //Use this to rotate Cam if needed.
        if (h > 0 && transform.position.x < MaxX)
        {
            transform.position += new Vector3(h, 0, 0);
            if (transform.position.x > MaxX) { transform.position = new Vector3(MaxX, 0, 0); } //This is set to MaxX position. If this obj uses different Y and Z values: Create variables for them and put those here.
        }

        if (h < 0 && transform.position.x > MinX)
        {
            transform.position += new Vector3(h, 0, 0);
            if (transform.position.x < MinX) { transform.position = new Vector3(MinX, 0, 0); } //This is set to MinX position. If this obj uses different Y and Z values: Create variables for them and put those here.
        }
    }
}
```

Component how you can adjust the cam settings can be done here. Currently there are no different settings set for different floors, but that is simple to do by getting floor Integer from the floor controller and using it to change the settings.

Canvas controller

The canvas controller is a component attached to the canvas object. This controller is the main screen switcher and is part of almost every function in the system. Most importantly, it holds the fading effect, that makes the transition for screen to another.

Function is this:

```
1 reference
private IEnumerator FadeInOut() //Simply a fade effect is c
{
    Color color = fadeImage.color;

    float t = 0f;
    while (t < fadeDuration)
    {
        t += Time.deltaTime;
        color.a = Mathf.Lerp(0f, 1f, t / fadeDuration);
        fadeImage.color = color;
        yield return null;
    }

    Fade_mode_Funcs();

    t = 0f;
    while (t < fadeDuration)
    {
        t += Time.deltaTime;
        color.a = Mathf.Lerp(1f, 0f, t / fadeDuration);
        fadeImage.color = color;
        yield return null;
    }

    color.a = 0f;
    fadeImage.color = color;
}
```

Fade_mode_Funcs() is used between the fading to make a smooth transition. Fade duration and color can be modified to be anything. Duration can be set from component, and color is set from set blank image. Color is only set to change the alpha channel only in this function.

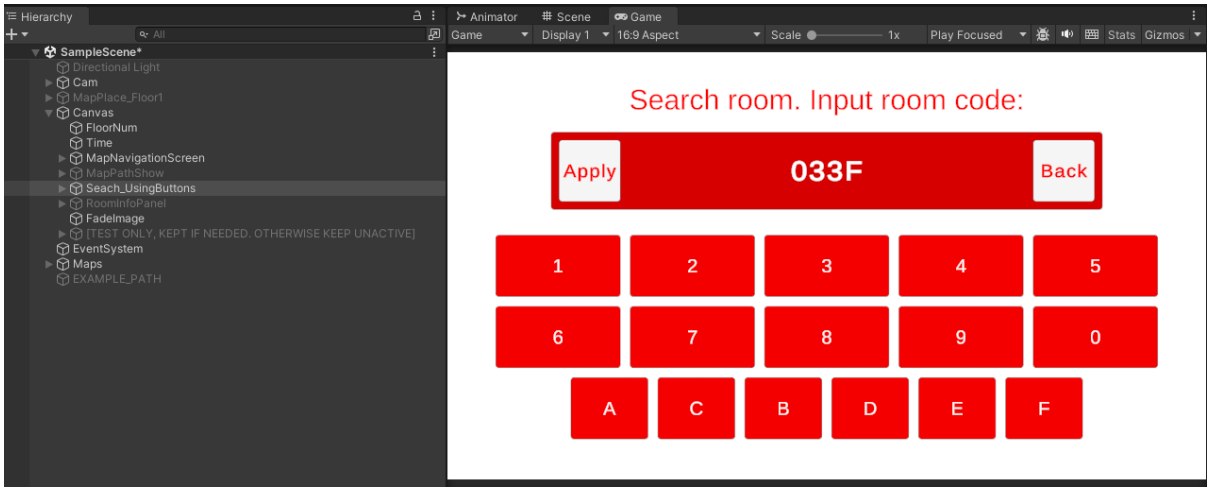
Changing screens

```
11 references
public enum Fade_Modes
{
    None = 0,
    FloorUp = 1,
    FloorDown = 2,
    SearchAct = 3,
    Floor0 = 4,
    Floor1 = 5,
    Floor2 = 6,
    SearchActBack = 7,
    ShowPathNow = 8,
    RoomShow = 9,
}

5 references
public void PlayFade(int mode) //This is attached to buttons that starts fade sequence
{
    if(mode == (int)Fade_Modes.FloorUp && Maps.GetComponent<FloorController>().Get_Flo
    else if (mode == (int)Fade_Modes.FloorDown && Maps.GetComponent<FloorController>()
    else if (mode == (int)Fade_Modes.SearchAct && !GetComponent<Search_Function>().IsS
    //else if (mode >= (int)Fade_Modes.Floor0 && mode <= (int)Fade_Modes.Floor2) { } /
    else
    {
        Fade_mode = mode;
        StartCoroutine(FadeInOut());
    }
}

1 reference
void Fade_mode_Funcs()
{
    if (Fade_mode == (int)Fade_Modes.FloorUp) { Maps.GetComponent<FloorController>().F
    else if (Fade_mode == (int)Fade_Modes.FloorDown) { Maps.GetComponent<FloorControll
    else if (Fade_mode == (int)Fade_Modes.SearchAct) { StartSearching(); }
    else if (Fade_mode >= (int)Fade_Modes.Floor0 && Fade_mode <= (int)Fade_Modes.Floor
    {
        Maps.GetComponent<FloorController>().SetFloorToBe(Fade_mode - 4);
        //StartSearching(); //Disables Search window. This was meant for old method. C
    }
    else if(Fade_mode == (int)Fade_Modes.SearchActBack)
    {
        //string pathToTake = GetComponent<Search_Function>().GetTarget();
        GetComponent<Search_Function>().StartWithoutReset();
        MapNav_Interface.SetActive(false);
        Maps.GetComponent<Path_Controller>().DeactivateAll();
        Maps.GetComponent<Path_Controller>().ResetNavigationVariables();
    }
    else if (Fade_mode == (int)Fade_Modes.ShowPathNow) { ShowPath(); }
    else if (Fade_mode == (int)Fade_Modes.RoomShow) { Show_RoomInformation(); }
    Fade_mode = 0; //This value should always reset.
}
```

Fade_modes hold Integer enums, what mode should be. Those are then used in the PlayFade() function, that is used by outside functions like search button, so users can start searching rooms. Playfade also checks for possible exceptions, what means outcomes when called function should not run.

Fade_mode_Funcs() is the function that runs the functions itself.

Room search



Room search happens in its window, that can be opened using a button with text: “Search”.

These are the functions in this screen: Buttons, Apply, Back, Error messages.

There is a variable called “SearchTarget”, that is used to store user input from buttons. This variable always starts empty when the user activates the search.

Buttons have all OnClick() function, that uses function below (Every button have been a ssign one letter/sign/number):

```
//In use at buttons that user inputs character/numbers to search for specific room.
1 reference
public void PadButton(string sign) //WARNING: Dont replace string=>char!. First it was char, BUT buttons cannot use it as function, if so. Buttons can use this func only using string!
{
    if(sign == "*") //Using this sign, deletes last character from the search string. Always use this char, that is not possible to use by user.
    {
        SearchTarget = SearchTarget.Remove(SearchTarget.Length - 1, 1);
        Debug.Log("Deleted last character from target string");
    }
    else if(SearchTarget.Length < 4) //else then add that char to Search target only if lenght is less that 4 since that is the Max length of rooms example: 033F.
    {
        SearchTarget += sign;
        Debug.Log("Added character to target: " + sign);
    }

    Search_Text.text = SearchTarget;
}
```

Back button as two functions, that depends on what “SearchTarget” currently is. Firstly, if “SearchTarget” is empty, it directs the user back to the main screen. Directing function can be seen below:

```
5 references
public void PlayFade(int mode) //This is attached to buttons that starts fade sequence. This is optional since fade sequence can be skipped by using funcs from => Fade_mode_Funcs().
{
    if(mode == (int)Fade_Modes.FloorUp && Maps.GetComponent<FloorController>().Get_FloorCount() >= 2) { Debug.Log("Cannot go up anymore. Max is 2. Current floor is " + Maps.GetComponent<FloorController>().Get_FloorCount()); }
    else if (mode == (int)Fade_Modes.FloorDown && Maps.GetComponent<FloorController>().Get_FloorCount() <= 0) { Debug.Log("Cannot go lower anymore. Min is 0. Current floor is " + Maps.GetComponent<FloorController>().Get_FloorCount()); }
    else if (mode == (int)Fade_Modes.SearchAct && !GetComponent<Search_Function>().IsSearchFieldEmpty()) { Debug.Log("Deleting one letter."); GetComponent<Search_Function>().PadButton("*"); }
    //else if (mode >= (int)Fade_Modes.Floor0 && mode <= (int)Fade_Modes.Floor2) { } //This should always work.
    else
    {
        Fade_mode = mode;
        StartCoroutine(FadeInOut());
    }
}
```

Secondly it does not direct user to main screen, if “SearchTarget” has something in it. It simply uses the button function and removes one character from the variable.

Using the star (*) sign removes the last character from that string.

Room and reservation object management

Inside Room controller, there are two class objects: “Room Data” and “Reservation”.

Room data contains the following:

```
[System.Serializable]
16 references
public class RoomData
{
    public string room_name;
    public GameObject room_obj;
    public int status;

    public List<Reservation> reservations = new();

    1 reference
    public RoomData(string roomName, GameObject roomObj, int status)
    {
        this.room_name = roomName;
        this.room_obj = roomObj;
        this.status = status;
    }
```

Variable room_name is the name of the stored room such as “033F”. Object means the room object in the 3d map space, where it is displayed to the user. Status integer points, whether room is reserved or not. 0 means free; 1 means reserved. Used integer instead of Boolean so future developers can expand the status system for example: Set 2 to mean, that specific room is completely out of use.

Rooms also have a specific reservation list, where each reservation is stored.

Reservation data contains the following:

```
[System.Serializable]
11 references
public class Reservation
{
    public string label;
    public float startTime;
    public float endTime;

    1 reference
    public Reservation(string label, float startTime, float endTime)
    {
        this.label = label;
        this.startTime = startTime;
        this.endTime = endTime;
    }
```

Variable “label” means reservation name. Both start and end time are as Float.

Times are stored as minutes such like real time is got using function below:

```
4 references
public static class TimeUtils //Helper
{
    5 references
    public static DateTime GetFinnishTime()
    {
        try
        {
            // Windows
            return TimeZoneInfo.ConvertTimeBySystemTimeZoneId(
                DateTime.UtcNow,
                "FLE Standard Time"
            );
        }
        catch
        {
            // Linux
            return TimeZoneInfo.ConvertTimeBySystemTimeZoneId(
                DateTime.UtcNow,
                "Europe/Helsinki"
            );
        }
    }

    1 reference
    public static int GetFinnishMinutesSinceMidnight()
    {
        DateTime finTime = GetFinnishTime();
        return finTime.Hour * 60 + finTime.Minute;
    }
```

Usage of these functions is used inside at other functions, that check reservation status for rooms. There is an made sheet for times:

```
7 references
public enum FinnishTimeSlots // These are times that are used to create start/end times for reservations. These are only hours as minutes like 6 hours is 360 minutes.
{
    H06_00 = 360,  // 6*60
    H07_00 = 420,  // 7*60
    H08_00 = 480,  // 8*60
    H09_00 = 540,  // 9*60
    H10_00 = 600,  // 10*60
    H11_00 = 660,  // 11*60
    H12_00 = 720,  // 12*60
    H13_00 = 780,  // 13*60
    H14_00 = 840,  // 14*60
    H15_00 = 900,  // 15*60
    H16_00 = 960,  // 16*60
    H17_00 = 1020, // 17*60
    H18_00 = 1080, // 18*60
    H19_00 = 1140, // 19*60
    H20_00 = 1200, // 20*60
    H21_00 = 1260  // 21*60
}
```

Adding reservations to specific room

As time of writing there is no real reservation gathering function from lukkari.fi in use. There are only references of old methods in the room controller that can be used to create the real gathering function.

What is in the project currently is the ability to add fictional reservations (But can be used in real situation) to specific room using this function:

```
Unity Script (1 asset reference) | 2 references
public class Room_Controller : MonoBehaviour
{
    3 references
    public bool AddReservationToRoom(string roomName, string label, float startTime, float endTime)
    {
        if (!roomLookupByName.TryGetValue(roomName, out RoomData room)) { Debug.LogWarning($"Room not found: {roomName}"); return false; }

        Reservation res = new Reservation(label, startTime, endTime);
        return room.TryAddReservation(res);
    }
```

Adding can be done like this:

```
//These are test reservations. Should be deleted, if using proper ones. Only use as reference to create adding function;
AddReservationToRoom("131", "Varaustesti", (float)FinnishTimeSlots.H11_00 + 47, (float)FinnishTimeSlots.H13_00+30);
AddReservationToRoom("131", "Varaustesti1", (float)FinnishTimeSlots.H13_00 + 51f, (float)FinnishTimeSlots.H15_00 + 30f);
AddReservationToRoom("131", "Varaustesti2", (float)FinnishTimeSlots.H15_00 + 45f, (float)FinnishTimeSlots.H17_00 + 15f);
```

First is the room name, where the reservation will be stored. Reservation will be added to mention reservation list; that is a room specific. Second is the reservation name that will be used to display for the user in the room info screen. Start and end time are also needed. For debug purposes, you can use Enums like seen above (Clock 11 = (float)FinnishTimeSlots.H11_00 = 660 minutes).

```
1 reference
public bool TryAddReservation(Reservation newRes)
{
    foreach (var res in reservations)
    {
        if (newRes.startTime < res.endTime && newRes.endTime > res.startTime)
        {
            return false;
        }
    }

    reservations.Add(newRes);
    return true;
}
```

Note here that if trying to add reservation, what is happening at same time as some other reservation, it is simply skipping it and not adding it. It is to suppress mistakes done possible during adding process.

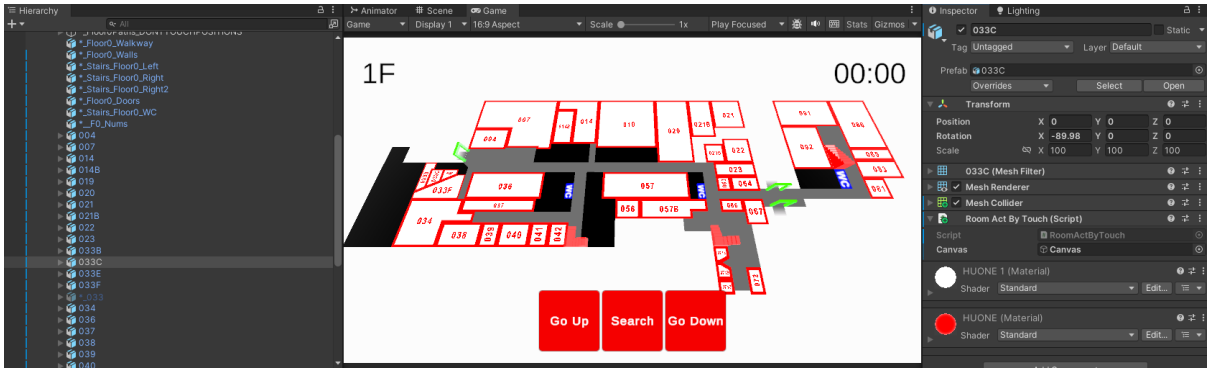
Automatic reservation status updater

```
void Start()
{
    RoomsUpdate();
    StartCoroutine(RoomsUpdateLoop());
}


IEnumerator RoomsUpdateLoop()
{
    while (true)
    {
        RoomsUpdate();
        yield return new WaitForSeconds(600f); // 10 minutes
    }
}


void Update()
{
    currentTime = TimeUtils.GetFinnishTime();
    ShowTime.text = $"{currentTime:HH:mm}";

    if (Input.GetKeyDown(KeyCode.Tab)) { RoomsUpdate(); }
}
```

There is a ten-minute updating time when it comes to updating rooms visually. This does not affect room info, but only rooms in map view. Time can be increased or decreased from line, that has comment.

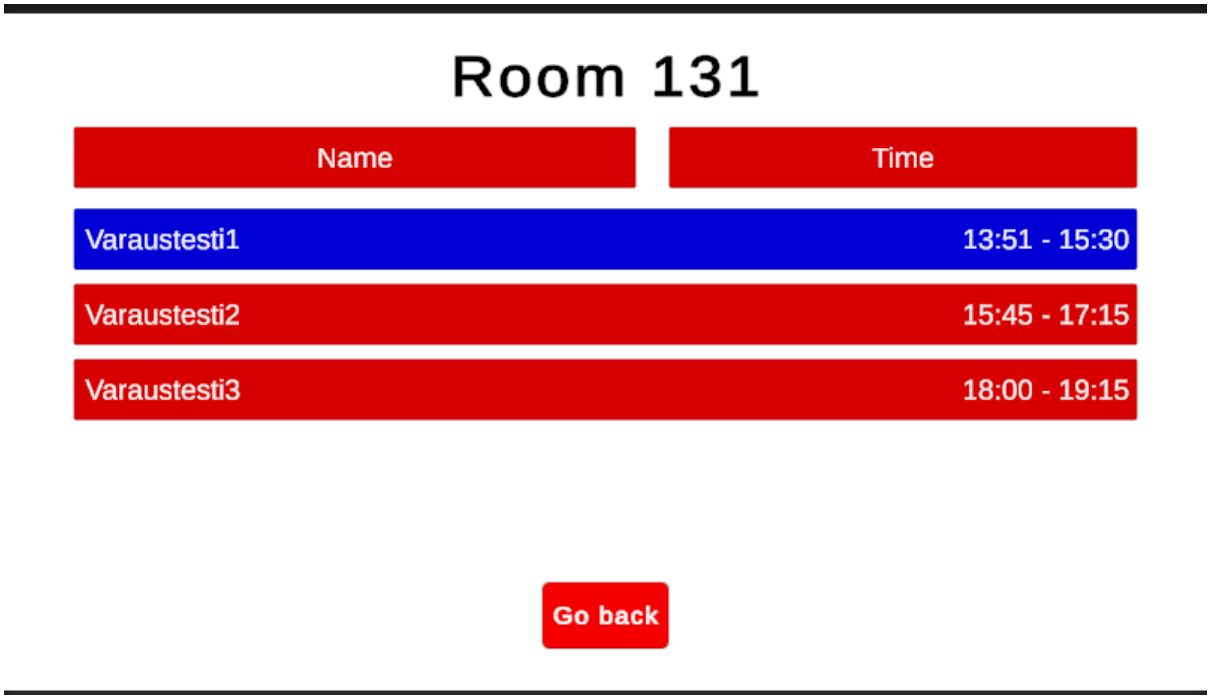
Room info screen

Room info is activated by clicking any room, that has components: “Room act by touch” and mesh collider.



Inside that component, room name is already taken from object name, so you only must add Canvas object to it. Everything else is automated.

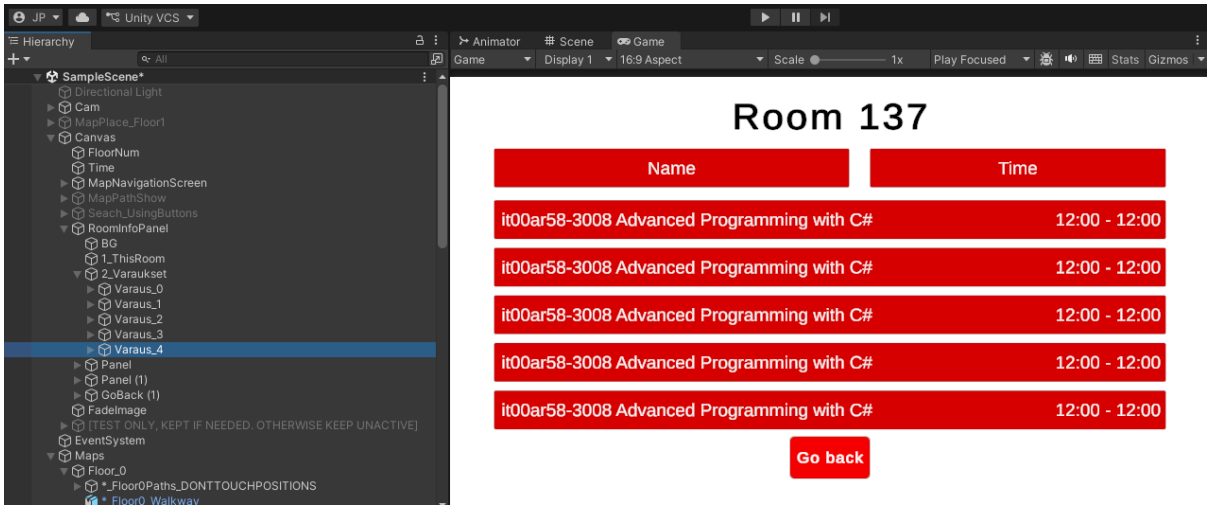
```
Unity Script (92 asset references) | 0 references
public class RoomActByTouch : MonoBehaviour
{
    [SerializeField] public GameObject Canvas;
    private Canvas_Controller canvasC;

    private string This_name;

    Unity Message | 0 references
    private void Awake()
    {
        This_name = gameObject.name;
        canvasC = Canvas.GetComponent<Canvas_Controller>();
    }

    Unity Message | 0 references
    void OnMouseOver()
    {
        //Debug.Log("Currently mouse is on " + This_name);

        if (Input.GetMouseButtonDown(0))
        {
            if(!canvasC.OnMap())
            {
                Debug.Log("Activating room: " + This_name);
                canvasC.SetRoomNameAtCanvas(This_name);
                canvasC.PlayFade(9);
            }
        }
    }
}
```



This is the current room info screen, that display all current and upcoming reservations. All passed reservations are not shown. Currently ongoing reservation are coloured blue.

```
1 reference
public void Show_RoomInformation() //Open the reservation info screen for specific room. Uses room_name to get right room.
{
    bool Go = !Room_Information.activeSelf;

    Room_Information.SetActive(Go);
    if(Go)
    {
        DeactivateAllReservationSlots();
        Debug.Log("Loading room information for: " + room_name);

        Room_Information.transform.GetChild(1).GetComponent<TextMeshProUGUI>().text = "Room " + room_name;

        List<Reservation> list = Maps.GetComponent<Room_Controller>().GetReservationsForRoomByName(room_name);

        if (list != null)
        {
            Debug.Log("Currently this room has: " + Maps.GetComponent<Room_Controller>().GetReservationsForRoomByName(room_name).Count + " reservations.");

            Transform reservationSlots = Room_Information.transform.GetChild(2); int i = 0; int MaxSlots = reservationSlots.childCount; Debug.Log("Max slots for reservations is " + MaxSlots);

            foreach (Reservation reservation in list)
            {
                if(!reservation.IsPassed())
                {
                    reservationSlots.GetChild(i).gameObject.SetActive(true);
                    reservationSlots.GetChild(i).GetChild(0).GetChild(0).GetComponent<TextMeshProUGUI>().text = reservation.label;
                    reservationSlots.GetChild(i).GetChild(0).GetChild(1).GetComponent<TextMeshProUGUI>().text = ProcessThisTime(reservation.startTime / 60) + " - " + ProcessThisTime(reservation.endTime / 60);

                    if (reservation.IsNow()) { reservationSlots.GetChild(i).GetChild(0).GetComponent<Image>().color = Color.blue; }
                    else { reservationSlots.GetChild(i).GetChild(0).GetComponent<Image>().color = Color.red; }

                    i++;
                    if(i > MaxSlots) { break; }
                }
            }

        }
        else { Debug.Log("This room does not have reservations."); }
    }
    else { Debug.Log("Exiting room information panel."); }
}
```

In above function reservations are processed before opening the info screen and where modifications can be made. If wish to display all reservations and ignore, is it passed or not, then replace if statement content inside the foreach loop with true. Current max slots for reservations are 5. If want to expand, then you must add a new child object for parent, that holds those slots:



Object “Varaus_4” is fifth and last slot. Simply copy that and move to place, where it can be seen. Thing is that you must arrange the current info content so more reservations can be seen.

Updating rooms status

Rooms can be visually seen whether it is reserved or not from its color. Currently all rooms have a white outline and a red center color. When the room is reserved, its center is green.

To update rooms, use this function:

```
1 reference
public void RoomsUpdate()
{
    foreach (var room in rooms)
    {
        bool occupied = room.UpdateStatusNow(); // updates status and returns true if occupied

        if (occupied)
        {
            room.SetReserved();
            //Debug.Log($"Room '{room.room_name}' is currently occupied."); //Use to test, if room is not getting right materials applied or is not getting reserved status;
        }
        else
        {
            room.SetFree();
            //Debug.Log($"Room '{room.room_name}' is free."); //Use to test, if room is not getting right materials applied or is not getting reserved status;
        }

        foreach (var res in room.reservations)
        {
            Debug.Log($"Reservation: {res.label}, {res.startTime/60}-{res.endTime/60}");
        }
    }
}
```

In function, it goes over all the rooms and its reservations. So, it is not only checking the status of the room but also updating it to up to date.

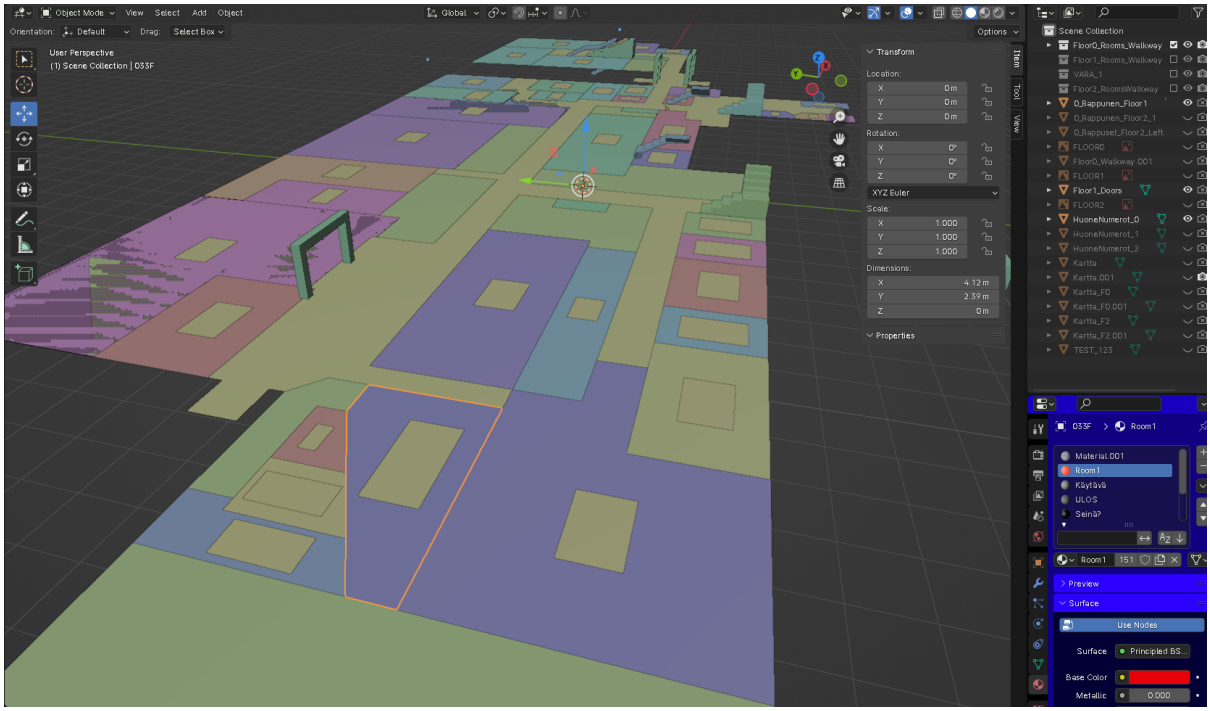
```
1 reference
public bool UpdateStatusNow() //Simply returns if room is being used.
{
    status = IsOccupiedNow() ? 1 : 0;
    return status == 1;
}
1 reference
private bool IsOccupiedNow() //Used at above.
{
    int now = TimeUtils.GetFinnishMinutesSinceMidnight();

    foreach (var res in reservations)
    {
        if (now >= res.startTime && now < res.endTime)
            return true;
    }

    return false;
}
```

Colors are assigned from the room's data class itself, so those had to be loaded from resource folder of the project since those must be static. So, to change the colors of the rooms, go to the resource/materials folder to find these materials.

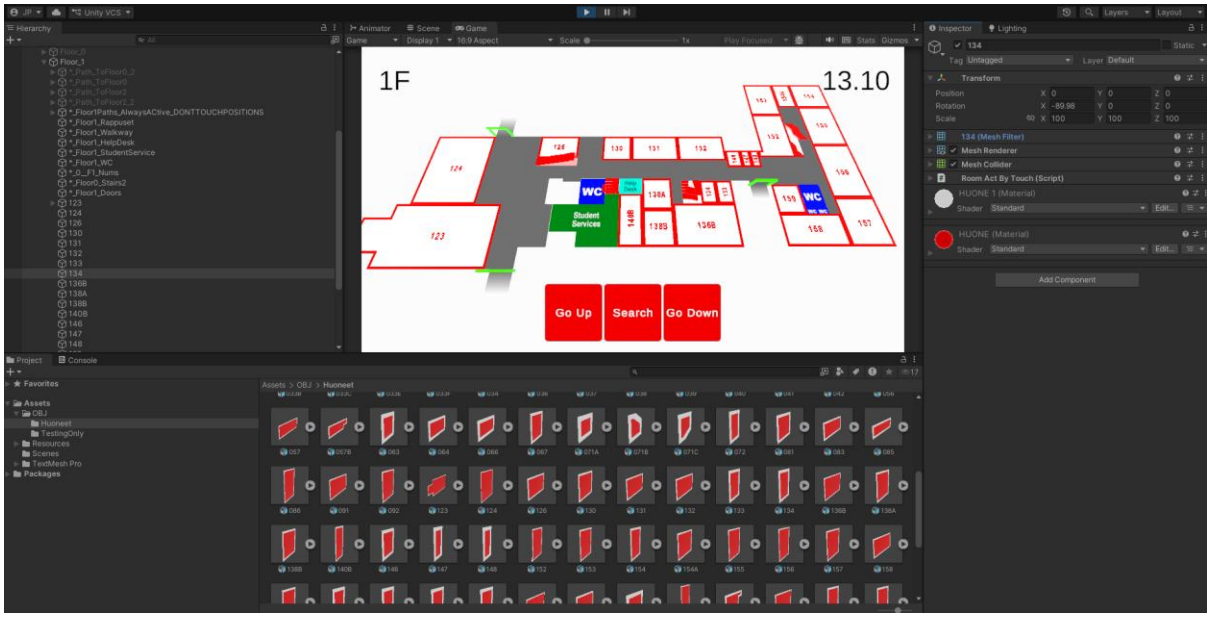
```
1 reference
public static void LoadMats() //This is loading the mats for rooms to use. Loading is called from Room_Controller since this class is not meant to use mono behavior.
{
    Huone_Mat = Resources.Load<Material>("Materials/HUONE"); Debug.Log("Loaded mat: " + Huone_Mat.name + ".");
    HuoneFound_Mat = Resources.Load<Material>("Materials/HUONE_RESERVED"); Debug.Log("Loaded mat: " + HuoneFound_Mat.name + ".");
    Huone_1_Mat = Resources.Load<Material>("Materials/HUONE 1"); Debug.Log("Loaded mat: " + Huone_1_Mat.name + ".");
}

private static Material Huone_Mat;
private static Material HuoneFound_Mat;
private static Material Huone_1_Mat;

2 references
public void SetReserved()
{
    status = 1;
    Renderer rend = room_obj.GetComponent<Renderer>();
    Material[] mats = rend.materials;

    if (mats[0].name.Contains("HUONE 1"))
    {
        mats[0] = HuoneFound_Mat;
        mats[1] = Huone_Mat;
    }
    else
    {
        mats[1] = HuoneFound_Mat;
        mats[0] = Huone_Mat;
    }

    rend.materials = mats;

    //Debug.Log("Found room " + room_name + ". Turning status into 1.");
}

1 reference
public void SetFree()
{
    status = 0;
    Renderer rend = room_obj.GetComponent<Renderer>();
    Material[] mats = rend.materials;

    if (mats[0].name.Contains("HUONE 1"))
    {
        mats[0] = Huone_1_Mat;
        mats[1] = Huone_Mat;
    }
    else
    {
        mats[0] = Huone_Mat;
        mats[1] = Huone_1_Mat;
    }

    rend.materials = mats;
}
```

Blender map objects

All 3D objects are made in Blender. Version 4.0 was used to create the map, so materials used with objects might look different when the newer version is in use. The screenshot is taken in solid viewport shading to highlight objects more clearly.



Objects are divided into folders that are named after the floor. Simply press it open to reveal its content. “VARA_1” folder is used to hold placeholders, test objects and other similarities.



Objects are imported into FBX format and brought as such to the project. Every room is a separate room object so if wanted to only change one room, this decision makes it very simple.

To replace one obj, open file explorer on that location. Then drop the new same named FBX file to the folder and replace the old object. Note if dropping it inside Unity project it simply treats it as a new copy and does not replace it.

Future and further development suggestions

1. Make real reservations only once from Awake() function. It is not a problem for the current system to handle repeating updates to reservation list, but it is much cheaper for less powerful machines, if loading only happens once. But it depends on what machine this program will be put on.