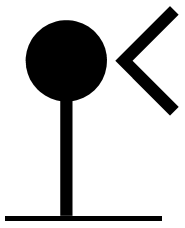
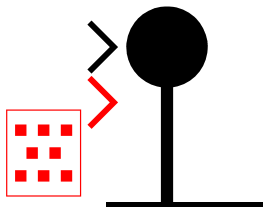
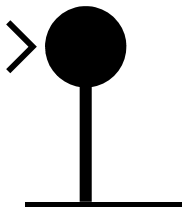


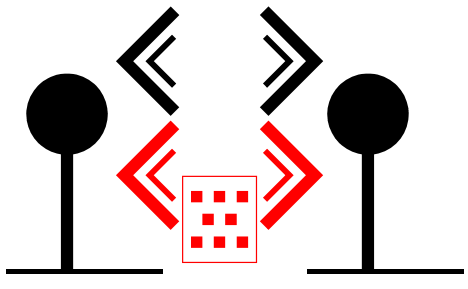
to speak



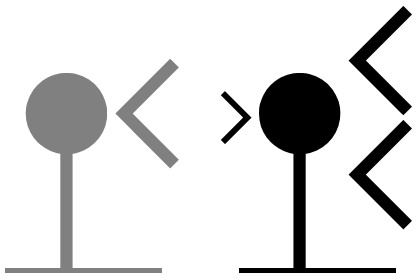
to hear through
an apparatus



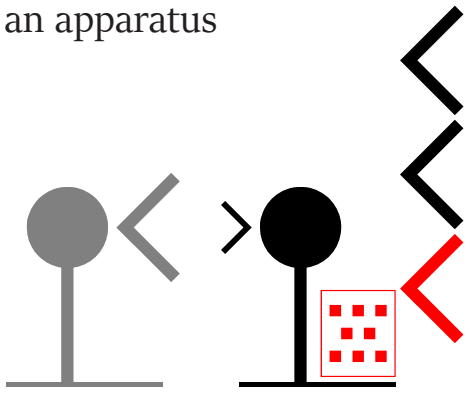
to hear



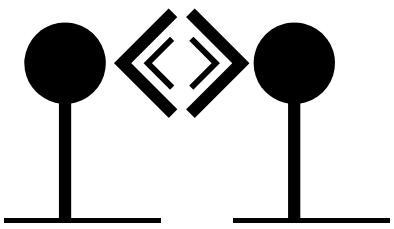
to correspond through
an apparatus



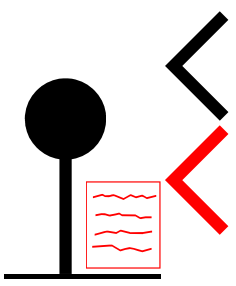
to translate



to translate another's speech
through an apparatus



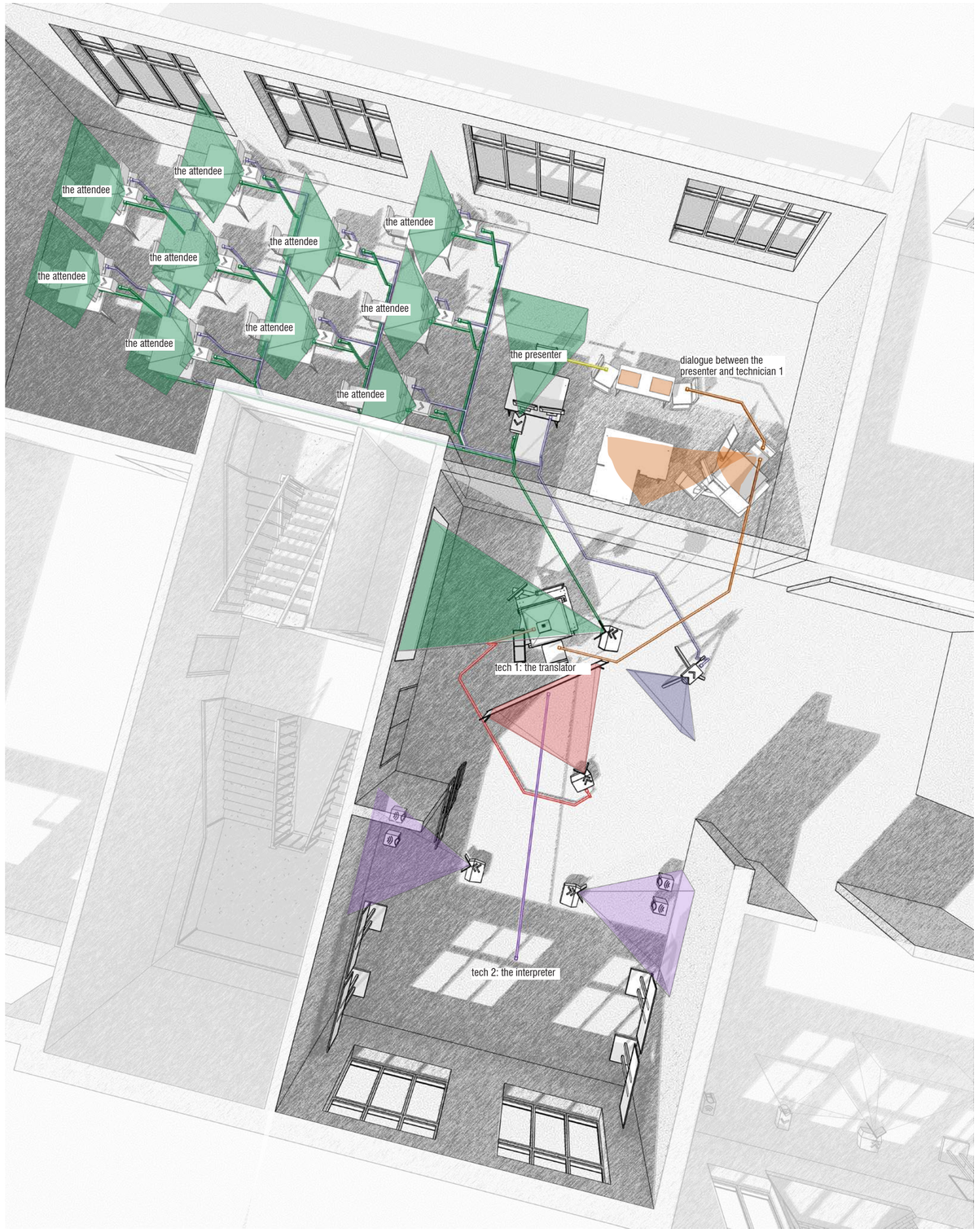
to correspond



to speak through
a manuscript

to speak through
an apparatus

to read a
manuscript



Spatial model for the dialectical computer, developed in Autodesk Revit.

abstract

To attempt to embody and replicate a spatial experience via a digitised format raises inescapable contradictions. Through the utilisation of digital media, the condition of disembodiment is unavoidable.

Whilst using the communication devices that create such 'digital spaces', (Zoom, Skype, Teams), the layers and barriers that exist between the user and the 'digital space' can be seen to cause a state of disembodiment. In doing so, this creates an experiential mode of presence that is encoded by technology. In response to this, the digital venue that will be created to house the Embodied Awareness and Space Symposium is conceived as a space through which to physically re-construct the computational processes that are embedded within digital tools such as Zoom and Skype. In doing so, the symposium constructs a critical response to the socio-political and experiential aspects that are embedded within the digital apparatus, attempting to transform the experience of disembodiment into a positive critique.

The digital means through which these shared experiences are mediated enact a series of translations that rely on languages of computation, ultimately resting on the foundation of binary operations performed by an electro-mechanical processor. As we correspond with a person, object, or drawing via the screen, we become reliant upon the media's digital translation processes.

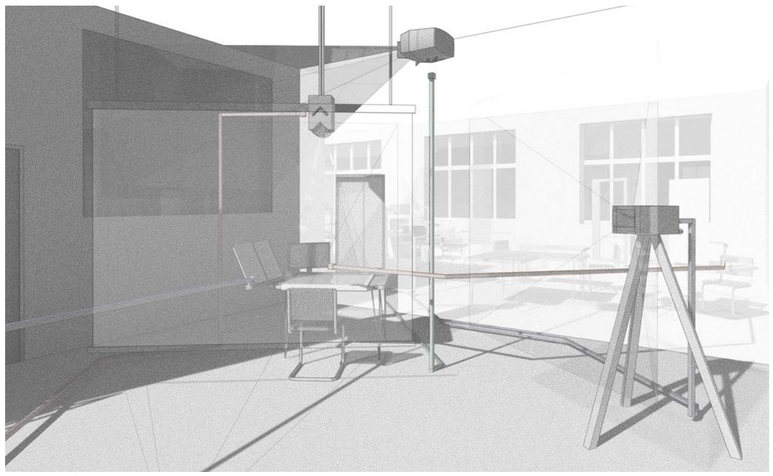
Our co-constitution with technology as a prosthesis, exerting agency in material and cognitive terms, is reflected in the entanglement between technology and culture. This entanglement suggests the possibility of a formal-materialist method: to explore the emergence of formal cultural ideologies that are bound with the material make-up of digital apparatus. This dual approach offers means to forensically excavate the underlying workings of digital languages as they are embedded in computational substrate. These languages may then be re-contextualised, and re-constituted through the construction of a creative practice installation, acting as a poetic critique of inherent dynamics of digital production.

Our inquiry responds to the dynamic of code, which exists both as executable text that is performed by the machine, while also holding the possibility for re-contextualisation. The fixed orthographic mark holds capacity for deferral, paradoxically enabling an 'open' interpretation by the reader. Extracting executable code from the machine offers the opportunity to re-work it toward an alternative, expanding the possibility for writing and interpretation – as the textuality of the writer and reader themselves is constituted by the media.

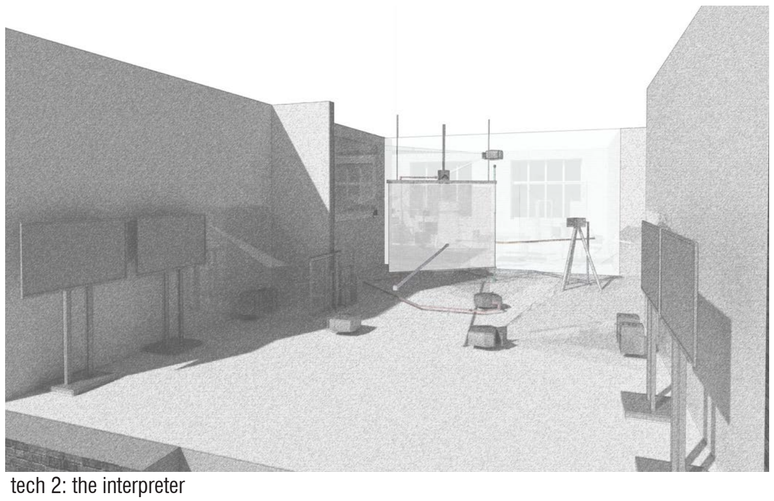
We aim to construct and explore modes of dialectical computation, developing methods of encoding that, through processes of translation and interpretation via differing media, may open up these possibilities within the previously fixed binary of digital logic.

How can a user exist and navigate within two or more spaces at once? How can this be translated, projected and experienced? The intention for this project is to focus on the translation processes inherent within the dynamics of computation, by fragmenting, reconstructing, and materially inhabiting the layers that occur when such a digitally embedded experience is enacted.

the presenter



tech 1: the translator



tech 2: the interpreter

a dialectical computer: the hybrid material/virtual venue

In re-contextualising the translation processes embedded in computation, we construct a dialectical computer to explore how they may be re-enacted and opened toward alternative ways of being.

Often within the dominant history of computation, a task is reduced to a calculable problem, encoded in an electro-mechanical apparatus that performs an automation of its processes. This may result in a reduction of difference, as the encoding of an original in according to the capacity of media catalyses a flattening of homogeneity, reshaping it according to the ontology of the machine and digital abstraction. We aim to build a system that remains open to a multiplicity of inputs and interpretations, re-contextualising the processes of computation in order to open the limits of possibility for designer and user.

the venue

The venue will form a hybrid material and virtual space where presenters can instruct a supplementary presentation to run alongside their main format, broadcast as a second feed. Comprising of two technicians, along with a set of media equipment, the dialectical computer enables the possibility of interpreting material contributed by the participants and enacting it within the space.

Our system is built as a re-construction of computational media, initially drawing from computer code. This code is recontextualised as text: an orthographic mark open to the readers interpretation and no longer subject to requirements of executability by machine.

As a starting point we utilise the C# code that forms Autodesk building modelling software, along with the html that describes webpages, and through a developing textual practice will turn them away from the machine and toward the specificity of the hybrid venue. Acknowledging the gestures, actions, instruction and performance that occurs through the lived experience of symposium.

process

Entering into dialogue with one technician, the participant may offer media including - but not limited to – text, audio, film, photographs, drawings and models which will then be interpreted and performed anew. We invite participants to contribute anything they wish to translate through this alternative form of computation. It might be a piece of work integral to their project, or an impression of the setting where they developed the work, or – reflecting on the dis-embodied nature of the conference – a reconstruction of the place they present from. The material could be left open to interpretation and reconstruction by the technicians, or more rigorously scripted in order to direct a choreographed second stream. These works and spatial experiences will be developed in dialogue with technician 1 – the translator – ahead of the event. It will then be encoded through our developing variations of code, eventually translated into instructions to be interpreted by the second technician.

Technician 1 will encode the presenter's original manuscript through iterative stages, gradually adapting the material to the apparatus in the hybrid venue. Running up to the event, a draft script will be formed that provides the basis for live coding during the symposium. This will then be re-written live and transmitted to the second technician – the interpreter – who will perform a live re-construction from the stream: adapting its content to media equipment including projectors, televisions, speakers, and methods for re-drawing/ modelling.

In the manner of typical computation, which offers a graphical user interface as the primary means of input, we offer a visual account of the venue as a kit of parts to be utilised by the participant. Beyond this we have also started to script the properties of each object in the shape of object oriented coding, written in a syntax that draws from c# language. We invite participants to engage with the visual user guide (overleaf) in tandem with their own preferred methods to develop an initial manuscript.

We will also provide documentation of our initial tests in the coming weeks to illustrate the use of code, offering participants the means to write instructions directly in emerging codes if they wished.

We encourage proposals that expand the existing set of objects, or combine them in order to generate desires spatial experiences and effects.

equipment

The following equipment is available for participants to script instructions for:

primary camera for feed
(position of camera may be modified, or be choreographed to perform movement through the space through the presentation)

2 projectors

6 speakers

4 televisions

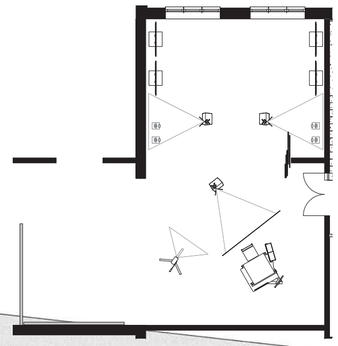
6 spotlights

8 drawing frames in various sizes (tbc)

6 podiums in various sizes (tbc)

4 12.5kg bags of clay in stone and terracotta

Building on an initial test which focusses on digital media, we will formulate means to re-create drawings and models live during the event.



timeframe

present until 31st March
At the present stage we invite participants to consider the material they wish to present in the venue, and how this could take the form of samples or spatial experiences that may be re-enacted in the venue.

1st and 2nd April
On these dates a test run will be performed to experiment with media in translation. The results and document will also be passed to participants to illustrate the possibilities and workflows emerging around the apparatus.

1st - 14th April
In dialogue with one technician as a translator, you can begin to formulate a draft plan for the second feed. This can take any form preferred by the author, such as a hybrid text and drawing akin to a screenplay. It may be scripted with a specific timeframe or left open for interpretation.

15th - 22nd April
Handover: from the complete screenplay, the translator will then complete a process of interpreting and encoding the participants desires into the our own code, to be used for communication of the performance to the second technician (the interpreter).

22nd - 24th April
The symposium: for the event, the translating technician will consult the draft material developed in dialogue with the participant, live-coding it for performance by the second technician, who will execute the instructions as they perceive them – performing an embodied reconstruction alongside the participant's primary conference feed.

to speak

to hear

to translate

to correspond

to speak through
an apparatus

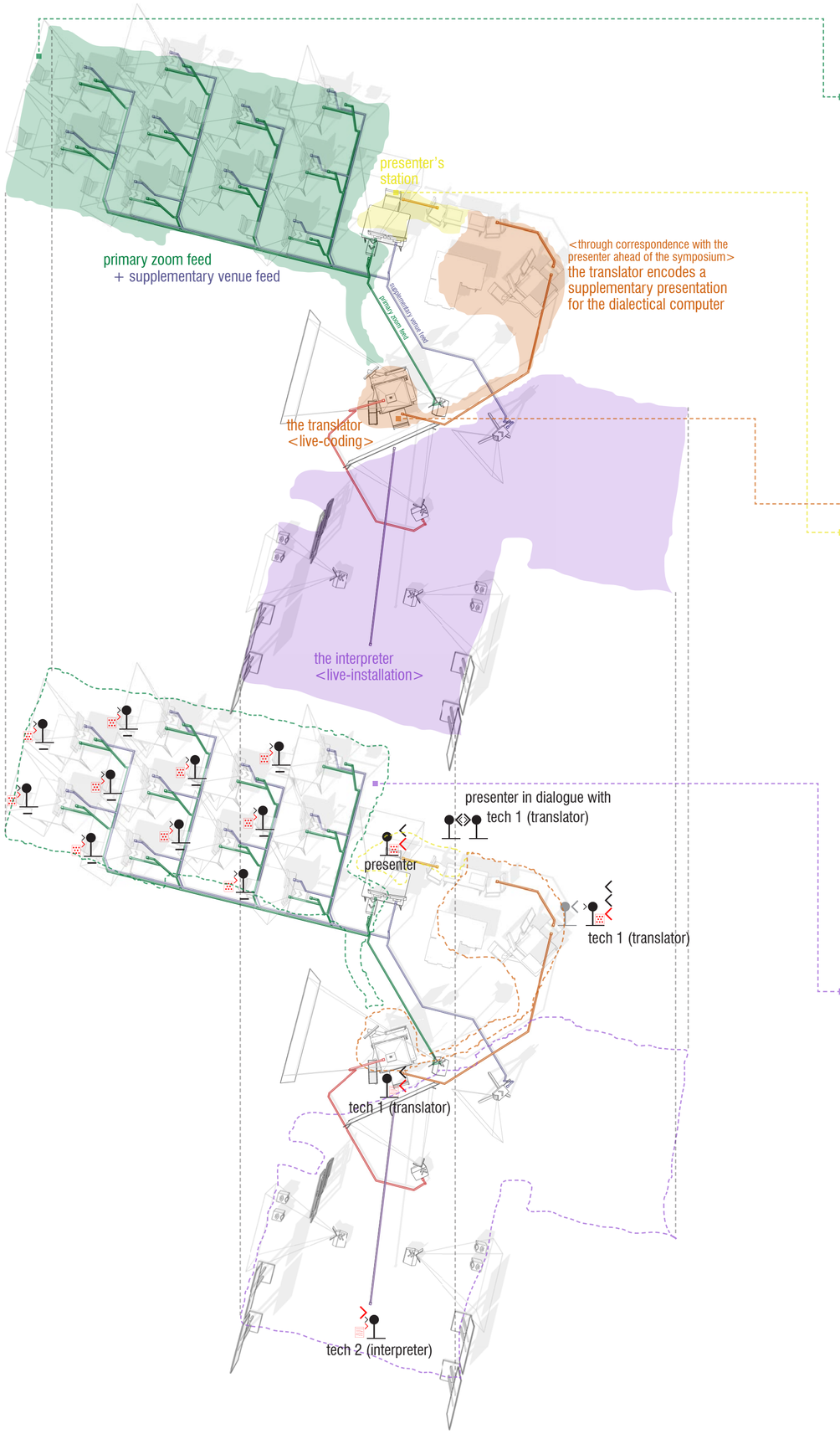
to hear through
an apparatus

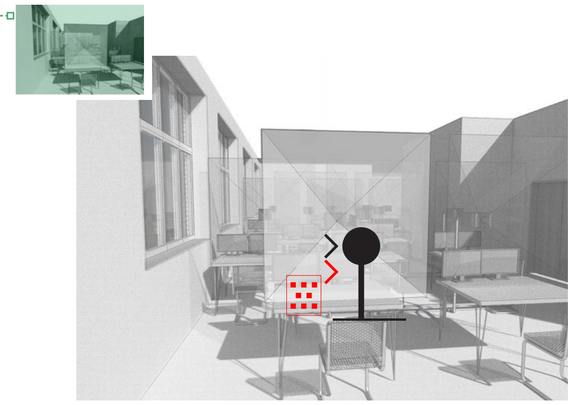
to correspond through
an apparatus

to translate another's speech
through an apparatus

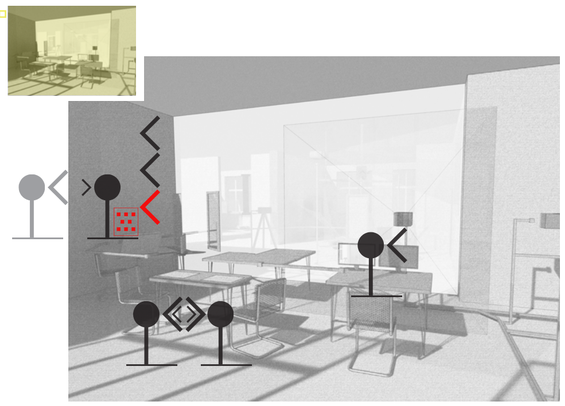
to speak through
a manuscript

to read a manuscript

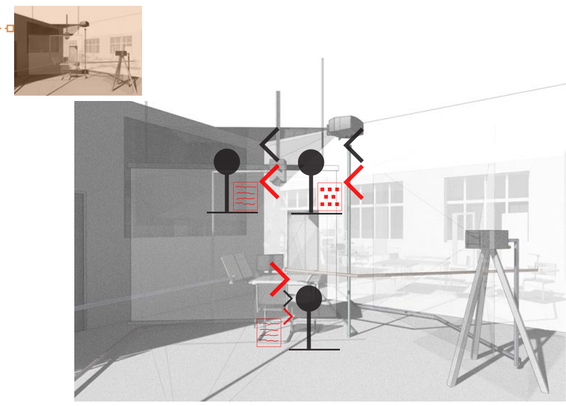




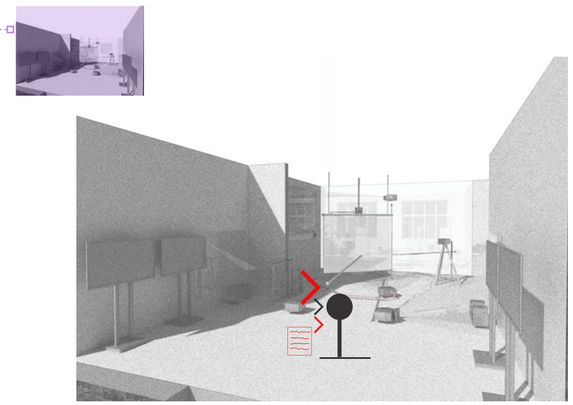
the attendee



the presenter



tech 1: the translator



tech 2: the interpreter

Appendix 1:
An initial draft of the object oriented code, forming a point of departure for the dialectical computer.

Meta-Structure

Document structure
From html

Drawing reconstruction method
Use Revit shortcuts in spatialised layout – find means to accelerate coded description for long handed reproduction by tech 2. Develop an expanded lexicon.

Meta-code

Timestamp
Digitised clock timer, minutes and seconds relative to start and end of presentation.
format = [hh.mm.ss] (double, double. double, double. double, double,);

Time

Start
Start = Time(--.--.--)

End
End = Time(--.--.--)

public void Add
Summary:
Adds the specified vector to this vector and returns the result.
Parameters:
source: The vector to add to this vector.
Returns:
The vector equal to the sum of the two vectors.
Remarks:
The added vector is obtained by adding each coordinate of the specified vector to the corresponding coordinate of this vector.

Add
public Autodesk.Revit.DB.XYZ **Add**(Autodesk.Revit.DB.XYZ source)

public void Subtract
Subtract

public void Present
= Time(--.--.--)

public void Duration
= (TimeStart) Substract (TimeEnd);

Is permanent? If so keep in position until end of presentation.
.toend

while.objectTime.Duration

Semantic version:
start, end, before, during, after

Coordinate systems

class XYZ
XYZ = [XYZ] (double, double, double);

class Plane
= infinite planar object,
For orthographic:
Plane XYZ = [XYZ] (double, double, double);

For non-orthographic: [develop equation to subtract/add coordinate system according to angle]

class Axis
Axis = Plane

class Vector
Vector.Origin [XYZ] (double, double, double);
Vector.Magnitude = (double);
Vector.Bearing = (double);
Vector.End = [XYZ] (double, double, double);

Vector trueNorth = new vector;

trueNorth.Origin [XYZ] (0, 0, 0);
trueNorth.Magnitude = (∞);
trueNorth.Bearing = (0);
trueNorth.End = [XYZ] (∞, ∞, 0);

class Rotation
Bearing = clockwise 360°. origin at Vector trueNorth

Yaw
Roll
Axis = Plane

Bearing

EuclideanGeometry
Euclidean = [Axis X, Axis X, Axis X} (Plane, Plane, Plane);

Plane EuclidX = [XYZ] (X=∞, Y=0, Z=0);
Plane EuclidY = [XYZ] (X=0, Y=∞, Z=0);
Plane EuclidZ = [XYZ] (X=0, Y=0, Z=∞);

Axis X = Plane X;
Axis Y = Plane Y;
Axis Z = Plane Z;

+ semantic description of space (between, in front of, behind...)

Venue.CoordSys = EuclideanGeometry;

*** develop method to place directly in front of interpreter technician's current position**
*paper relative to technician

class Interpreter
position = XYZ;
direction = rotation from Venue.CoordSys

PaperEuclid = new EuclideanGeometry;

Plane PaperEuclidX = from (paper.getsurface)
{
X = minvalue
}

Plane PaperEuclidY= from (paper.getsurface)
{
X = minvalue
}

Plane PaperEuclidZ = paper.getsurface;

PaperEuclid(PaperEuclidX, PaperEuclidY, PaperEuclidZ);

Paper.CoordSys = PaperEuclid;
Axis X = Plane X;
Axis Y = Plane Y;
Axis Z = Plane Z;

ModelEuclid = new EuclideanGeometry;

Plane ModelEuclidX = [XYZ] (X=∞, Y=0, Z=0);
Plane ModelEuclidY = [XYZ] (X=0, Y=∞, Z=0);
Plane ModelEuclidZ = [XYZ] (X=0, Y=0, Z=∞);

ModelEuclid(ModelEuclidX, ModelEuclidY, ModelEuclidZ);

Methods:
HardcodeXYZ, or get podium/worksurface reference to reset

Model.CoordSys = ModelEuclid;

Class Position
Reference origin = (string);
Position.Coord [XYZ] (double, double, double);

Object position reference

Classes:

Camera
  Position (XYZ);
  Rotation (double);
  Target (Position XYZ);
  Start (Time);
  End (Time);
  Duration (Time);
  Input();

Projector
  Position (XYZ);
  Rotation (double);
  Target (Position XYZ);
  Start (Time);
  End (Time);
  Duration (Time);
  Input();

Light
  Position (XYZ);
  Rotation (double);
  Target (Position XYZ);
  Start (Time);
  End (Time);
  Duration (Time);
  Input();
  Level(double)

Speaker
  Position (XYZ);
  Rotation (double);
  Target (Position XYZ);
  Start (Time);
  End (Time);
  Duration (Time);
  Input();
  Level(double)

Camera
  Position (XYZ);
  Rotation (double);
  Target (Position XYZ);
  Start (Time);
  End (Time);
  Duration (Time);

Television
  Position (XYZ);
  Rotation (double);
  Target (Position XYZ);
  Start (Time);
  End (Time);
  Duration (Time);
  Input();
  Level(double)

Text
  Position (XYZ);
  Rotation (double);
  Target (Position XYZ);
  Start (Time);
  End (Time);
  Duration (Time);

Drawing
  CoordSys();
  Position (XYZ);
  Rotation (double);
  Target (Position XYZ);
  Start (Time);
  End (Time);
  Duration (Time);

Model
  CoordSys();
  Position (XYZ);
  Rotation (double);
  Target (Position XYZ);
  Start (Time);
  End (Time);
  Duration (Time);

Data types:

Number (double or long)
Text
Boolean (true/false)
Spatial diagram
Emergent from gestures operating apparatus: instruction, action, memory, recollection, improvisation, ambiguity

Functions:
For
While
If

Array
List

Equals
Less than
More than

Until
Before
After

Syntax

Initial draft of script:

include Time;
include EuclideanXYZ;

// Declare objects

new Projector (blueProjector);
new Projector (redProjector);
new Projector (greenProjector);

new Speaker (blueSpeaker);
new Speaker (redSpeaker);
new Speaker (greenSpeaker);
new Speaker (orangeSpeaker);
new Speaker (cyanSpeaker);
new Speaker (purpleSpeaker);

new Television (blueTelevision);
new Television (redTelevision);
new Television (greenTelevision);
new Television (orangeTelevision);

new Light (blueLight);
new Light (redLight);
new Light (greenLight);
new Light (orangeLight);

new Drawingframe (blueDrawingframe);
new Drawingframe (redDrawingframe);
new Drawingframe (greenDrawingframe);
new Drawingframe (orangeDrawingframe);
new Drawingframe (cyanDrawingframe);
new Drawingframe (purpleDrawingframe);

new Drawing (blueDrawing);
new Drawing (redDrawing);
new Drawing (greenDrawing);
new Drawing (orangeDrawing);
new Drawing (cyanDrawing);
new Drawing (purpleDrawing);

new Podium (bluePodium);
new Podium (redPodium);
new Podium (greenPodium);
new Podium (orangePodium);
new Podium (cyanPodium);
new Podium (purplePodium);

new Model (blueModel);
new Model (redModel);
new Model (greenModel);
new Model (orangeModel);
new Model (cyanModel);
new Model (purpleModel);

new Time = testinstall;
testinstall.Time.Start = 00.00.00
testinstall.Time.End= 00.20.00

Appendix 2
Excerpts of code used with Autodesk Revit's Application Programming Interface: a means to hardcode a model directly through text.

E:\PhD\1\Coding\Revit\Live coding\Hardcode_dwelling_build_1\Hardcode_dwelling_build_1\Class1.cs

```
using System;
using Autodesk.Revit.UI;
using Autodesk.Revit.DB;
using Autodesk.Revit.UI.Selection;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using Autodesk.Revit.DB.Events;
using Autodesk.Revit.DB.Architecture;

namespace Hardcode_dwelling_build_1
{
    [Autodesk.Revit.Attributes.Transaction(Autodesk.Revit.Attributes.TransactionMode.Manual)]
    [Autodesk.Revit.DB.Macros.AddInId("B57B5E5D-DA21-4BE0-94DC-312CBA592D5E")]
    public partial class ThisDocument
    {
        private void Module_Startup(object sender, EventArgs e)
        {

        }

        private void Module_Shutdown(object sender, EventArgs e)
        {

        }

        #region Revit Macros generated code
        private void InternalStartup()
        {
            this.Startup += new System.EventHandler(Module_Startup);
            this.Shutdown += new System.EventHandler(Module_Shutdown);
        }
        #endregion

        public void HrdCdDw1()

        {
            // in-file macro handle
            UIApplication uiapp = this.Application;
            UIDocument uidoc = uiapp.ActiveUIDocument;
            Document doc = uidoc.Document;

            // Get the handle of the application
            Autodesk.Revit.ApplicationServices.Application application = doc.Application;


            // Create Axonometric View //

            using (Transaction tAxo = new Transaction(doc, "Create Axonometric View"))
            {
                tAxo.Start();
                XYZ eye = new XYZ(20.6414075243214, -1.15042343338695, 21.8796790905289);
                XYZ forward = new XYZ(-0.577350269189626, 0.577350269189626, -0.577350269189626);
                XYZ axoup = new XYZ(-0.408248290463863, 0.408248290463863, 0.816496580927726);

                ViewOrientation3D viewOrientation3D = new ViewOrientation3D(eye, axoup, forward);
                ViewFamilyType viewFamilyType3D = new FilteredElementCollector(doc).OfClass(typeof(ViewFamilyType))
                    .Cast<ViewFamilyType>().FirstOrDefault<ViewFamilyType>(x => ViewFamily.ThreeDimensional == x.ViewFamily);

                View3D Axonometric = View3D.CreateIsometric(doc, viewFamilyType3D.Id);
                Axonometric.SetOrientation(viewOrientation3D);
                Axonometric.Name = "Axonometric View";

                View viewTemplate = (from v in new FilteredElementCollector(doc).OfClass(typeof(View))
                                     .Cast<View>()
                                     where v.IsTemplate == true && v.Name == "1 to 50 Axo"
                                     select v).First();

                Axonometric.ViewTemplateId = viewTemplate.Id;

                tAxo.Commit();
            }
            TaskDialog.Show("stage", "Create Axonometric View");


            // Create Levels //

            using (Transaction tLevels = new Transaction(doc, "Create Levels"))
            {
                tLevels.Start();
                Level levelFoundation = Level.Create(doc, -4.5);
                levelFoundation.Name = "levelFoundation";
                Level levelFooting = Level.Create(doc, -2);
                levelFooting.Name = "levelFooting";
                Level levelExtGround = Level.Create(doc, -1);
                levelExtGround.Name = "levelExtGround";
                Level levelGround = Level.Create(doc, 0);
                levelGround.Name = "levelGround";
                Level levelFirst = Level.Create(doc, 10);
                levelFirst.Name = "levelFirst";
                Level levelRoof = Level.Create(doc, 19.75);
                levelRoof.Name = "levelRoof";
                tLevels.Commit();

                TaskDialog.Show("stage", "Create Levels");


                // Create Plan Views //

                using (Transaction tPlans = new Transaction(doc, "Create Plan Views"))
                {
                    tPlans.Start();

                    FilteredElementCollector collector = new FilteredElementCollector(doc);
                    IList<Element> viewFamilyTypes = collector.OfClass(typeof(ViewFamilyType)).ToElements();
                    ElementId floorPlanId = new ElementId(-1);
                    foreach (Element e in viewFamilyTypes)
                    {
                        ViewFamilyType v = e as ViewFamilyType;

                        if (v != null && v.ViewFamily == ViewFamily.FloorPlan)
                        {
                            floorPlanId = e.Id;
                            break;
                        }
                    }

                    View viewplan150Template = (from vpl in new FilteredElementCollector(doc).OfClass(typeof(View))
                                                .Cast<View>()
                                                where vpl.IsTemplate == true && vpl.Name == "1 50 Plan"
                                                select vpl).First();

                    ViewPlan groundfloorView = ViewPlan.Create(doc, floorPlanId, levelGround.Id);
                    PlanViewRange gfPv = groundfloorView.GetViewRange();
                    ElementId gftopClipPlane = gfPv.GetLevelId(PlanViewPlane.TopClipPlane);
                    gfPv.SetOffset(PlanViewPlane.TopClipPlane, 4);
                    gfPv.SetOffset(PlanViewPlane.CutPlane, 4);
                    gfPv.SetLevelId(PlanViewPlane.TopClipPlane, PlanViewRange.Unlimited);
                    groundfloorView.SetViewRange(gfPv);
                    groundfloorView.ViewTemplateId = viewplan150Template.Id;

                    ViewPlan firstfloorView = ViewPlan.Create(doc, floorPlanId, levelFirst.Id);
                    PlanViewRange ffPv = firstfloorView.GetViewRange();
                    ElementId fftopClipPlane = ffPv.GetLevelId(PlanViewPlane.TopClipPlane);
                    ffPv.SetOffset(PlanViewPlane.TopClipPlane, 4);
                    ffPv.SetOffset(PlanViewPlane.CutPlane, 4);
                    ffPv.SetLevelId(PlanViewPlane.TopClipPlane, PlanViewRange.Unlimited);
                    firstfloorView.SetViewRange(ffPv);
                    firstfloorView.ViewTemplateId = viewplan150Template.Id;

                    ViewPlan rooflevelView = ViewPlan.Create(doc, floorPlanId, levelRoof.Id);
                    PlanViewRange rlPv = firstfloorView.GetViewRange();
                    ElementId rltopClipPlane = rlPv.GetLevelId(PlanViewPlane.TopClipPlane);
                    rlPv.SetOffset(PlanViewPlane.TopClipPlane, 16);
                    rlPv.SetOffset(PlanViewPlane.CutPlane, 22);
                    rlPv.SetLevelId(PlanViewPlane.TopClipPlane, PlanViewRange.Unlimited);
                    rooflevelView.SetViewRange(ffPv);
                    rooflevelView.ViewTemplateId = viewplan150Template.Id;

                    XYZ pla = new XYZ(-3, -3, 0);
                    XYZ plb = new XYZ(-3, 32.5, 0);
                    XYZ plc = new XYZ(22, 32.5, 0);
                    XYZ pld = new XYZ(22, -3, 0);
                    Line plab = (Line.CreateBound(pla, plb));
                    Line plbc = (Line.CreateBound(plb, plc));
                    Line plcd = (Line.CreateBound(plc, pld));
                    Line plda = (Line.CreateBound(pld, pla));
                    CurveLoop planloop = new CurveLoop();
                    planloop.Append(plab);
                    planloop.Append(plbc);
                    planloop.Append(plcd);
                    planloop.Append(plda);

                    ViewCropRegionShapeManager vcrShapeMgr = firstfloorView.GetCropRegionShapeManager();
                    if (vcrShapeMgr.CanHaveShape) vcrShapeMgr.SetCropShape(planloop); else TaskDialog.Show("error", "plan crop incomplete");

                    tPlans.Commit();
                }
                TaskDialog.Show("stage", "Create Plan Views");


                // wall midpoint corner datum set
                XYZ a = new XYZ(0, 0, 0);
                XYZ b = new XYZ(a.X, a.Y + 26.5, a.Z);
                XYZ c = new XYZ(a.X + 16, b.Y, a.Z);
                XYZ d = new XYZ(c.X, a.Y, a.Z);
                Line ab = (Line.CreateBound(a, b));
                Line bc = (Line.CreateBound(b, c));
                Line cd = (Line.CreateBound(c, d));
                Line da = (Line.CreateBound(d, a));


                // Create Section Views //

                ViewFamilyType vft = new FilteredElementCollector(doc).OfClass(typeof(ViewFamilyType)).Cast<ViewFamilyType>()
                    .FirstOrDefault<ViewFamilyType>(x => ViewFamily.Section == x.ViewFamily);

                View viewsection150Template = (from vsaa in new FilteredElementCollector(doc).OfClass(typeof(View))
                                               .Cast<View>()
                                               where vsaa.IsTemplate == true && vsaa.Name == "1 50 Section"
                                               select vsaa).First();

                using (Transaction tSectionAA = new Transaction(doc))
                {
                    tSectionAA.Start("Create Section View");

                    BoundingBoxXYZ sectionBox = new BoundingBoxXYZ();
                    sectionBox.Min = new XYZ(2, -3, -8.5);
                    sectionBox.Max = new XYZ(17, 28.5, 24);
                    XYZ v = b - a;

                    double minZ = sectionBox.Min.Z;
                    double maxZ = sectionBox.Max.Z;
                    double w = v.GetLength();
                    double h = maxZ - minZ;
                    double offset = 16;

                    XYZ min = new XYZ(-w, minZ - offset, -offset);
                    XYZ max = new XYZ(w, maxZ + offset, 0);

                    XYZ viewrhspoint = new XYZ(v.X + 18.05, 15, v.Z);
                    XYZ direction = v.Normalize();
                    XYZ up = XYZ.BasisZ;
                    XYZ viewdir = direction.CrossProduct(up);

                    Transform t = Transform.Identity;
                    t.Origin = viewrhspoint;
                    t.BasisX = direction;
                    t.BasisY = up;
                    t.BasisZ = viewdir;

                    sectionBox.Transform = t;
                    sectionBox.Min = min;
                    sectionBox.Max = max;

                    ViewSection sectionAA = ViewSection.CreateSection(doc, vft.Id, sectionBox);
                    sectionAA.ViewTemplateId = viewsection150Template.Id;

                    tSectionAA.Commit();
                }

                using (Transaction tSectionBB = new Transaction(doc))
                {
                    tSectionBB.Start("Create Section View");

                    BoundingBoxXYZ sectionBox = new BoundingBoxXYZ();
                    sectionBox.Min = new XYZ(2, -3, -8.5);
                    sectionBox.Max = new XYZ(17, 28.5, 24);
                    XYZ v = a - b;

                    double minZ = sectionBox.Min.Z;
                    double maxZ = sectionBox.Max.Z;
                    double w = v.GetLength();
                    double h = maxZ - minZ;
                    double offset = 16;

                    XYZ min = new XYZ(-w, minZ - offset, -offset);
                    XYZ max = new XYZ(w, maxZ + offset, 0);

                    XYZ viewrhspoint = new XYZ(v.X - 1.6, 15, v.Z);
                    XYZ direction = v.Normalize();
                    XYZ up = XYZ.BasisZ;
                    XYZ viewdir = direction.CrossProduct(up);

                    Transform t = Transform.Identity;
                    t.Origin = viewrhspoint;
                    t.BasisX = direction;
                    t.BasisY = up;
                    t.BasisZ = viewdir;

                    sectionBox.Transform = t;
                    sectionBox.Min = min;
                    sectionBox.Max = max;

                    ViewSection sectionBB = ViewSection.CreateSection(doc, vft.Id, sectionBox);
                    sectionBB.ViewTemplateId = viewsection150Template.Id;

```

E:\PhD\1\Coding\Revit\Live coding\Hardcode_dwelling_build_1\Hardcode_dwelling_build_1\Class1.cs

```
                    tSectionBB.Commit();
                }

                using (Transaction tSectionCC = new Transaction(doc))
                {
                    tSectionCC.Start("Create Section View");

                    BoundingBoxXYZ sectionBox = new BoundingBoxXYZ();
                    sectionBox.Min = new XYZ(-3.5, -3.5, -8.5);
                    sectionBox.Max = new XYZ(19.5, 27.35, 33);

                    XYZ v = d - a;

                    XYZ min = sectionBox.Min;
                    XYZ max = sectionBox.Max;

                    XYZ midpointX = (sectionBox.Max + sectionBox.Min) / 2;
                    XYZ viewmidpoint = new XYZ(0, midpointX.Y - 15.22, 0);

                    XYZ direction = v.Normalize();
                    XYZ up = XYZ.BasisZ;
                    XYZ viewdir = direction.CrossProduct(up);

                    Transform t = Transform.Identity;
                    t.Origin = viewmidpoint;
                    t.BasisX = direction;
                    t.BasisY = up;
                    t.BasisZ = viewdir;

                    sectionBox.Transform = t;

                    ViewSection sectionCC = ViewSection.CreateSection(doc, vft.Id, sectionBox);
                    sectionCC.ViewTemplateId = viewsection150Template.Id;

                    tSectionCC.Commit();
                }

                using (Transaction tSectionDD = new Transaction(doc))
                {
                    tSectionDD.Start("Create Section View");

                    BoundingBoxXYZ sectionBox = new BoundingBoxXYZ();
                    sectionBox.Min = new XYZ(-3.5, -3.5, -8.5);
                    sectionBox.Max = new XYZ(19.5, 27.35, 33);

                    XYZ v = a - d;

                    XYZ min = sectionBox.Min;
                    XYZ max = sectionBox.Max;

                    XYZ midpointX = (sectionBox.Max + sectionBox.Min) / 2;
                    XYZ viewendpoint = new XYZ(16, midpointX.Y + 8.55, 0); //3.43

                    XYZ direction = v.Normalize();
                    XYZ up = XYZ.BasisZ;
                    XYZ viewdir = direction.CrossProduct(up);

                    Transform t = Transform.Identity;
                    t.Origin = viewendpoint;
                    t.BasisX = direction;
                    t.BasisY = up;
                    t.BasisZ = viewdir;

                    sectionBox.Transform = t;

                    ViewSection sectionDD = ViewSection.CreateSection(doc, vft.Id, sectionBox);
                    sectionDD.ViewTemplateId = viewsection150Template.Id;

                    tSectionDD.Commit();
                }
                TaskDialog.Show("stage", "Create Section Views");


                // Create Foundations //

                FilteredElementCollector foundationwallcollector = new FilteredElementCollector(doc);
                foundationwallcollector = new FilteredElementCollector(doc);
                foundationwallcollector.OfClass(typeof(WallType));
                var foundationwalltypeelements = from element in foundationwallcollector where element.Name == "Wall-Fnd_600Con_Footing" select element;
                WallType foundationwall = foundationwalltypeelements.Cast<WallType>().ElementAt<WallType>(0);

                using (Transaction tFoundations = new Transaction(doc, "Create Foundations"))
                {
                    tFoundations.Start();
                    Wall foa = Wall.Create(doc, ab, foundationwall.Id, levelFoundation.Id, 2.5, 0, false, true);
                    Wall fob = Wall.Create(doc, bc, foundationwall.Id, levelFoundation.Id, 2.5, 0, false, true);
                    Wall foc = Wall.Create(doc, cd, foundationwall.Id, levelFoundation.Id, 2.5, 0, false, true);
                    Wall fod = Wall.Create(doc, da, foundationwall.Id, levelFoundation.Id, 2.5, 0, false, true);
                    tFoundations.Commit();
                }
                TaskDialog.Show("stage", "Create Foundations" + Environment.NewLine + "type : " + foundationwall.Name);


                // Create Wall Footings //

                FilteredElementCollector footingwallcollector = new FilteredElementCollector(doc);
                footingwallcollector = new FilteredElementCollector(doc);
                footingwallcollector.OfClass(typeof(WallType));
                var footingwalltypeelements = from element in footingwallcollector where element.Name == "Wall-Ext_1ft_Cavity" select element;
                WallType footingwall = footingwalltypeelements.Cast<WallType>().ElementAt<WallType>(0);

                using (Transaction tFootings = new Transaction(doc, "Create Footings"))
                {
                    tFootings.Start();
                    Wall foa = Wall.Create(doc, ab, footingwall.Id, levelFooting.Id, 2, 0, false, true);
                    Wall fob = Wall.Create(doc, bc, footingwall.Id, levelFooting.Id, 2, 0, false, true);
                    Wall foc = Wall.Create(doc, cd, footingwall.Id, levelFooting.Id, 2, 0, false, true);
                    Wall fod = Wall.Create(doc, da, footingwall.Id, levelFooting.Id, 2, 0, false, true);
                    tFootings.Commit();
                }
                TaskDialog.Show("stage", "Create Wall Footings" + Environment.NewLine + "type : " + footingwall.Name);


                // Create Ground Floor Slab //

                //internal corner datum set
                XYZ ia = new XYZ(a.X + 0.5, a.Y + 0.5, a.Z);
                XYZ ib = new XYZ(b.X + 0.5, b.Y - 0.5, a.Z);
                XYZ ic = new XYZ(c.X - 0.5, ib.Y, a.Z);
                XYZ id = new XYZ(d.X - 0.5, ia.Y, a.Z);
                Line iab = (Line.CreateBound(ia, ib));
                Line ibc = (Line.CreateBound(ib, ic));
                Line icd = (Line.CreateBound(ic, id));
                Line ida = (Line.CreateBound(id, ia));

                FloorType groundFloorSlab = new FilteredElementCollector(doc).OfClass(typeof(FloorType))
                    .First<Element>(e => e.Name.Equals("Floor-Grnd-Bearing_65Scr-90Ins-125Conc-50SBld-150Hcore")) as FloorType;

                using (Transaction tGroundSlab = new Transaction(doc, "Create Ground Floor Slab"))
                {
                    CurveArray slabCurves = application.Create.NewCurveArray();
                    slabCurves.Append(iab);
                    slabCurves.Append(ibc);
                    slabCurves.Append(icd);
                    slabCurves.Append(ida);
                    XYZ normal = XYZ.BasisZ;

                    tGroundSlab.Start();
                    Floor grndslab = doc.Create.NewFloor(slabCurves, groundFloorSlab, levelGround, false, normal);
                    grndslab.get_Parameter(BuiltInParameter.FLOOR_HEIGHTABOVELEVEL_PARAM).Set(0);
                    tGroundSlab.Commit();
                    TaskDialog.Show("stage", "Create Ground Floor Slab" + Environment.NewLine + "type : " + groundFloorSlab.Name);

                    XYZ bpa = new XYZ(b.X + 1, b.Y, a.Z);
                    XYZ bpb = new XYZ(c.X - 1, c.Y, a.Z);
                    XYZ dpa = new XYZ(d.X - 1, d.Y, a.Z);
                    XYZ dpb = new XYZ(a.X + 1, d.Y, a.Z);
                    Line bbpa = (Line.CreateBound(b, bpa));
                    Line bpbc = (Line.CreateBound(bpb, c));
                    Line ddpa = (Line.CreateBound(d, dpa));
                    Line dpba = (Line.CreateBound(dpb, a));
                    Line bpabpb = (Line.CreateBound(bpa, bpb));
                    Line dpadpb = (Line.CreateBound(dpa, dpb));


                    // Create External Masonry Walls //

                    FilteredElementCollector wallcollector = new FilteredElementCollector(doc);
                    wallcollector = new FilteredElementCollector(doc);
                    wallcollector.OfClass(typeof(WallType));
                    var walltypeelements = from element in wallcollector where element.Name == "Wall-Ext_1ft_Cavity" select element;
                    WallType standardwall = walltypeelements.Cast<WallType>().ElementAt<WallType>(0);

                    using (Transaction tWall = new Transaction(doc, "Create Wall"))
                    {
                        tWall.Start();
                        // Wall A Gable
                        XYZ wanormal = -XYZ.BasisX;
                        XYZ[] apts = new XYZ[] {
                            new XYZ(a.X, 0, 0),
                            new XYZ(a.X, 0, 20.376),
                            new XYZ(a.X, 13.626, 29.726),
                            new XYZ(a.X, 26.876, 20.376),
                            new XYZ(a.X, 26.876, 0),
                        };

                        IList<Curve> aprofile = new List<Curve>();

                        XYZ aq = apts[apts.Length - 1];

                        foreach (XYZ ap in apts)
                        {
                            aprofile.Add(Line.CreateBound(aq, ap));
                            aq = ap;
                        }

                        Wall wa = Wall.Create(doc, aprofile, standardwall.Id, levelGround.Id, true, wanormal);
                        Parameter waoffset = wa.get_Parameter(BuiltInParameter.WALL_BASE_OFFSET);
                        waoffset.Set(0);

                        Wall wbprta = Wall.Create(doc, bbpa, standardwall.Id, levelGround.Id, 20, 0, false, true);
                        Wall curtwNjoist = Wall.Create(doc, bpabpb, standardwall.Id, levelFirst.Id, 1, 9, false, false);
                        Wall wbprtb = Wall.Create(doc, bpbc, standardwall.Id, levelGround.Id, 20, 0, false, true);

                        // Wall C Gable
                        XYZ wcnormal = XYZ.BasisX;
                        XYZ[] pts = new XYZ[] {
                            new XYZ(c.X, 0, 0),
                            new XYZ(c.X, 0, 20.376),
                            new XYZ(c.X, 13.626, 29.726),
                            new XYZ(c.X, 26.876, 20.376),
                            new XYZ(c.X, 26.876, 0),
                        };

                        IList<Curve> profile = new List<Curve>();

                        XYZ qq = pts[pts.Length - 1];

                        foreach (XYZ pp in pts)
                        {
                            profile.Add(Line.CreateBound(qq, pp));
                            qq = pp;
                        }

                        Wall wc = Wall.Create(doc, profile, standardwall.Id, levelGround.Id, true, wcnormal);
                        Parameter wcoffset = wc.get_Parameter(BuiltInParameter.WALL_BASE_OFFSET);
                        wcoffset.Set(0);

                        Wall wd = Wall.Create(doc, da, standardwall.Id, levelGround.Id, 10, 0, false, false);
                        Wall wdprta = Wall.Create(doc, ddpa, standardwall.Id, levelFirst.Id, 10, 0, false, false);
                        Wall curtwSjoist = Wall.Create(doc, dpadpb, standardwall.Id, levelFirst.Id, 1, 9, false, false);
                        Wall wdprtb = Wall.Create(doc, dpba, standardwall.Id, levelFirst.Id, 10, 0, false, false);

                        tWall.Commit();
                        TaskDialog.Show("stage", "Create External Masonry Walls" + Environment.NewLine + "type : " + standardwall.Name);


                        // Create Internal Walls at Ground Floor Level //

                        //internal wall ground floor datum set
                        XYZ intGa = new XYZ(id.X, ia.Y + 5.75, 0);
                        XYZ intGaoff = new XYZ(id.X, ia.Y + 3.4125, 0);
                        XYZ intGb = new XYZ(ia.X + 6.25, intGa.Y, 0);
                        XYZ intGc = new XYZ(intGb.X, intGb.Y + 13, 0);
                        XYZ intGcoff = new XYZ(intGb.X, intGb.Y + 10.8348, 0);
                        XYZ intGd = new XYZ(intGb.X + 2, intGc.Y, 0);
                        XYZ intGe = new XYZ(intGd.X, intGd.Y - 2, 0);
                        XYZ intGf = new XYZ(intGb.X, intGe.Y, 0);
                        XYZ intGbh = new XYZ(intGb.X + 2, intGb.Y, 0);
                        XYZ intGbi = new XYZ(intGbh.X, id.Y, 0);

                        Line intGab = (Line.CreateBound(intGa, intGb));
                        Line intGbc = (Line.CreateBound(intGb, intGc));
                        Line intGbcoff = (Line.CreateBound(intGb, intGcoff));
                        Line intGcd = (Line.CreateBound(intGc, intGd));
                        Line intGde = (Line.CreateBound(intGd, intGe));
                        Line intGef = (Line.CreateBound(intGe, intGf));
                        Line intGhi = (Line.CreateBound(intGbh, intGbi));
                        Line intGcf = (Line.CreateBound(intGc, intGf));

                        FilteredElementCollector intwall05collector = new FilteredElementCollector(doc);
                        intwall05collector = new FilteredElementCollector(doc);
                        intwall05collector.OfClass(typeof(WallType));
                        var intwall05typeelements = from element in intwall05collector where element.Name == "Wall-Int_0.5ft_Partition" select element;
                        WallType int05wall = intwall05typeelements.Cast<WallType>().ElementAt<WallType>(0);

                        FilteredElementCollector intwall033collector = new FilteredElementCollector(doc);
                        intwall033collector = new FilteredElementCollector(doc);
                        intwall033collector.OfClass(typeof(WallType));
                        var intwall033typeelements = from element in intwall033collector where element.Name == "Wall-Int_0.33ft_Partition" select element;
                        WallType int033wall = intwall033typeelements.Cast<WallType>().ElementAt<WallType>(0);

                        FilteredElementCollector brickwall101collector = new FilteredElementCollector(doc);
                        brickwall101collector = new FilteredElementCollector(doc);
                        brickwall101collector.OfClass(typeof(WallType));
                        var brickwall101typeelements = from element in brickwall101collector where element.Name == "Wall-Int_0.33ft_Brick" select element;
                        WallType brickwall101 = brickwall101typeelements.Cast<WallType>().ElementAt<WallType>(0);
```

E:\PhD\1\Coding\Revit\Live coding\Hardcode_dwelling_build_1\Hardcode_dwelling_build_1\Class1.cs

```
                                        trussCrvs.Append(ibibtruss);
                                        trussCrvs.Append(iatrussibtruss);
                                        trussCrvs.Append(iaiatruss);

                                        ModelCurveArray footPrintToModelCurveMapping = new ModelCurveArray();
                                        FootPrintRoof footprintTruss = doc.Create.NewFootPrintRoof(trussCrvs, levelRoof, trussType, out footPrintToModelCurveMapping);

                                        var iterator = footPrintToModelCurveMapping.ForwardIterator();
                                        iterator.Reset();

                                        var i = 1;
                                        while (iterator.MoveNext())
                                        {
                                            i++;
                                            var modelCurve = iterator.Current as ModelCurve;
                                            if (i % 2 != 0)
                                            {
                                                footprintTruss.set_DefinesSlope(modelCurve, true);
                                                footprintTruss.set_SlopeAngle(modelCurve, 0.7);
                                            }
                                            else
                                            {
                                                footprintTruss.set_DefinesSlope(modelCurve, false);
                                            }
                                        }

                                        XYZ trussb = new XYZ(ia.X + 1.9154605263158, ia.Y - 0.507565789473684, a.Z);
                                        XYZ trussc = new XYZ(trussb.X + 2.423026315789474, trussb.Y, a.Z);
                                        XYZ trussd = new XYZ(trussc.X + 2.423026315789474, trussb.Y, a.Z);
                                        XYZ trusse = new XYZ(trussd.X + 2.423026315789474, trussb.Y, a.Z);
                                        XYZ trussf = new XYZ(trusse.X + 2.423026315789474, trussb.Y, a.Z);
                                        XYZ trussg = new XYZ(trussf.X + 2.423026315789474, trussb.Y, a.Z);

                                        ElementTransformUtils.CopyElement(doc, footprintTruss.Id, trussb);
                                        ElementTransformUtils.CopyElement(doc, footprintTruss.Id, trussc);
                                        ElementTransformUtils.CopyElement(doc, footprintTruss.Id, trussd);
                                        ElementTransformUtils.CopyElement(doc, footprintTruss.Id, trusse);
                                        ElementTransformUtils.CopyElement(doc, footprintTruss.Id, trussf);
                                        ElementTransformUtils.CopyElement(doc, footprintTruss.Id, trussg);

                                        tTrusses.Commit();
                                    }
                                    TaskDialog.Show("stage", "Create Roof Trusses" + Environment.NewLine + "type : " + trussType.Name);


                                    // Create Roof //

                                    FilteredElementCollector rfcollector = new FilteredElementCollector(doc);
                                    rfcollector.OfClass(typeof(RoofType));
                                    var rooftypeelements = from element in rfcollector where element.Name == "Roof_Generic-125mm" select element;
                                    RoofType roofType = rooftypeelements.Cast<RoofType>().ElementAt<RoofType>(0);

                                    XYZ rfea = new XYZ(ea.X, ea.Y - 0.5, a.Z);
                                    XYZ rfeb = new XYZ(eb.X, eb.Y + 0.5, a.Z);
                                    XYZ rfec = new XYZ(ec.X, ec.Y + 0.5, a.Z);
                                    XYZ rfed = new XYZ(ed.X, ed.Y - 0.5, a.Z);
                                    Line rfeab = (Line.CreateBound(rfea, rfeb));
                                    Line rfebc = (Line.CreateBound(rfeb, rfec));
                                    Line rfecd = (Line.CreateBound(rfec, rfed));
                                    Line rfeda = (Line.CreateBound(rfed, rfea));

                                    using (Transaction tRoof = new Transaction(doc, "Create Roof"))
                                    {
                                        tRoof.Start();

                                        CurveArray footprint = application.Create.NewCurveArray();
                                        footprint.Append(rfeab);
                                        footprint.Append(rfebc);
                                        footprint.Append(rfecd);
                                        footprint.Append(rfeda);

                                        ModelCurveArray footPrintToModelCurveMapping = new ModelCurveArray();
                                        FootPrintRoof footprintRoof = doc.Create.NewFootPrintRoof(footprint, levelRoof, roofType, out footPrintToModelCurveMapping);

                                        var iterator = footPrintToModelCurveMapping.ForwardIterator();
                                        iterator.Reset();

                                        var i = 1;
                                        while (iterator.MoveNext())
                                        {
                                            i++;
                                            var modelCurve = iterator.Current as ModelCurve;
                                            if (i % 2 != 0)
                                            {
                                                footprintRoof.set_DefinesSlope(modelCurve, true);
                                                footprintRoof.set_SlopeAngle(modelCurve, 0.7);
                                            }
                                            else
                                            {
                                                footprintRoof.set_DefinesSlope(modelCurve, false);
                                            }
                                        }

                                        tRoof.Commit();
                                    } /*transaction troof end*/
                                    TaskDialog.Show("stage", "Create Roof" + Environment.NewLine + "type : " + roofType.Name);


                                    // Create Exterior Curtain Walls //

                                    WallType curtainwall = new FilteredElementCollector(doc).OfClass(typeof(WallType))
                                        .First<Element>(e => e.Name.Equals("Curtain_Wall-Exterior_Glazing")) as WallType;

                                    using (Transaction tCurtainWall = new Transaction(doc, "Create Exterior Curtain Walls"))
                                    {
                                        tCurtainWall.Start();
                                        Wall curtwbgrnd = Wall.Create(doc, bpabpb, curtainwall.Id, levelGround.Id, 9, 0, false, false);
                                        Wall curtwbslab = Wall.Create(doc, bpabpb, curtainwall.Id, levelGround.Id, 1, 9, false, false);
                                        Wall curtwbfrst = Wall.Create(doc, bpabpb, curtainwall.Id, levelFirst.Id, 9, 0, false, false);
                                        Wall curtwd = Wall.Create(doc, dpadpb, curtainwall.Id, levelFirst.Id, 9, 0, false, false);
                                        tCurtainWall.Commit();
                                    } /*transaction tCurtainWall end*/
                                    TaskDialog.Show("stage", "Create Exterior Curtain Walls" + Environment.NewLine + "type : " + curtainwall.Name);

                                    // Create Stair //
                                    ElementId newStairsId = null;

                                    using (StairsEditScope newStairsScope = new StairsEditScope(doc, "Create Stair"))
                                    {
                                        newStairsId = newStairsScope.Start(levelGround.Id, levelFirst.Id);

                                        using (Transaction stairsTrans = new Transaction(doc, "Add Runs and Landings to Stairs"))
                                        {
                                            stairsTrans.Start();
                                            // Create a sketched run for the stairs
                                            IList<Curve> bdryCurves = new List<Curve>();
                                            IList<Curve> riserCurves = new List<Curve>();
                                            IList<Curve> pathCurves = new List<Curve>();
                                            XYZ pnt1 = new XYZ(0.5, 21, 0);
                                            XYZ pnt2 = new XYZ(0.5, 8, 0);
                                            XYZ pnt3 = new XYZ(3.5, 21, 0);
                                            XYZ pnt4 = new XYZ(3.5, 8, 0);

                                            // boundaries
                                            bdryCurves.Add(Line.CreateBound(pnt1, pnt2));
                                            bdryCurves.Add(Line.CreateBound(pnt3, pnt4));

                                            // risers hrdcd
                                            XYZ strRi1a = new XYZ(0.5, 21, 0);
                                            XYZ strRi1aa = new XYZ(3.5, 21, 0);
                                            XYZ strRi1b = new XYZ(0.5, 20, 0);
                                            XYZ strRi1bb = new XYZ(3.5, 20, 0);
                                            XYZ strRi1c = new XYZ(0.5, 19, 0);
                                            XYZ strRi1cc = new XYZ(3.5, 19, 0);
                                            XYZ strRi1d = new XYZ(0.5, 18, 0);
                                            XYZ strRi1dd = new XYZ(3.5, 18, 0);
                                            XYZ strRi1e = new XYZ(0.5, 17, 0);
                                            XYZ strRi1ee = new XYZ(3.5, 17, 0);
                                            XYZ strRi1f = new XYZ(0.5, 16, 0);
                                            XYZ strRi1ff = new XYZ(3.5, 16, 0);
                                            XYZ strRi1g = new XYZ(0.5, 15, 0);
                                            XYZ strRi1gg = new XYZ(3.5, 15, 0);
                                            XYZ strRi1h = new XYZ(0.5, 14, 0);
                                            XYZ strRi1hh = new XYZ(3.5, 14, 0);
                                            XYZ strRi1i = new XYZ(0.5, 13, 0);
                                            XYZ strRi1ii = new XYZ(3.5, 13, 0);
                                            XYZ strRi1j = new XYZ(0.5, 12, 0);
                                            XYZ strRi1jj = new XYZ(3.5, 12, 0);
                                            XYZ strRi1k = new XYZ(0.5, 11, 0);
                                            XYZ strRi1kk = new XYZ(3.5, 11, 0);
                                            XYZ strRi1l = new XYZ(0.5, 10, 0);
                                            XYZ strRi1ll = new XYZ(3.5, 10, 0);
                                            XYZ strRi1m = new XYZ(0.5, 9, 0);
                                            XYZ strRi1mm = new XYZ(3.5, 9, 0);
                                            XYZ strRi1n = new XYZ(0.5, 8, 0);
                                            XYZ strRi1nn = new XYZ(3.5, 8, 0);

                                            riserCurves.Add(Line.CreateBound(strRi1a, strRi1aa));
                                            riserCurves.Add(Line.CreateBound(strRi1b, strRi1bb));
                                            riserCurves.Add(Line.CreateBound(strRi1c, strRi1cc));
                                            riserCurves.Add(Line.CreateBound(strRi1d, strRi1dd));
                                            riserCurves.Add(Line.CreateBound(strRi1e, strRi1ee));
                                            riserCurves.Add(Line.CreateBound(strRi1f, strRi1ff));
                                            riserCurves.Add(Line.CreateBound(strRi1g, strRi1gg));
                                            riserCurves.Add(Line.CreateBound(strRi1h, strRi1hh));
                                            riserCurves.Add(Line.CreateBound(strRi1i, strRi1ii));
                                            riserCurves.Add(Line.CreateBound(strRi1j, strRi1jj));
                                            riserCurves.Add(Line.CreateBound(strRi1k, strRi1kk));
                                            riserCurves.Add(Line.CreateBound(strRi1l, strRi1ll));
                                            riserCurves.Add(Line.CreateBound(strRi1m, strRi1mm));
                                            riserCurves.Add(Line.CreateBound(strRi1n, strRi1nn));

                                            //stairs path curves
                                            XYZ pathEnd0 = (pnt1 + pnt3) / 2.0;
                                            XYZ pathEnd1 = (pnt2 + pnt4) / 2.0;
                                            pathCurves.Add(Line.CreateBound(pathEnd0, pathEnd1));

                                            StairsRun newRun1 = StairsRun.CreateSketchedRun(doc, newStairsId, levelGround.Elevation, bdryCurves, riserCurves, pathCurves);

                                            IList<Curve> r2bdryCurves = new List<Curve>();
                                            IList<Curve> r2riserCurves = new List<Curve>();
                                            IList<Curve> r2pathCurves = new List<Curve>();
                                            XYZ r2pnt1 = new XYZ(3.55, 8, 0);
                                            XYZ r2pnt2 = new XYZ(3.55, 10, 0);
                                            XYZ r2pnt3 = new XYZ(6.55, 8, 0);
                                            XYZ r2pnt4 = new XYZ(6.55, 10, 0);

                                            // boundaries
                                            r2bdryCurves.Add(Line.CreateBound(r2pnt1, r2pnt2));
                                            r2bdryCurves.Add(Line.CreateBound(r2pnt3, r2pnt4));

                                            // risers hrdcd
                                            XYZ strRi2a = new XYZ(3.55, 8, 0);
                                            XYZ strRi2aa = new XYZ(6.55, 8, 0);
                                            XYZ strRi2b = new XYZ(3.55, 9, 0);
                                            XYZ strRi2bb = new XYZ(6.55, 9, 0);
                                            XYZ strRi2c = new XYZ(3.55, 10, 0);
                                            XYZ strRi2cc = new XYZ(6.55, 10, 0);

                                            r2riserCurves.Add(Line.CreateBound(strRi2a, strRi2aa));
                                            r2riserCurves.Add(Line.CreateBound(strRi2b, strRi2bb));
                                            r2riserCurves.Add(Line.CreateBound(strRi2c, strRi2cc));

                                            //stairs path curves
                                            XYZ r2pathEnd0 = (r2pnt1 + r2pnt3) / 2.0;
                                            XYZ r2pathEnd1 = (r2pnt2 + r2pnt4) / 2.0;
                                            r2pathCurves.Add(Line.CreateBound(r2pathEnd0, r2pathEnd1));

                                            StairsRun newRun2 = StairsRun.CreateSketchedRun(doc, newStairsId, newRun1.TopElevation, r2bdryCurves, r2riserCurves, r2pathCurves);

                                            // Add a landing between the runs
                                            CurveLoop landingLoop = new CurveLoop();
                                            XYZ p1 = new XYZ(6.55, 8, 0);
                                            XYZ p2 = new XYZ(6.55, 5, 0);
                                            XYZ p3 = new XYZ(0.5, 5, 0);
                                            XYZ p4 = new XYZ(0.5, 8, 0);
                                            Line curve_1 = Line.CreateBound(p1, p2);
                                            Line curve_2 = Line.CreateBound(p2, p3);
                                            Line curve_3 = Line.CreateBound(p3, p4);
                                            Line curve_4 = Line.CreateBound(p4, p1);
                                            landingLoop.Append(curve_1);
                                            landingLoop.Append(curve_2);
                                            landingLoop.Append(curve_3);
                                            landingLoop.Append(curve_4);
                                            StairsLanding newLanding = StairsLanding.CreateSketchedLanding(doc, newStairsId, landingLoop, newRun1.TopElevation);

                                            stairsTrans.Commit();
                                        }
                                        // A failure preprocessor is to handle possible failures during the edit mode commitment process.
                                        newStairsScope.Commit(new StairsFailurePreprocessor());
                                    } /*transaction stairsTrans end*/
                                    TaskDialog.Show("stage", "Create Stair");


                                    // Create Railings //

                                    XYZ ffiaarail = new XYZ(ffiaa.X, ffiaa.Y - 0.133, a.Z);
                                    XYZ ffibbrail = new XYZ(ffibb.X + 0.133, ffibb.Y - 0.133, a.Z);
                                    XYZ fficcrail = new XYZ(fficc.X + 0.133, fficc.Y + 0.16677, a.Z);
                                    XYZ fficcrailoffa = new XYZ(fficcrail.X - 0.16677, fficcrail.Y, a.Z);
                                    XYZ fficcrailoffb = new XYZ(fficcrailoffa.X, fficcrailoffa.Y - 0.16677, a.Z);

                                    Line ffiaaffibbrail = (Line.CreateBound(ffiaarail, ffibbrail));
                                    Line ffibbfficcrail = (Line.CreateBound(ffibbrail, fficcrail));
                                    Line fficcrailfficcrailoffa = (Line.CreateBound(fficcrail, fficcrailoffa));
                                    Line fficcrailoffafficcrailoffb = (Line.CreateBound(fficcrailoffa, fficcrailoffb));

                                    XYZ ffiddrail = new XYZ(ffidd.X + 0.133, ffidd.Y, ffidd.Z);
                                    XYZ ffieerail = new XYZ(ffiee.X + 0.133, ffiee.Y + 0.133, ffiee.Z);
                                    XYZ ffiffrail = new XYZ(ffiff.X, ffiff.Y + 0.133, ffiff.Z);
                                    Line ffiddrailffieerail = (Line.CreateBound(ffiddrail, ffieerail));
                                    Line ffieerailffiffrail = (Line.CreateBound(ffieerail, ffiffrail));

                                    FilteredElementCollector railingcollector = new FilteredElementCollector(doc);
                                    railingcollector = new FilteredElementCollector(doc);
                                    railingcollector.OfClass(typeof(RailingType));
                                    var railingtypeelements = from element in railingcollector where element.Name == "931mm" select element;
```

E:\PhD\1\Coding\Revit\Live coding\Hardcode_dwelling_build_1\Hardcode_dwelling_build_1\Class1.cs

```
                                    RailingType railing931 = railingtypeelements.Cast<RailingType>().ElementAt<RailingType>(0);

                                    using (Transaction tRailing = new Transaction(doc, "Create Railings"))
                                    {
                                        tRailing.Start();

                                        CurveLoop railingLoopac = new CurveLoop();
                                        railingLoopac.Append(ffiaaffibbrail);
                                        railingLoopac.Append(ffibbfficcrail);
                                        railingLoopac.Append(fficcrailfficcrailoffa);
                                        railingLoopac.Append(fficcrailoffafficcrailoffb);
                                        Railing.Create(doc, railingLoopac, railing931.Id, levelFirst.Id);

                                        CurveLoop railingLooppdf = new CurveLoop();
                                        railingLooppdf.Append(ffiddrailffieerail);
                                        railingLooppdf.Append(ffieerailffiffrail);
                                        Railing.Create(doc, railingLooppdf, railing931.Id, levelFirst.Id);

                                        tRailing.Commit();
                                    }
                                    TaskDialog.Show("stage", "Create Railings");


                                    // Place Front Door //

                                    Family drfm = new FilteredElementCollector(doc).OfClass(typeof(Family)).Cast<Family>()
                                        .FirstOrDefault(q => q.FamilyCategoryId.IntegerValue == (int)BuiltInCategory.OST_Doors
                                                        && q.Name == "Doors_ExtSgl_w-Glazing_Bars_3") as Family;

                                    FamilySymbol drfs = drfm.GetFamilySymbolIds().Select(q => doc.GetElement(q)).Cast<FamilySymbol>().First(q => q.Name == "910x2110mm");

                                    using (Transaction tDoor = new Transaction(doc, "Place Front Door"))
                                    {
                                        tDoor.Start();

                                        string drxco = "3";
                                        string dryco = "0.5";

                                        // Convert coordinates to double and create XYZ point.
                                        double drx = double.Parse(drxco);
                                        double dry = double.Parse(dryco);
                                        XYZ xyz = new XYZ(drx, dry, levelGround.Elevation);

                                        FamilyInstance drfi = doc.Create.NewFamilyInstance(xyz, drfs, wd, levelGround, Autodesk.Revit.DB.Structure.StructuralType.NonStructural);

                                        tDoor.Commit();
                                    }

                                    using (Transaction tDoorbuga = new Transaction(doc, "Create Door Instance"))
                                    {
                                        tDoorbuga.Start();

                                        Parameter WALL_BASE_OFFSET = wd.get_Parameter(BuiltInParameter.WALL_BASE_OFFSET);
                                        if (WALL_BASE_OFFSET == null)
                                        {
                                            WALL_BASE_OFFSET.Set(-0.1);
                                        }
                                        tDoorbuga.Commit();
                                    }

                                    using (Transaction tDoorbugb = new Transaction(doc, "Create Door Instance"))
                                    {
                                        tDoorbugb.Start();

                                        Parameter WALL_BASE_OFFSET = wd.get_Parameter(BuiltInParameter.WALL_BASE_OFFSET);
                                        if (WALL_BASE_OFFSET != null)
                                        {
                                            WALL_BASE_OFFSET.Set(0);
                                        }
                                        tDoorbugb.Commit();
                                    }
                                    //                           TaskDialog.Show("stage", "Place Front Door");


                                    // Place Bath Door //

                                    Family drintfm = new FilteredElementCollector(doc).OfClass(typeof(Family)).Cast<Family>()
                                        .FirstOrDefault(q => q.FamilyCategoryId.IntegerValue == (int)BuiltInCategory.OST_Doors
                                                        && q.Name == "Doors_IntSgl") as Family;

                                    FamilySymbol drintfs = drintfm.GetFamilySymbolIds().Select(q => doc.GetElement(q)).Cast<FamilySymbol>().First(q => q.Name == "810x2110mm");

                                    using (Transaction tDoorint = new Transaction(doc, "Place Door Instance"))
                                    {
                                        tDoorint.Start();

                                        string drxco = "8.5";
                                        string dryco = "4";

                                        double drx = double.Parse(drxco);
                                        double dry = double.Parse(dryco);
                                        XYZ xyz = new XYZ(drx, dry, levelGround.Elevation);

                                        // Find the nearest hosting wall to the insertion point
                                        FilteredElementCollector collector = new FilteredElementCollector(doc);
                                        collector.OfClass(typeof(Wall));

                                        List<Wall> walls = collector.Cast<Wall>().Where(wl => wl.LevelId == levelGround.Id).ToList();
                                        Wall wall = null;
                                        double distance = double.MaxValue;

                                        foreach (Wall w in walls)
                                        {
                                            double proximity = (w.Location as LocationCurve).Curve.Distance(xyz);

                                            if (proximity < distance)
                                            {
                                                distance = proximity;
                                                wall = w;
                                            }
                                        }

                                        FamilyInstance drintfi = doc.Create.NewFamilyInstance(xyz, drintfs, wall, levelGround, Autodesk.Revit.DB.Structure.StructuralType.NonStructural);

                                        tDoorint.Commit();
                                    }
                                    TaskDialog.Show("stage", "Place Bath Door");


                                    // Place Window //

                                    using (Transaction tWindow = new Transaction(doc, "Create Window"))
                                    {
                                        tWindow.Start();

                                        Family wndfm = new FilteredElementCollector(doc).OfClass(typeof(Family)).Cast<Family>()
                                            .FirstOrDefault(q => q.FamilyCategoryId.IntegerValue == (int)BuiltInCategory.OST_Windows
                                                            && q.Name == "Windows_Sgl_Plain") as Family;
                                        if (wndfm == null)
                                        {
                                            TaskDialog.Show("Error", "no family");
                                            return;
                                        }

                                        FamilySymbol wnfs = wndfm.GetFamilySymbolIds().Select(q => doc.GetElement(q)).Cast<FamilySymbol>().First(q => q.Name == "910x525mm");

                                        string xCoord = "11.5";
                                        string yCoord = "0";

                                        double x = double.Parse(xCoord);
                                        double y = double.Parse(yCoord);
                                        XYZ xyz = new XYZ(x, y, levelGround.Elevation + 5.2);

                                        FamilyInstance window = doc.Create.NewFamilyInstance(xyz, wnfs, wd, Autodesk.Revit.DB.Structure.StructuralType.NonStructural);

                                        tWindow.Commit();
                                    }
                                    TaskDialog.Show("stage", "Place Window");


                                    // Cast Kitchen Counters //

                                    XYZ wrkta = new XYZ(ib.X, ib.Y, 0);
                                    XYZ wrktb = new XYZ(wrkta.X + 2.1, wrkta.Y, 0);
                                    XYZ wrktc = new XYZ(wrktb.X, wrktb.Y - 6.77, 0);
                                    XYZ wrktd = new XYZ(wrkta.X, wrktc.Y, 0);
                                    Line wrktab = (Line.CreateBound(wrkta, wrktb));
                                    Line wrktbc = (Line.CreateBound(wrktb, wrktc));
                                    Line wrktcd = (Line.CreateBound(wrktc, wrktd));
                                    Line wrktda = (Line.CreateBound(wrktd, wrkta));

                                    using (Transaction twrktCast = new Transaction(doc, "Cast Kitchen Counters"))
                                    {
                                        XYZ wrktpnormal = XYZ.BasisZ;

                                        CurveArray wrktpCurves = application.Create.NewCurveArray();
                                        wrktpCurves.Append(wrktab);
                                        wrktpCurves.Append(wrktbc);
                                        wrktpCurves.Append(wrktcd);
                                        wrktpCurves.Append(wrktda);

                                        twrktCast.Start();
                                        Floor wrktCast = doc.Create.NewFloor(wrktpCurves, hearthCasttype, levelFirst, false, wrktpnormal);
                                        wrktCast.get_Parameter(BuiltInParameter.FLOOR_HEIGHTABOVELEVEL_PARAM).Set(3.2);

                                        XYZ wrktaoff = new XYZ(wrkta.X, wrkta.Y - 0.33, 0);
                                        XYZ wrktboff = new XYZ(wrkta.X + 1.8, wrktb.Y - 0.33, 0);
                                        XYZ wrktcoff = new XYZ(wrktd.X, wrktd.Y + 0.33, 0);
                                        XYZ wrktdoff = new XYZ(wrktd.X + 1.8, wrktd.Y + 0.33, 0);
                                        Line wrktaboff = (Line.CreateBound(wrktaoff, wrktboff));
                                        Line wrktcdoff = (Line.CreateBound(wrktcoff, wrktdoff));

                                        Wall wrktpab = Wall.Create(doc, wrktaboff, cncwall.Id, levelFirst.Id, 3, 0, false, true);
                                        Wall wrktpcd = Wall.Create(doc, wrktcdoff, cncwall.Id, levelFirst.Id, 3, 0, false, true);

                                        twrktCast.Commit();
                                    }
                                    TaskDialog.Show("stage", "Cast Kitchen Counters");


                                    // Place Furniture //

                                    using (Transaction tFurniture = new Transaction(doc, "Place Furniture"))
                                    {
                                        tFurniture.Start();

                                        // Ground Floor

                                        // FBed
                                        Family FBedFA = new FilteredElementCollector(doc).OfClass(typeof(Family)).Cast<Family>()
                                            .FirstOrDefault(q => q.FamilyCategoryId.IntegerValue == (int)BuiltInCategory.OST_Furniture
                                                            && q.Name == "Furniture_Bed_1") as Family;

                                        FamilySymbol FBedFS = FBedFA.GetFamilySymbolIds().Select(q => doc.GetElement(q)).Cast<FamilySymbol>().First(q => q.Name == "1346x1880x355mm-Double");
                                        XYZ FBedCO = new XYZ(-13.77, 13.22, 0);
                                        FamilyInstance FBed = doc.Create.NewFamilyInstance(FBedCO, FBedFS, grndslab, 0);

                                        LocationPoint FBedLP = FBed.Location as LocationPoint;
                                        XYZ FBedppt = new XYZ(FBedLP.Point.X, FBedLP.Point.Y, 0);
                                        Line FBedaxis = Line.CreateBound(FBedppt, new XYZ(FBedppt.X, FBedppt.Y, FBedppt.Z + 10));
                                        ElementTransformUtils.RotateElement(doc, FBed.Id, FBedaxis, (Math.PI / 180) * (270));

                                        // FurnitureStoolRectangular
                                        Family FStoolRectangularFA = new FilteredElementCollector(doc).OfClass(typeof(Family)).Cast<Family>()
                                            .FirstOrDefault(q => q.FamilyCategoryId.IntegerValue == (int)BuiltInCategory.OST_Furniture
                                                            && q.Name == "Furniture_Stool_Rectangular") as Family;

                                        FamilySymbol FStoolRectangularFS = FStoolRectangularFA.GetFamilySymbolIds().Select(q => doc.GetElement(q)).Cast<FamilySymbol>().First(q => q.Name == "650x500x850mm");
                                        XYZ FStoolRectangularCO = new XYZ(-9.3, 15, 0);
                                        FamilyInstance FStoolRectangular = doc.Create.NewFamilyInstance(FStoolRectangularCO, FStoolRectangularFS, grndslab, 0);

                                        LocationPoint FStoolRectangularLP = FStoolRectangular.Location as LocationPoint;
                                        XYZ FStoolRectangularppt = new XYZ(FStoolRectangularLP.Point.X, FStoolRectangularLP.Point.Y, 0);
                                        Line FStoolRectangularaxis = Line.CreateBound(FStoolRectangularppt, new XYZ(FStoolRectangularppt.X, FStoolRectangularppt.Y, FStoolRectangularppt.Z + 10));
                                        ElementTransformUtils.RotateElement(doc, FStoolRectangular.Id, FStoolRectangularaxis, (Math.PI / 180) * (270));

                                        // FurnitureDresser
                                        Family FDresserFA = new FilteredElementCollector(doc).OfClass(typeof(Family)).Cast<Family>()
                                            .FirstOrDefault(q => q.FamilyCategoryId.IntegerValue == (int)BuiltInCategory.OST_Furniture
                                                            && q.Name == "Furniture_Dresser") as Family;

                                        FamilySymbol FDresserFS = FDresserFA.GetFamilySymbolIds().Select(q => doc.GetElement(q)).Cast<FamilySymbol>().First(q => q.Name == "1830x457x1525mm");
                                        XYZ FDresserCO = new XYZ(-11.9, -9.775, 0);
                                        FamilyInstance FDresser = doc.Create.NewFamilyInstance(FDresserCO, FDresserFS, grndslab, 0);

                                        LocationPoint FDresserLP = FDresser.Location as LocationPoint;
                                        XYZ FDresserppt = new XYZ(FDresserLP.Point.X, FDresserLP.Point.Y, 0);
                                        Line FDresseraxis = Line.CreateBound(FDresserppt, new XYZ(FDresserppt.X, FDresserppt.Y, FDresserppt.Z + 10));
                                        ElementTransformUtils.RotateElement(doc, FDresser.Id, FDresseraxis, (Math.PI / 180) * (180));

                                        // FurnitureBookshelves
                                        Family FBookshelvesFA = new FilteredElementCollector(doc).OfClass(typeof(Family)).Cast<Family>()
                                            .FirstOrDefault(q => q.FamilyCategoryId.IntegerValue == (int)BuiltInCategory.OST_Furniture
                                                            && q.Name == "Furniture_Bookshelves") as Family;

                                        FamilySymbol FBookshelvesFS = FBookshelvesFA.GetFamilySymbolIds().Select(q => doc.GetElement(q)).Cast<FamilySymbol>().First(q => q.Name == "1850x390x1850mm");
                                        XYZ FBookshelvesCO = new XYZ(-22.2, 13.82, 0);
                                        FamilyInstance FBookshelves = doc.Create.NewFamilyInstance(FBookshelvesCO, FBookshelvesFS, grndslab, 0);

                                        LocationPoint FBookshelvesLP = FBookshelves.Location as LocationPoint;
                                        XYZ FBookshelvesppt = new XYZ(FBookshelvesLP.Point.X, FBookshelvesLP.Point.Y, 0);
                                        Line FBookshelvesaxis = Line.CreateBound(FBookshelvesppt, new XYZ(FBookshelvesppt.X, FBookshelvesppt.Y, FBookshelvesppt.Z + 10));
                                        ElementTransformUtils.RotateElement(doc, FBookshelves.Id, FBookshelvesaxis, (Math.PI / 180) * (270));

                                        // FTableCoffeeGlass
                                        Family FTableCoffeeGlassFA = new FilteredElementCollector(doc).OfClass(typeof(Family)).Cast<Family>()
                                            .FirstOrDefault(q => q.FamilyCategoryId.IntegerValue == (int)BuiltInCategory.OST_Furniture
                                                            && q.Name == "Furniture_Table_Coffee_Glass") as Family;

                                        FamilySymbol FTableCoffeeGlassFS = FTableCoffeeGlassFA.GetFamilySymbolIds().Select(q => doc.GetElement(q)).Cast<FamilySymbol>().First(q => q.Name == "1200x760x330mm");
                                        XYZ FTableCoffeeGlassCO = new XYZ(24.24, -8, 0);
                                        FamilyInstance FTableCoffeeGlass = doc.Create.NewFamilyInstance(FTableCoffeeGlassCO, FTableCoffeeGlassFS, grndslab, 0);

                                        LocationPoint FTableCoffeeGlassLP = FTableCoffeeGlass.Location as LocationPoint;
                                        XYZ FTableCoffeeGlassppt = new XYZ(FTableCoffeeGlassLP.Point.X, FTableCoffeeGlassLP.Point.Y, 0);
                                        Line FTableCoffeeGlassaxis = Line.CreateBound(FTableCoffeeGlassppt, new XYZ(FTableCoffeeGlassppt.X, FTableCoffeeGlassppt.Y, FTableCoffeeGlassppt.Z + 10));
                                        ElementTransformUtils.RotateElement(doc, FTableCoffeeGlass.Id, FTableCoffeeGlassaxis, (Math.PI / 180) * (90));
```