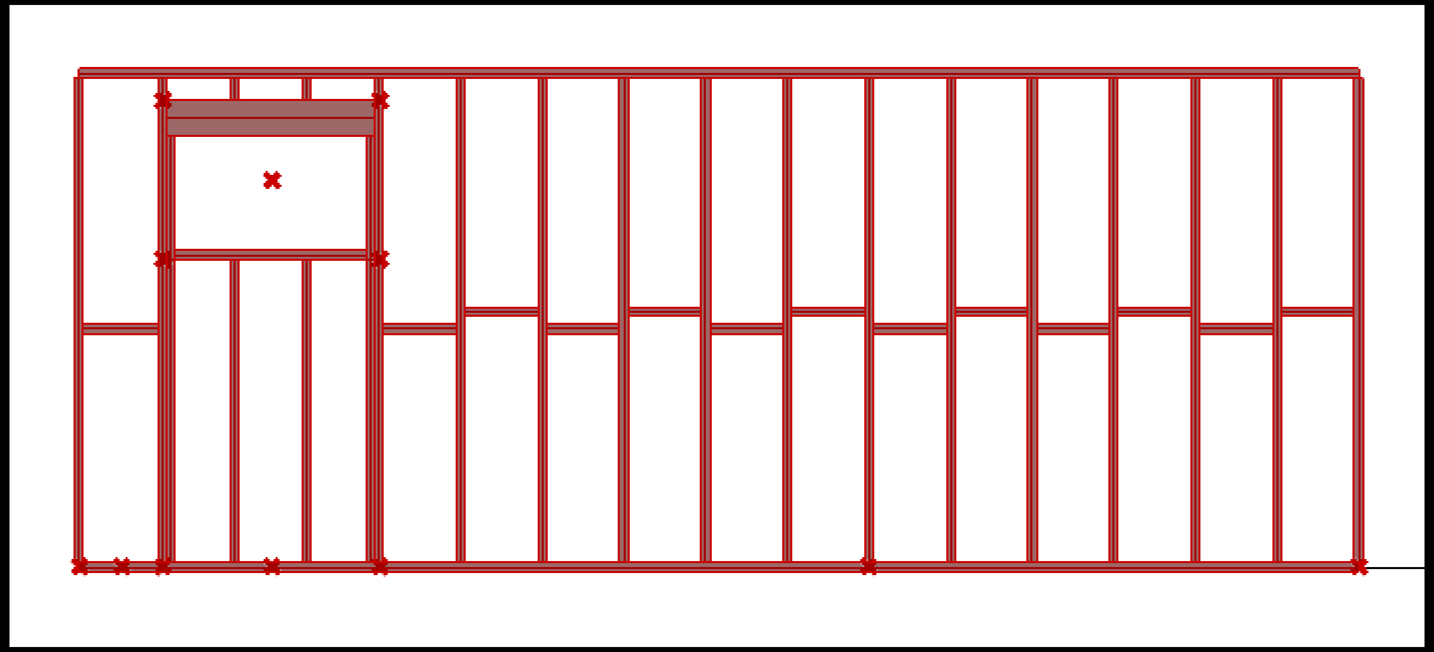
OPTIMIZATION OF MATERIAL & LABOR EFFICIENCY USING WITH SALVAGED TIMBER

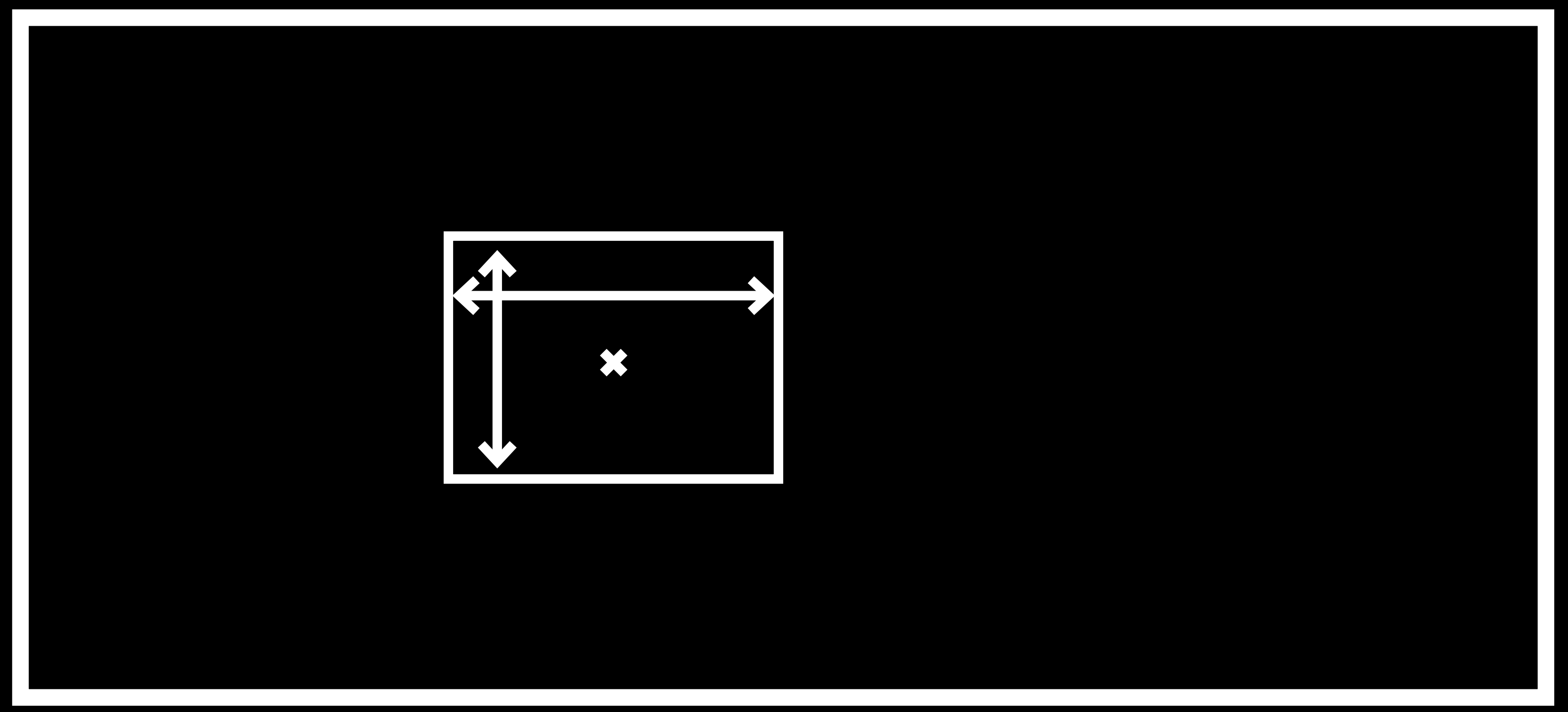
GOAL

The goal of this experiment is to train an ML-Agent to optimize the usage of salvage timber on a timber frame. The experiment will involve defining a random size of timber frame, spawning random window positions on the wall frame, assigning random values for window height and length, and moving the window around in order to find the best position and size to use the salvage timber minimal offcuts. The agent will keep trying until 90% of salvaged timber is used without offcuts. The procedure will be repeated multiple times to train the agent.



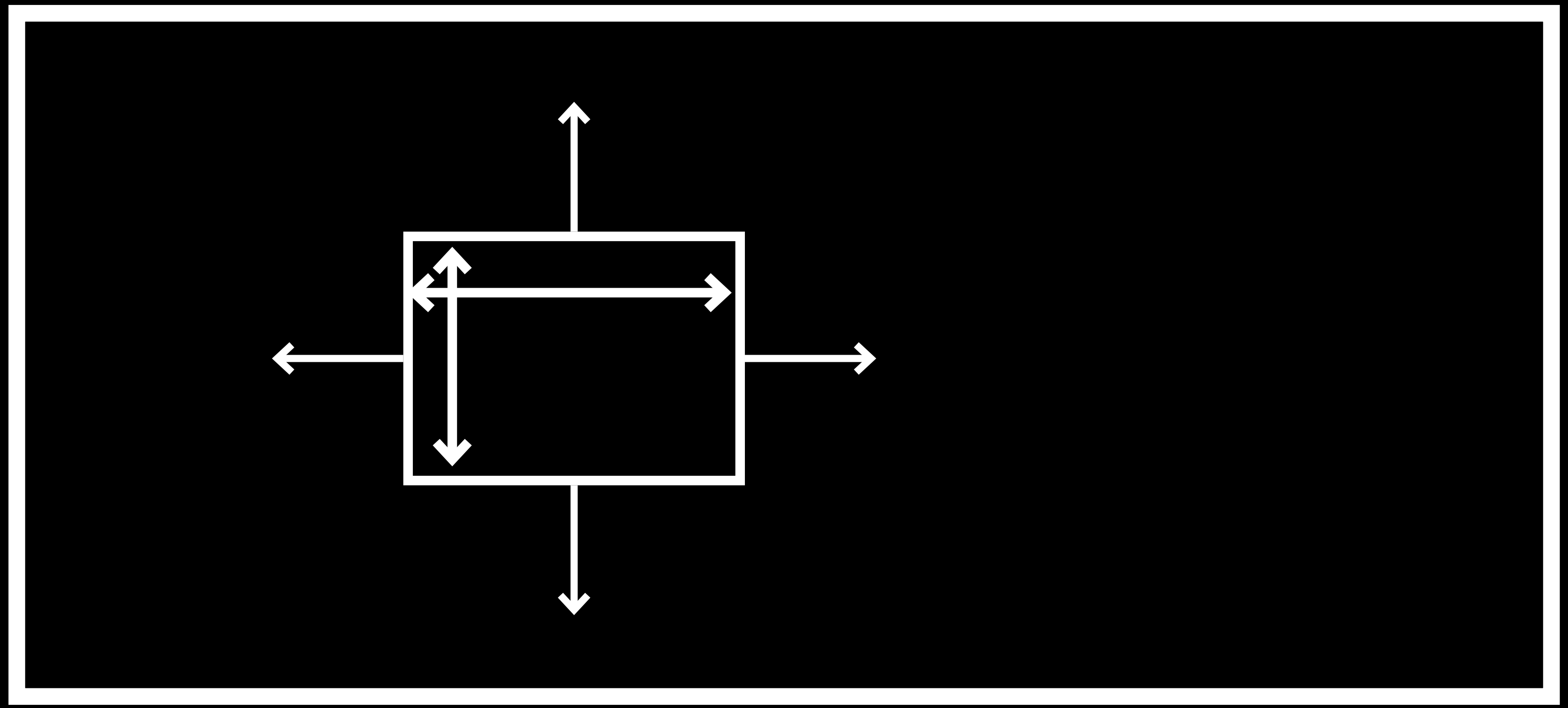
STEP 1: DEFINE A RANDOM SIZE OF TIMBER FRAME

The first step is to define a random size for the timber frame. The size can be defined by specifying the height, width, and length of the frame. This step is important because the size of the timber frame will influence the placement and sizing of the windows, which will ultimately impact the amount of salvage timber that can be used.



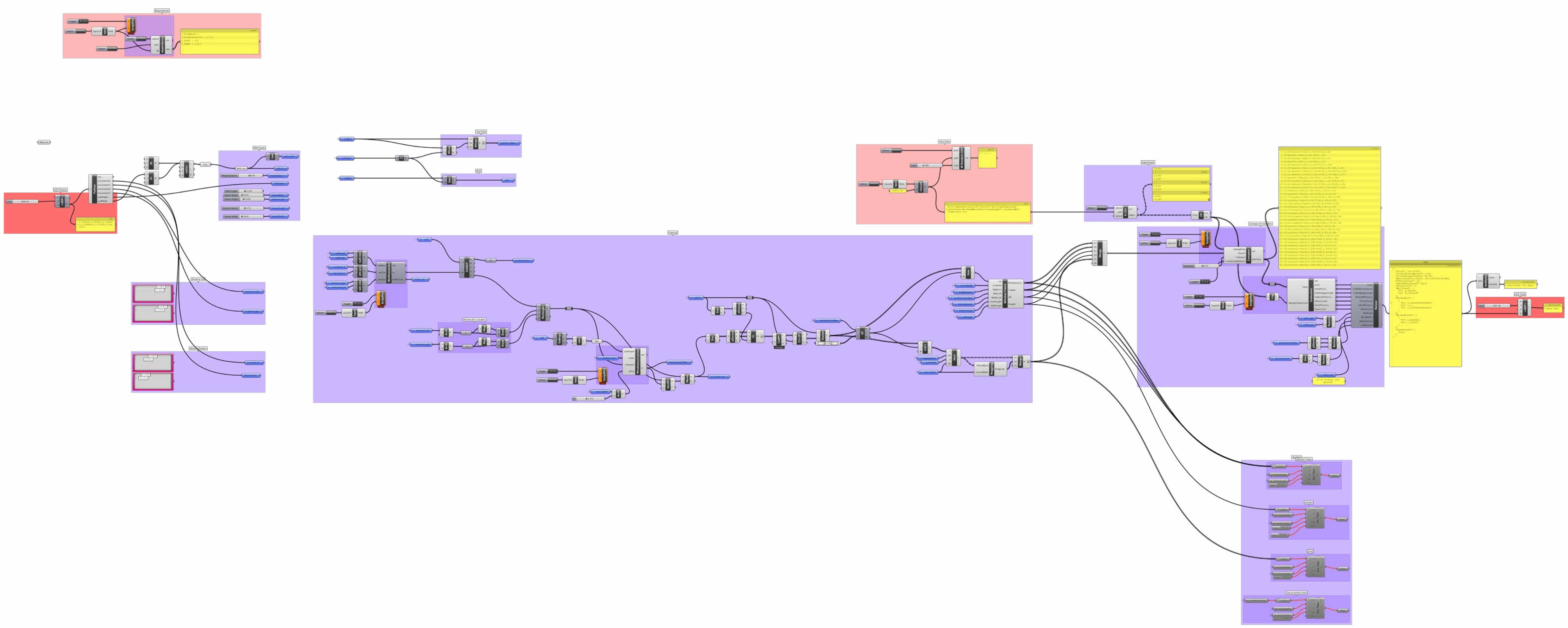
STEP 2: SPAWN RANDOM WINDOW POSITIONS ON THE WALL FRAME.

This can be achieved by selecting random locations on the wall where windows can be placed. The number of windows is expected to be defined randomly, depending on the size of the timber frame. Although this function is currently not implemented.



STEP 3: MOVE & RESIZE WINDOW

Once the window position, height, and length are defined, the ML-Agent will start to move the window around and resize in order to find the best position and size to use the salvage timber minimal offcuts. The agent will use its learning algorithm to determine the best position and size for the window. If the agent can use 90% of the salvaged timber without any offcuts, it will move to the next episode. Otherwise, it will continue trying until it finds the best solution.



```csharp
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using Unity.Mathematics;
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Actuators;
using Unity.MLAgents.Sensors;
using Random = UnityEngine.Random;


public class Train : Agent
{

    [SerializeField][Range(0.01f, 1.0f)] private float _windowMoveSpeed = 0.1f;
    [SerializeField][Range(0.01f, 0.5f)] private float _windowSizeSpeed = 0.1f;

    private float _wallHeight;
    private float _wallWidth;

    private float _windowInitPosX;
    private float _windowInitPosY;
    private float _windowInitSizeX;
    private float _windowInitSizeY;

    private float _windowPosX;
    private float _windowPosY;
    private float _windowSizeX;
    private float _windowSizeY;

    private string _previousDataFromGh = "";
    private string _lastDataFromGh = "";
    private bool _dataFromGhIsChanged = false;


    [System.Serializable]
    public class GhData
    {
        public float Score;
        public float TotalOffcutsAmount;
        public float TotalSalvageLength;
        public float MaterialEfficiency;
        public int OffcutsCount;
        public float LaborEfficiency;
        public int ReuseCount;
        public int MinCutRatio;
        public Vector2 WallScale;
        public Vector3[] WindowPos;
        public Vector2[] WindowScale;
        public bool[] IsAtBounds;
    }


    private GhData _ghData;

    public override void OnEpisodeBegin()
    {
        // randomize wall height & length
        _wallWidth = Random.Range(3f, 10f);
        _wallHeight = Random.Range(2.1f, 3.5f);

        // randomize window initial scale
        _windowInitSizeX = Random.Range(0.5f, 1.5f);
        _windowInitSizeY = Random.Range(0.5f, 1.0f);

        // randomize window initial position
        float tenPercWallWidth = _wallWidth * 0.1f;
        float tenPercWallHeight = _wallHeight * 0.1f;
        float halfWindowSizeX = _windowSizeX * 0.5f;
        float halfWindowSizeY = _windowSizeY * 0.5f;

        _windowInitPosX = Random.Range(halfWindowSizeX + tenPercWallWidth, _wallWidth - halfWindowSizeX - tenPercWallWidth);
        _windowInitPosY = Random.Range(halfWindowSizeY + tenPercWallHeight, _wallHeight - halfWindowSizeY - tenPercWallHeight);

        _windowPosX = _windowInitPosX;
        _windowPosY = _windowInitPosY;
        _windowSizeX = _windowInitSizeX;
```



```csharp
        _windowSizeY = _windowInitSizeY;

        gameObject.GetComponent<Gh_IO>().msgToGh = $"{_windowPosX},{_windowPosY},{_windowSizeX},{_windowSizeY},{_wallHeight},{_wallWidth}";
    }

    public override void CollectObservations(VectorSensor sensor)
    {
        // only observe until package is received
        if (_dataFromGhIsChanged)
        {
            // efficiency = 6
            sensor.AddObservation(_ghData.TotalOffcutsAmount);
            sensor.AddObservation(_ghData.TotalSalvageLength);
            sensor.AddObservation(_ghData.MaterialEfficiency);
            sensor.AddObservation(_ghData.OffcutsCount);
            sensor.AddObservation(_ghData.LaborEfficiency);
            sensor.AddObservation(_ghData.ReuseCount);
            sensor.AddObservation(_ghData.MinCutRatio);

            // position & scale = 7
            sensor.AddObservation(_ghData.WallScale.x);
            sensor.AddObservation(_ghData.WallScale.y);
            foreach (var winPos in _ghData.WindowPos)
            {
                sensor.AddObservation(winPos.x);
                sensor.AddObservation(winPos.y);
                sensor.AddObservation(winPos.z);
            }

            foreach (var winScale in _ghData.WindowScale)
            {
                sensor.AddObservation(winScale.x);
                sensor.AddObservation(winScale.y);
            }
        }
    }

    public override void OnActionReceived(ActionBuffers actions)
    {
        // window param
        _windowPosX += actions.ContinuousActions[0] * Time.deltaTime * _windowMoveSpeed;
        _windowPosY += actions.ContinuousActions[1] * Time.deltaTime * _windowMoveSpeed;
        _windowSizeX += actions.ContinuousActions[2] * Time.deltaTime * _windowSizeSpeed;
        _windowSizeY += actions.ContinuousActions[3] * Time.deltaTime * _windowSizeSpeed;

        // send to grasshopper
        gameObject.GetComponent<Gh_IO>().msgToGh = $"{_windowPosX},{_windowPosY},{_windowSizeX},{_windowSizeY},{_wallHeight},{_wallWidth}";

        // get from grasshopper
        string dataFromGh = gameObject.GetComponent<Gh_IO>().msgFromGh;

        // check for message changes
        _previousDataFromGh = _lastDataFromGh;
        _lastDataFromGh = dataFromGh;
        if (_lastDataFromGh != _previousDataFromGh)
        {
            _dataFromGhIsChanged = true;
        }

        // if data is changed
        if (!_dataFromGhIsChanged) return;
        _ghData = JsonUtility.FromJson<GhData>(dataFromGh);

        SetReward(_ghData.Score);
        // if any of the window smaller than 200x200, end episode
        if (_ghData.WindowScale.Any(winScale => winScale.x < 0.2f || winScale.y < 0.2f))
        {
            AddReward(-50);
            ResetWindow();
        }

        //if any of the window is touching boundary, end episode
        if (_ghData.IsAtBounds.Any(isAtBound => isAtBound))
        {
            AddReward(-50);
            ResetWindow();
        }

```



```csharp
        if (_ghData.MinCutRatio >= 90)
        {
            AddReward(+100);
            EndEpisode();
        }
    }

    public override void Heuristic(in ActionBuffers actionOut)
    {
        ActionSegment<float> continuousActions = actionOut.ContinuousActions;
        continuousActions[0] = Input.GetAxisRaw("Horizontal");
        continuousActions[1] = Input.GetAxisRaw("Vertical");
    }

    private void ResetWindow()
    {
        _windowPosX = _windowInitPosX;
        _windowPosY = _windowInitPosY;
        _windowSizeX = _windowInitSizeX;
        _windowSizeY = _windowInitSizeY;
    }
}
```

UDP SERVER

UNITY & GRASSHOPPER DATA COMMUNICATION

To achieve real-time communication between Unity and Grasshopper, a UDP server was setup. UDP (User Datagram Protocol) is a connectionless protocol that allows for fast and efficient data transfer without the need for a dedicated connection. The server was configured to listen on a specific port for incoming messages from Grasshopper and Unity.

Grasshopper sends a JSON file to Unity containing the necessary data for the training process. JSON (JavaScript Object Notation) is a lightweight data-interchange format that is easy to read and write. The JSON file contains data such as socre, salvage timber usage and other attributes necessary for the ML-Agent to observe. The JSON format was chosen for Unity due to Unity have existing JSON Parser.

Unity sends a simple CSV (Comma-Separated Values) file format string back to Grasshopper. The CSV file contains data related to the translation & scale data of window and wall. The CSV file format was chosen for its simplicity and ease of parsing.

On both platforms, the messages are parsed and the data is used to update the program.

GRASSHOPPER --> UNITY

```
{
 "Score": 149.90686,
 "TotalOffcutsAmount": 3.06,
 "TotalSalvageLength": 55.89,
 "MaterialEfficiency": 182.64705882352942,
 "OffcutsCount": 16,
 "LaborEfficiency": 22.5,
 "ReuseCount": 7,
 "MinCutRatio": 33.333333333333329,
 "WallScale": {
  "x": 6.665805,
  "y": "2.575211"
 },
 "WindowPos": [
  {
   "x": 1.0002009999999997,
   "y": 0.0,
   "z": 2.0183169999999997
  }
 ],
 "WindowScale": [
  {
   "x": 0.8269151,
   "y": 1.128883
  }
 ],
 "IsAtBounds": [
  false
 ]
}
```

UNITY --> GRASSHOPPER

1.000201, 2.018317, 1.128883, 0.8269151, 2.575211, 6.665805

(WindowPosX), (WindowPosY), (WindowSizeX), (WindowSizeY), (WallHeight), (WallWidth)

SCALABILITY & LIMITATION

The grasshopper script can adapt to different timber structure logic, with little Unity ML-Agent logic code changes. This ensures a scalability and adaptability of the project. For example, a machine learning model could be trained on a dataset of complex timber structures and used to generate new designs that are structurally sound and visually appealing.

However a major limitation of trained models in timber frame design is their inability to adapt to drastically different timber structures. Although the model is able to scale with mostly similar wall frames, the model cannot adapt to drastically different timber structures. This means that the model needs to be re-trained if the timber frame changes drastically. This can be time-consuming and costly, and it limits the flexibility of the model.

FUTURE DEVELOPMENT

I believe that the ML-Agent Reinforcement learning approach also holds great potential for optimizing timber frames on more complex timber structures that are irregular. This technology has already been proven effective in optimizing simple structures, but extending it to more complex structures will require a more robust and sophisticated approach.

One of the key challenges that will need to be addressed is the integration of load analysis, wind analysis, and solar analysis using tools such as the grasshopper plugin, Ladybug. With developed foundation of this research experiment, it would be easier to implement these analysis tools in the future. These analyses will need to be integrated into the ML-Agent Reinforcement learning model to ensure that the system is able to generate optimal solutions that are robust to real-world environmental conditions.

Furthermore, it will be important to determine how different types of analysis can affect the structure formation of the timber and how well it can be adapted to real-world scenarios after being trained with real data. This will require a comprehensive analysis of the data generated by the model and the identification of the key factors that contribute to optimal timber frame structures.

REFERENCES

Huang, C. (2021). Reinforcement Learning for Architectural Design-Build—Opportunity of Machine Learning in a Material-informed Circular Design Strategy. 171–180. https://doi.org/10.52842/conf.caadria.2021.1.171

Wang, D., & Snooks, R. (n.d.). INTUITIVE BEHAVIOR_The Operation of Reinforcement Learning in Generative Design Processes. 10.

Wang, D., & Snooks, R. (2021). Artificial Intuitions of Generative Design: An Approach Based on Reinforcement Learning. In P. F. Yuan, J. Yao, C. Yan, X. Wang, & N. Leach (Eds.), Proceedings of the 2020 DigitalFUTURES (pp. 189–198). Springer Singapore. https://doi.org/10.1007/978-981-33-4400-6_18