Building a probabilistic language for kids

Quang Long Ho Ngo, sciper : 310781
Supervisor : Richard Davis

Introduction

The COVID-19 pandemic serves as one of many examples where basic understanding of probability has advantages. This understanding allows people to make informed decisions and predictions in the face of uncertainty. For example, estimating the spreading of the virus or the effectiveness of the vaccine involves probabilistic thinking.

More generally, we are building more and more computational models that are based on probability, most of which are associated with machine learning of some sort.Building these models normally requires an understanding in fields of math such as analysis or linear algebra, where concepts may be abstract and difficult to grasp for people with little math training. An important part of the population lacks these concepts and therefore can't develop complex probabilistic models.

Probabilistic programming languages are an attempt to overcome this problem. Probabilistic programming languages make it easier for people to build and evaluate probabilistic models by providing a high-level interface for specifying and manipulating these models. They come with many features which make it simpler to build and evaluate probabilistic modes, such as block-box inference algorithms. Pyro[1] is one of the most popular of these languages and is used at Uber for their probabilistic calculations.

Currently, the existing probabilistic programming languages are not perfect and many of them are still too complicated to use. Furthermore, they require the user to have a strong background in programming. However, a simpler tool might be useful for many people. If we take the perspective of education, having a probabilistic tool that is able to correct and develop a model might help students understand probability from a practical standpoint. The students would be able to quickly prototype and test different probabilistic models for a particular problem.
The goal of this project is to provide a tool that is able to build and evaluate probabilistic models without doing any programming. In this paper, we will begin by exploring what is expected when teaching probability from an education science view and existing Bayesian network creation tools. Then we will detail design choices that were made when building the tool and some technicalities. We will end by discussing what the tool is currently able to do and improvement perspectives are there.

[1] https://www.uber.com/en-CH/blog/pyro/

Literature Review

In order to develop such a tool, we need to understand how probability is taught and how it is perceived by people learning it. We will discuss how probability is taught and perceived and more importantly at what age it should be introduced.

Piaget is a central figure when we come to learn about education science. His famous works include his theory of cognitive development[2]. His contributions to the field may be less accurate with today's views on education, but the foundations are still relevant. What we are looking for on his work is his viewpoint on teaching probability[3].
According to Piaget and Inhelder[4], children do not have an inborn notion of chance. However, it is acquired after the acquisition of the corresponding opposite notions. This means that chance events can be grasped only after identifying a set of phenomena which are not random events. Probability is the combination of two concepts : chance and operations.
A child's knowledge will make the transition from reversible and composable operations to the irreversible ones. In order to understand this transition, we will have to link it to Piaget's theory of cognitive development. The main idea is that a child goes from a first state (situated before 7-8 y.o) where he can't grasp the idea of reversibility to a third stage (from 11-12 y.o) in which he is now capable of thinking in a more formal way. By applying logical concepts to his everyday life. The notion of chance (as seen by Piaget) is the opposition of two types of causality : Chance is distinct from determinism. Chance comes with laws, and it is opposed to the notion of miracles. Children tend to understand mechanisms by creating a causal model. We can see it with the question they love to ask : "Why ?".

Teaching probability to children can be hard as they lack the notion of chance. This problem can be perceived in the English school system at a primary level[5]. Objectives for most pupils at the age of 11 were first changed in 1985 to now include simple terms in probability and understanding of elementary notions of probability. Textbooks have then been updated to include more content on probability and probabilistic vocabulary was introduced at a younger age. However, in the following 11 years, the study of probability was progressively scaled back in primary schools. There were multiple problems that are related, but we will only look into one of them that is directly relevant to developing probabilistic tools. The problem is the misconceptions about probability[6]. We will now take people at a high school level, because as we have seen it is quite hard to teach probability at a younger age and attempts in doing it were not successful.

[2] https://en.wikipedia.org/wiki/Piaget%27s_theory_of_cognitive_development
[3] https://www.stat.auckland.ac.nz/~iase/publications/17/8D1_SANM.pdf
[4] https://en.wikipedia.org/wiki/B%C3%A4rbel_Inhelder
[5] https://www.tandfonline.com/doi/abs/10.1080/0305764042000289938
[6] https://link.springer.com/chapter/10.1007/0-387-24530-8_11

By the end of high school, students should have learned about combinatorics and several notions in probability such as: conditional probability, independence, discrete and random variables. However, high school students tend to have misconceptions that can be categorized.

Representativeness
Students tend to think that even a small sample should respect the law of large numbers. They tend to have a mental representation of what a random sequence should look like. This heuristic doesn't seem to improve from 14 y.o students who have never studied probability to 18 y.o students who have studied probability for about a month.

Equiprobability bias
Students tend to forget to combine different results if they are in different order, not realizing that they are the same event. For example, by throwing 2 independent dice, it is more likely to obtain a 5 and a 6 rather than having a 5 twice. The typical error is to think these two events are equally likely.

Causal Reasoning and the Fallacy of the Time Axis
Students tend to believe that an event A that has occurred after an event B has no influence on the result of B. In the setup, where an urn contains two white balls and two red balls. We pick up two balls at random, one after the other without replacement. The probability that the first ball is red, given that the second ball is also red is sometimes answered as ½. Showing that the reasoning is having information on event A gives no information on event B.

These misconceptions can be addressed with probabilistic tools by providing a visual representation of the relationship between different variables and the different outcomes of a model. Also, the student would be able to experiment with many scenarios that are known to create a particular misconception and see how the probability changes, which can help them better understand the underlying concepts.

There are existing tools that enable the construction of Bayesian networks. The most complete of them is BayesBox[7]. The main problem is that it is proprietary software which means that it is harder to customize for different usage and the usage may be limited by the financial aspect. Nonetheless, it is a good tool. It supports both discrete and continuous variables. There are custom functions and observed values can be entered and processed by the inference algorithm. This project was inspired by BayesBox and the main goal was to transform it into something that is more user-friendly, in particular for someone that has limited understanding of probability.

[7] https://demo.bayesfusion.com/bayesbox.html

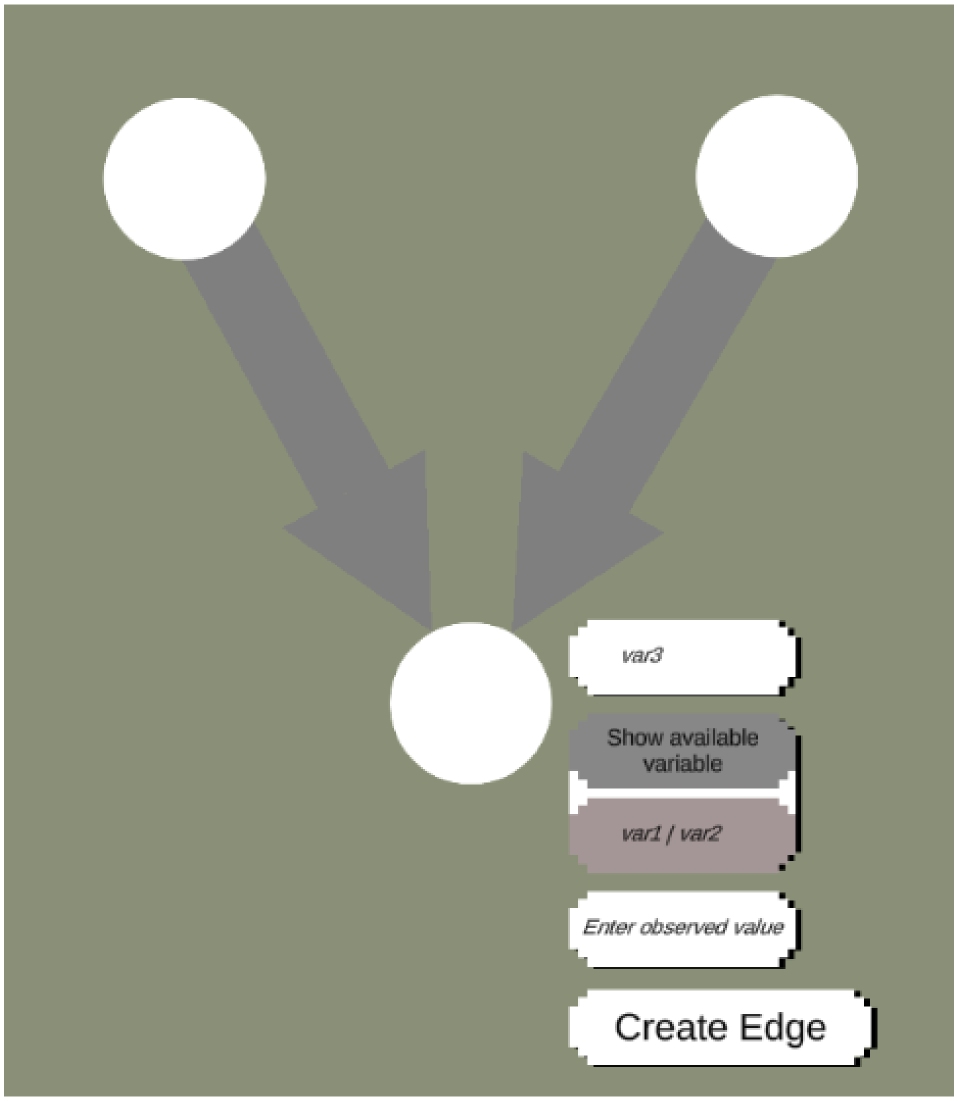
BayANet[8] is an open-source[9] tool that tries to achieve the same goal. But this tool is way less complete than BayerBox. The interface is way simpler which is good because the goal is clearer and creating nodes and edges is more straightforward. But the problem with this tool comes from the probabilistic side. Each value needs to be entered manually and there is no custom function. This leads to less abstraction and it becomes harder to create more complex models. These two tools were the main inspiration for this project. BayesBox for the functionalities it offers and BayANet for its simple and friendly interface.

System design

This tool uses Unity as a front-end engine, it enables rapid UI development and advanced links to the back-end. It allows the tool to be compiled for different platforms and can even be uploaded as a live demo on the unity website so that people can try it without installing it. The probability calculations are performed by infer.NET[10] because dealing with these calculations can be tricky and often leads to multiple errors if it is implemented incorrectly. The main difficulty of this project is therefore to create a bridge between the UI and the library infer.NET. Many design choices needed to be made. Because including all infer.NET functionalities would make the UI design harder because an inexperienced user should be able to use it or at least with only some minutes of explanation.

The user is able to create nodes and edges that represent the Bayesian network. Edges are directed and represent links between different nodes. There is only one kind of edge and this means that we don’t carry the meaning of the link between 2 nodes via the edge. To specify what type of connections 2 nodes have, we use custom functions.

[8] http://www.cs.man.ac.uk/~gbrown/bayes_nets/#
[9] https://github.com/aleksieva/bayes_nets
[10] https://dotnet.github.io/infer/



In the image above, we can see the custom function “var1 | var2" which represents how the two nodes above influence the third one. This image also shows that the user can enter an observed value in the third node which will trigger the infer.NET inference algorithm and update the value of all other nodes. One thing that the user can’t do is build a cyclic graph because it doesn’t make sense in Bayesian networks to have cycles. A ProbaViz button is at the bottom and when the user clicks on it, it will display the inferred probabilities of all the nodes in the console.

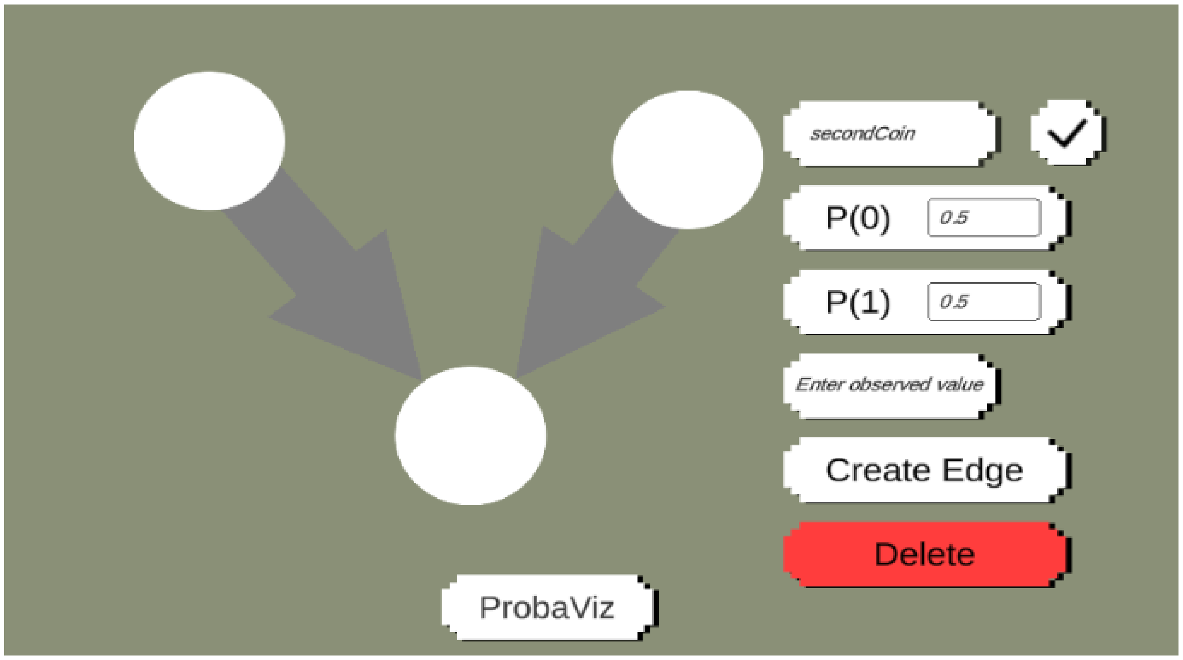
To illustrate all functionalities, we will create a model that comes from the infert.NET examples which is the two coins model[11].

[11] https://dotnet.github.io/infer/userguide/Two%20coins%20tutorial.html

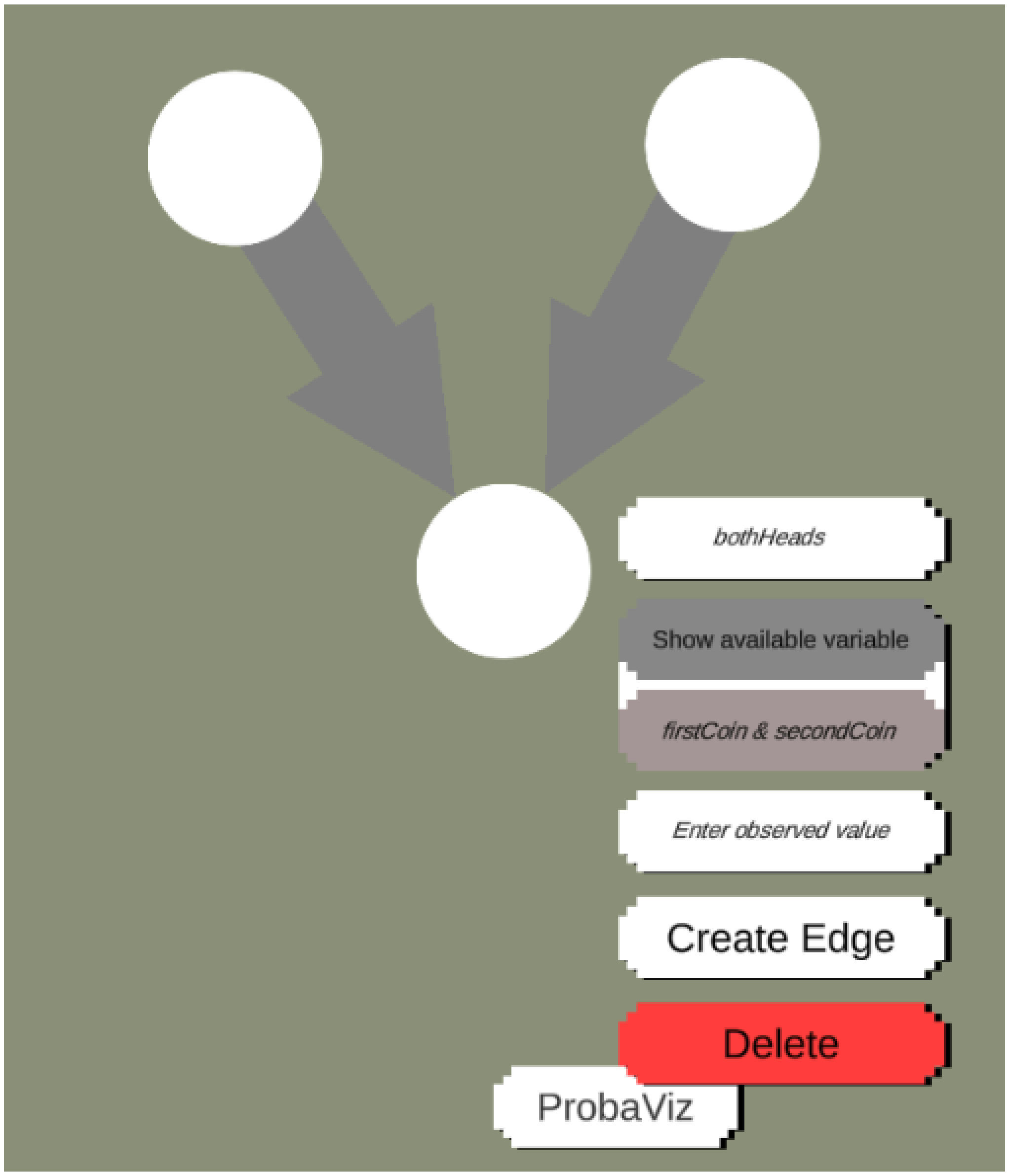
```
Variable<bool> firstCoin = Variable.Bernoulli(0.5);
Variable<bool> secondCoin = Variable.Bernoulli(0.5);
Variable<bool> bothHeads = firstCoin & secondCoin;
InferenceEngine engine = new InferenceEngine();
Console.WriteLine("Probability both coins are heads:
"+engine.Infer(bothHeads));
```

Here is the code that is used in the example, and now we are going to rebuild it with the tool.

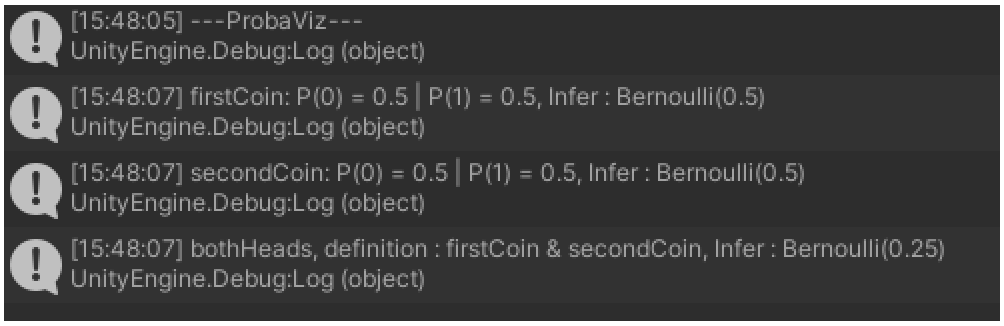
Step 1 :
We create all the nodes and the edges, then we name the nodes. The default probability is already 0.5 for each outcome and this corresponds to our model so we don't have to input anything.



Step 2 :
We need to specify the function for bothHeads, because the default function will be firstCoin|secondCoin but we want firstCoin&secondCoin.

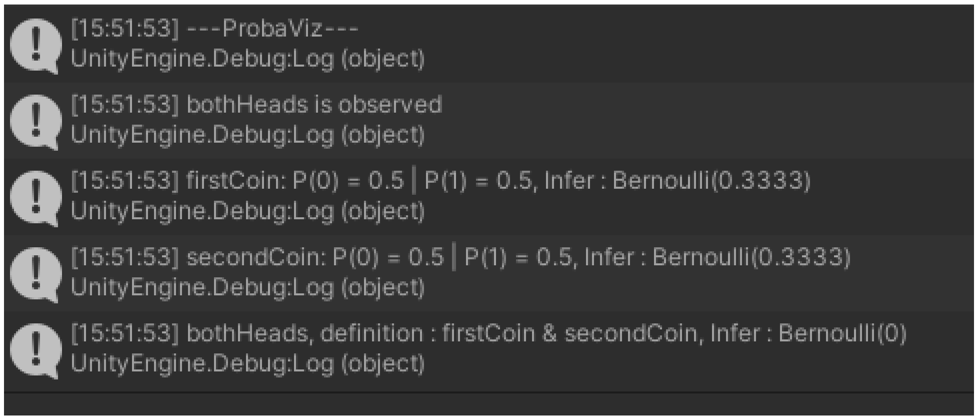


Step 3 :
We will now click on ProbaViz and it will display the different probability in the console



```
[15:48:05] ---ProbaViz---
UnityEngine.Debug:Log (object)
[15:48:07] firstCoin: P(0) = 0.5 | P(1) = 0.5, Infer : Bernoulli(0.5)
UnityEngine.Debug:Log (object)
[15:48:07] secondCoin: P(0) = 0.5 | P(1) = 0.5, Infer : Bernoulli(0.5)
UnityEngine.Debug:Log (object)
[15:48:07] bothHeads, definition : firstCoin & secondCoin, Infer : Bernoulli(0.25)
UnityEngine.Debug:Log (object)
```

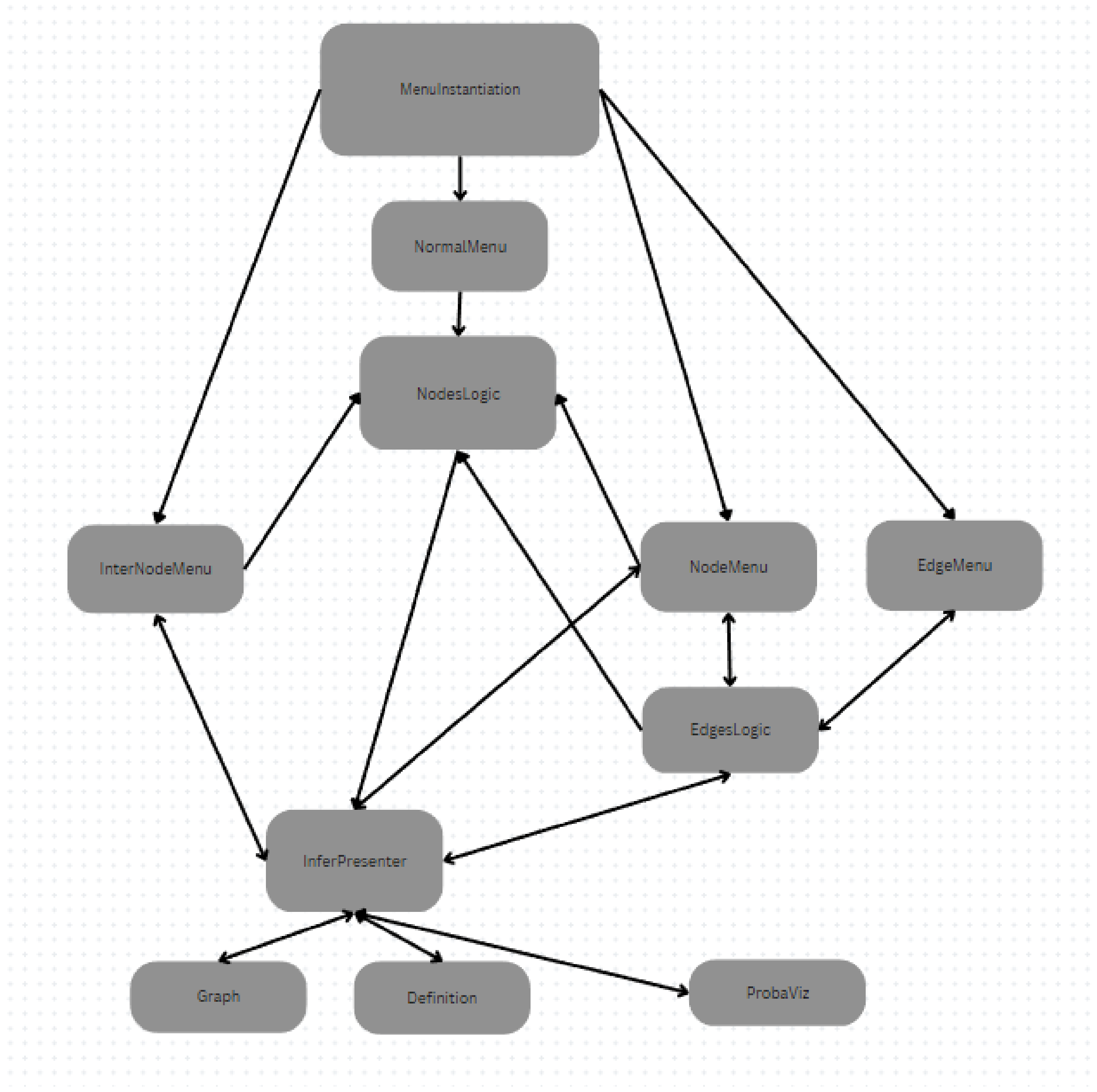
Step 4 :
We can enter an observed value in the bothHeads node and put a 0. Now if we click on ProbaViz again, the probabilities have changed because of the new observation.



```
[15:51:53] ---ProbaViz---
UnityEngine.Debug:Log (object)
[15:51:53] bothHeads is observed
UnityEngine.Debug:Log (object)
[15:51:53] firstCoin: P(0) = 0.5 | P(1) = 0.5, Infer : Bernoulli(0.3333)
UnityEngine.Debug:Log (object)
[15:51:53] secondCoin: P(0) = 0.5 | P(1) = 0.5, Infer : Bernoulli(0.3333)
UnityEngine.Debug:Log (object)
[15:51:53] bothHeads, definition : firstCoin & secondCoin, Infer : Bernoulli(0)
UnityEngine.Debug:Log (object)
```

Code implementation

We will now discuss technical aspects and the organization of the code. The class diagram is given in the next image.
The role of MenuInstantiaton is to manage the menus. It will spawn one of the 4 kinds of menus depending on where the user clicked. Node has 2 menus, one for roots and the other one for non-roots. Each time a menu is spawned, it is linked to the corresponding GameObject and can talk to it. The main roles of menus are to input values, create new edges in the case of node menus, and delete objects.
Many arrows are here because we often need to retrieve all the edges or nodes for computation. This is often done with the GameObject.findObjectsWithTag function because every node and edge are tagged.
We then have a graph and a definition class which are the data logic. The graph class will give information on the state of the graph and if it is cyclic or not. The definition class will process the custom function input from the user and interpret it to give a infer.NET variable back.
To link these 2 classes with the rest of Unity classes, there is the inferPresenter class. Its main role is to forward information between the back-end and the front-end. When ProbaViz is called, it will also ask for information to inferPresenter before outputting probabilities.



Menus
When the user right click on the application, the MenuInstantiation [12]class will check the position of the mouse and check if there is an object underneath with the raycast function from Unity. It will also check that there is only one open menu at a time.

Nodes[13]
There are 2 types of nodes. Roots and non-roots. The difference is that roots can specify their probabilities for the value 0 and 1. When non-roots can only define a definition which makes use

[12] https://github.com/lipefree/bachelor_project/blob/main/Assets/Scripts/Managers/MenuInstantiation.cs
[13] https://github.com/lipefree/bachelor_project/blob/main/Assets/Scripts/Units/Nodes/NodesLogic.cs

of the parent nodes. This design allows the user to make meaningful models as it chooses for the user the kind of each node based on the graph representation. Each node can set a new name which will be used to be referred to when writing a definition in a non-root node. When a definition is entered, it will be checked and interpreted by the definition class.
Nodes come with default values, the initial probability is a Bernoulli with p=0.5 and the initial definition function will be parent1 | parent2 | … depending on the number of parents. We can also enter an observed value and the infer engine will be notified. The observed value can be erased to have a random variable again.

Edges[14]
To have a flexible UI, edges are generated via Unity's mesh system. An edge will always have a node at the base and a node at the tip. Moving the nodes will change the shape of the edge depending on the new distance and new angles between the nodes. When creating an edge, to confirm the build we need to click on a node that will be the node tip. If the user clicks elsewhere the edge will disappear. If a node is found, it will check if being attached to that node will create a cyclic graph by calling the graph class via the presenter.

Definition[15]
This file also contains a Lexer class, the role of the lexer is to consume an input and output a list of tokens and strings. These tokens will help process the input. Tokens are symbols (identifier, integer literal, delimiter or operations) or indicate file organization with spaces and a token that represent we are at the end of the input. There is also a notDefined token which represents the case where we can't process a part of the input which will later result in an error. Calling the parse method from the lexer will then give a list of tuples, where each tuple has the type of the token and its content in the form of a string.

Then we have the definition class with interpret as the main method. This method needs an environment which is the link between all variables and their names. Each token will be consumed one by one and their effect will be interpreted in C#. The interpreter also supports parenthesis, which require the code to handle priority of operation. Each time it encounters an open parenthesis, it will look for the closing one. If there is a problem with the parenthesis, the definition won't be accepted.

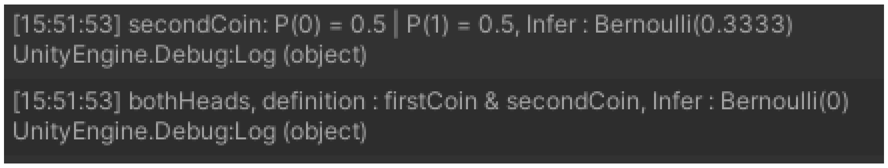
However, with this design the interpreter only supports Variable<bool> as results. To support more types, we would need to implement a complete interpreter with a lexer, parser, type checker and name analyzer. This would also enable the usage of infer.NET function from the user. An alternative to the current design would be to use code "blocks" which would be inspired

[14] https://github.com/lipefree/bachelor_project/blob/main/Assets/Scripts/Units/Edges/EdgesLogic.cs
[15] https://github.com/lipefree/bachelor_project/blob/main/Assets/Scripts/System/Definition.cs

by Blockly[16]. It will be easier to build a more generalized interpreter because we receive tokens directly, and there can be blocks that can represent some infer.NET functions directly such as various random variables generation. It will require more work on the UI for the input, as we want a simple interface and the number of blocks can become unmanageable if we include too many functionalities.

Graph[17]
The main purpose of the graph class is cycle detection in a graph. But it also gives many helper functions to get the roots of the graph, parents of a given node, etc. To detect if there is a cycle, we perform a topological sort. Internally the topological sort[18], first runs a depth-first search that also records the time for each node. The time represents the moment where we have searched for all the children of a particular node. This time is then used to perform topological sort. The logic being, if a child has a time that is less than his parent time, then we have a cycle.

Probaviz[19]
This class represents the tool that is used to give information about the different node's probability. For the moment, it only prints to the console, but the idea is to perform a visual representation for the user. It is coded using the async functionality from C#, because the call to infer.NEt is quite heavy and we don't want unity to wait for the result to move on. The implementation is quite simple, because the variables are defined using the interpreter. The function infer that comes from infer.NET gives the information we need to output. For the moment, it only prints the kind of variable we have, Bernoulli(0.5) for example. It will print differently if the node is a root or not.



```
[15:51:53] secondCoin: P(0) = 0.5 | P(1) = 0.5, Infer : Bernoulli(0.3333)
UnityEngine.Debug:Log (object)

[15:51:53] bothHeads, definition : firstCoin & secondCoin, Infer : Bernoulli(0)
UnityEngine.Debug:Log (object)
```

In the above image, we see two different print formats. The first one is a root and we specify the probabilities and the infer result. It is useful to have both in the case where we enter an observed value. The node bothHeads was observed to be false which influenced the secondCoin value, because the definition of bothHeads involves the node secondCoin. The left probabilities will then indicate the initial values of the variables and the user can observe how much it differs.

[16] https://developers.google.com/blockly
[17] https://github.com/lipefree/bachelor_project/blob/main/Assets/Scripts/System/Tree.cs
[18] https://en.wikipedia.org/wiki/Topological_sorting
[19] https://github.com/lipefree/bachelor_project/blob/main/Assets/Scripts/Units/ProbaViz.cs

Conclusion

The tool that we were able to develop is in between BayesBox and BayANet. The UI is simple, but we have inference and custom functions. This balance makes the life of the user easier, as some functionalities come with default values. Like the custom function or the nodes’ names. If we had more time, the ability to use continuous random variables would be the next step. It would allow the user to model more complex networks. ProbaViz should also be upgraded to something more complete that shows results on the playing screen instead of the console. A lot can also be done for the nodes and the edges, for example the thickness of the edge can represent the contribution of one node on another node. A tutorial should also be included to introduce new users to the tool and the ability to construct multiple models at the same time with a folder system should also be added.
Transitioning from Unity application to a web version would be a good option, because people can just create models from the browser without downloading which makes it more accessible and we could enable multiple users on the same model but it will be more complicated from a technical standpoint (maintaining user data, server management).
A pure scala approach would be meaningful if we are planning to build a huge network and want to enable way more tools. But even then using Unity could still be working. This tool is not oriented towards heavy use so optimization is not a priority.
User feedback on the tool should be one of the next steps, now that the tool has a first working version. These feedbacks could indicate what part of the UI is good and where to improve, and maybe even what relevant functionalities should be explored next.
As the tool is also supposed to be targeted towards education,a more specialized UI that eases the integration into math problem solving. It could be used as an alternative to standard ways of solving problems and show the student what is more important to notice in the problem.