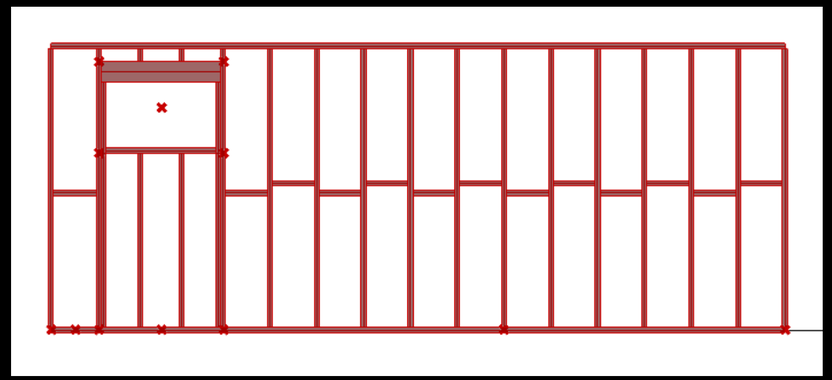
OPTIMIZATION OF MATERIAL & LABOR EFFICIENCY USING WITH SALVAGED TIMBER

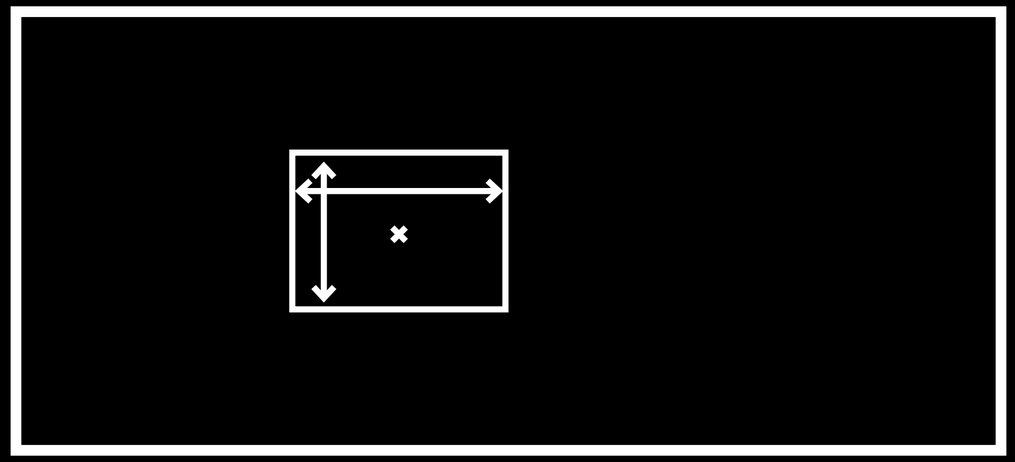
GOAL

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.



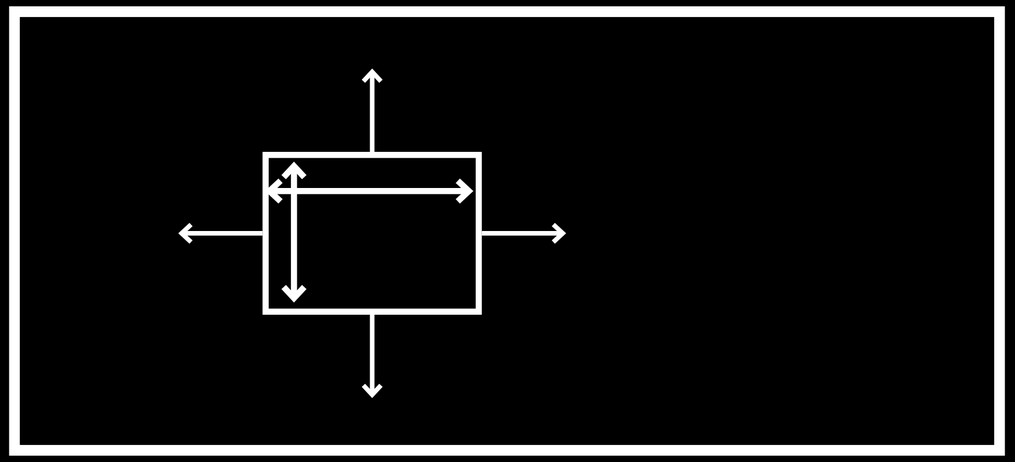
STEP 1: DEFINE A RANDOM SIZE OF TIMBER FRAME

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.

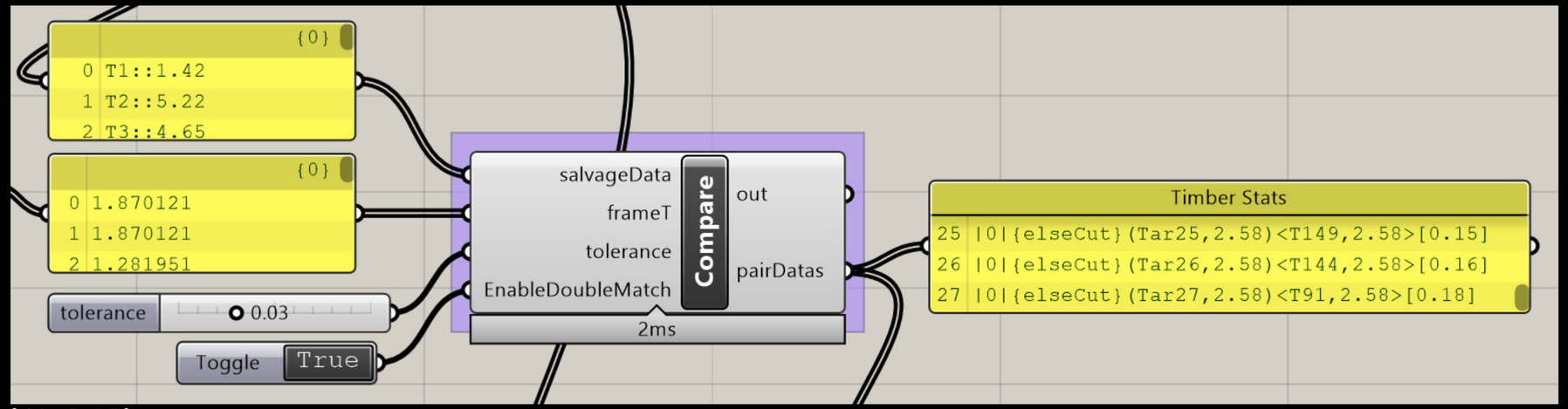
This can be achieved by selecting random locations on the wall where windows can be placed. The number of windows is expected to be defined randomly, depending on the size of the timber frame. Although this function is currently not implemented.



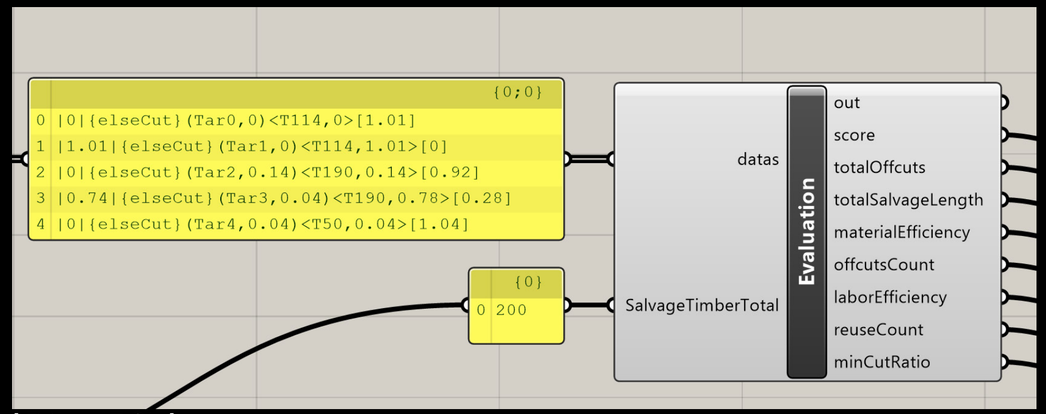
STEP 3: MOVE & RESIZE WINDOW

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.

PREVIOUS COMPONENTS & DATA HANDELING

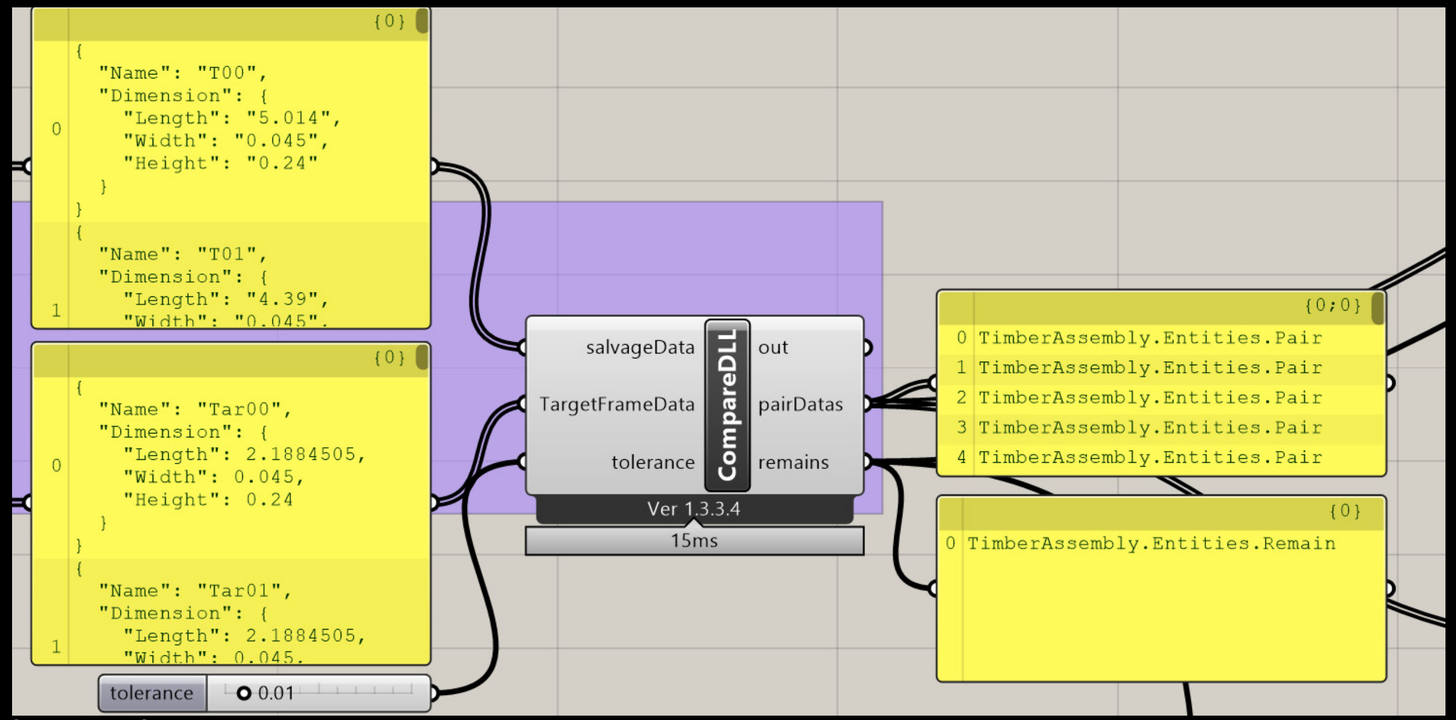


[COMPARE]



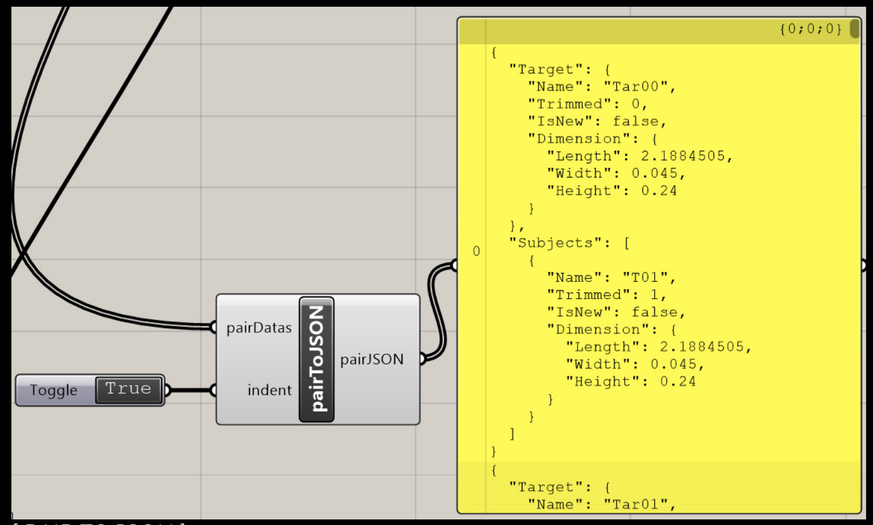
[EVALUATION]

NEW COMPONENTS & DATA HANDELING

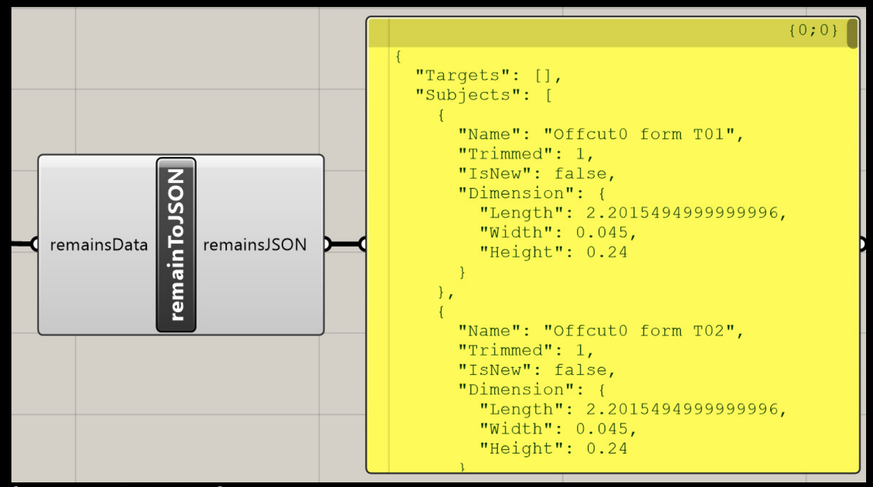


[COMPARE]

DATA DECODE



[PAIR TO JSON]

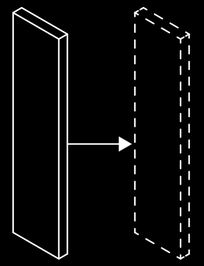


[REMAIN TO JSON]

MATCHING LOGIC

```
using System.Linq;
using System.Reflection;
using TimberAssembly;
using TimberAssembly.Entities;

private void RunScript(List<string> salvageData, List<string> TargetFrameData, double tolerance, ref object pairDatas, ref object
remains)
{
  // version display
  Assembly assembly = Assembly.GetAssembly(typeof(TimberAssembly.Match));
  string version = assembly.GetName().Version.ToString();
  Component.Message = "Ver " + version;

  // deserialize
  List<Agent> salvageAgents = Parser.DeserializeToAgents(salvageData);
  List<Agent> targetAgents = Parser.DeserializeToAgents(TargetFrameData);


  // tolerance limit
  if (tolerance > smallestDimension(salvageAgents))
  {
    string message = "Tolerance is too large. Tolerance is set to " + smallestDimension(salvageAgents).ToString();
    Component.AddRuntimeMessage(GH_RuntimeMessageLevel.Warning, message);
    tolerance = smallestDimension(salvageAgents);
  }

  // Matching operation
  Match matchOperation = new Match(targetAgents, salvageAgents, tolerance);

  Remain remain = new Remain();
  List<Pair> matchedPairs = matchOperation.ExactMatch(ref remain); // when subject == target
  matchedPairs.AddRange(matchOperation.SecondMatchSlow(ref remain)); // when subject 1 + subject 2 == target
  matchedPairs.AddRange(matchOperation.CutToTarget(ref remain)); // when subject > target
  matchedPairs.AddRange(matchOperation.ExtendToTarget(ref remain)); // when subject < target
  // Output data
  pairDatas = matchedPairs;
  remains = remain
}
```

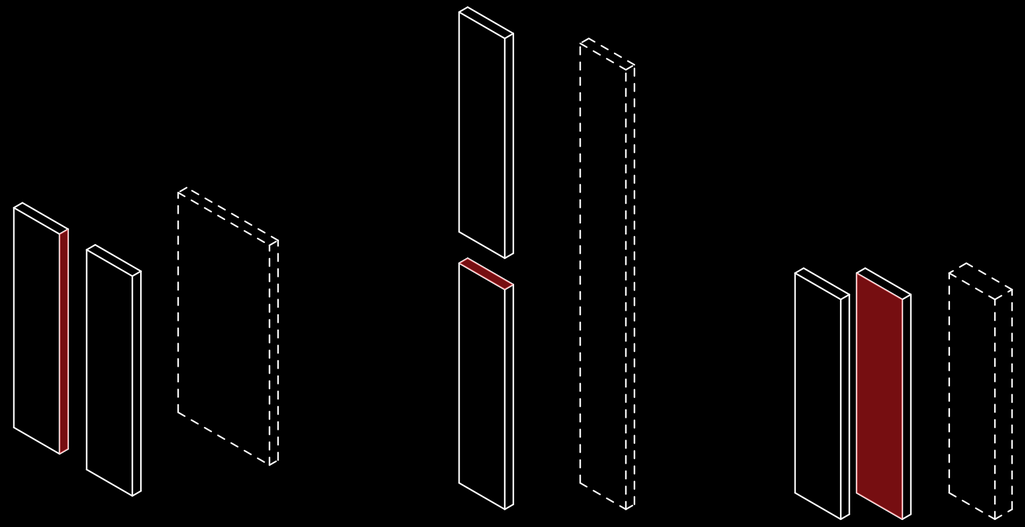
```
private double smallestDimension(List<Agent> agents)
{
  double smallestDimension = double.MaxValue;
  foreach (var agent in agents)
  {
    if (agent.Dimension.ToList().Min() < smallestDimension)
    {
      smallestDimension = agent.Dimension.ToList().Min();
    }
  }
  return smallestDimension;
}
```



1. [EXACT MATCH]

   When subject == target

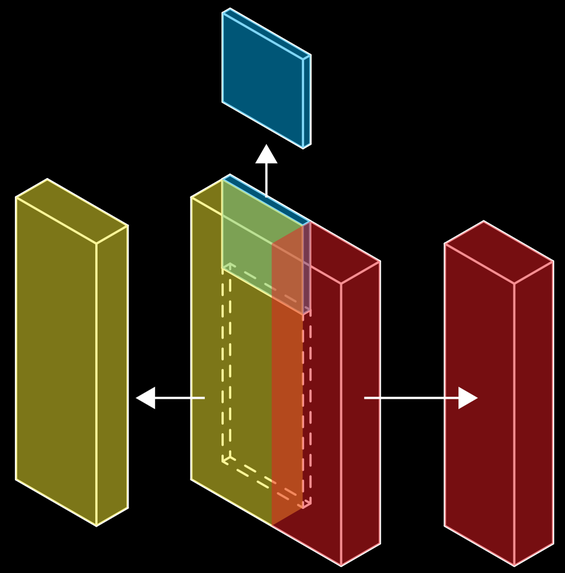
```
List<Pair> matchedPairs = matchOperation.ExactMatch(ref remain);
```



2. [SECOND MATCH]

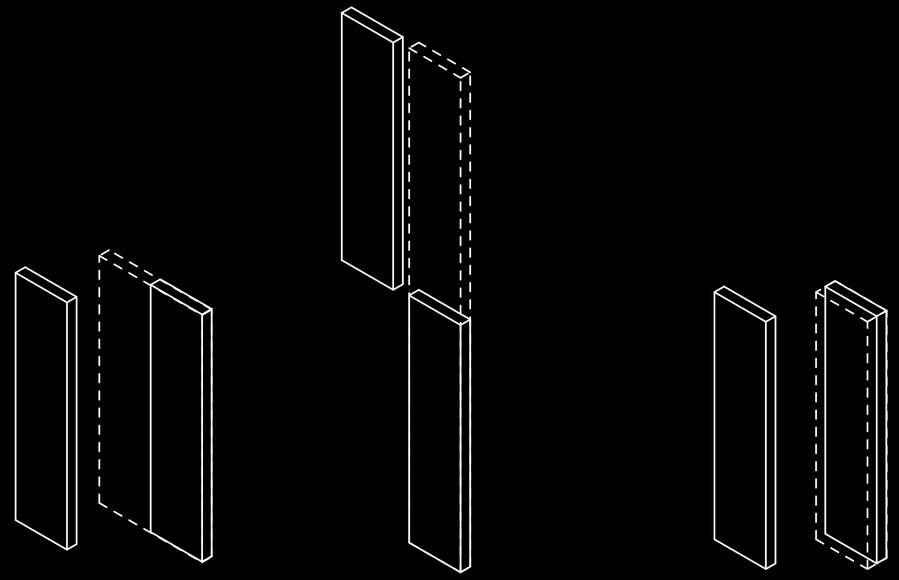
   When subject A + subject B == target

```
matchedPairs.AddRange(matchOperation.SecondMatchSlow(ref remain));
```



3. [CUT TO TARGET] (create offcuts)
   When subject > target

```
matchedPairs.AddRange(matchOperation.SecondMatchSlow(ref remain));
```



4. [EXTEND TO TARGET] (introduce new timber)
   When subject < target

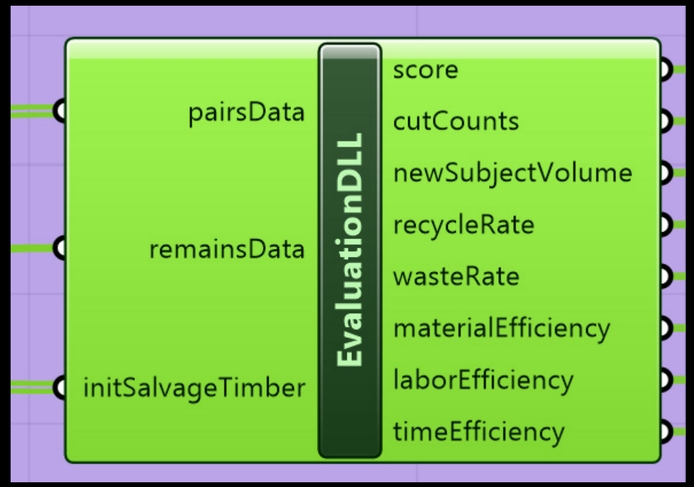
```
matchedPairs.AddRange(matchOperation.CutToTarget(ref remain));
```

Note:
you don't have to follow these order, it can be alter as needed.
This arrangement could also be alter by ML-Agent.

EVALUATION LOGIC

```
using System.Linq;
using TimberAssembly;
using TimberAssembly.Entities;

private void RunScript(List<object> pairsData, object remainsData, List<string> initSalvageTimber, ref object score, ref object
cutCounts, ref object newSubjectVolume, ref object recycleRate, ref object wasteRate, ref object materialEfficiency, ref object
laborEfficiency, ref object timeEfficiency)
{
  List<Agent> initSalvage = Parser.DeserializeToAgents(initSalvageTimber);
  Remain remain = (Remain) remainsData;
  List<Pair> pairs = pairsData.OfType<Pair>().ToList();

  Evaluate evaluate = new Evaluate(pairs, remain, initSalvage);

  cutCounts = evaluate.GetCutCount();
  newSubjectVolume = evaluate.GetNewSubjectVolume();
  recycleRate = evaluate.GetRecycleRateVolume();
  wasteRate = evaluate.GetWasteRateByVolume();
  materialEfficiency = evaluate.EvaluateEfficiencyByVolume();
  laborEfficiency = evaluate.EvaluateEfficiencyByCutCount();
  timeEfficiency = evaluate.EvaluateEfficiencyByTime(0.2, 0.05);

  score = GetOverallScore(evaluate, 0.2, 0.05);
}
```

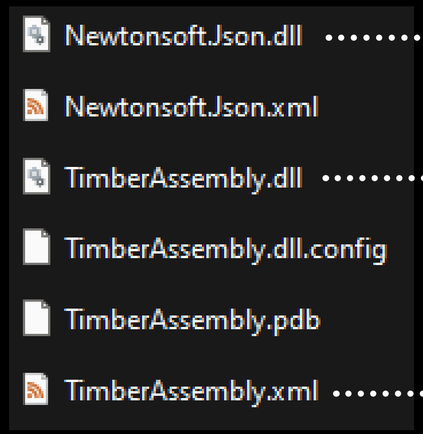
[EVALUATION]




```
private double GetOverallScore(Evaluate eval, double timePerSubject, double timePerCut)
{
  // These weights should add up to 1
  double wasteRateWeight = 0.2;
  double recycleRateWeight = 0.25;
  double materialEfficiencyWeight = 0.25;
  double laborEfficiencyWeight = 0.15;
  double timeEfficiencyWeight = 0.15;

  // Calculate each component of the overall score
  double wasteRateScore = (1 - eval.GetWasteRateByVolume()) * wasteRateWeight;
  double recycleRateScore = eval.GetRecycleRateVolume() * recycleRateWeight;
  double materialEfficiencyScore = eval.EvaluateEfficiencyByVolume() * materialEfficiencyWeight;
  double laborEfficiencyScore = eval.EvaluateEfficiencyByCutCount() * laborEfficiencyWeight;
  double timeEfficiencyScore = eval.EvaluateEfficiencyByTime(timePerSubject, timePerCut) * timeEfficiencyWeight;

  // Sum up all the components to get the overall score
  double overallScore = wasteRateScore + recycleRateScore + materialEfficiencyScore + laborEfficiencyScore + timeEfficiencyScore;

  return overallScore;
}
```

[GET OVERALL SCORE]

CODE BASE
[API]



[DLL FILE]

```csharp
public static class Parser
{
  public static List<Agent> DeserializeToAgents(List<string> jsons) {...}

  public static List<string> SerializeAgentPairs(List<Pair> pairs, bool indent = false) {...}

  public static List<string> SerializeAgentRemains(Remain remains, bool indent = false) {...}
}
```

[PARSER]

```csharp
public class Match
{
  public List<Agent> TargetAgents { get; set; }
  public List<Agent> SalvageAgents { get; set; }
  public double Tolerance { get; set; }

  public Match(List<Agent> targetAgents, List<Agent> salvageAgents, double tolerance = 0.01)
  {
      TargetAgents = targetAgents;
      SalvageAgents = salvageAgents;
      Tolerance = tolerance;
  }
  public List<Pair> ExactMatch(ref Remain remains) {...}

  public List<Pair> SecondMatchSlow(ref Remain remains) {...}

  public List<Pair> SecondMatchFast(ref Remain remains) {...}

  public List<Pair> CutToTarget(ref Remain remain) {...}

  public List<Pair> RemainMatch(Remain previousRemains) {...} // DEPRECATED! Use ExtendToTarget Instead.

  public List<Pair> ExtendToTarget(ref Remain remain) {...}
}
```

[MATCH]

```csharp
public class Evaluate
{
  private readonly List<Pair> _pairs;
  private readonly Remain _remains;
  private readonly List<Agent> _totalInitialSubjects;

  public Evaluate(List<Pair> pairs, Remain remains, List<Agent> totalInitialSubjects)
  {
      _pairs = pairs;
      _remains = remains;
      _totalInitialSubjects = totalInitialSubjects;
  }

  public int GetCutCount() {...}

  public double GetUsedSubjectVolume() {...}

  public double GetWasteVolume() {...}

  public double GetSubjectInitVolume() {...}

  // Waste Rate = Waste Volume / Total Volume
  // (Lower the better)
  public double GetWasteRateByVolume() {...}

  // Get total volume of new timbers.
  public double GetNewSubjectVolume() {...}

  // Get the percentage of used timbers compare to initial timbers by volume.
  // (Higher the better)
  public double GetRecycleRateVolume() {...}

  // MaterialEfficiency = 1 - Waste Rate
  // (Higher the better)
  public double EvaluateEfficiencyByVolume() {...}

  // Labor Efficiency = (Number of timbers - Number of cuts) / Number of timbers
  // (Higher the better)
  public double EvaluateEfficiencyByCutCount() {...}

  /// <summary>
  /// Time Efficiency = Perfect Time / Actual Time
  /// (Higher the better)
  /// </summary>
  /// <param name="timePerSubject">Time taken for each timber to be install</param>
  /// <param name="timePerCut">Time taken for each timber to be cut</param>
  public double EvaluateEfficiencyByTime(double timePerSubject, double timePerCut) {...}
}
```

[EVALUATE]

CODE BASE
[CLASS OBJECT]

```cs
public class Agent
{
    public string Name { get; set; }
    public int Trimmed { get; set; }
    public bool IsNew { get; set; }
    public Dimension Dimension { get; set; }

    public Agent(string name = null, Dimension dimension = null, int trimmed = 0, bool isNew = false)
    {
        Name = name;
        Dimension = dimension;
        Trimmed = trimmed;
        IsNew = isNew;
    }

    public override string ToString() {...}

    public double Volume() {...}
}
```

[AGENT]

```cs
public class Pair
{
    public Agent Target { get; set; }
    public List<Agent> Subjects { get; set; }

    public Pair(Agent target = null, List<Agent> subjects = null)
    {
        Target = target;
        Subjects = subjects;
    }
}
```

[PAIR]

```cs
public class Remain
{
    public List<Agent> Targets { get; set; }
    public List<Agent> Subjects { get; set; }
}
```

[REMAIN]

```cs
public class Dimension
{
    public double Length { get; set; }
    public double Width { get; set; }
    public double Height { get; set; }

    public Dimension() { }

    public Dimension(double length, double width, double height)
    {
        Length = length;
        Width = width;
        Height = height;
    }

    public Dimension(List<double> dimensions)
    {
        if (dimensions.Count != 3)
            throw new ArgumentException($"Dimension take exactly 3 values, but has {dimensions.Count} values.");
        Length = dimensions[0];
        Width = dimensions[1];
        Height = dimensions[2];
    }

    public List<double> ToList() {...}

    public override string ToString() {...}

    public bool IsAnyLargerThan(Dimension dimension) {...}

    public bool IsAnyLargerOrEqualThan(Dimension dimension) {...}

    public bool IsAnySmallerThan(Dimension dimension) {...}

    public bool IsAnySmallerOrEqualThan(Dimension dimension) {...}

    public bool IsAllLargerThan(Dimension dimension) {...}

    public bool IsAllLargerOrEqualThan(Dimension dimension) {...}

    public bool IsAllSmallerThan(Dimension dimension) {...}

    public bool IsAllSmallerOrEqualThan(Dimension dimension) {...}

    public bool Equality(Dimension dimension, double tolerance = 0.01) {...}

    public Dimension Absolute() {...}

    public double GetVolume() {...}

    public void Subtract(Dimension dimension) {...}

    public void Add(Dimension dimension) {...}

    public static Dimension Zero() {...}

    public static Dimension GetSum(Dimension dimension1, Dimension dimension2) {...}

    public static Dimension GetSum(List<Dimension> dimensions) {...}

    public static Dimension GetDifference(Dimension dimension1, Dimension dimension2) {...}
}
```

[DIMENSION]