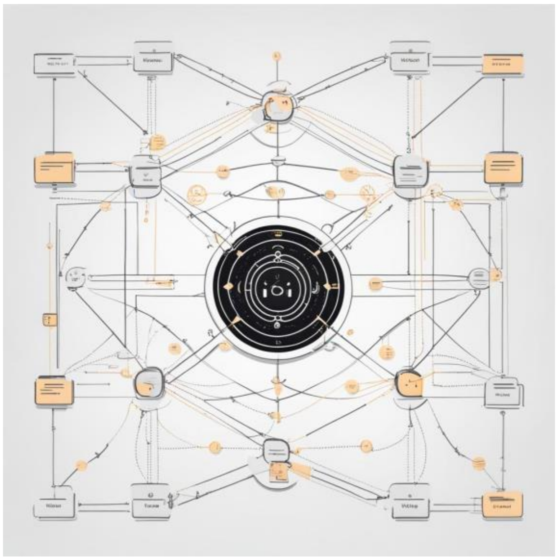
UNIVERSITY OF WEST ATTICA

&

UNIVERSITY OF LIMOGES

Master's Thesis

Visual scripting tools for machine learning development



Panagiotis Kyrkos

(aivc20013)

Supervisor: Anastasios Kesidis

Athens, September 2023

Visual scripting tools for machine learning development

Μέλη εξεταστικής επιτροπής συμπεριλαμβανομένου και του Εισηγητή

Η διπλωματική εργασία εξετάστηκε επιτυχώς από την κάτωθι Εξεταστική Επιτροπή

Α/α	ΟΝΟΜΑ ΕΠΩΝΥΜΟ	ΒΑΘΜΙΔΑ/ΙΔΙΟΤΗΤΑ	ΨΗΦΙΑΚΗ ΥΠΟΓΡΑΦΗ
1	Αναστάσιος Κεσίδης	Καθηγητής	Anastasios Kesidis Digitally signed by Anastasios Kesidis Date: 2023.10.09 10:51:27 +03'00'
2	Πάρις Μαστοροκώστας	Καθηγητής	Paris Mastorokostas Digitally signed by Paris Mastorokostas Date: 2023.10.09 12:18:39 +03'00'
3	Παναγιώτα Τσελέντη	ΕΔΙΠ	Panagiota Tselenti Digitally signed by Panagiota Tselenti Date: 2023.10.09 12:21:53 +03'00'

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Παναγιώτης Κύρκος του Αποστόλου με αριθμό μητρώου aivc20013 φοιτητής του Προγράμματος Μεταπτυχιακών Σπουδών Τεχνητή Νοημοσύνη και Οπτική Υπολογιστική του Τμήματος Μηχανικών Πληροφορικής της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής, δηλώνω ότι:

«Είμαι συγγραφέας αυτής της μεταπτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Ο Δηλών

[illegible]

Abstract

In computer science and not only, many and various software exist to design and implement different system using purely visual methods. These methods can greatly vary depending on the task to be solved and the field that the specific software is made for. The purpose of this thesis is the analysis, the design and the development of a visual scripting tool for machine learning development. At first various basic terms and information are explained about artificial intelligence, visual scripting tools and game engines. Current state is also described along with the current similar tools that exist. Afterwards, the Unity game engine is described, with which the visual scripting tool was implemented with. The basic mechanisms of Unity are explicated as they are required for the proper understanding of the implementation of the tool. Next the goal of the thesis is described, that is to say the analysis, the utilization and the design of the mentioned tool. Later, the implementation of Decision Tree Module Designer is discussed in detail, as to how the software is designed, how the interface works and looks and the user controls. Moreover, the core features are shown and explained along with the simulation mode. Then some early work in progress is shown that was implemented. Finally, the ways of possible utilization of such a tool are presented and analyzed in depth, followed by the future plans for the tool and how it can be improved with new features and better interface.

Table of contents

Table of Images

Chapter 1 Introduction

The purpose of this thesis, is the analysis, design and the implementation of a visual scripting tool for machine learning. Although there are many and various software around artificial intelligence, almost all focus on other aspects instead of the visual part, and those that do focus on visual aspect, they do not try to mix with the actual implementation of the machine learning algorithms. The software that was created as part of this thesis called Panoptes, is a visual tool to design, implement and adjust machine learning algorithms and more specifically decision trees. The tool was designed using the Unity game engine. All necessary features and mechanisms of Unity are described which are needed for the proper understanding of the implementation of the tool. Afterwards, the benefits of such a tool are described in detail and later on the implementation, user interface, controls, graphics and the current version features are analyzed in depth. These are then followed by the feature plans for the development of the software, that if implemented can turn the tool into the ultimate tool for designing and implement various machine learning systems.

In this chapter, all the basic and necessary information about artificial intelligence, visual design tools, game engines will be discussed. This chapter is important for the proper understanding of the upcoming chapters, especially the ones that go in detail about the tool. Apart from the mentioned information, similar tools that exist and implement part of the software will also be covered in the chapter.

1.1 Artificial Intelligence

In the realm of technological advancements, one concept has captured the imagination of scientists, innovators, and society at large: Artificial Intelligence (AI). As we navigate the complexities of the modern world, AI has emerged as a groundbreaking discipline that aims to replicate human intelligence and cognitive abilities in machines. It holds the promise of transforming the way we live, work, and interact with the world around us.

Artificial Intelligence refers to the development and deployment of intelligent machines that can perform tasks that typically require human intelligence. These machines are designed to perceive their environment, reason, learn from experience, and make decisions based on the information available to them. Through the amalgamation of various disciplines, including computer science, mathematics, and cognitive science, AI seeks to create systems that can emulate human-like thinking, problem-solving, and decision-making capabilities.

At its core, AI encompasses a range of techniques and methodologies aimed at enabling machines to exhibit intelligent behavior. These techniques include machine learning, where algorithms are trained on vast amounts of data to identify patterns and make predictions, as well as natural language processing, which enables machines to understand and generate human language. Additionally, AI encompasses computer vision, robotics, expert systems, and many other subfields, each contributing to different aspects of intelligent machine capabilities.

The applications of AI are far-reaching and diverse. In healthcare, AI-powered systems can analyze medical data, assist in disease diagnosis, and aid in the development of personalized treatment plans. In finance, AI algorithms can analyze market trends, optimize investment strategies, and detect fraudulent transactions. In transportation, AI can improve the efficiency of traffic management, enable autonomous vehicles, and enhance logistics operations. These are just a few examples among countless others that showcase the potential of AI to revolutionize industries and enhance human capabilities.

However, the development and deployment of AI also raise significant questions and challenges. Ethical considerations, such as privacy, bias, and accountability, come to the forefront as AI systems make decisions that impact individuals and society as a whole. The fear of job displacement due to automation and concerns about the concentration of power in the hands of AI systems also loom large. It is crucial to address these challenges and develop frameworks that ensure responsible and ethical AI development and deployment.

1.1.1 Machine Learning
Machine learning is a subset of artificial intelligence that focuses on the development of algorithms and statistical models that enable computers to learn and make predictions or decisions without being explicitly programmed. It is a field that explores how machines can automatically learn and improve from experience or data, allowing them to adapt and perform tasks more accurately and efficiently over time.

At its core, machine learning involves training models on large datasets to recognize patterns, extract meaningful insights, and make predictions or take actions based on that knowledge. These models are designed to learn from examples and observations, identify underlying relationships or trends, and generalize that knowledge to make predictions or decisions on new, unseen data.

There are various types of machine learning algorithms, each with its own characteristics and applications. Supervised learning is one common approach, where models are trained using labeled data, with each example paired with the correct output or target value. The model learns to map inputs to outputs by finding patterns in the labeled data, enabling it to make predictions on new, unlabeled data.

Unsupervised learning, on the other hand, deals with unlabeled data, and the goal is to discover hidden patterns or structures within the dataset. These algorithms aim to identify clusters, associations, or anomalies in the data, providing insights into the underlying structure of the information.

Another approach is reinforcement learning, which involves training an agent to interact with an environment and learn by receiving feedback in the form of rewards or punishments. The agent learns through trial and error, gradually optimizing its actions to maximize the cumulative reward it receives.

Machine learning finds applications across numerous domains. In image and speech recognition, machine learning models can be trained to accurately classify and understand visual or audio data. In natural language processing, machine learning algorithms enable machines to understand and generate human language, powering applications like chatbots and language translation. Machine learning is also widely used in recommendation systems, fraud detection, sentiment analysis, and many other areas where data-driven predictions or decisions are required.

In summary, machine learning is a branch of artificial intelligence that focuses on enabling computers to learn and improve from data without explicit programming. By leveraging algorithms and statistical models, machines can identify patterns, make predictions, and make decisions based on the knowledge acquired from training datasets. Through its various techniques and applications, machine learning has the potential to drive innovation and provide valuable insights across numerous fields.

1.1.2 Deep Learning
Deep learning is a branch of artificial intelligence that focuses on creating algorithms and models inspired by the human brain's neural networks. It involves training complex computational systems called neural networks to recognize patterns

and make intelligent decisions. By using layers of interconnected nodes and large amounts of data, deep learning enables machines to learn and improve their performance over time without explicit programming. It has found applications in various fields, including image and speech recognition, natural language processing, and autonomous driving.

1.1.3 Neural Networks

A neural network [1] [2], also known as an artificial neural network (ANN), is a computational model inspired by the structure and functioning of the human brain's biological neural networks. It is a fundamental component of deep learning and plays a crucial role in various machine learning applications.

At its core, a neural network consists of interconnected nodes, or artificial neurons, organized into layers. These layers are typically categorized into three types: input layer, hidden layers, and output layer. The input layer receives the initial data or input features, while the output layer produces the desired output or prediction. The hidden layers, positioned between the input and output layers, perform complex computations and enable the network to learn and extract meaningful representations from the data.

Each node in a neural network receives inputs, processes them using an activation function [3], and produces an output signal. The activation function determines the node's output based on a weighted sum of its inputs, which are adjusted by values known as weights and biases. The weights represent the strength of the connections between nodes, while biases allow for fine-tuning the node's response.

During the training process, the neural network learns to adjust the weights and biases iteratively to minimize the difference between its predictions and the desired outputs. This optimization is typically achieved using algorithms like backpropagation, which calculate the gradients of the network's error with respect to its weights and biases, and update them accordingly.

By stacking multiple layers and nodes, neural networks can learn hierarchical representations of data. Lower layers capture simple features, while higher layers extract more abstract and complex patterns. This ability to automatically learn and represent data in multiple levels of abstraction is one of the key strengths of neural networks.

Neural networks are capable of solving a wide range of tasks, including classification, regression, pattern recognition, and sequence generation. Different types of neural networks have been developed for specific tasks, such as convolutional neural networks (CNNs) for image analysis, recurrent neural networks (RNNs) for sequential data, and generative adversarial networks (GANs) for generating new content.

The success of neural networks stems from their ability to learn from data, generalize to unseen examples, and make predictions or decisions without being explicitly programmed. With the availability of large datasets and advances in computational power, neural networks have become increasingly powerful tools for solving complex problems in various fields, driving advancements in artificial intelligence and machine learning.

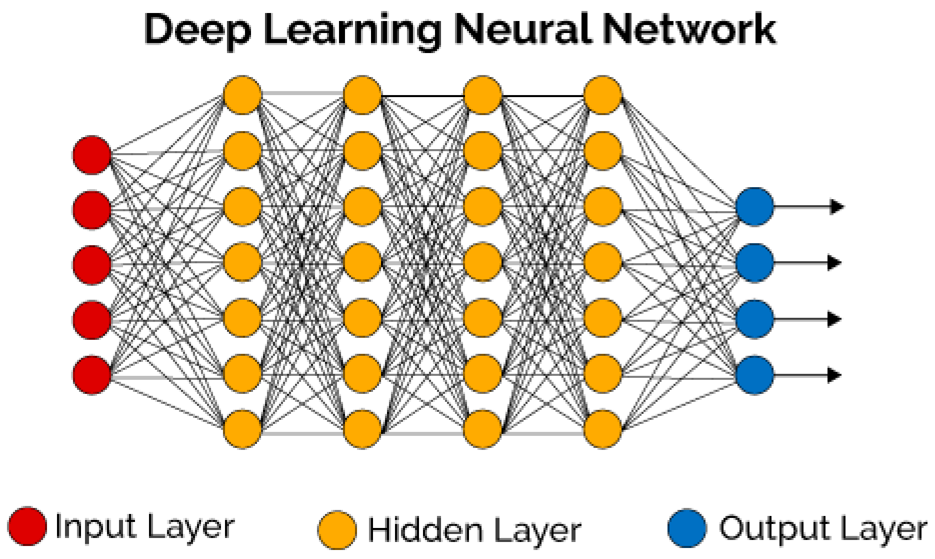


Image 1: Deep Learning Neural Network Example

1.2 Visual Design Tools

In computer science and not only, many and various software exist to design and implement different system using purely visual methods. These methods can greatly vary depending on the task to be solved and the field that the specific software is made for. These tools, usually make heavy use of the user interface, along with mouse and keyboard shortcuts. Some of them can also use touch screen or voice commands for the design part. Various tools will be mentioned and although they maybe not seem connected with each other, all these tools use visual methods to design and implement systems, assets, etc.

1.2.1 UML

The unified modeling language (UML) [4] is a general-purpose modeling language that is intended to provide a standard way to visualize the design of a system.

UML, although not a software, it is used extensively across computer science to design systems, both in paper and in computer using specific software.

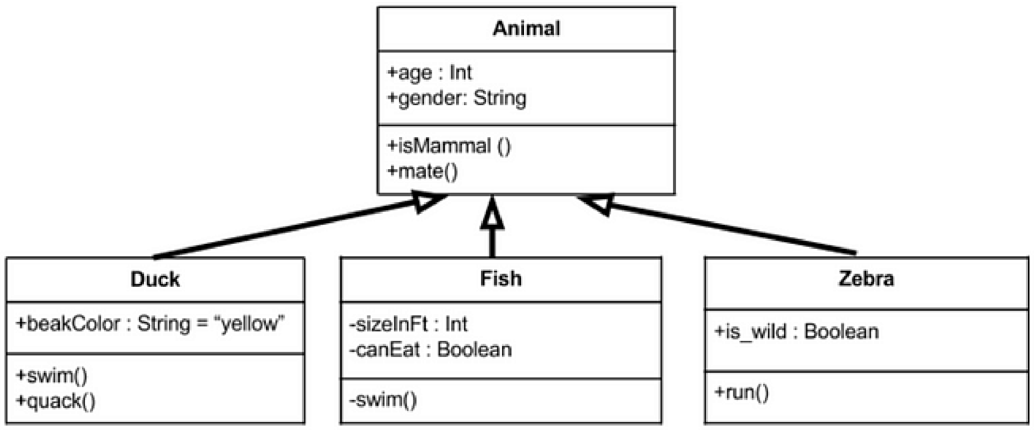


Image 2: UML example [5]

1.2.2 2D/3D Image editing and modelling tools

Something completely different than the mentioned UML, is software that is used in both 2D image editing and 3D modelling. Some known tools for image processing are Microsoft Paint, Gimp, Adobe Photoshop and for 3D modelling, Blender, 3ds max or AutoCAD. All these tools, allow the manipulation and the implementation of various things, either that be images or 3D models using purely visual methods.

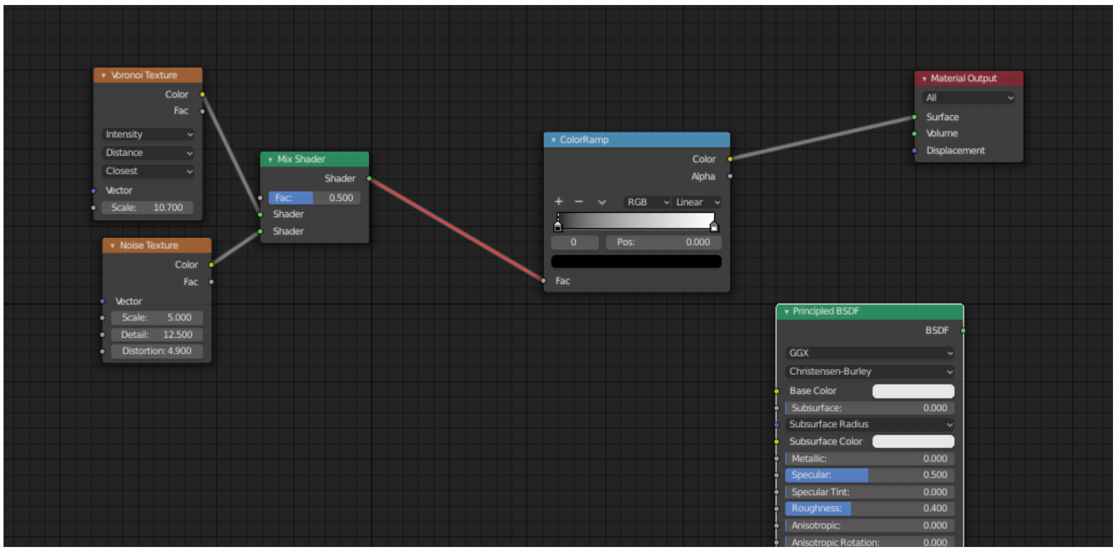


Image 3:Blender Shader Editor

1.2.3 Sound design tools

Even for sound there are tools to help with the design of it. These tools usually show to the user the waves of a sound and the user can manipulate it using the mouse to alter it or add effects to it.

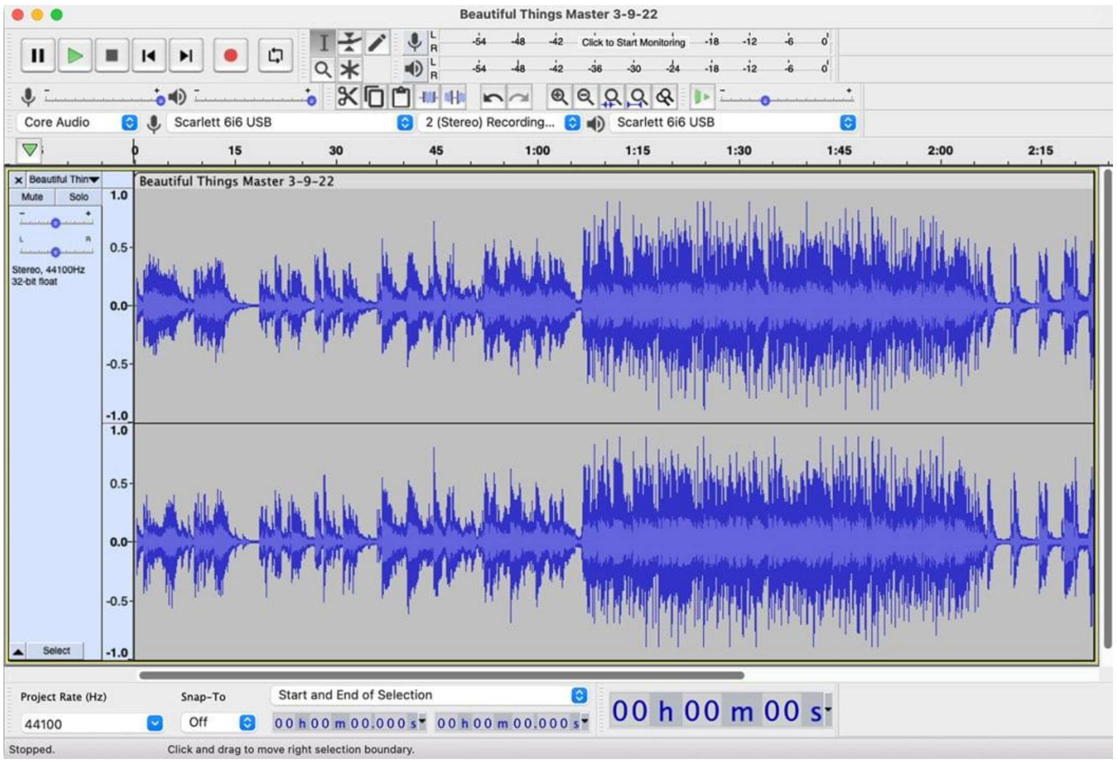


Image 4:Audacity Audio Editor

1.2.4 Game engines

Game engines is one of the best examples of tools where the user designs something using purely visual aspects of the software. The most common example is the level design tools inside game engines, where the user designs the 3D or 2D levels of the game. However, many game engines, have extra “sub” tools for the creation of other aspects of the game as well. For example one of these sub tools, are “Quest” design tools, with which the user usually drags and drops nodes to create tree/path based diagrams, which is then is translated to game code. Other game engines, such as the Unreal Engine have node based systems to even design actual code.

1.2.5 Database/Code Design

Another area where visual design tools are used are database tools and specific code IDEs. For databases, there are plenty development environments where the user can design using diagrams and drag and drop mechanisms to design table relationships and then automatically the software can generate the needed scripts. A couple of examples are SQL Server Management Studio (SSMS) or MySQL Workbench. For code it is not very common to have visual design tools, but there are examples either in the form of game for educational purposes and others.

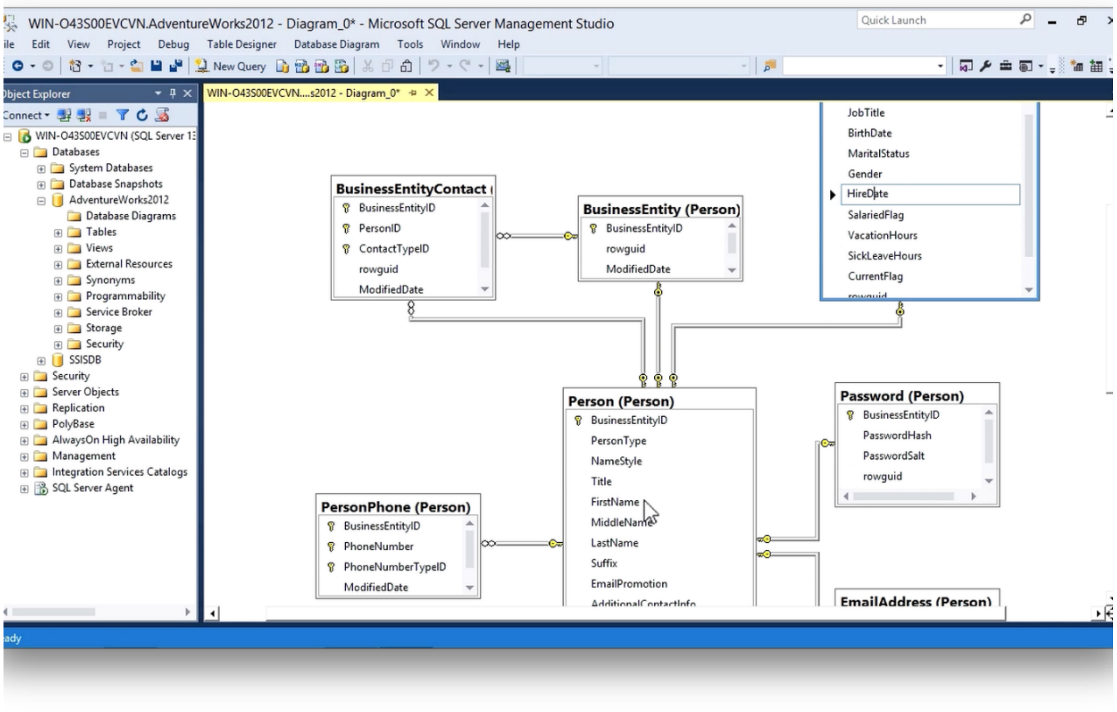


Image 5: SQL Server Management Studio Diagram Mode

1.3 Game Engines

As mentioned above, game engines [6] is probably one of the best examples that make use of custom tools to design various sub systems of the game using visual design, that being from a game level, to the sound system of a game or actual code using node based editors. Also since the tool that was created as part of this thesis is created using the Unity engine, a short mention of the 3 most popular game engines will follow.

A gaming engine is a software development environment, also referred to as a “game architecture” or “game framework,” with settings and configurations that optimize and simplify the development of video games across a variety of programming languages. A gaming engine may include a 2D or 3D graphics rendering engine that’s compatible with different import formats, a physics engine that simulates real-world activities, artificial intelligence (AI) that automatically responds to the player’s actions, a sound engine that controls sound effects, an animation engine, and a host of other feature.

1.3.1 Unity

Unity [7] [8] is a powerful and popular cross-platform game engine and development tool used to create video games, simulations, and interactive experiences. It provides a comprehensive set of tools and features that

Image 6:Unity Logo

enable developers to build games for various platforms, including mobile devices, computers, consoles, and even virtual reality (VR) and augmented reality (AR) devices.

Unity offers a visual editor that allows developers to create and manipulate game objects, design levels, set up scenes, and implement game mechanics without the need for extensive coding. However, it also provides a robust scripting API in C# (C sharp) for more advanced programming and customization.

Key features of Unity include:

1. Cross-platform development: Unity allows developers to create games for multiple platforms using a single codebase, saving time and effort in porting games to different devices.
2. Visual editor: Unity's intuitive visual editor allows designers and artists to create and edit game assets, scenes, and user interfaces without extensive programming knowledge.
3. Asset Store: Unity has a vast Asset Store where developers can find a wide range of ready-to-use assets, such as 3D models, textures, audio clips, and scripts, to enhance their projects.
4. Scripting in C#: Unity uses C# as its primary scripting language, which provides a powerful and widely supported programming framework for implementing game logic and behavior.
5. Physics and animation systems: Unity includes built-in physics and animation systems that simplify the creation of realistic movements, collisions, and interactions within the game world.
6. Multiplayer and networking support: Unity offers features and tools for developing multiplayer games, including networking capabilities for both local and online multiplayer experiences.
7. 2D and 3D capabilities: Unity supports both 2D and 3D game development, allowing developers to create a wide range of game genres and styles.
8. Extensibility and integration: Unity can be extended through custom tools, plugins, and third-party integrations, enabling developers to incorporate additional functionality or connect with external services and APIs.

Unity has a large and active community of developers who share knowledge, tutorials, and resources, making it a popular choice for both indie developers and larger game studios. It has been used to create numerous successful games and is known for its versatility, performance, and ease of use

Image 7: Unity Tech Demo "Enemies"

1.3.2 Unreal Engine

Unreal Engine [9] is a powerful and widely used game engine developed by Epic Games. It provides a comprehensive suite of tools and features for creating high-quality video games, simulations, and interactive experiences. Unreal Engine is renowned for its advanced graphics capabilities, versatility, and flexibility.

Here are some key features of Unreal Engine:

1. Real-time rendering: Unreal Engine excels in rendering realistic and immersive visuals in real-time, making it suitable for creating visually stunning games and experiences.
2. Blueprint visual scripting: Unreal Engine offers a node-based visual scripting system called Blueprint, which allows developers and designers to create game logic, behavior, and interactions without extensive coding. It is a beginner-friendly approach that simplifies the development process.
3. C++ programming: Unreal Engine provides a powerful and robust programming framework in C++, allowing developers to write code for more complex systems and optimize performance.
4. High-fidelity graphics: Unreal Engine supports advanced rendering techniques, including dynamic lighting, physically based materials, global illumination, and post-processing effects, enabling developers to achieve highly realistic visuals.
5. Animation and cinematics: Unreal Engine offers a comprehensive animation system that allows developers to create complex character animations, cinematic sequences, and cutscenes. It supports skeletal animation, inverse kinematics, and facial animation.

6. Blueprints visual scripting: Unreal Engine offers a node-based visual scripting system called Blueprints, which allows developers and designers to create game logic, behavior, and interactions without extensive coding. It is a beginner-friendly approach that simplifies the development process.
7. Physics simulation: Unreal Engine includes a robust physics simulation system that enables realistic object interactions, rigid body dynamics, cloth simulation, and destruction effects.
8. Virtual Reality (VR) and Augmented Reality (AR) support: Unreal Engine provides built-in support for developing VR and AR experiences, making it a popular choice for creating immersive virtual reality games and applications.
9. Multiplatform development: Unreal Engine supports multiple platforms, including PC, consoles, mobile devices, and VR/AR devices, allowing developers to target a wide range of platforms with a single codebase.
10. Marketplace and community: Unreal Engine has a thriving marketplace where developers can access a vast library of assets, plugins, and tools to enhance their projects. The community surrounding Unreal Engine is active and supportive, providing tutorials, documentation, and forums for knowledge sharing.

Image 8: Unreal Engine 5 showcase

Unreal Engine has been used to create numerous critically acclaimed and commercially successful games, ranging from small indie titles to large-scale productions. It offers a comprehensive set of tools and features that empower developers to bring their creative visions to life. The latest version of the engine is Unreal Engine 5 as of 2023.

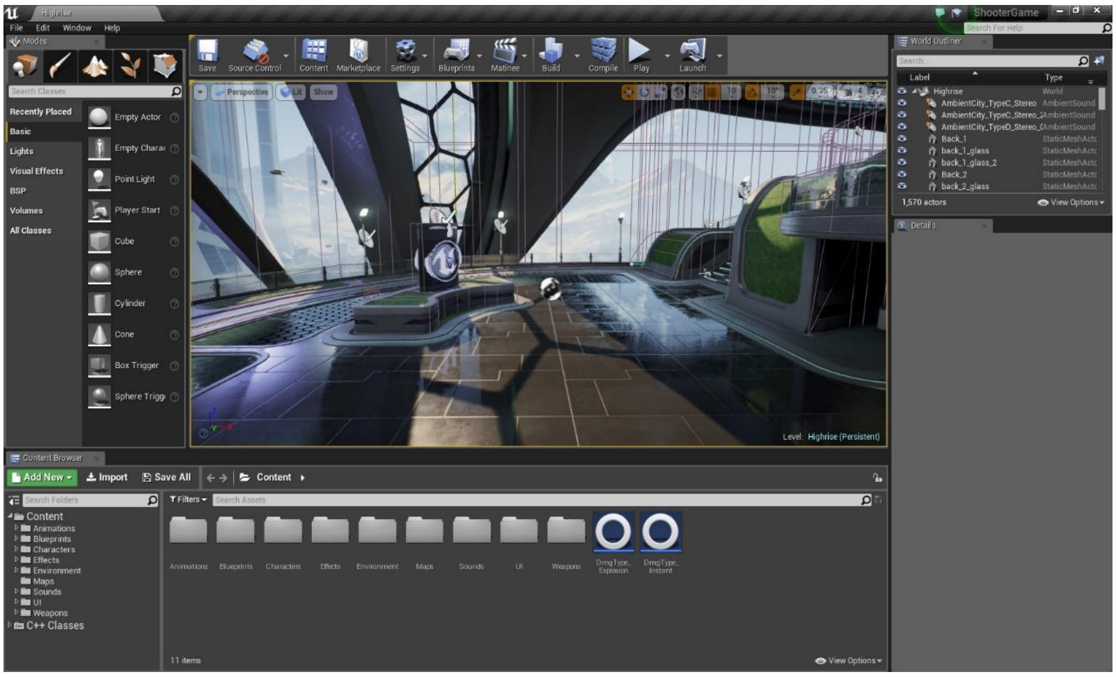


Image 9: Unreal Engine 5 user interface

Unreal Engine Design Tools

Unreal Engine is probably the best example of a tool that contains various visual design sub tools for almost everything. There is the main blueprint design tool that the user can use to design actual code using a node based editor. These nodes are then translated into the native game engine language C++.

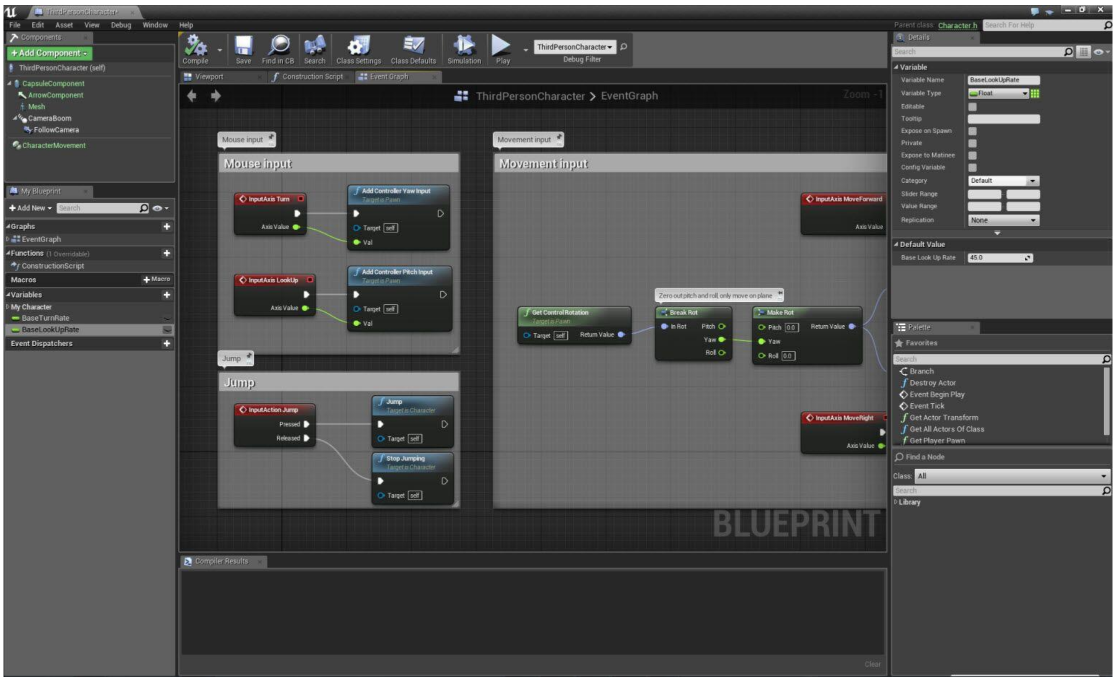


Image 10: Unreal Engine 5 Blueprint editor

Another common sub tool is material/shader editor where the user can design materials and shaders which are then translated into graphic shader code which is used in 3D design.

Audio editor is another sub tool, which is used to design various sound systems and connect sounds to create complex sequences or add effects to existing ones.

There are also tools for animations and user interface design, where the user can design animations for the 3D models and user interface sub tool can be used to design the whole for the game.

To conclude these are only some of the tools that Unreal Engine has to help with various systems. This makes design for many amateur and inexperienced users much easier, without the need of technical knowledge.

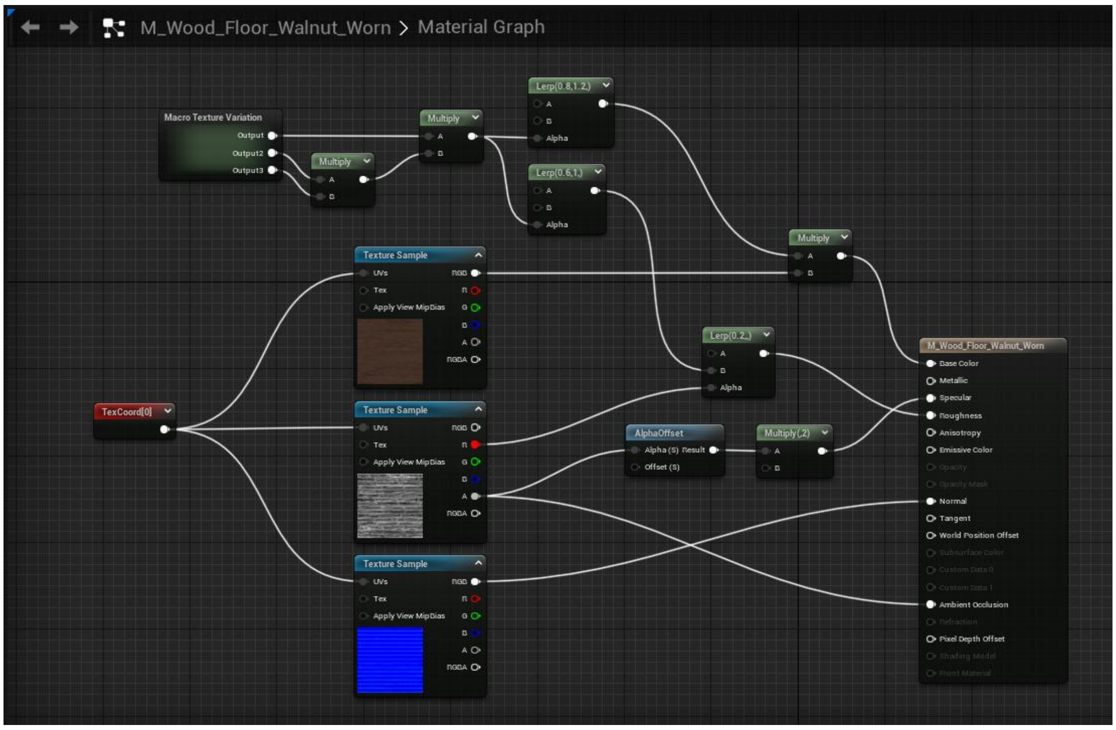


Image 11:Unreal Engine 5 Shader editor

1.4 Visual Scripting tools

As mentioned above, there are many examples of tools that make use of a visual way to design systems in many and various fields. Visual scripting tools are tools focused on code such as the Unreal Engine Blueprint Editor that was mentioned before.

Visual scripting tools are beneficial for several reasons, especially in the context of programming and game development:

1. Accessibility: Visual scripting tools lower the barrier to entry for individuals who are not proficient in traditional coding languages. They enable artists, designers, and non-programmers to create complex logic and functionality without the need to write code manually. This accessibility democratizes the

development process and encourages more people to participate in creating interactive experiences.

2. Rapid Prototyping: Since the process is more visual and intuitive, designers and developers can quickly experiment with different ideas and see the results in real-time. This iterative approach speeds up the development cycle and helps identify potential design flaws or improvements early on.
3. No Syntax Errors: Visual scripting tools eliminate syntax-related errors commonly found in traditional coding. Since the logic and structure are visually represented, the chances of making syntax mistakes are significantly reduced. This leads to fewer bugs and a more reliable development process.
4. Collaboration: They facilitate collaboration between programmers and other team members. Artists, designers, and content creators can use visual scripting to implement their ideas directly, reducing the need for constant back-and-forth with programmers. This collaborative environment fosters efficient teamwork and enables a seamless integration of various elements in a project.
5. Educational Tool: They server as excellent educational tools, especially for beginners learning programming concepts. It provides a more tangible and visual representation of code execution, helping newcomers grasp fundamental programming principles without feeling overwhelmed by syntax and code structures.
6. Flexibility and Experimentation: Visual scripting tools often come with a wide range of pre-built components and functions that users can combine creatively. This flexibility allows developers to experiment and explore different solutions, leading to innovative and unique approaches to problem-solving.
7. Game Development: In the game development industry, visual scripting is particularly useful for creating gameplay mechanics, AI behaviors, event handling, and other interactive elements. Game designers can visually map out complex game logic and interactions, empowering them to create captivating and interactive game experiences.
8. Code Comprehension: Visual scripting can also help programmers better understand complex codebases. By visually representing the underlying code's logic, it becomes easier for developers to follow the flow and relationships between different components, making maintenance and debugging more manageable.

Overall, visual scripting tools are valuable because they make programming more accessible to a broader audience, enable rapid prototyping, improve collaboration, reduce errors, and enhance the creative potential of developers and designers alike. They serve as powerful aids in both the learning process and the development of complex interactive experiences.

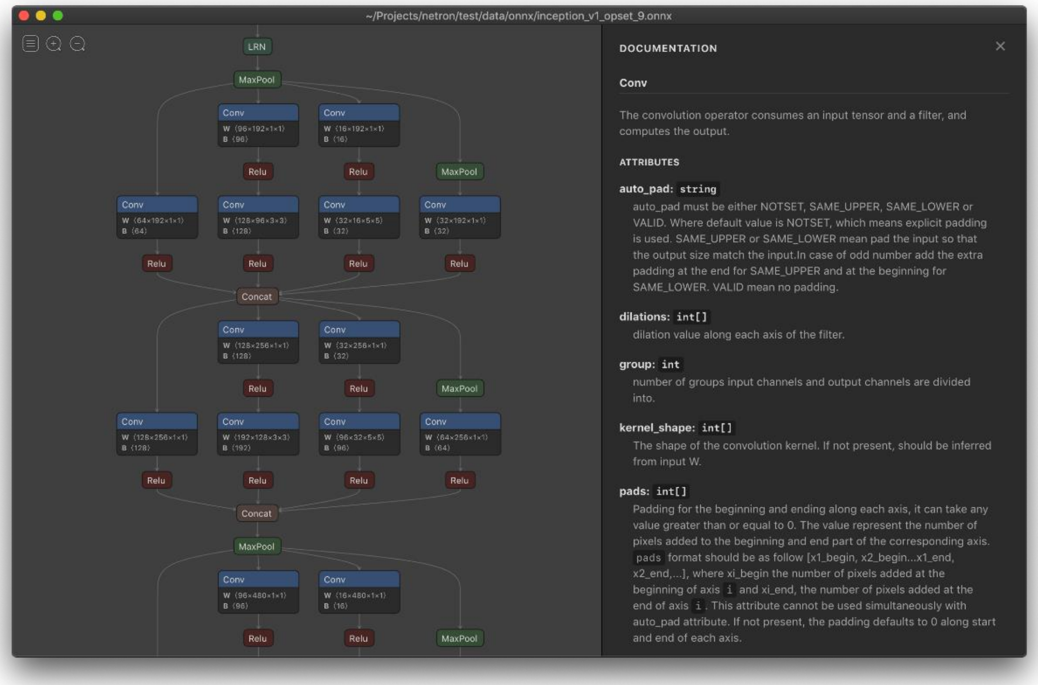
1.5 Current State

The purpose of this thesis is to create a tool for the design of various machine learning systems. Some similar tools are described below to show the current state on the visual design tools.

1.5.1 Netron

Netron [10] is an open-source tool for visualizing neural network models. It supports various deep learning frameworks, such as TensorFlow, PyTorch, ONNX, and Caffe. Netron allows you to load trained models and explore their architecture, inspect layers, and view inputs and outputs. It provides an intuitive interface for understanding the structure and flow of information in neural networks.

Although it is not a tool to design neural network models, it helps with the visualization of them which is very similar, yet only a part of the design process.



DOCUMENTATION

Conv

The convolution operator consumes an input tensor and a filter, and computes the output.

ATTRIBUTES

auto_pad: `string`
auto_pad must be either NOTSET, SAME_UPPER, SAME_LOWER or VALID. Where default value is NOTSET, which means explicit padding is used. SAME_UPPER or SAME_LOWER mean pad the input so that the output size match the input.In case of odd number add the extra padding at the end for SAME_UPPER and at the beginning for SAME_LOWER. VALID mean no padding.

dilations: `int[]`
dilation value along each axis of the filter.

group: `int`
number of groups input channels and output channels are divided into.

kernel_shape: `int[]`
The shape of the convolution kernel. If not present, should be inferred from input W.

pads: `int[]`
Padding for the beginning and ending along each axis, it can take any value greater than or equal to 0. The value represent the number of pixels added to the beginning and end part of the corresponding axis. `pads` format should be as follow [x1_begin, x2_begin...x1_end, x2_end,...], where xi_begin the number of pixels added at the beginning of axis `i` and xi_end, the number of pixels added at the end of axis `i`. This attribute cannot be used simultaneously with auto_pad attribute. If not present, the padding defaults to 0 along start and end of each axis.

Image 12:Netron visualization mode

1.5.2 TensorSpace

TensorSpace [11] is a JavaScript library that focuses on 3D visualizations of neural network models. It allows you to create interactive 3D representations of models and their layers, providing a unique perspective for visualizing and understanding neural networks. TensorSpace integrates with popular deep learning libraries like TensorFlow.js and Keras.

Again similar to Netron, TensorSpace is used for the visualization part but goes one step further by visualizing the networks in 3D space. It also provides various realtime examples.

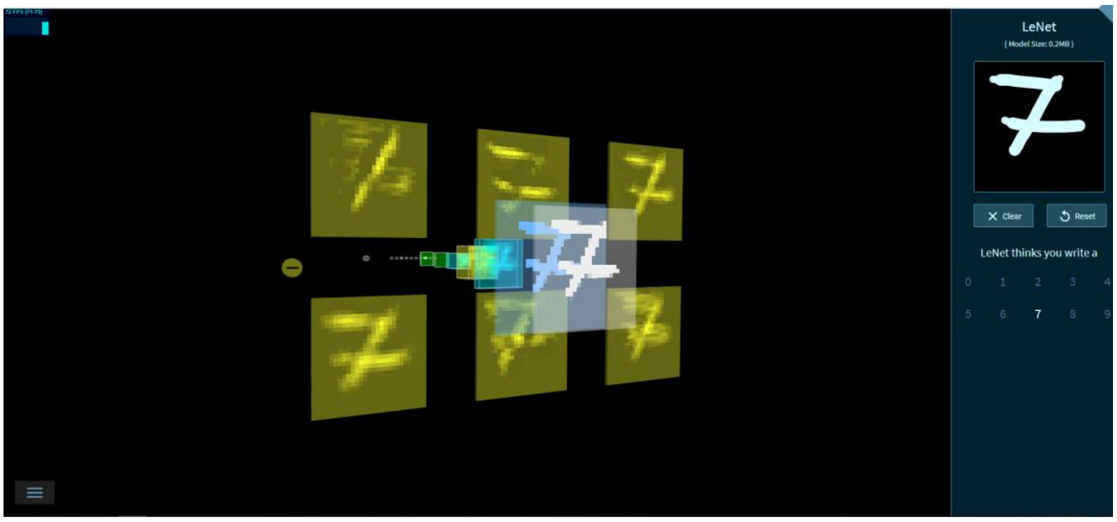


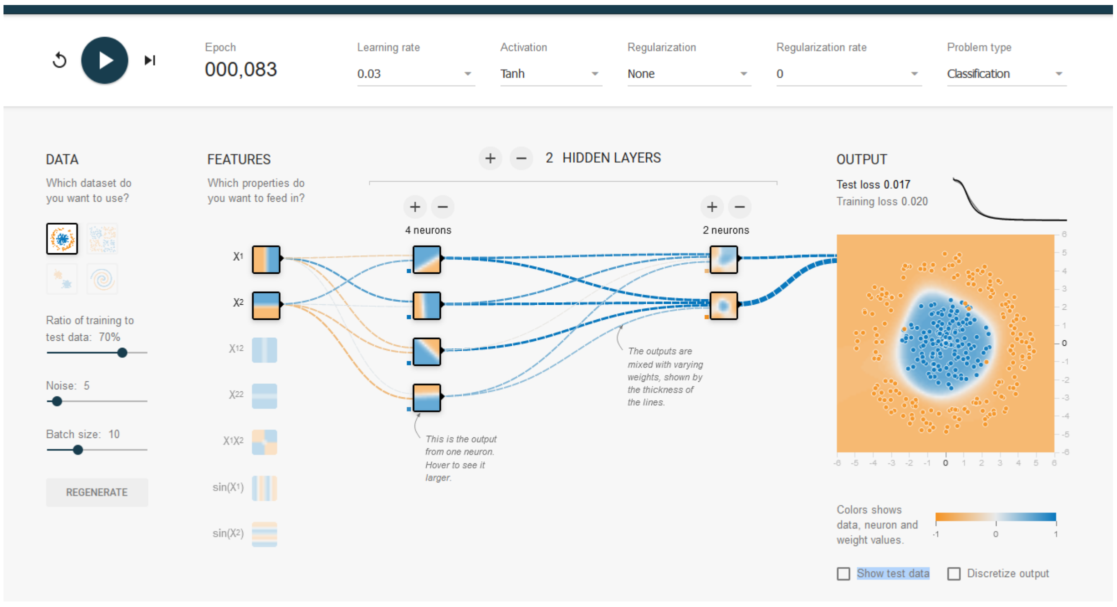
Image 13: TensorSpace playground number draw example

1.5.3 Gephi

Although not specifically designed for neural networks, Gephi [12] is a powerful graph visualization tool that can be useful for visualizing network architectures and their connections. It allows you to create interactive graph visualizations, analyze network properties, and explore complex relationships within neural networks.

1.5.4 TensorFlow Playground

TensorFlow Playground [13] is a web-based interactive visualization tool provided by Google that allows users to experiment with various neural network architectures and hyperparameters using a simple GUI. It's great for understanding how different configurations affect the behavior of neural networks in real-time.


Epoch 000,083
Learning rate 0.03
Activation Tanh
Regularization None
Regularization rate 0
Problem type Classification

DATA
Which dataset do you want to use?
Ratio of training to test data: 70%
Noise: 5
Batch size: 10
REGENERATE

FEATURES
Which properties do you want to feed in?

2 HIDDEN LAYERS
4 neurons
2 neurons

OUTPUT
Test loss 0.017
Training loss 0.020

This is the output from one neuron. Hover to see it larger.

The outputs are mixed with varying weights, shown by the thickness of the lines.

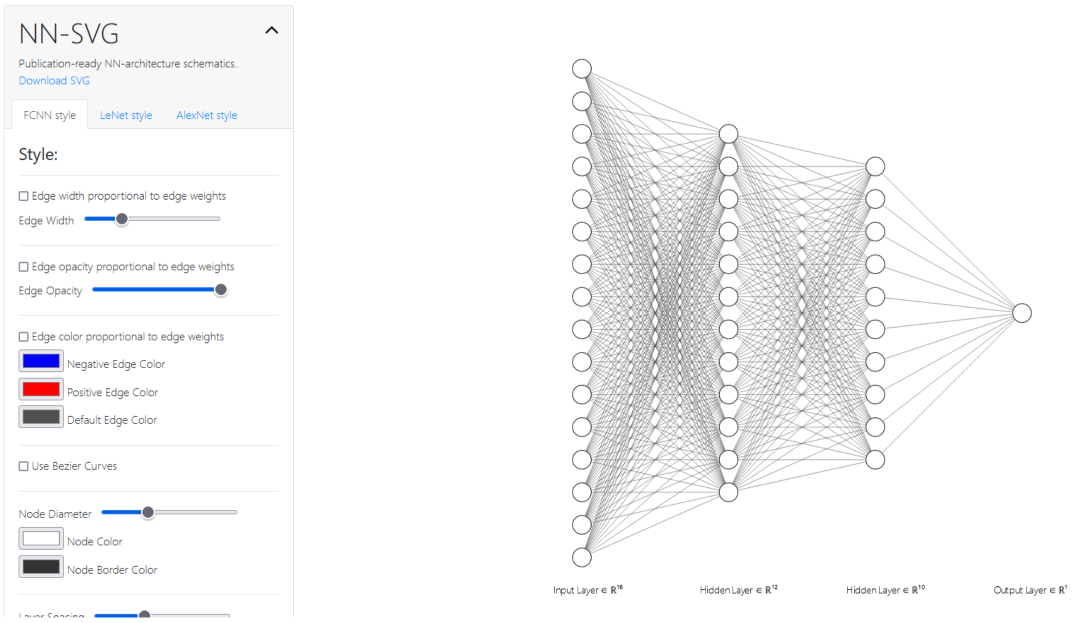
Colors shows data, neuron and weight values.

Show test data
Discretize output

Image 14: TensorFlow Playerground interface

1.5.5 NN-SVG

NN-SVG [14] is an online tool that helps visualize neural network architectures by generating Scalable Vector Graphics (SVG) representations of the models. Users can customize the layout and design to create informative visualizations. Again, although just a visualization tool, the result if very helpful to understand the basic structure of a neural network and modify the visualization of it such as the edge width.



NN-SVG

Publication-ready NN-architecture schematics.

Download SVG

FCNN style | LeNet style | AlexNet style

Style:

☐ Edge width proportional to edge weights

Edge Width

☐ Edge opacity proportional to edge weights

Edge Opacity

☐ Edge color proportional to edge weights

🟦 Negative Edge Color

🟥 Positive Edge Color

⬛ Default Edge Color

☐ Use Bezier Curves

Node Diameter

⬜ Node Color

⬛ Node Border Color

Image 15: NN-SVG Interface

Chapter 2 Unity Engine

In the first chapter Unity game engine was mentioned. Unity, is the game engine that was selected for the creation of the software prototype. In this chapter, certain Unity mechanisms and terms will be described in more detail.

2.1 General Terms

To begin with, some basic terms will be described that are important and useful across all game engines.

2.1.1 3D Model

Image 16: Retro phone 3D model

A 3D model is a digital representation of a three-dimensional object or scene. It is created using specialized software or 3D modeling tools, which allow users to construct and manipulate objects in a virtual space. 3D models can be highly detailed and realistic, capturing the shape, texture, and appearance of real-world objects or imaginary creations.

In a 3D model, objects are typically represented using polygons, which are flat surfaces connected by edges and vertices. These polygons are combined to form the overall shape of the model. Additional details, such as textures, colors, and lighting, can be applied to the model to enhance its realism. 3D models are usually created using specialized software such as 3ds Max, Blender, Maya and others.

2.1.2 Texture

In 3D modeling, texture refers to the surface characteristics and appearance applied to a 3D model. It is a 2D image that is mapped onto the surface of a 3D object to give it color, pattern, detail, and realism. Textures provide visual information about the material, such as its roughness, smoothness, reflectivity, and other surface properties.

Textures can be created using image-editing software or generated procedurally through algorithms. They can range from simple colors and patterns to highly detailed images that replicate real-world materials like wood, metal, fabric, or skin. Common types of textures include diffuse maps (color and pattern information), specular maps (highlighting areas of reflectivity), normal maps (creating the illusion of surface details), and displacement maps (altering the geometry to create surface irregularities).

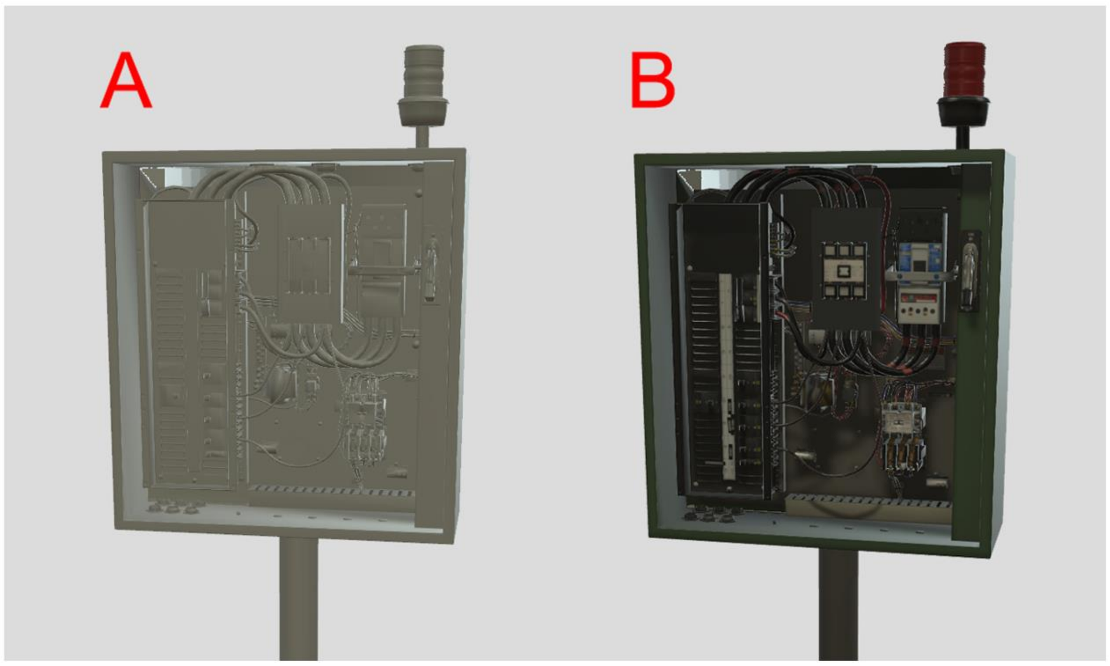


Image 17: 3D model without (A) and with a texture (B)

2.2 Unity Specific Terms

In this paragraph, certain basic and specific to unity term will be explained which are the GameObjects, Components and Scenes.

2.2.1 GameObject

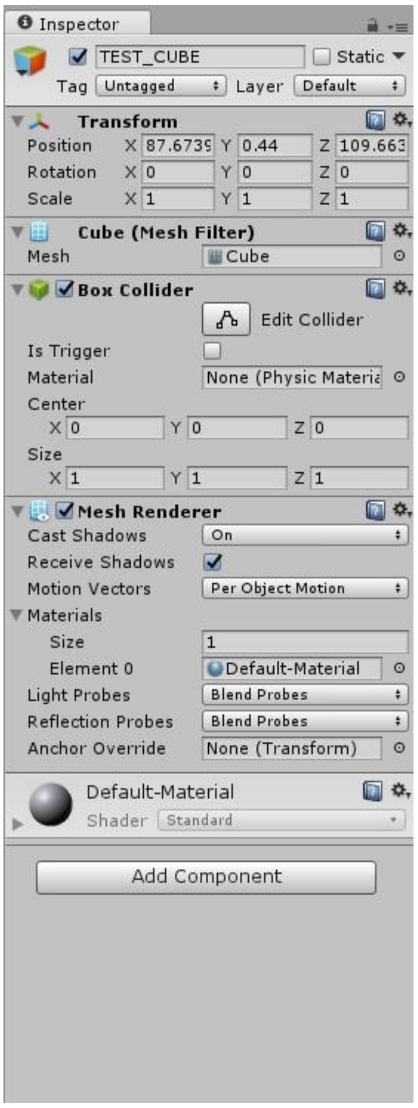
In the Unity game engine, a GameObject is a fundamental building block and the basic unit of any scene or game object hierarchy. It represents an entity or object in the virtual world of a game.

A GameObject is an empty container that can hold components, which define the behavior and functionality of the object. Components are attached to a GameObject to give it properties and capabilities such as rendering, physics, scripting, audio, and more. Examples of components include Mesh Renderer (for visual rendering), Rigidbody (for physics simulation), Collider (for collision detection), and Script (for custom behavior using scripts written in languages like C#).

By attaching different combinations of components to a GameObject, you can define how it looks, behaves, interacts with other objects, and responds to user input or game events.

2.2.2 Components

A GameObject has Components which are essentially classes that implement most of the functionality that is related to the GameObject. Once a GameObject is instantiated it has a default Component called Transform. Transform is a class that contains all the needed information about the 3D position, rotation and scaling of the object.



```
Inspector
TEST_CUBE   Static
Tag Untagged   Layer Default
Transform
Position  X 87.6739  Y 0.44  Z 109.663
Rotation  X 0  Y 0  Z 0
Scale     X 1  Y 1  Z 1
Cube (Mesh Filter)
Mesh  Cube
Box Collider
Edit Collider
Is Trigger
Material  None (Physic Materia
Center
  X 0  Y 0  Z 0
Size
  X 1  Y 1  Z 1
Mesh Renderer
Cast Shadows  On
Receive Shadows  ✓
Motion Vectors  Per Object Motion
Materials
  Size  1
  Element 0  Default-Material
Light Probes  Blend Probes
Reflection Probes  Blend Probes
Anchor Override  None (Transform)
Default-Material
Shader  Standard
Add Component
```

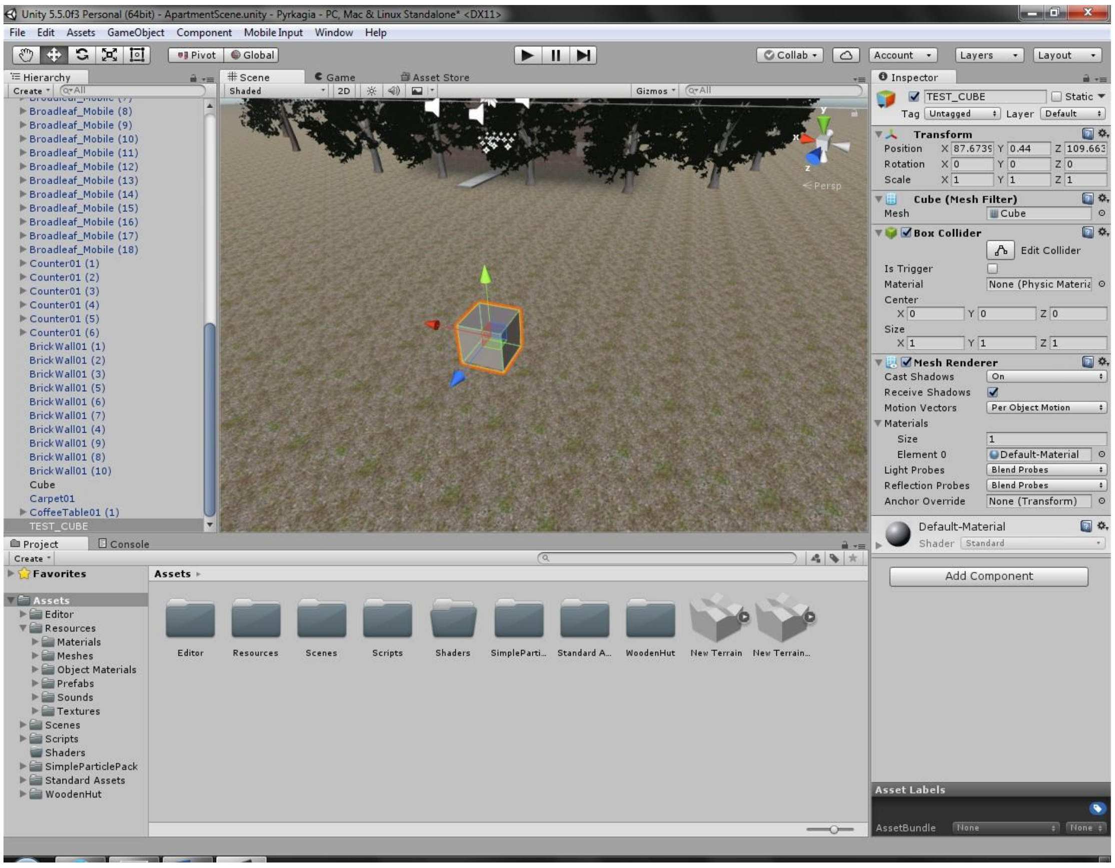
Image 18: Unity Inspector Window

In Image 18, the inspector window is shown. Inspector window is the window that shows all information about a GameObject. In this image, a cube is selected from the current Scene. In the top part of the window certain basic information about the GameObject exist such as Tag and Layer. Below these information, all the components are shows which are collapsible panels. Transform is the component that was explained above. Mesh Filter is a required component to show a 3D model. Afterwards there is the Box Collider component which gives an object the ability to detect collisions. Finally there is the Mesh Renderer component, which is required in order to properly show the selected model in the Mesh Filter component. Another smaller panel can also be noticed which is the 3D object's material and shader.

It is important to note that only public variables are shown and can be modified in the inspector which they can be even altered during game mode. There is also the Debug view in the inspector window where private and protected variables are shown, but cannot be modified.

2.2.3 Scenes

A scene refers to a self-contained environment or level within a game or application. It represents a specific portion or area of your project where you can place and organize various game objects, characters, terrain, lights, and other elements. Scenes in Unity allow you to break down your project into smaller, manageable parts, making it easier to design and develop different parts of your game or application separately. For example, you might have one scene for the main menu, another for the game level, and yet another for a settings screen. Each scene in Unity consists of a collection of GameObjects, which are the fundamental building blocks of your game or application. GameObjects can represent characters, objects, cameras, lights, and more. By placing and configuring these GameObjects within a scene, you define the visual and interactive aspects of your project. Scenes also control the flow and progression of your game. You can transition between scenes to create a seamless experience, such as moving from a menu scene to a gameplay scene or from one level to another.


Unity 5.5.0f3 Personal (64bit) - ApartmentScene.unity - Pyrkagia - PC, Mac & Linux Standalone* <DX11>
File Edit Assets GameObject Component Mobile Input Window Help
Pivot Global
Collab Account Layers Layout
Hierarchy
Scene Game Asset Store
Shaded 2D Gizmos
Inspector
Project Console
Favorites
Assets
Editor Resources Scenes Scripts Shaders SimpleParti... Standard A... WoodenHut New Terrain New Terrain...
Add Component
Asset Labels
AssetBundle None None

TEST_CUBE Static
Tag Untagged Layer Default
Transform
Position X 87.6739 Y 0.44 Z 109.663
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Cube (Mesh Filter)
Mesh Cube
Box Collider
Edit Collider
Is Trigger
Material None (Physic Materia
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1
Mesh Renderer
Cast Shadows On
Receive Shadows ✓
Motion Vectors Per Object Motion
Materials
Size 1
Element 0 Default-Material
Light Probes Blend Probes
Reflection Probes Blend Probes
Anchor Override None (Transform)
Default-Material
Shader Standard

Image 19: Unity scene window

Chapter 3 Thesis Purpose

The purpose of this thesis is to design a system that provides the user to design and implement various machine learning algorithms in a visual design/scripting way. As mentioned in chapter 1.4, visual scripting tools make things a lot easier to the user both inexperienced and experienced and are especially useful to quickly create prototypes even without great knowledge, as no code is needed. Before it is explained how such a system can be designed, it is important to mention what a machine learning algorithm is along with examples of the most common ones.

3.1 Machine Learning Algorithms

Machine learning algorithms are computational algorithms designed to analyze and interpret data, recognize patterns, and make predictions or decisions without explicit programming instructions. These algorithms are a core component of machine learning, a field of artificial intelligence that focuses on developing systems capable of learning and improving from data.

Here are some common machine learning algorithms:

1. Linear Regression: This algorithm is used for regression tasks, where the goal is to predict a continuous output variable based on input features by fitting a linear equation to the data.
2. Logistic Regression: Logistic regression is used for classification tasks, where the goal is to predict discrete class labels based on input features. It models the relationship between the features and the probability of belonging to a particular class.
3. Decision Trees: Decision trees [15] are versatile algorithms that can be used for both classification and regression tasks. They create a tree-like model of decisions based on features to reach the predicted output.
4. Random Forest: Random forest is an ensemble learning method that combines multiple decision trees. It improves accuracy and reduces overfitting by aggregating predictions from individual trees.
5. Support Vector Machines (SVM): SVM is a popular algorithm for classification tasks. It finds an optimal hyperplane that separates different classes with the maximum margin.
6. K-Nearest Neighbors (KNN): KNN is a simple algorithm that classifies new data points based on their proximity to labeled data points. It assigns the class label based on the majority vote of its k nearest neighbors.
7. Naive Bayes: Naive Bayes is a probabilistic classifier based on Bayes' theorem. It assumes that the features are conditionally independent, simplifying the computation of probabilities.
8. Neural Networks: Neural networks are a class of algorithms inspired by the structure and function of biological brains. They consist of interconnected nodes or "neurons" organized in layers and are capable of learning complex patterns and relationships in data.
9. Gradient Boosting Algorithms: Algorithms like AdaBoost, Gradient Boosting Machines (GBM), and XGBoost build an ensemble of weak prediction models, combining their predictions to produce a stronger model.

10. Clustering Algorithms: Clustering algorithms, such as K-Means, Hierarchical Clustering, and DBSCAN, group similar data points together based on their characteristics or distances.

3.2 Design

The design of a system is important because it determines the system's architecture, functionality, performance, reliability, scalability, and user experience. A well-designed system is organized, efficient, robust, user-friendly, and easier to maintain and scale. It plays a crucial role in achieving optimal performance, meeting functional requirements, and ensuring a positive user experience.

3.2.1 Modules

Designing a system that does more than one thing often leads to cramped systems that are either not properly scalable or not very user friendly. To solve this, the system can be splitted into modules, where each module will be the “designer” of a specific machine learning algorithm. The modules will not affect each other and should be purely distinguishable from each other. However, each module should have the ability to cooperate with other modules on an “upper” level designer.

Each module being separate can give a clean architecture and also scalability so that the modules can be added in the future or adjusted separately without breaking each other.

3.2.2 Module Designer

Designer is the most important tool in a visual scripting/design system, as it provides the user with an interface to design whatever he wants. A designer is consisted of three parts, the visualization of the system that is to be designed e.g. nodes, lines , the main interface part where the user can create and modify the system and the code conversion part that converts whatever the user has designed into actual code or script so that it can be executed somewhere.

3.2.3 Module Designer Visualization

As mentioned, visualization is fundamentally how a system is shown to the user graphically such as nodes, lines, graphs etc. Visualization obviously, should be different for each module, as each machine algorithm has different requirements and components, so the visualization should be different. Although in most similar software visualization is done in 2D, it would be best for the system to make use of both a 2D perspective and a 3D one. 2D visualization would help getting a better idea of the module architecture/design, while the 3D view would be great to get a better view and understanding of certain machine learning algorithms, including using the extra Z dimension to better organize the module's items.

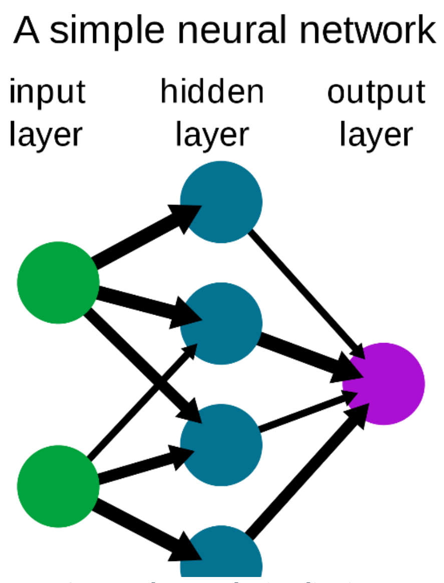


Image 20: Neural Network Visualization

Some examples for the visualization of machine learning algorithms follow.

Neural Network Visualization

For example a neural network could be visualized as the following image. Neural networks are consisted of an input layer, hidden layer and output layer, along with their connections. All these layers should be visualized to the user. Moreover the interface should also include more information such as the weights, activation function and more detailed nodes e.g. instead of a hidden layer it could show all hidden layers and the neurons of them.

Decision Tree Visualization

Decision trees are another common machine learning algorithm and the visualization of it could be very similar to the neural network one. Essentially both are graphs so the visualization part is very similar, which are nodes connected with lines. However they function differently so not all things are going to be the same. Neural networks are more complicated and so they should contain more information.

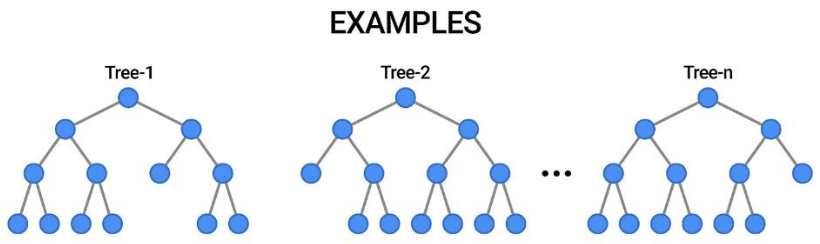


Image 21: Decision Trees Visualization

K-Nearest Neighbors (KNN)

As both neural networks and decision trees are visualized in a similar way, another machine learning algorithm K-Nearest Neighbors, is a good example of how a module can be very different from another. KNN is visualized using a 2D plot that contain usually colored dots or different symbols to distinguish each category.

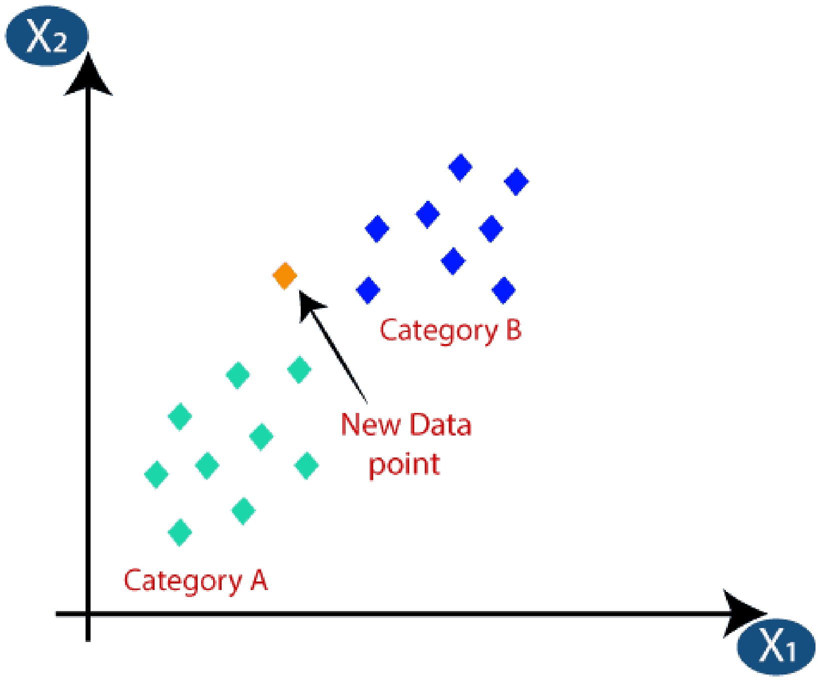


Image 22: KNN Visualization

3.2.4 Module Designer Interface

As stated, a designer is consisted of a “design” interface that is used to actually design the system. Again this varies depending on the system that is to be designed as every machine learning algorithm has different requirements.

Neural Network Designer Interface

A designer interface for a neural network for example could contain input panels or a form of wizard to create a neural network and adjust parameters such as the activation function, amount of nodes on each layer, which nodes are connected with each other, weights, etc. Also a set of buttons can control the visibility of the network, such as how the nodes are shown or a button to toggle a "higher level" view so that hidden layers are not shown in detailed.

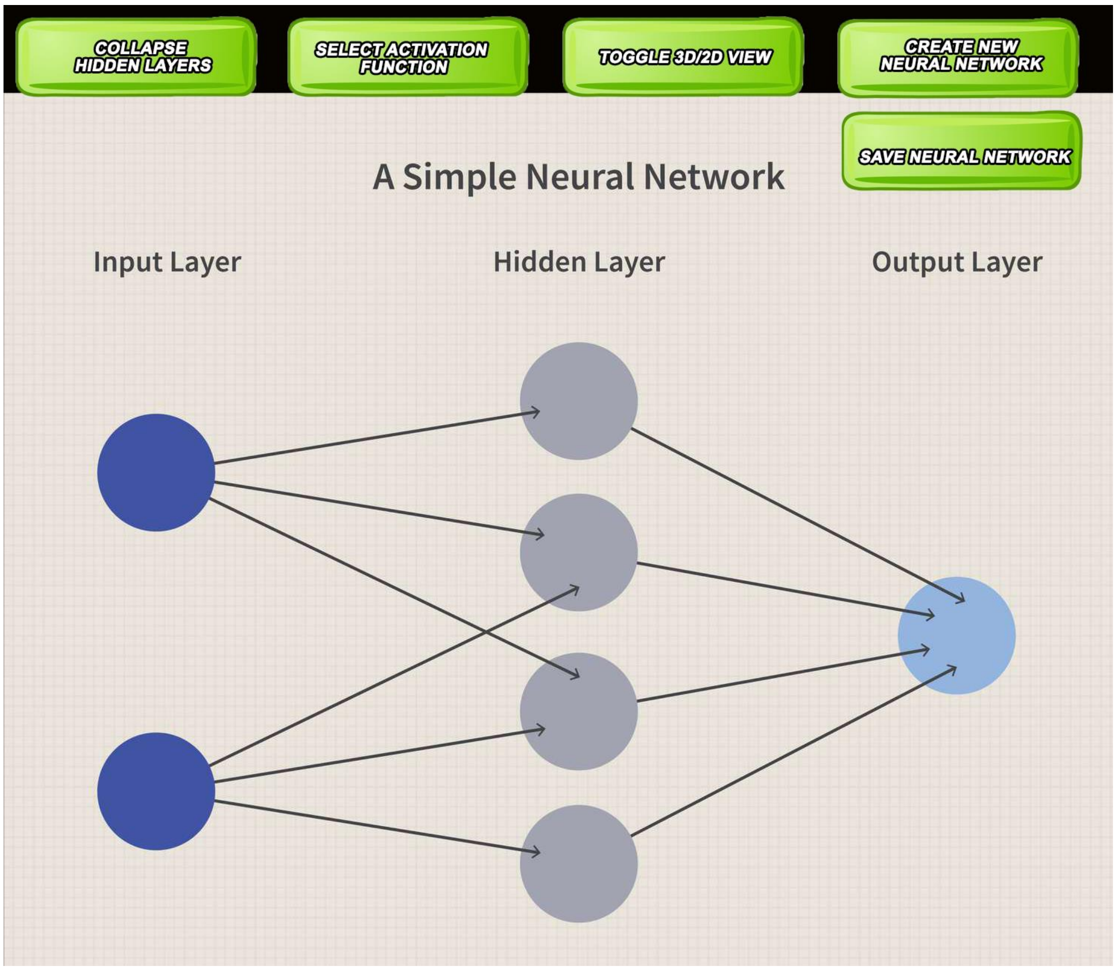


Image 23: Neural Network Designer Interface Mock

3.2.5 Module Designer Code Conversion

The final part of the module designer is the code conversion part. The idea behind it, is that once the user designs the wanted system using the interface described

above, the system will then be converted into actual code or some sorts of scripting language that can be run in a software. As mentioned in all the previous parts, this part as well highly depends on the machine learning algorithm in question. As the neural network was described in more detailed, an example will be given for it.

For the neural network, each node that is shown in the user should be a separate classes which could help in the visualization of it, or for performance reasons could be arrays. At first, once the user creates a network, the network will be visualized. In order to be visualized the code behind it needs to be created first so it can be the classes or arrays mentioned above. In more detail a class could be a Neuron or a

Synapse along with the needed properties, but this is that if the system is design using an OOP architecture.

3.2.6 Upper Level Designer

As pointed it out earlier, each module should be separate so that it does not affect other modules. Although modules are independent and should work separately for each machine learning algorithm, an upper level designer can exist to combine them. So the upper level designer in essence is a way for the user to design a larger scale system by combining each module he created with each other. For example a neural network output could feed the input nodes of another neural network. It is only logical, that not all modules can be combined with all other modules, so it is very important to have certain interfaces for each module. These interfaces can determine which module can be connected with other modules.

Converters

Converters are a special type of node in the designer that can be used to convert data from one form to another. This is very useful especially since most data require some sort of manipulation in order for the modules to be connected.

Operators

Operators are another type of special node in the designer. Their purpose is to apply various operations to data from the output of a module. Operations could vary from simple addition or subtraction to complex math functions.

The following image shows how the upper level designer can look.

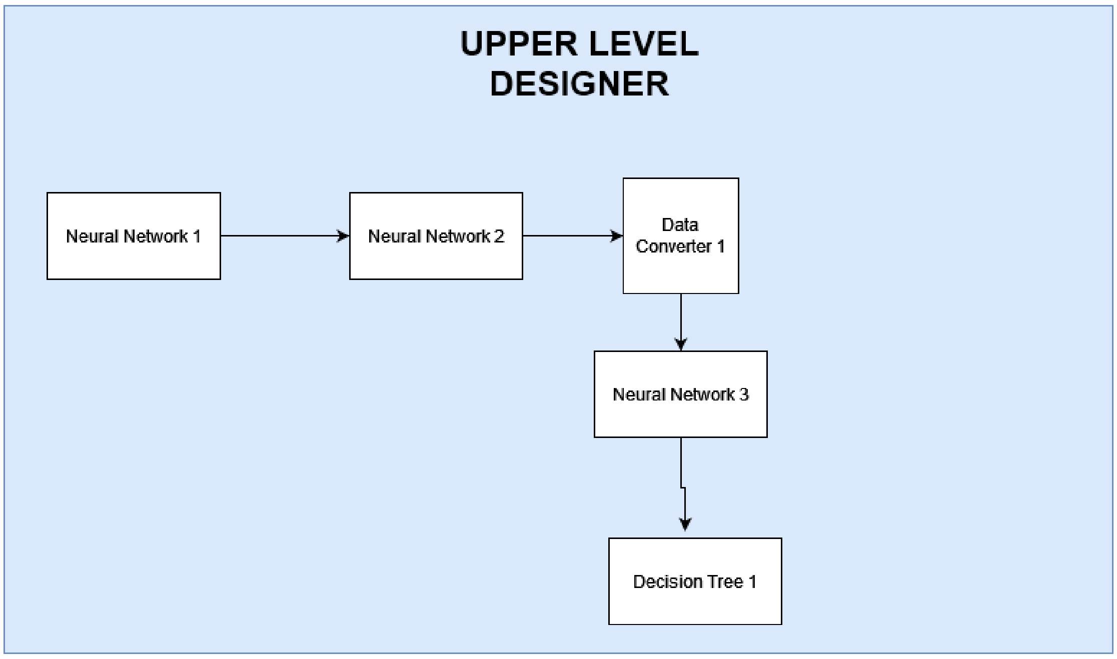


Image 24: Upper Level Designer

In the above image, an example is shown where Neural Network 1 connects and feeds its output to Neural Network 2. Afterwards Neural Network 2 passes its output through a data converter. Data converter then converts the data into a suitable form of data to be then fed as input in Neural Network 3. Finally Neural Network 3 output is connected to a decision tree.

Interface

The user interface of the designer should contain some sort of module selection menu in the form of a dropdown list for example. The user should also be able to connect and move modules around.

Another function the system could have is to select a node of the system and then "zoom in" to go into details of it. By zooming in the node could then be converted in real time into the detailed view of the selected node such as a neural network.

3.3 Benefits

The benefits of the system that is analyzed within this chapter as described next in detail.

No Coding

This is usually the most important aspect of a visual scripting tool. No code means users without knowledge of programming or scripting languages can use the system to design various machine learning algorithms. Still, knowledge of machine learning is required in order to properly design and parameterize an algorithm. It is common for mathematicians, data analysts often to lack technical knowledge, so visual scripting closes this gap. Moreover, it also makes the tool more accessible to younger ages. Also, no coding means that the user can focus on the algorithm itself without having to worry about usual technical issues, such as compilation and logical errors.

Visualization

Visualization is strongly connected with the no coding part, as the user uses the interface to design algorithms. Visualization of both 3D and 2D can help the user to better understand a system, which can also help in the design process. Also, using existing examples, it can also act as an educational tool to explain the design of each machine learning algorithm. Moreover by having various preferences the user can adjust colors and other graphical aspects to make the software and algorithms' design more to his liking. Furthermore, visualization should be implemented in a way to allow the user to fully explore a machine learning algorithm both in a high level and a low level e.g. in case of a neural network, user could simply see the basic layers and by zooming in, all neurons could be shown along with detailed connections.

User Interface and User Experience

Interface and user experience (UX) are among the most important aspects of a software, especially one that is focused on visual scripting. By providing the user with a clean interface design and use of mouse and keyboard shortcuts, user experience should be optimal. Also if user interface is implemented properly, it can allow users with disabilities to make use of the software either by using proper interface principles e.g. font, colors, etc... or by even making use of voice commands.

Prototyping

All tools that use visual design systems are great for prototyping and a lot faster than common methods such as normal programming. It is important to remember, that prototyping is probably one of the most important if not the most important aspects of AI. AI design takes a lot of tries to properly implement something, so having a quick and flexible way to do it can save a huge amount of time.

Scalability

As mentioned, scalability is one of the most important parts of any system. Many systems fail on this part, as they are usually designed to solve specific issues, usually with limited development time, hence developers usually make compromises and break architecture patterns which means bugs and scalability issues. Modules solve this issue, because they are separate sub systems which allow each system to work separately, hence each module can be expanded, improved or solve issues from technical aspect separately. Also new modules can be added without breaking existing architecture.

Abstract Implementation

As the system is described how it can work, it can be implemented in various technologies as long as they can support the visualization part. This includes game engines, custom graphical engines or common graphical frameworks such as OpenGL/WebGL. That means the software can be implemented to work in all operating systems, web (hosted in a website) and work on different hardware as long as they support graphics.

Also by using proper and clean code for an object oriented architecture and by following the SOLID [16] principles, it means that the software can be shared with collaborators, extended, modified, tested, and refactored with fewer complications.

Lightweight

As long as a proper implementation architecture is used and the technology used for graphics is GPU accelerated the software should be able to run perfectly and without performance issues in most hardware.

Chapter 4 Panoptes

In this chapter, Panoptes will be analyzed. Panoptes is the software that was implemented as part of this thesis. The software was mentioned Panoptes from a character from Greek mythology. He is often referred to as "Panoptes the All-Seeing" or "Panoptes the Giant" [17]. In Greek mythology, Panoptes was a giant with a hundred eyes all over his body. He was said to be a servant of the goddess Hera. Similar to how Panoptes was referred as "Panoptes the All-Seeing", the software allows to “see” everything from low level to higher level, hence the all-seeing.

4.1 Introduction

Panoptes was implemented in Unity that was mentioned in Chapter 2 on version 2022.2.17f1. The software currently supports the design, visualization and simulation of a decision trees and the visualization of a neural network. Panoptes current language is English.

4.1.1 Choice of game engine

There are many game engines to choose from, but Unity was a good choice as it provides 3D graphics and it is very lightweight so the final executable can run smoothly on hardware with even limited computational resources (CPU, GPU, RAM). On the contrary, game engines such as Unreal Engine or Cry Engine require more computational resources. Furthermore, Unity language is C#, which is a very versatile, object oriented, strong typed and also has a big community with many libraries and frameworks to use. Moreover, Unity can also generate final executables for all operating systems (Windows, MacOS, Linux), all game platforms (Play Station, XBox, etc), all mobile operating systems (Android, IOS) as well as WebGL to allow deployment on web. Unity also supports VR and AR, which can provide a unique user experience.

4.2 Folder Structure

This section describes how the folders are organized in the software. Folder organization is an important part of the implementation of the project as it helps the programmers to work more effectively and makes the software management and scaling easier. Note that only the “top-level” folder structure is explained as there are a lot more subfolders within these ones.

- Assets
  - Resources
    - Materials
    - Meshes
    - Prefabs
    - Textures
  - Scenes
  - Prefabs
  - Scripts
    - Classes
    - MonoBehaviour

Assets

This is the first folder that is created from Unity and is the main folder that contains all elements of the game.

Resources

Is used to store all assets that are usually needs loading during runtime.

Materials

Contains all the materials used in the game from GameObjects.

Meshes

Contains all the 3D models used in the game.

Prefabs

Contains all the game Prefabs. Prefabs are combinations of many GameObjects which in essence form a small tree of other GameObjects.

Textures

Contains all the textures, meaning all the “images” used on 3D models or 2D interface.

Scenes

Contains all the scenes/levels used in the software.

Scripts

Contains all the code. It is divided in Classes and MonoBehaviour sub folders.

Classes folder contains all the folders that do not inherit from the MonoBehaviour Unity class, which means they cannot be added as a component in a GameObject. This includes all domain classes for decision tree and neural network and includes helper functions along with proper sub classes to allow the implementation of repository and factory patterns.

In contrast, MonoBehaviour folder contains all the classes that inherit from MonoBehaviour that can be used as components in GameObject, which include user interface manipulation and actual game mechanisms.

4.3 Design

The classes are structured in a way to follow the SOLID [16] principles as much as possible. Factory, repository and observer design patterns are used as well as dependency injection logic.

Memory tree vs Game Objects

In various parts across the chapter, the term “memory tree” is used. Memory tree is referred to the actual domain model classes that have to do with the decision tree.

The term is used to distinguish itself from the gameobjects of Unity engine which are used in the visualization part but also act as a bridge to the actual domain models.

Factory Pattern

Factories are used in creation of objects, to allow a single place to create objects.

Repository Pattern

Repositories are used to write or read data from a source, in this case system text files.

Observer Pattern

Observer pattern is used to allow communication between objects, specifically monobehaviour gameobjects. Also, this pattern is used extensively in Unity engine itself.

Dependency Injection Pattern

Due to how Unity works, the typical dependency injection cannot be applied as seen in other software such as web or desktop applications in .net or java frameworks. However, a similar approach was used in most cases to allow objects independent of their dependencies.

Validators

Validators allow for validations to happen in a specific class and be separated from the main code. Currently this approach is used only in connecting nodes which includes various checks.

Exception Handling

Although not a design pattern, exception handling is done broadly in the software to minimize user exceptions and show proper messages. Also, the console panel explained in the next paragraph also shows exceptions show that the user can see if an exception occurs.

4.4 User Interface

Before mechanisms and features are analyzed, it is important that user interface is properly explained so that the features are fully comprehensible.

4.4.1 Main Interface

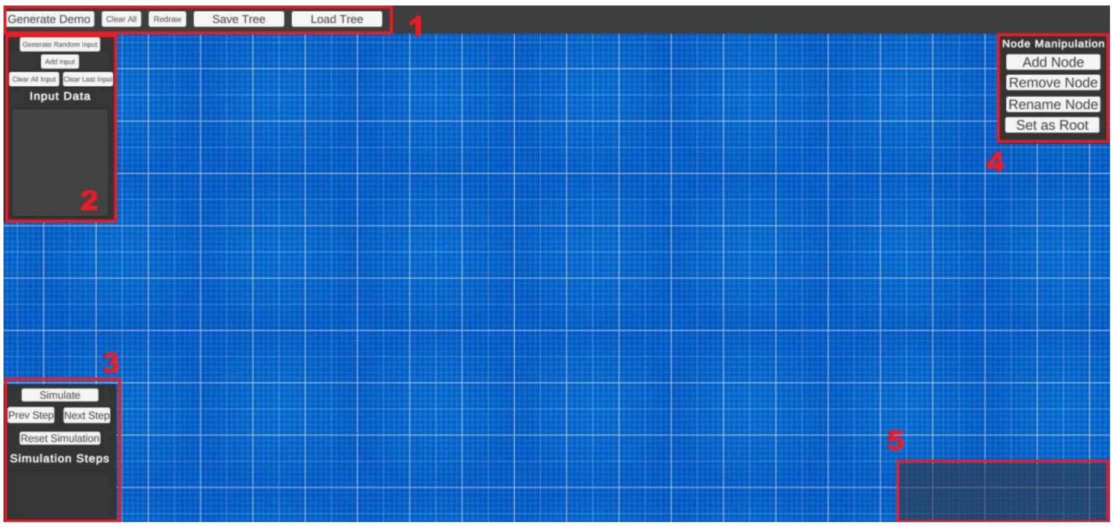


Image 25: User interface numbered

In the above Image 25 the main user interface is shown with numbered sections.

Section 1 – Top Panel

Top panel includes five buttons.

- **Generate Demo Button**: Generates a predefined decision tree demo.
- **Clear All Button**: Clears everything up for the current decision tree, which includes both memory objects and gameobjects.
- **Redraw Button**: Redraws current tree. This can also be used to "reformat" a tree as it uses a specific algorithm to draw the nodes.
- **Save Tree Button**: Saves the current tree. When pressed it shows a small input panel for user to enter the name.

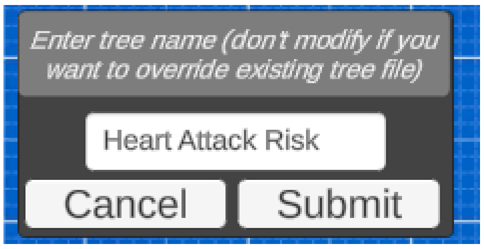

Enter tree name (don't modify if you want to override existing tree file)
Heart Attack Risk
Cancel
Submit

Image 26: User input panel for saving

- **Load Tree Button**: Loads a saved tree. When pressed it shows a small panel where the user can select one of the available saved trees from a dropdown list.

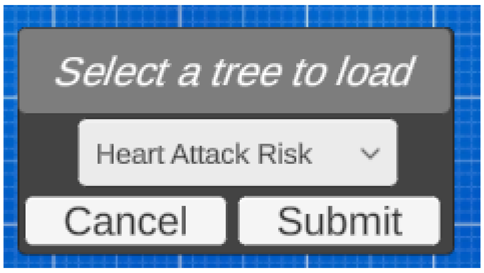


Image 27: User panel for tree loading

Section 2 – Input Data Panel

Input data panel is used for input insertion for the current tree.

- **Generate Random Input Button**: Generates a random input pattern for the tree.
- **Add Input Button**: Shows a pop up so that the user can select a value from a dropdown list based on current node, based on the available branches.

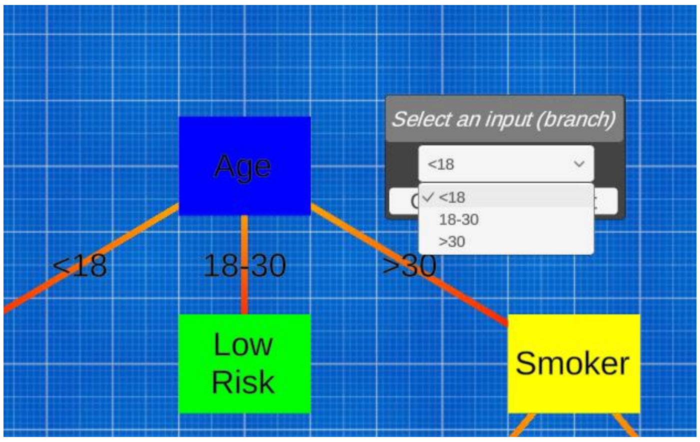


Image 28: Add input button panel

- **Clear All Input Button**: Clears all input data.
- **Clear Last Input Button**: Clears the last value of the input data.

- **Input Data Panel**: Contains all input data, steps used for simulation.

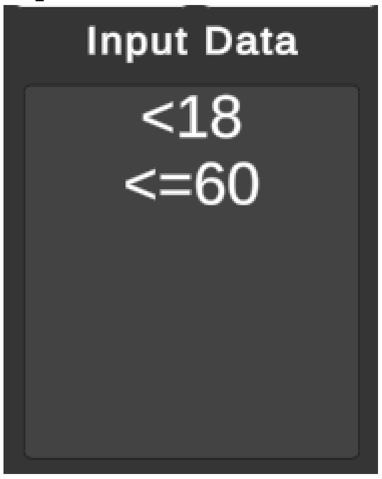


Image 29: Input data panel with data

Section 3 – Simulation Panel

Simulation panel includes all the interface elements that have to do about the tree simulation.

- **Simulate Button**: Starts the simulation of the current tree.
- **Prev Step Button**: Once simulation is started, it returns to the previous state/step.
- **Next Step Button**: Once simulation is started, it proceeds to the next state/step.
- **Reset Simulation Button**: Resets the simulation.
- **Simulation Steps Panel**: Shows current simulation steps.

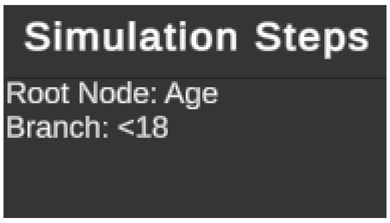

Simulation Steps
Root Node: Age
Branch: <18

Image 30: Simulation steps panel

Section 4 – Node Manipulation Panel

Node manipulation panel has all interface elements that has to do with node manipulation.

- **Add Node Button**: Adds a new node. The user can put the desired name in the pop up panel input field.

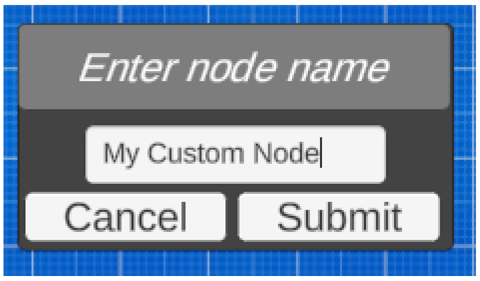
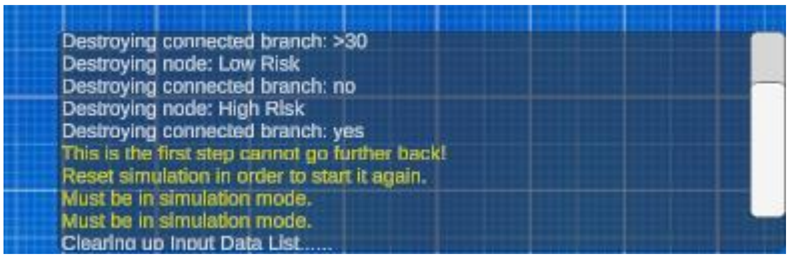


Image 31: New node button panel

- **Remove Node Button**: Removes the currently selected node.
- **Rename Node Button**: Renames a node with the specified new name.
- **Set as Root Button**: Sets the selected node as root node.

Section 5 – Console Panel

Console panel is used to show various messages from logging. It includes different coloring depending on the log type. White is used for normal logging, yellow for warning and red for errors and exceptions.



```
Destroying connected branch: >30
Destroying node: Low Risk
Destroying connected branch: no
Destroying node: High Risk
Destroying connected branch: yes
This is the first step cannot go further back!
Reset simulation in order to start it again.
Must be in simulation mode.
Must be in simulation mode.
Clearing up Input Data List.....
```

Image 32: Console panel

4.4.2 Tree Visualization

Tree visualization is referred to all elements used in visualization of tree.

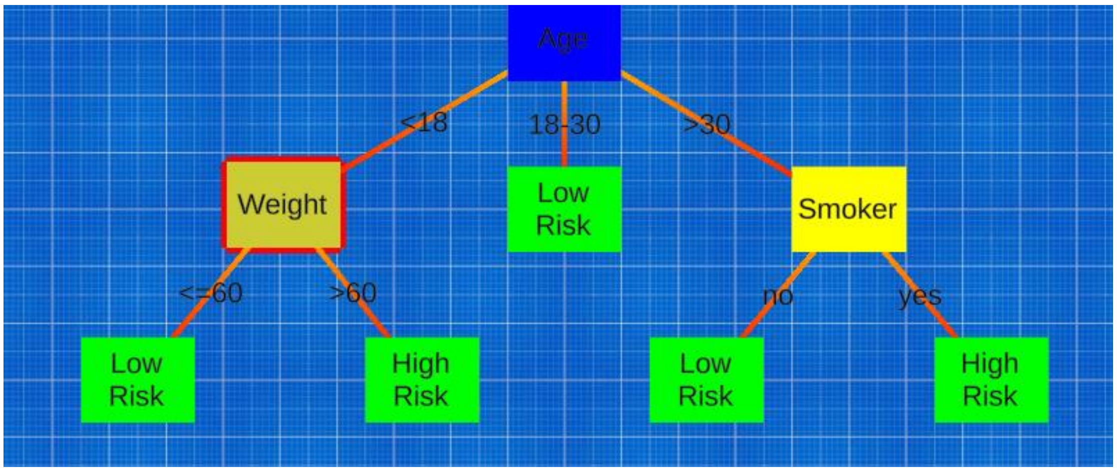


Image 33: Tree visualization with Weight node selected

In the above Image 33, the visualization of the demo tree is shown. In the image certain elements can be shown, such as the cubes and lines connecting them. Rectangles represent the tree nodes while the lines represent the branches connecting the nodes.

Node Visualization

As mentioned, nodes are represented by rectangles and are colorized differently so that the user can easily understand what type of node they are.

- **Blue Node**: Represents the root node.
- **Yellow Node**: Represents a decision node.
- **Green Node**: Represents a leaf node.
- **Node with red outline**: If a node has a red outline it means that it is either selected by the user, or is used in the current simulation step.

Also, nodes can be manipulated and moved around freely by the user.

Branches Visualization

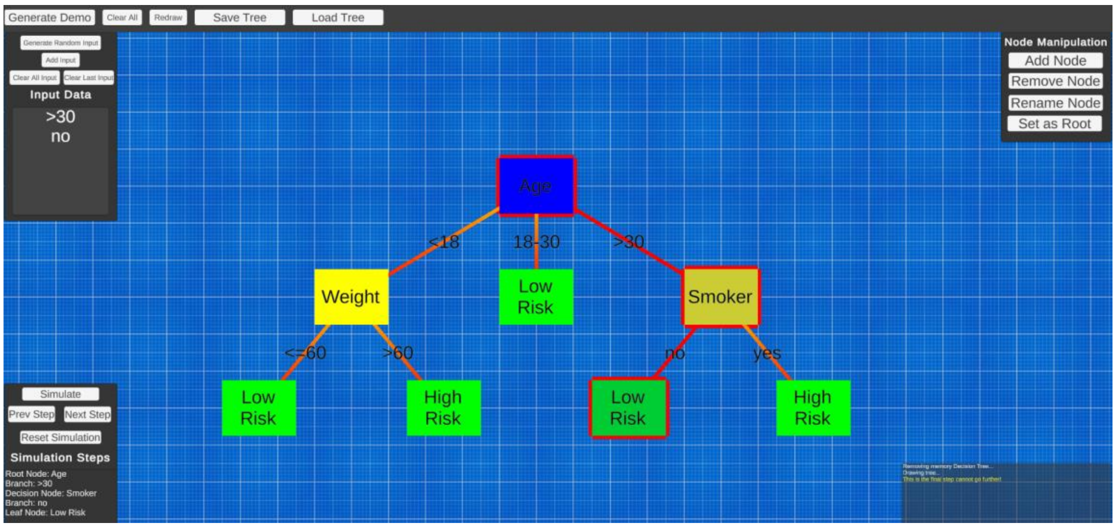
Lines represent the branches of the trees. Their coloring represents the branch direction.

- **Line coloring**: Yellow color is used to show the source of the branch and Red color is used to show the target of the branch.
- **Line text**: The text centered on top of the line represents the branch value.

Thins to note is that lines hence branches, cannot exist on their own and always shown when two nodes are connected. Moreover, lines update themselves whenever a node is used and update their start and end point coordinates accordingly. Lines will also update their text so that when moved, text is always in the center of the line.

Simulation Visualization

Simulation is a core aspect of the software and it needs a way to be visualized so that the user can actually see the visualization.



>30
no

Root Node: Age
Branch: >30
Decision Node: Smoker
Branch: no
Leaf Node: Low Risk

Removing memory Decision Tree...
Drawing tree...
This is the final step cannot go further!

Image 34: Simulation Visualization

As seen in the above Image 34, the red outlines for nodes and red lines are used to visualize the current simulations steps. Simulation will be discussed in details in the coming paragraphs.

4.5 Controls

Controls is referred to how the user interacts with the software with either keyboard or mouse.

Camera

Camera is currently set to Orthographic mode. In order to move the camera around the mouse wheel can be pressed and then drag the mouse. Mouse wheel is also used to zoom in and out depending on the roll direction. Note that even though the camera is only Orthographic, the application is fully 3D, which means that changing a simple setting in Unity inspector can alter the mode to Perspective. Currently Perspective is disabled due to the lack of fully proper controls such as rotating camera and moving in 3D space with full freedom.

Nodes

Node manipulation can be done using the mouse and keyboard.

- **Selecting Node:** Can be done by pressing the left click when the cursor is over a node. When the node is selected, a red outline should show around the node.
- **Deselecting Node:** Can be done by various ways, but mainly when pressing left or right click while the cursor is at a blank space.
- **Deleting Node:** Can be done by selecting a node and pressing the “Delete” key on keyboard. Same result can be achieved using the “Remove Node” button. Node coloring will also change the affected nodes such as the parent of deleted node depending on whether the nodes has changed type e.g. from leaf node to decision node.
- **Moving Node:** By selecting a node with left click and holding the left click, the node can be dragged around and follow the cursor. Moving a node will update the connection if any accordingly.
- **Node Connect:** To connect a node with another node, the user has to first select a node and then right click when the cursor is over the desired node. That connects the selected node to the node that the cursor is currently on top of. If the nodes are successfully connected, a new branch/line should show connecting the nodes. Node coloring will also change depending on whether any of the nodes has changed type e.g. from leaf node to decision node.
- **Node Disconnect**: Disconnecting works exactly as connecting, but instead of creation new branches it removes existing ones.

4.6 Features

In its current version Panoptes is focused on the design, implementation and simulation of decision tree module. Below all the mechanisms that are implemented are shown and described.

4.6.1 Tree Creation

Decision tree creation currently supports an example “Heart Attack Risk” demo and also supports creation from scratch. In the first image below, the red rectangle highlights the button used to generate the predefined demo. The second image shows what happens when the button is pressed, which is to create a predefined tree as shown.

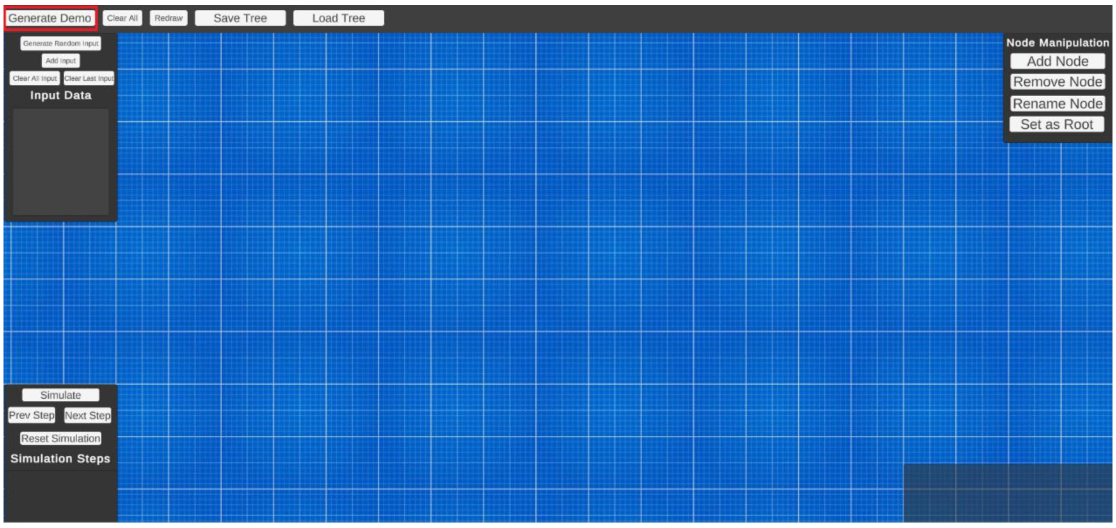


Image 35: Tree creation demo (before button clicked)

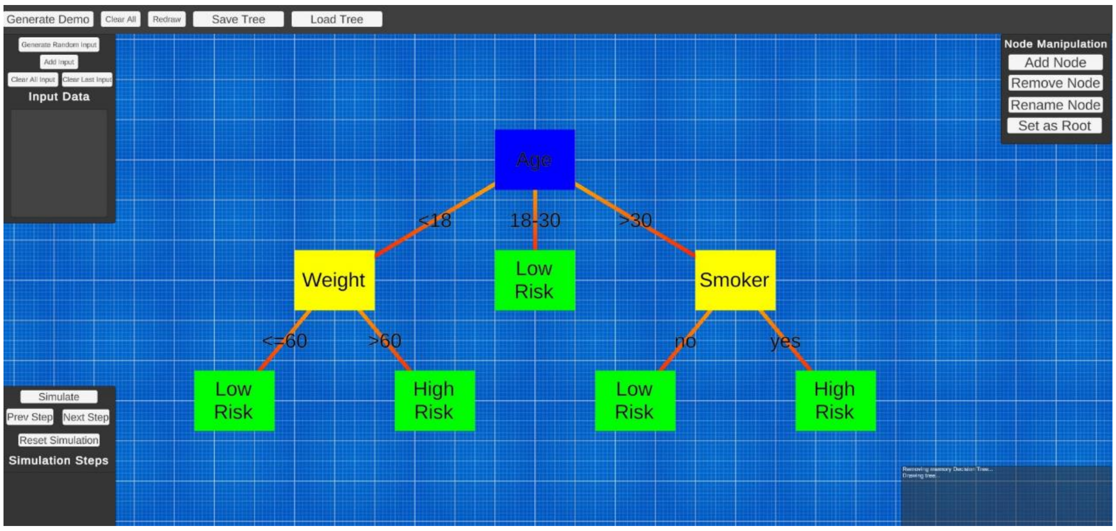


Image 36: Tree creation demo (after button clicked)

Creation from scratch is simply done by adding a new node manually.

4.6.2 Tree Saving

Tree saving allows the user to save the current tree. If a tree is already loaded, then the popup input field is already populated with that tree's name. If the same name is used, then the file will be overwritten. For the saving the proper repository is used, which serializes the tree class and saves it to a .json file in

C:\Users\<LOCAL_USERNAME>AppData\LocalLow\Panagiotis Kyrkos\Panoptes\Saved\Modules\Decision Tree.

If the directory doesn't exist as it happens in the first saving, it will be created. Currently saving node gameobjects locations is not supported.

4.6.3 Tree Loading

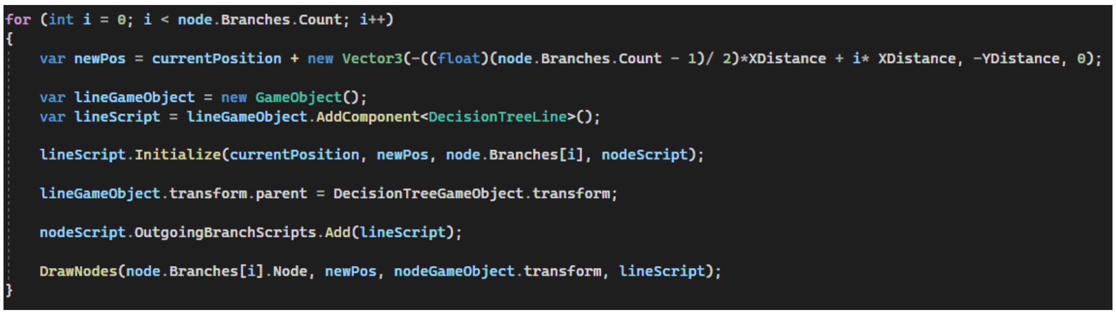
Tree loading allows the user to load a saved tree. When the appropriate button is pressed a popup shows with a dropdown list that is populated with the names of the saved trees. The user can then select one of the values in the dropdown and load the tree. Currently since saving node gameobjects locations is not supported, all the newly loaded trees use an algorithm to draw the nodes. Note that both saving and loading use a framework called Newtonsoft [18] for the .json serialization.

4.6.4 Clear All

Clear all is a function that allows the user to erase the current tree, which includes everything both memory tree and the all related gameobjects such as nodes, lines, texts etc.

4.6.5 Automatic drawing

Automatic drawing is a function that draws a loaded tree or redraws the current tree. It uses an algorithm that creates and connects all needed elements both on memory and gameobjects. Redraw function is used when selecting the demo generation or when the user loads a saved tree.



```
for (int i = 0; i < node.Branches.Count; i++)
{
    var newPos = currentPosition + new Vector3(-((float)(node.Branches.Count - 1)/ 2)*XDistance + i* XDistance, -YDistance, 0);

    var lineGameObject = new GameObject();
    var lineScript = lineGameObject.AddComponent<DecisionTreeLine>();

    lineScript.Initialize(currentPosition, newPos, node.Branches[i], nodeScript);

    lineGameObject.transform.parent = DecisionTreeGameObject.transform;

    nodeScript.OutgoingBranchScripts.Add(lineScript);

    DrawNodes(node.Branches[i].Node, newPos, nodeGameObject.transform, lineScript);
}
```

Image 37: Redraw function code snippet

The above code snippet shows a part of the function that does the positioning of the next to be created node. The math function is currently fairly simple and does not currently handles all cases properly. Also the redraw function, as shown in the snippet it uses recursion.

In the images below it is shown how the tree redraws itself when pressing the redraw button after the user deletes certain nodes. The "Redraw" button is pressed after each node deletion.

1. First the “Weight” decision node is deleted.
2. Then the “Smoker” decision node is deleted.
3. Finally the “High Risk” node is deleted.

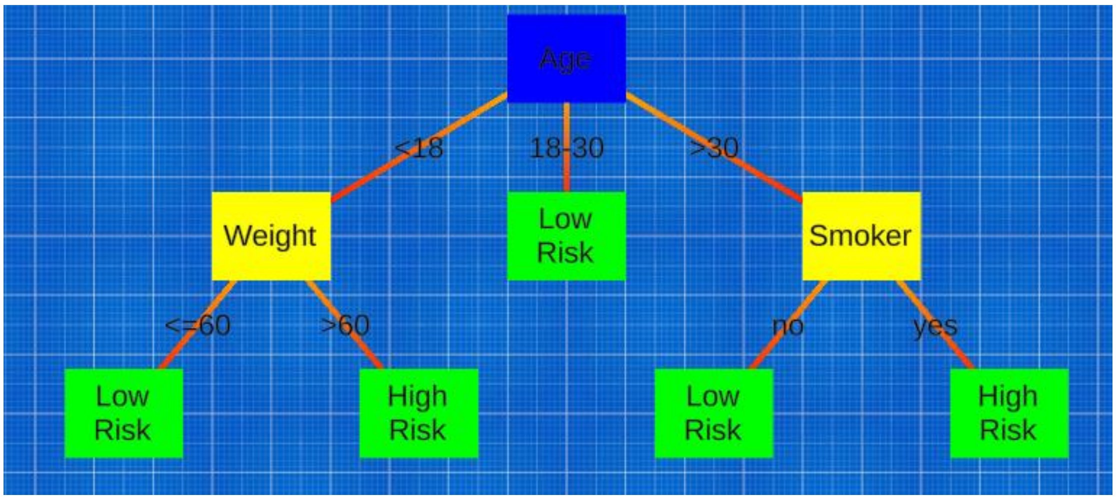


Image 38: Demo tree default redraw

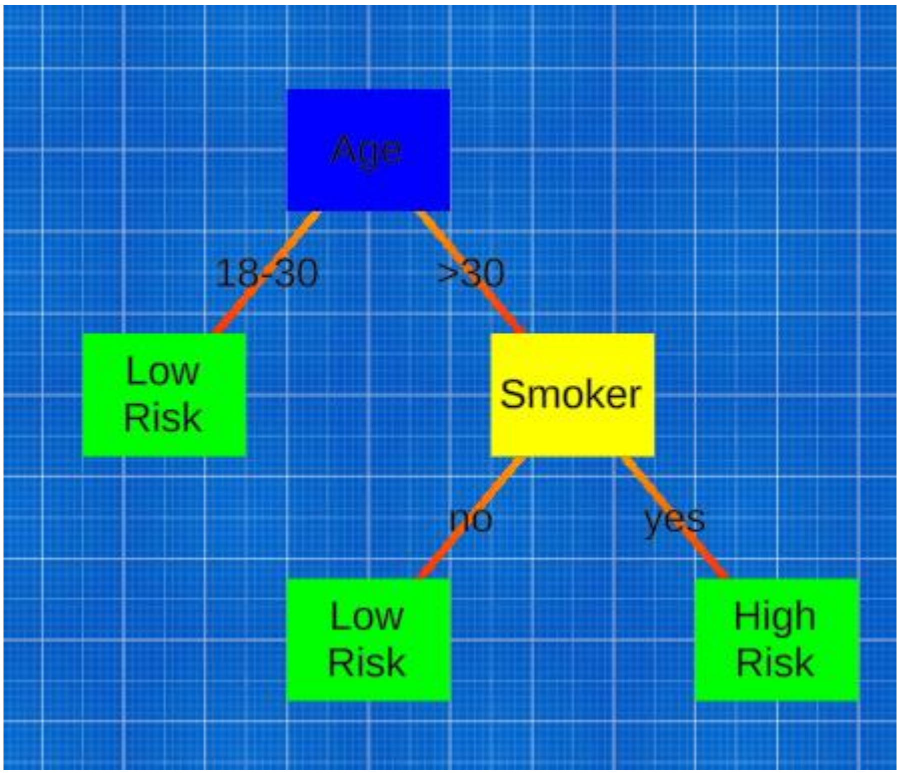


Image 39: Demo tree after removing Weight node and Redrawing

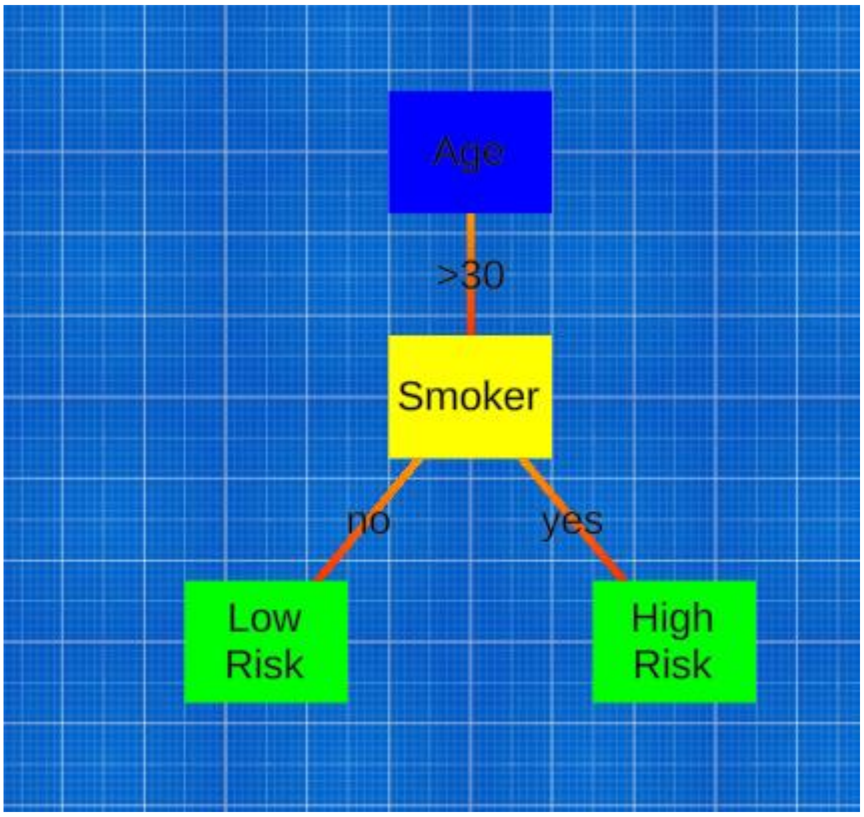


Image 40: Demo tree after removing Weight and Smoker nodes and Redrawing

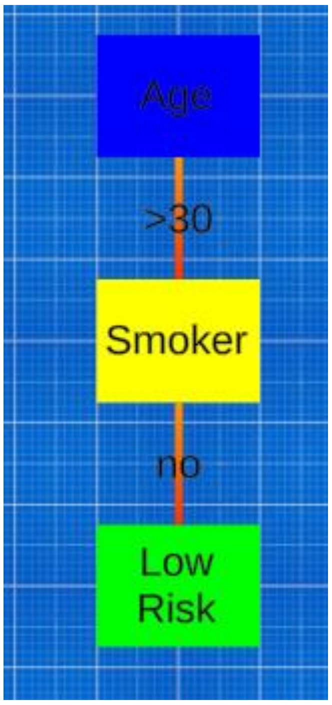


Image 41: Demo tree after removing Weight, Smoker and High Risk nodes and Redrawing

4.6.6 Node Manipulation

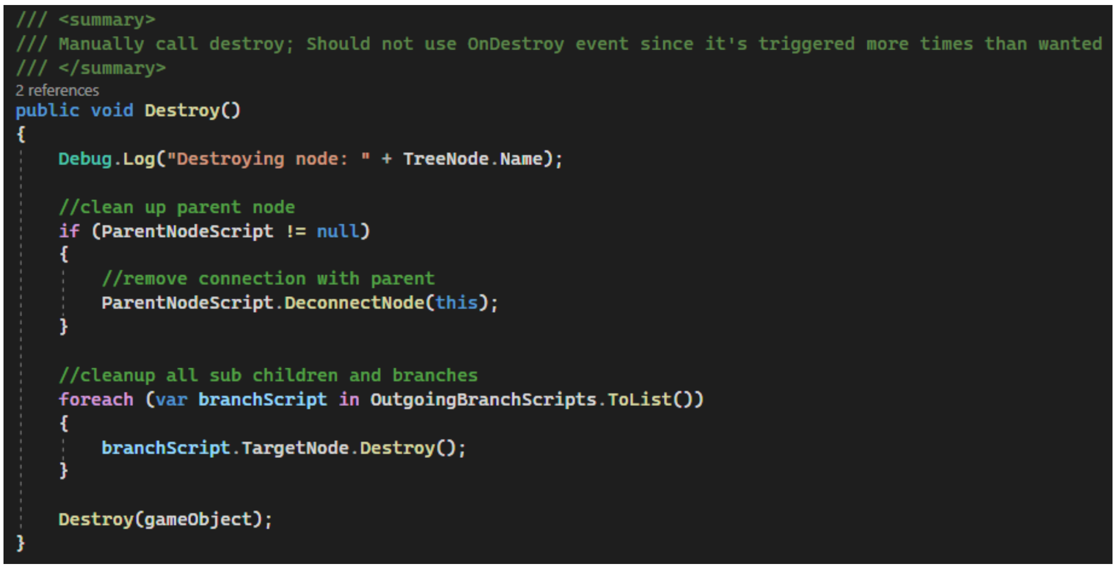
Node manipulation includes all the actions that are related to tree nodes.

Adding Node

New node is supported which can be done by pressing the “Add Node” button. When adding node, an input can be given by the user which determines the node name/value. If the user has already selected a node when adding a node, then the newly created node will be connected to the selected one. The user will also be shown a second popup to insert the branch value. The first node to be added is automatically set as the root node.

Deleting Node

Deleting node can be achieved by pressing the “Remove Node” button or pressing the “Delete” keyboard key. This action requires a node to be selected and this node cannot be the root node. When deleting a node, all the direct connected outgoing branches and all sub children nodes and their branches will be removed using a recursive function.



```
/// <summary>
/// Manually call destroy; Should not use OnDestroy event since it's triggered more times than wanted
/// </summary>
2 references
public void Destroy()
{
    Debug.Log("Destroying node: " + TreeNode.Name);

    //clean up parent node
    if (ParentNodeScript != null)
    {
        //remove connection with parent
        ParentNodeScript.DeconnectNode(this);
    }

    //cleanup all sub children and branches
    foreach (var branchScript in OutgoingBranchScripts.ToList())
    {
        branchScript.TargetNode.Destroy();
    }

    Destroy(gameObject);
}
```

Image 42: Node Delete code snippet

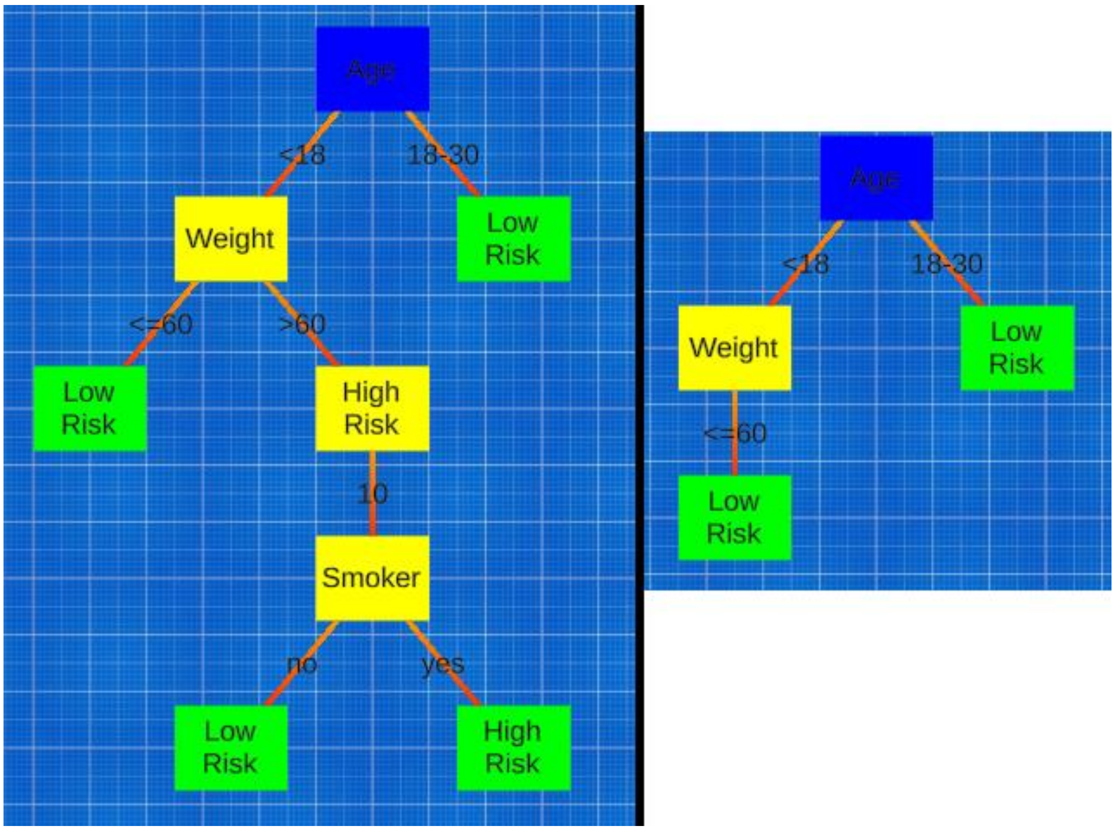


Image 43: Node visualization before and after delete of node "High Risk"

Rename Node

Rename node is a simple function to rename the selected node. Can be done via pressing the "Rename Node" button.

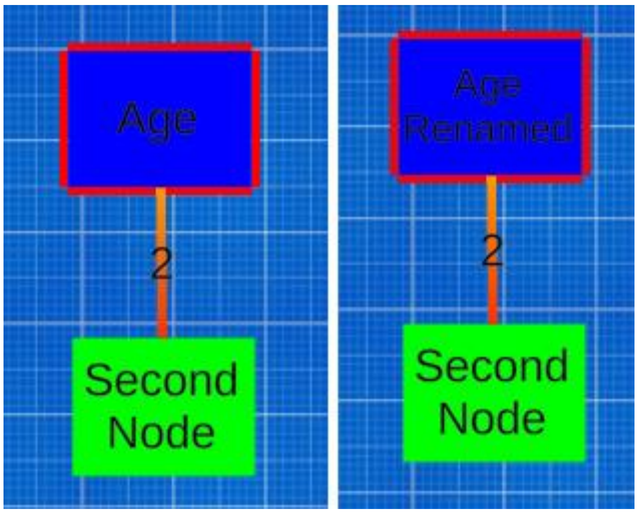


Image 44: Node before and after rename

Set Root Node

Set root node changes the current root node of the tree. Can be done by pressing the "Set as Root" button. Requires a selected node.

4.6.7 Node Connections

Node connections are the mechanism that allows the user to connect a node to another node. Connecting two nodes will also update the memory tree as well as the gameobjects. Connecting nodes require certain checks in order to be allowed.

These checks are:

- Both source and target node need to be non nullable. This means that the user needs to have selected node (source node) and cursor when right clicking be on top of another node (target node).
- Source node and target node cannot be the same. Connecting a node to itself is not allowed.
- Target node cannot be the tree's root node. Root node is the start of the tree, so no other node can connect to it.
- Target node must not be the parent of the source node. This would lead to a bidirectional connection of nodes which is not allowed on decision trees.
- Target node having a parent. If a target node is connected, it means it already has a parent node and only one parent is allowed per node.

4.6.8 Input Data

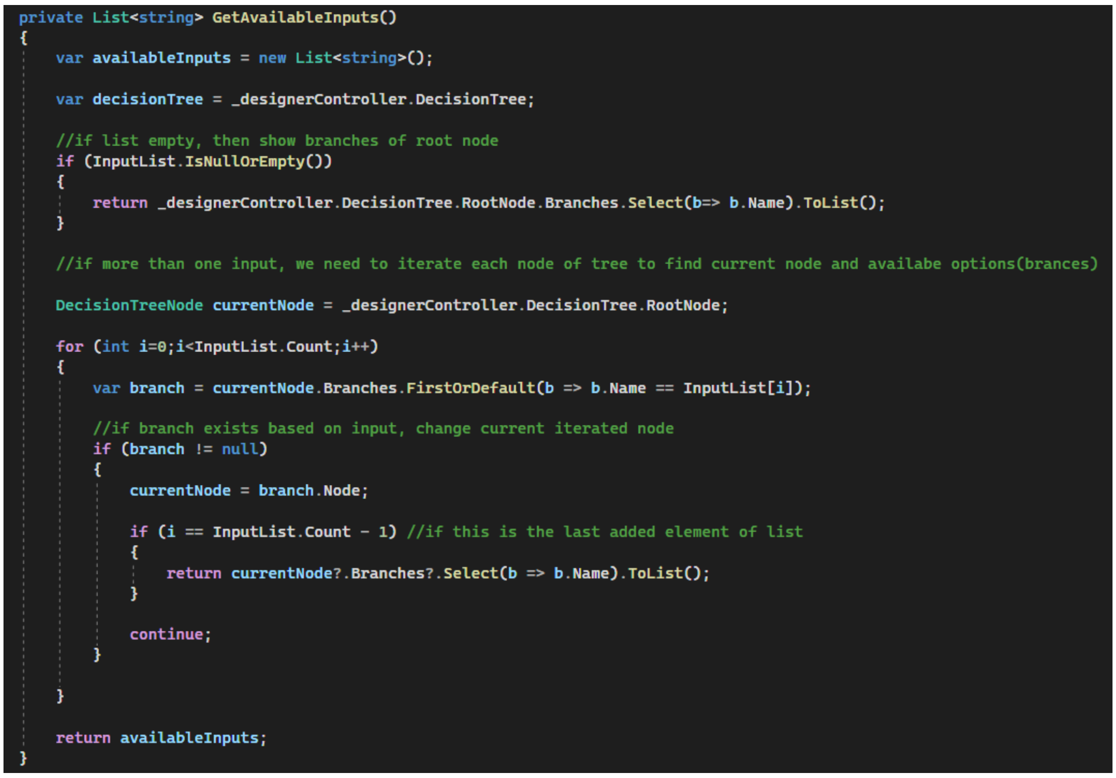
Input data are all the elements that are related to input added by the user and used in simulation. The current input values are shown in the left top panel "Input Data". The following actions are available:

Generate Random Input

Creates a random input for the current tree. This can be used to populate quickly with data for a simulation.

Add Input

Manually add input. When the "Add Input" button is pressed the system will check the current input data values and will iterate the tree to find which node is the last one based on that data. If the node found is not a leaf node which means it has more paths to proceed, the user is shown a popup with a dropdown populated with these available branch values (inputs).



```
private List<string> GetAvailableInputs()
{
    var availableInputs = new List<string>();

    var decisionTree = _designerController.DecisionTree;

    //if list empty, then show branches of root node
    if (InputList.IsNullOrEmpty())
    {
        return _designerController.DecisionTree.RootNode.Branches.Select(b=> b.Name).ToList();
    }

    //if more than one input, we need to iterate each node of tree to find current node and availabe options(brances)

    DecisionTreeNode currentNode = _designerController.DecisionTree.RootNode;

    for (int i=0;i<InputList.Count;i++)
    {
        var branch = currentNode.Branches.FirstOrDefault(b => b.Name == InputList[i]);

        //if branch exists based on input, change current iterated node
        if (branch != null)
        {
            currentNode = branch.Node;

            if (i == InputList.Count - 1) //if this is the last added element of list
            {
                return currentNode?.Branches?.Select(b => b.Name).ToList();
            }

            continue;
        }

    }

    return availableInputs;
}
```

Image 45: Get Available Inputs code snippet

Clear All Input

Clears all the current input data. Can be run by pressing the “Clear All Input” button.

Clear Last Input

Clears the last added input. Can be run by pressing the “Clear Last Input” button.

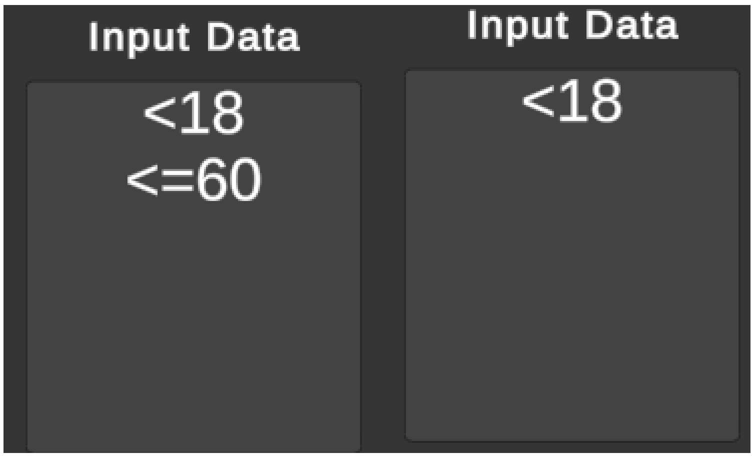


Image 46: Clear last input example

4.6.9 Simulation

Simulation is one of the core aspects of the applications. The simulation uses the input data described in 4.6.8 and simulates the run on the current tree. Simulation’s main purpose is to visualize which can aid the user to have a good understanding of the machine learning algorithm that is shown, in this case the decision tree. Simulation supports a step-by-step visualization with the ability to go to both previous and next steps.

A demo is shown below with step-by-step pictures.

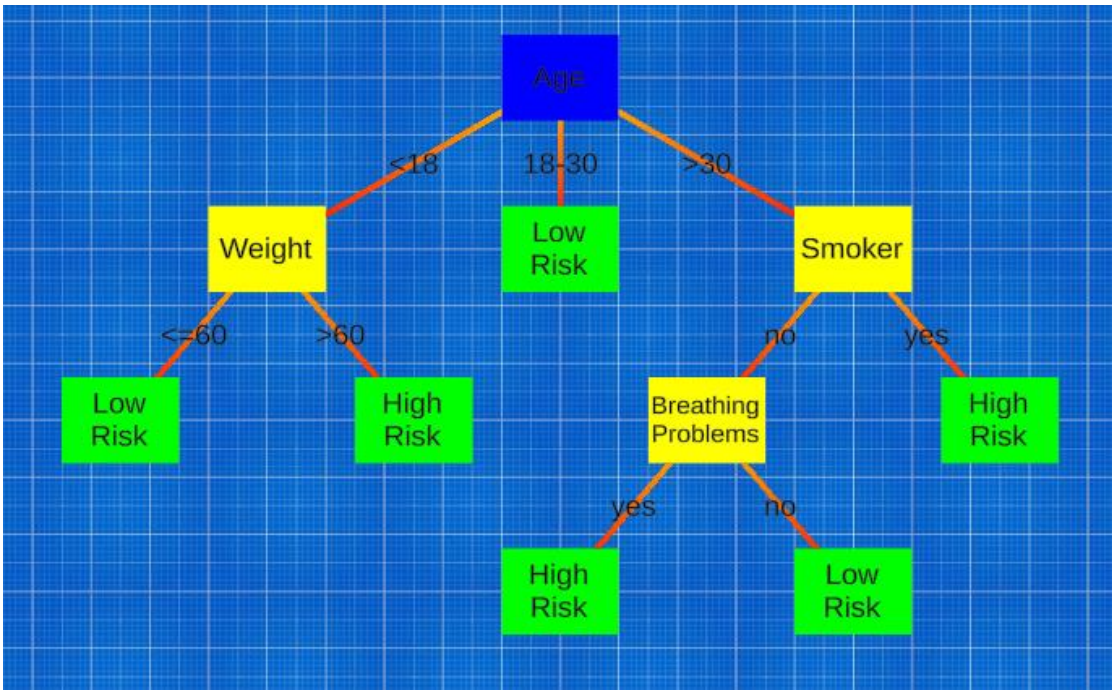
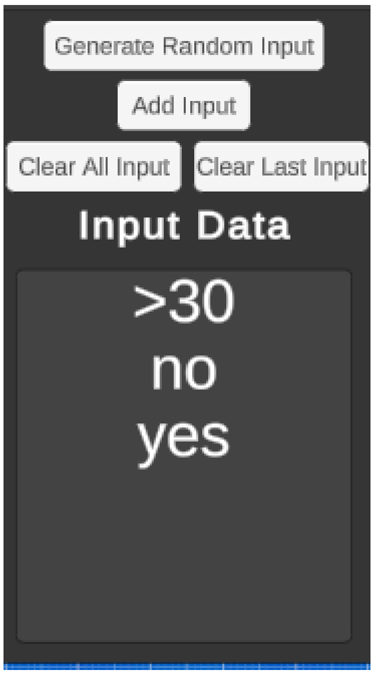


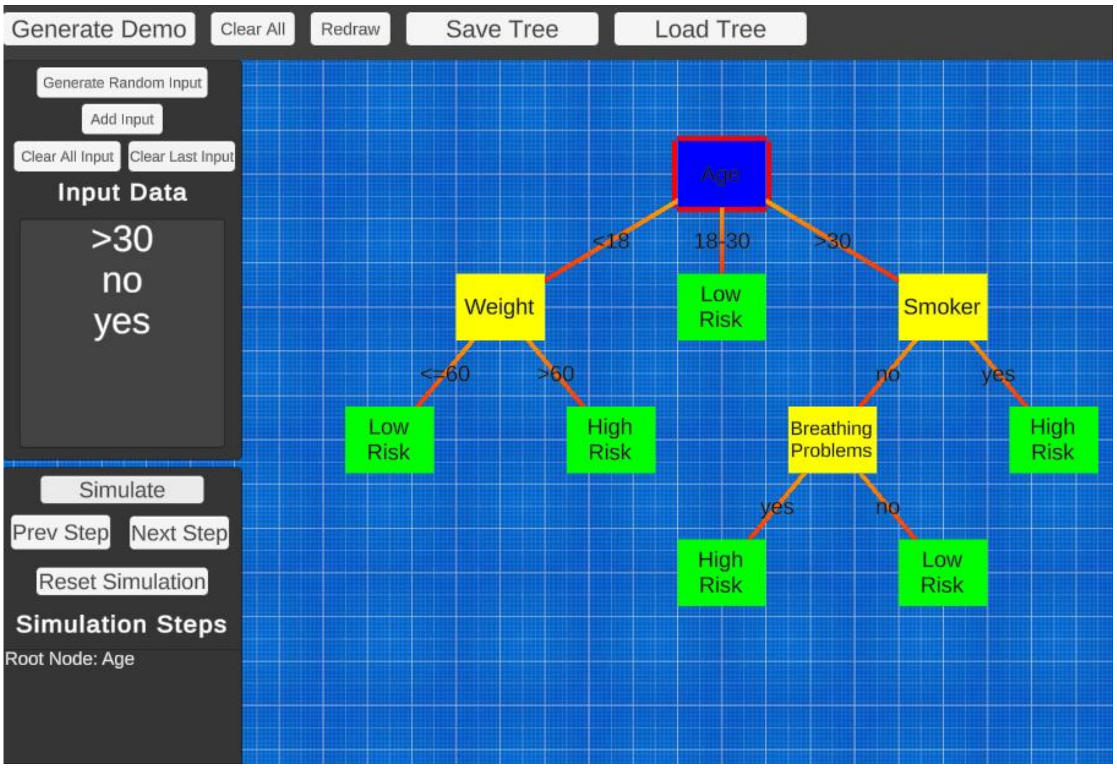
Image 47: Simulation demo tree



Input Data

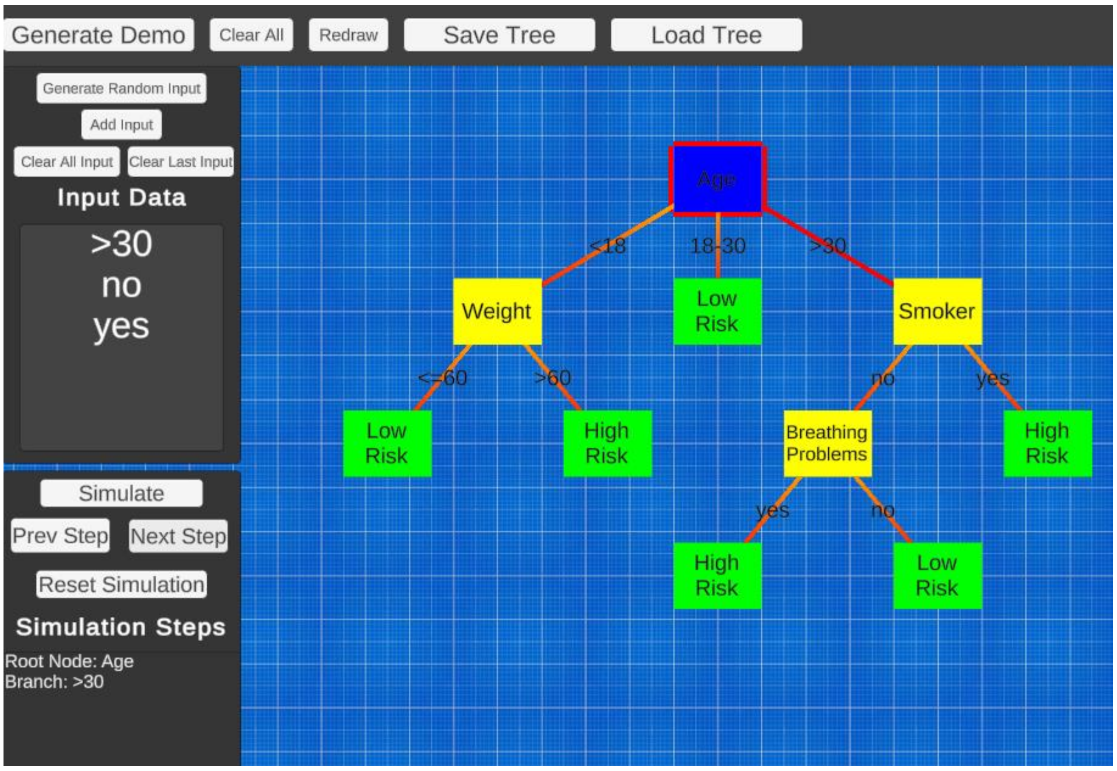
>30
no
yes

Image 48: Simulation demo input data



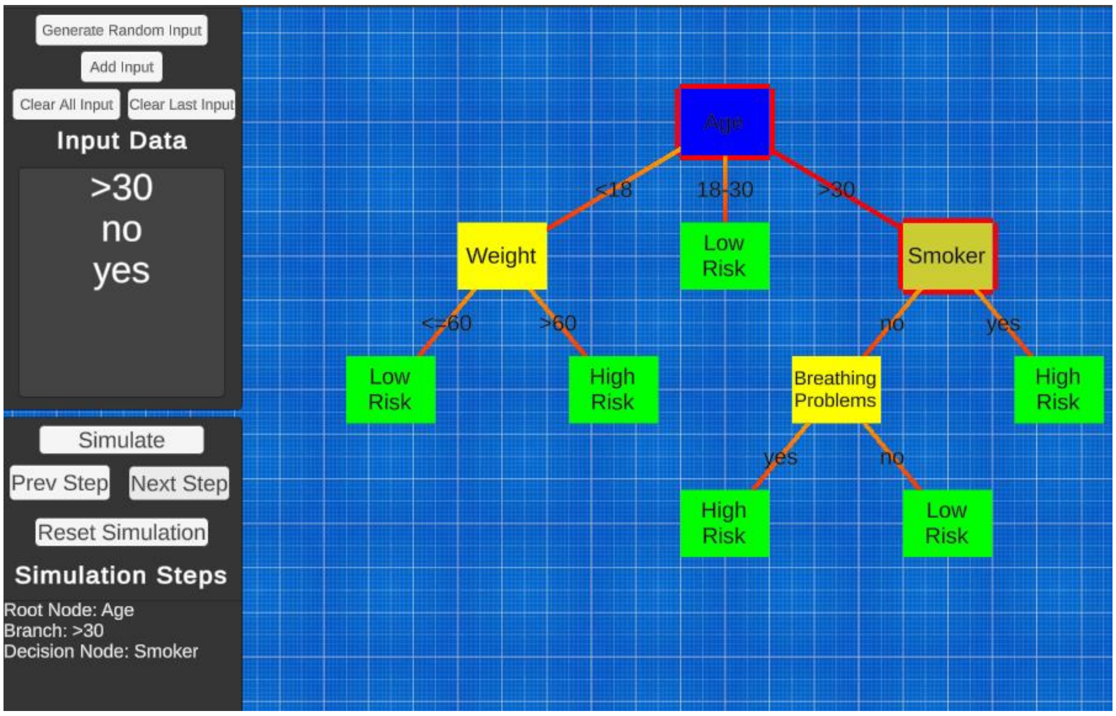
Root Node: Age

Image 49: Simulation demo after starting Simulation



Root Node: Age
Branch: >30

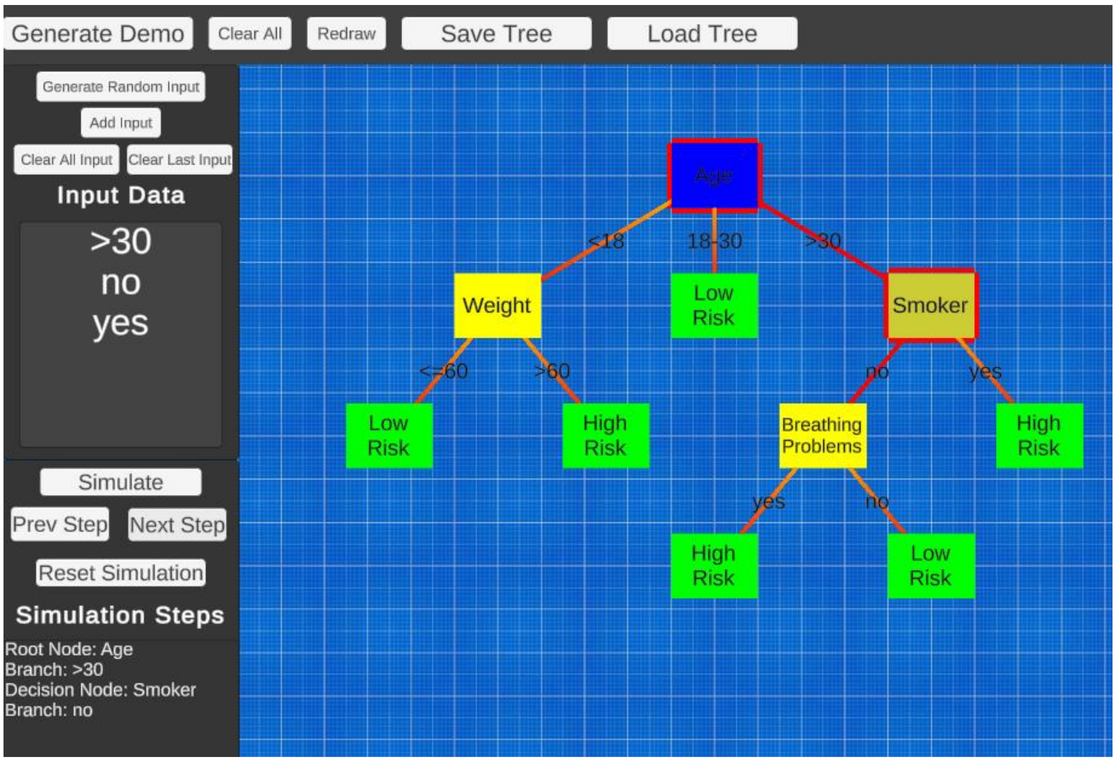
Image 50: Simulation demo step 2



>30
no
yes

Root Node: Age
Branch: >30
Decision Node: Smoker

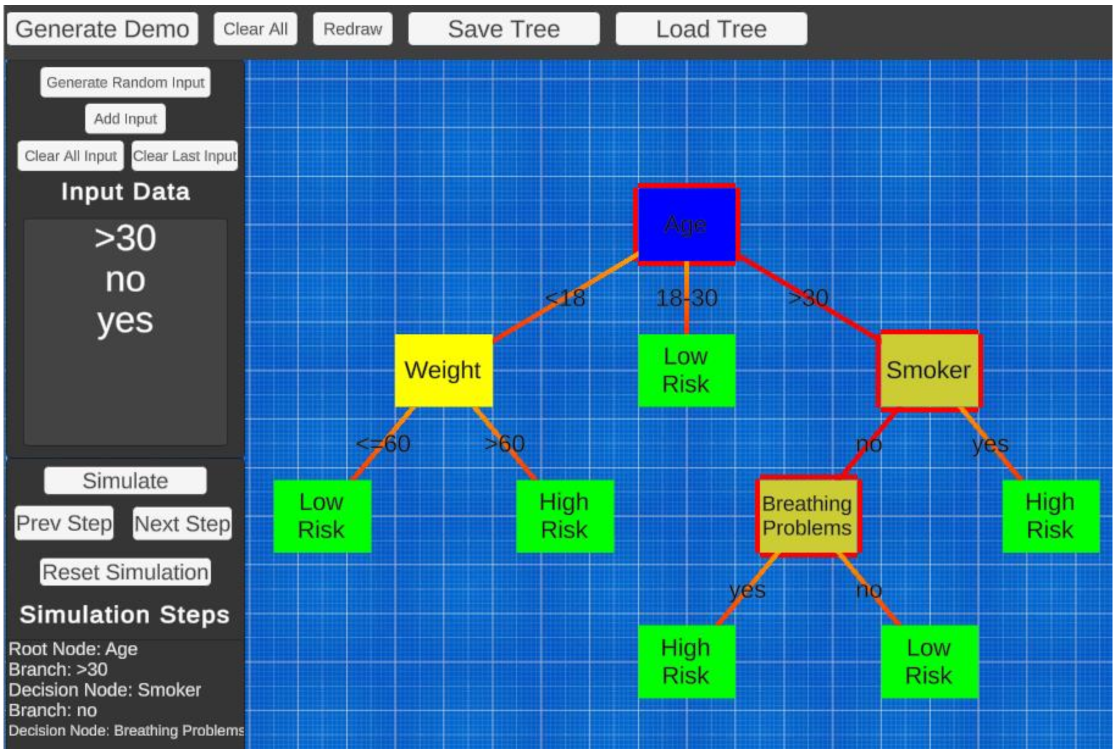
Image 51: Simulation demo step 3



>30
no
yes

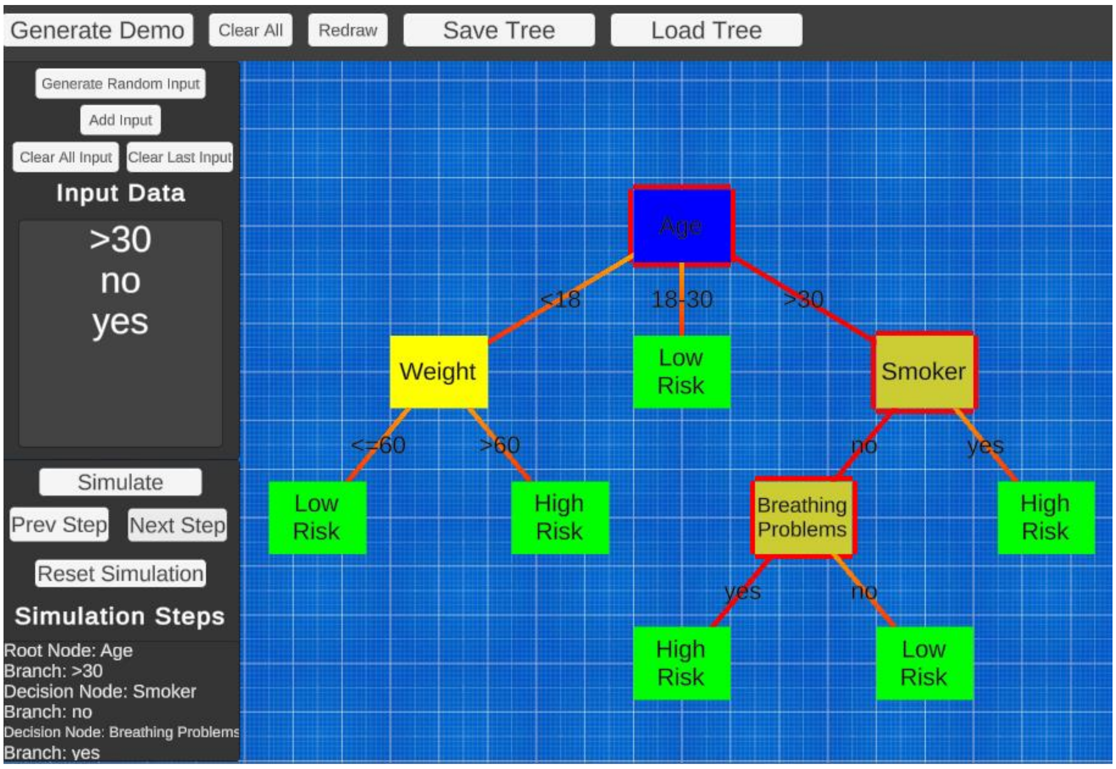
Root Node: Age
Branch: >30
Decision Node: Smoker
Branch: no

Image 52: Simulation demo step 4



Root Node: Age
Branch: >30
Decision Node: Smoker
Branch: no
Decision Node: Breathing Problems

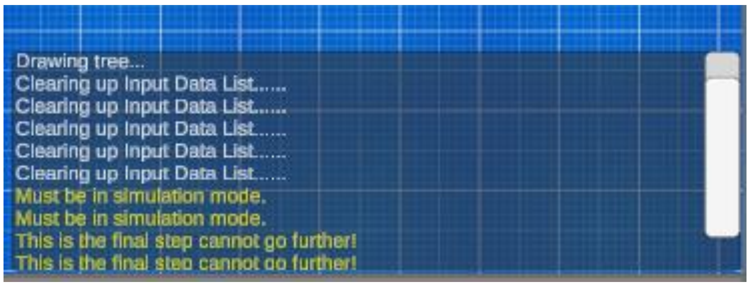
Image 53: Simulation demo step 5



Root Node: Age
Branch: >30
Decision Node: Smoker
Branch: no
Decision Node: Breathing Problems
Branch: yes

Image 54: Simulation demo step 6

The last image shows the simulation end. Each image step is after pressing the “Next Step”. The highlighted in red is the current activated nodes and branches in each step. After the end is reached, pressing the “Next Step” button throws a warning.



```
Drawing tree...
Clearing up Input Data List......
Clearing up Input Data List......
Clearing up Input Data List......
Clearing up Input Data List......
Clearing up Input Data List......
Must be in simulation mode.
Must be in simulation mode.
This is the final step cannot go further!
This is the final step cannot go further!
```

Image 55: Console error after pressing "Next Step" in simulation panel while in final step

Chapter 5 Future Development Plan

Panoptes that was discussed in the previous chapter, is only a very early prototype of the design system that was analyzed in this Thesis. Being an early prototype there are many things that can be improved and added in all aspects. Despite the fact that being an early prototype, the software was implemented in such a way that allows easy expansion and reusing code and systems that are already implemented. Future add-ons and improvements are discussed in the following paragraphs.

5.1 Designers

In this section, all plans for both system and module designers will be examined.

More Modules

Probably the most important is to add more modules as there are many machine learning algorithms such as Linear regression, Logistic regression, SVM algorithm, Naïve Bayes algorithm, KNN algorithm, K-means, Random forest algorithm, Dimensionality reduction algorithms and others.

Entity Grouping

A useful add-on would be a function to allow "grouping" of entities. Grouping can contain naming and also be visualized using an outline that shows the name of group. Groups should also include a mechanism to allow the module nodes to be "combined" into one node when the user zooms out and show in detail when the user zooms in. Grouping should be implemented both for System Designer and Module Designers.

Templates

Templates are useful in designer applications so that the user can use a template so that he does not have to start to design something from scratch. Templates can either be system created or user created.

System Designer

Currently system designer is still very work in progress. Features such as loading and saving need to be added. Data Manipulators and Operators can also be implemented to allow for things such as math functions, array manipulations, etc.

5.2 General Improvements

Panoptes can be improved a lot further on some general aspects, to become a much more mature, easier to control and more accessible software. Some of the improvements that can be done are explained in detail afterwards.

User Interface

User interface can be improved in all aspects of the software and contain more information, smoother graphics and transitions to provide the user with the optimal experience. This includes supports for different resolutions

Settings

More settings and preferences mean more control for the user. Settings to control aspects such as the colors, font of labels, meshes used in visualization.

Accessibility

Supporting more languages by adding a Localization system will greatly help the software to be accessible for more people. Also, many people have disabilities of various kinds, so improving the software in such way that it provides proper access to user with disabilities. That can be different colors for people with eye color blindness, more sounds, different fonts that are clearer and larger. Moreover, a voice controlled could help with people that have body disabilities and cannot properly used mouse or keyboard.

Sounds

Sounds are not included in this version but they could be added to allow a better experience. That can include sounds for simple pop-up messages to error warnings. This is directly connected with accessibility and disabilities as it will allow more users to use the software.

Exporting to other systems

In this version the software was designed to work with Windows. Unity supports exporting and creating an executable for other OS such as MacOS, Linux, Mobile, IOs, WebGL and game platforms. Although the code is not fully compatible for each system, only minor changes are required in order to work.

VR/AR

Virtual Reality (VR) and Augmented Reality (AR) are very trending technologies these past few years. Although not fully embraced by most users yet, both technologies seem promising. Many applications use VR and AR to improve user experience.

Testing other Game Engines

Although Panoptes was implemented with Unity, it could worth investigating and experimenting with implementing the same system on a different game engine or technology to see if there are more benefits, either from performance or graphics.

Chapter 6 Conclusion

Artificial Intelligence, despite being is an old field in computer science, the last years due to the great advances in hardware and software, it has been rapidly developing and expanding to allow technology to reach a whole new level. This advancement, requires proper tools in order to allow proper and full usage at their full potential of the artificial intelligence and machine learning algorithms. Although many tools exist for various purposes, most of them lack a proper interface or have no at all and still do not cover many of algorithms. Advancement in technology, also brought many changes in game engines and allow easier, faster, better and more optimized ways to create games and "serious" games. It is important to always keep in mind that game engines are not used exclusively for game development and that they are great tools to be used in visualization of other software as well.

Visual scripting is also another core system and software design approach that seems to be developing in the last years. Although, it may not be a good option for very large software, it can be used to design and implement smaller difficult and complex systems without the user having knowledge in coding. It also allows for rapid deployment due to the nature of it, usually being with simple buttons and controls. They are also a great for educational purposes as users tend to understand things a lot better when shown in a visual way and even more when combing with audio stimulus as well.

Utilizing game engines to create visual scripting tools for artificial intelligence is an innovative way to move forward. Artificial intelligence, is often hard to understand and gets very complex, so having a visual aid, not only it can help with educational purposes to allow users to better comprehend the artificial intelligence domain, but can also help with system design and implementation. Even experienced users tend to have trouble creating such systems, since they usually require good knowledge of things such as programming or math. Visually designing a system takes the programming out of the picture, so that the user can fully focus on design the system without having distractions caused from programming such as errors, exceptions, etc.

The developed system "Panoptes" that was analyzed, designed and implemented as part of this thesis, achieves creating a system that utilizes Unity game engine to create a visual scripting and design tool for artificial intelligence and machine learning algorithms. Simulation mode also achieves a great way to visualize how a machine learning algorithm works, so that even a new to the field user can fully understand how it works.

While developing the "Panoptes", certain difficulties were encountered such as finding a proper way to bind the actual decision tree data structure to the visual aspect of (Unity game objects) and have it behave as one so all the changes done on the visual design area are respectively done in the actual tree structure as well and vice versa. Furthermore, a small yet important problem that came up was the branch visualization. Although in other technologies or game engines, this would normally be a simple task of drawing a simple line between points, in Unity it proved challenging as each line/branch is essentially a new game object which needs extra handling as to how it is connected to the tree data structure branch and the cleanup of it as well. Another difficulty was creating a way to properly draw and connect the nodes automatically from the data structure when no nodes are drawn, which was successfully implemented using a custom algorithm that handles most cases. Lastly, the final important issue that was encountered, was the business logic and writing all the rules and validations needed so that to prevent the user doing actions that do not make sense. It was solved by properly implementing all needed checks and not allowing the user to do those actions.

Apart from the problems that were encountered, developing a software such as "Panoptes" was a challenging task. It required proper code structure as well as following good coding practices so that the tool is as expandable as possible without many changes or issues. The tool was implemented in such a way that is possible to easily create designers similar machine learning algorithms that use graphs as visualization. Moreover, developing a visual design tool, essentially is creating an "editor" for a user, so the design and implementation requires knowledge of various fields in computer science, mainly in programming and computer graphics but for user interface as well so that the tool, needs to be as self-explanatory to user as possible.

Furthermore, Unity game engine proved a great technology to build the software as it provides a lightweight executable that is suitable to most desktop computers without needing high requirements for hardware, as well as being able to export it to many and different platforms very easily. It is also easy to work with graphics and has many capabilities as to extend the software in the future to provide the optimal user experience. Finally, C# programming language combined with .Net framework that are used for the programming part that is supported by Unity were great, robust and flexible technologies to program the software. They provided a way to focus on the actual development and design of the software without having a lot of technical issues that are found in other programming languages or technologies.

In conclusion, as artificial intelligence is greatly progressing, software similar to "Panoptes" will greatly aid in the understanding, design and implementation of it and allow more rapid growth of the field which can lead to new technologies and innovations.

Chapter 7 Appendix

7.1 Domain Classes and Utils

7.1.1 Decision Tree

```
using System;

namespace Modules.DecisionTree
{
    [Serializable]
    public class DecisionTree
    {

        public string Name { get; set; }

        public DecisionTreeNode RootNode { get; set; }

        public DecisionTree() { }

        public DecisionTree(string name)
        {
            Name = name;
        }

    }
}


using System;

namespace Modules.DecisionTree
{
    [Serializable]
    public class DecisionTreeBranch
    {
        public DecisionTreeNode Node { get; set; }

        public string Name { get; set; }

        public DecisionTreeBranch() { }

        public DecisionTreeBranch(string name)
        {
            Name = name;
        }

        public DecisionTreeBranch(string name, string nodeName)
        {
            Name = name;
            Node = new DecisionTreeNode(nodeName);
        }

    }
}


using System;
using System.Collections.Generic;
using System.Linq;

namespace Modules.DecisionTree
```

```
{
    [Serializable]
    public class DecisionTreeNode
    {

        public string Name { get; set; }

        //if no children - then leaf node
        public List<DecisionTreeBranch> Branches { get; set; }

        public bool IsRootNode { get; set; } = false;
        public bool IsLeafNode => Branches?.Any() != true && !IsRootNode;

        public bool IsDecisionNode => !IsLeafNode && !IsRootNode;

        public DecisionTreeNode() { }

        public DecisionTreeNode(string name)
        {
            Name = name;
        }

        public string GetNodeType()
        {
            if (IsRootNode) { return "Root Node"; }
            if (IsDecisionNode) { return "Decision Node"; }
            if (IsLeafNode) { return "Leaf Node"; }
            return string.Empty;
        }

        public bool IsParentOf(DecisionTreeNode child)
        {
            if (child == null)
            {
                return false;
            }

            if (Branches.IsNullOrEmpty())
            {
                return false;
            }


            return Branches.Any(b => b.Node == child);
        }

    }
}


using Modules.DecisionTree;
using System.Collections.Generic;

public static class DecisionTreeUtil
{

    /// <summary>
    /// Creates a random input list for the specified decision tree
    /// </summary>
    /// <param name="decisionTree"></param>
    /// <returns></returns>
    public static List<string> CreateRandomInputForDecisionTree(Decision-
Tree decisionTree)
    {
        var node = decisionTree.RootNode; //temp node
        var inputList = new List<string>();

        while (!node.IsLeafNode)
        {
            var randomBranchIndex = RandomUtil.GetRandomPosi-
tive(node.Branches.Count); //get random branch
            inputList.Add(node.Branches[randomBranchIndex].Name);
            node = node.Branches[randomBranchIndex].Node;
        }

        return inputList;
    }
}
```

7.1.2 Decision Tree Factories

```
using Modules.DecisionTree;
using System.Collections.Generic;

public static class DecisionTreeFactory
{
    public static DecisionTree CreateDemoHeartAttackRiskDecisionTree()
    {
        var tree = new DecisionTree("Heart Attack Risk");

        var firstBranch = new DecisionTreeBranch("<18")
        {
            Node = new DecisionTreeNode("Weight")
            {
                Branches = new List<DecisionTreeBranch>
                {
                    new DecisionTreeBranch("<=60","Low Risk"),
                    new DecisionTreeBranch(">60","High Risk")
                }
            }
        };

        var secondBranch = new DecisionTreeBranch("18-30", "Low Risk");

        var thirdBranch = new DecisionTreeBranch(">30")
        {
            Node = new DecisionTreeNode("Smoker")
            {
                Branches = new List<DecisionTreeBranch>
                {
                    new DecisionTreeBranch("no","Low Risk"),
                    new DecisionTreeBranch("yes", "High Risk")
                }
            }
        };

        tree.RootNode = new DecisionTreeNode("Age")
        {
            IsRootNode = true,
            Branches = new List<DecisionTreeBranch> { firstBranch, sec-
ondBranch, thirdBranch }
        };
        return tree;
    }
}
```

7.1.3 Decision Tree Repositories

```
using Modules.DecisionTree;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using UnityEngine;

public class DecisionTreeRepository
{

    public static void SaveToFile(DecisionTree tree)
    {
        try
        {
            var jsonTree = Newtonsoft.Json.JsonConvert.SerializeOb-
ject(tree);

            var filePath = FileIOUtil.GetSavePathForFile(tree.Name, Mod-
uleType.DecisionTree);

            Directory.CreateDirectory(Path.GetDirectoryName(filePath));

            StreamWriter writer = new StreamWriter(filePath, false);

            writer.WriteLine(jsonTree);

            writer.Close();
        }
        catch (Exception ex)
        {
            Debug.LogException(ex);
        }
    }

    public static DecisionTree LoadFromFile(string treeName)
    {
        try
        {
            var filePath = FileIOUtil.GetSavePathForFile(treeName, Module-
Type.DecisionTree);

            Directory.CreateDirectory(Path.GetDirectoryName(filePath));

            StreamReader reader = new StreamReader(filePath);

            var jsonTree = reader.ReadToEnd();

            reader.Close();

            return Newtonsoft.Json.JsonConvert.DeserializeObject<Decision-
Tree>(jsonTree);
        }
        catch (Exception ex)
        {
            Debug.LogException(ex);
            return null;
        }
    }

    public static List<string> GetAllNames()
    {
        try
        {
            var filePath = FileIOUtil.GetSavePath(ModuleType.Decision-
Tree);

            var files = Directory.GetFiles(filePath);

            //remove path keep only file name without extension
            return files.Select(f => Path.GetFileNameWithoutExten-
sion(f)).ToList();
        }
        catch (Exception ex)
        {
            Debug.LogException(ex);
            return new List<string>();
        }
    }

}
```

7.1.4 Generic Utils

```
public static class FileIOUtil
{
    public static string ConvertIllegalFileCharactersToUnderscores(string
filename)
    {
        var newFilename = filename;

        foreach (char c in System.IO.Path.GetInvalidFileNameChars())
        {
            newFilename = newFilename.Replace(c, '_');
        }

        return newFilename;
    }

    /// <summary>
    /// Converts the given filename to appropriate filename (removes ille-
gal characters) and returns the full path
    /// </summary>
    /// <param name="filename"></param>
    /// <returns></returns>
    public static string GetSavePathForFile(string filename,ModuleType
moduleType)
    {
        return
            $"{GetSavePath(moduleType)}/" +
            $"{ConvertIllegalFileCharactersToUnderscores(filename)}.json";
    }

    public static string GetSavePath(ModuleType moduleType)
    {
        return
            $"{FileSaveContants.FileSavePath}/" +
            $"{FileSaveContants.FileSaveFolderName}/" +
            $"{FileSaveContants.ModulesFolderName}/" +
            $"{ModuleTypeUtil.ModuleTypeToName(moduleType)}";
    }
}


public static class ModuleTypeUtil
{
    public static string ModuleTypeToName(ModuleType type)
    {
        return type switch
        {
            ModuleType.DecisionTree => "Decision Tree",
            ModuleType.NeuralNetwork => "Neural Network",
            _ => "Undefined",
        };
    }
}


using System;

/// <summary>
/// Utility Class to handle Random operations
/// </summary>
public static class RandomUtil
{
    private static Random rnd = new Random();
    public static int GetRandom()
    {
        return rnd.Next();
    }

    public static int GetRandomPositive(int maxValue)
    {
        return rnd.Next(maxValue);
    }

    public static int GetRandomRange(int minValue,int maxValue)
    {
        return rnd.Next(minValue, maxValue);
    }
}
```

7.1.5 Generic Extensions

```
using System.Collections.Generic;

public static class ListExtensions
{
    public static bool IsNullOrEmpty<T>(this List<T> list)
    {
        return list == null || list.Count == 0;
    }
}
```

7.1.6 Decision Tree Validators

```
using Modules.DecisionTree;

/// <summary>
/// Validator for node connection
/// </summary>
public static class DecisionTreeNodeConnectionValidator
{
    public static ValidationResult Validate(DecisionTreeNode sourceNode,
DecisionTreeNode targetNode,bool targetNodeHasParent)
    {
        //validations
        if (sourceNode == null || targetNode == null)
        {
            return new ValidationResult("Trying to connect null nodes!");
        }

        if (sourceNode == targetNode)
        {
            return new ValidationResult("Cannot connect the node to it-
self!");
        }

        //check if target node is parent of source node
        if (targetNode.IsParentOf(sourceNode))
        {
            return new ValidationResult("Target node is parent to source
node and bidirectional connection is not allowed on decision trees!");
        }

        if (targetNodeHasParent && !sourceNode.IsParentOf(targetNode))
        {
            return new ValidationResult("Cannot connect to a node that is
already connected!");
        }

        if (targetNode.IsRootNode)
        {
            return new ValidationResult("Cannot connect node to root
node!");
        }

        return new ValidationResult();

    }
}
```

7.1.7 Constants

```
using UnityEngine;

public static class FileSaveContants
{
    public static string FileSavePath = Application.persistentDataPath;
    public const string FileSaveFolderName = "Saved";
    public const string ModulesFolderName = "Modules";
}
```

7.1.8 Enums

```
public enum ModuleType
```

```
{
    DecisionTree = 1,
    NeuralNetwork = 2
}
```

7.1.9 Neural Network

```
using System;
using System.Security.Cryptography;

namespace Assets.Scripts.Classes.NeuralNetwork
{
    public class CryptoRandom
    {
        public double RandomValue { get; set; }

        public CryptoRandom()
        {
            using (RNGCryptoServiceProvider p = new RNGCryptoServicePro-
vider())
            {
                Random r = new Random(p.GetHashCode());
                this.RandomValue = r.NextDouble();
            }
        }
    }
}


namespace Assets.Scripts.Classes.NeuralNetwork
{
    public class Dendrite
    {
        public double Weight { get; set; }

        public Dendrite()
        {
            CryptoRandom n = new CryptoRandom();
            this.Weight = n.RandomValue;
        }
    }
}


using System.Collections.Generic;

namespace Assets.Scripts.Classes.NeuralNetwork
{
    public class Layer
    {
        public List<Neuron> Neurons { get; set; }
        public int NeuronCount => Neurons.Count;

        public Layer(int numNeurons)
        {
            Neurons = new List<Neuron>(numNeurons);
        }
    }
}


using System;
using System.Collections.Generic;
```

```
namespace Assets.Scripts.Classes.NeuralNetwork
{
    public class Neuron
    {
        public List<Dendrite> Dendrites { get; set; }
        public double Bias { get; set; }
        public double Delta { get; set; }
        public double Value { get; set; }

        public int DendriteCount => Dendrites.Count;

        public Neuron()
        {
            Random n = new Random(Environment.TickCount);
            this.Bias = n.NextDouble();

            this.Dendrites = new List<Dendrite>();
        }
    }
}


using System;
using System.Collections.Generic;
using System.Text;

namespace Assets.Scripts.Classes.NeuralNetwork
{
    public class NeuralNetwork
    {
        public List<Layer> Layers { get; set; }
        public double LearningRate { get; set; }
        public int LayerCount => Layers.Count;
        public string Name { get; set; }

        public NeuralNetwork(double learningRate, int[] layers)
        {
            if (layers.Length < 2) return;

            Name = "NeuralNetwork" + new Guid();


            this.LearningRate = learningRate;
            this.Layers = new List<Layer>();

            for (int l = 0; l < layers.Length; l++)
            {
                Layer layer = new Layer(layers[l]);
                this.Layers.Add(layer);

                for (int n = 0; n < layers[l]; n++)
                    layer.Neurons.Add(new Neuron());

                layer.Neurons.ForEach((nn) =>
                {
                    if (l == 0)
                        nn.Bias = 0;
                    else
                        for (int d = 0; d < layers[l - 1]; d++)
                            nn.Dendrites.Add(new Dendrite());
                });
            }
        }

        private double Sigmoid(double x)
        {
            return 1 / (1 + Math.Exp(-x));
        }

        public double[] Run(List<double> input)
        {
            if (input.Count != this.Layers[0].NeuronCount) return null;

            for (int l = 0; l < Layers.Count; l++)
            {
                Layer layer = Layers[l];

                for (int n = 0; n < layer.Neurons.Count; n++)
                {
                    Neuron neuron = layer.Neurons[n];

                    if (l == 0)
                        neuron.Value = input[n];
                    else
                    {
                        neuron.Value = 0;
                        for (int np = 0; np < this.Layers[l - 1].Neu-
rons.Count; np++)
                            neuron.Value = neuron.Value + this.Layers[l -
1].Neurons[np].Value * neuron.Dendrites[np].Weight;

                        neuron.Value = Sigmoid(neuron.Value + neu-
ron.Bias);
                    }
                }
            }

            Layer last = this.Layers[this.Layers.Count - 1];
            int numOutput = last.Neurons.Count;
            double[] output = new double[numOutput];
            for (int i = 0; i < last.Neurons.Count; i++)
                output[i] = last.Neurons[i].Value;

            return output;
        }

        public bool Train(List<double> input, List<double> output)
        {
            if ((input.Count != this.Layers[0].Neurons.Count) || (out-
put.Count != this.Layers[this.Layers.Count - 1].Neurons.Count)) return
false;

            Run(input);

            for (int i = 0; i < this.Layers[this.Layers.Count - 1].Neu-
rons.Count; i++)
            {
                Neuron neuron = this.Layers[this.Layers.Count - 1].Neu-
rons[i];

                neuron.Delta = neuron.Value * (1 - neuron.Value) * (out-
put[i] - neuron.Value);
```

```
                for (int j = this.Layers.Count - 2; j > 2; j--)
                {
                    for (int k = 0; k < this.Layers[j].Neurons.Count; k++)
                    {
                        Neuron n = this.Layers[j].Neurons[k];

                        n.Delta = n.Value *
                                  (1 - n.Value) *
                                  this.Layers[j + 1].Neurons[i].Den-
drites[k].Weight *
                                  this.Layers[j + 1].Neurons[i].Delta;
                    }
                }
            }

            for (int i = this.Layers.Count - 1; i > 1; i--)
            {
                for (int j = 0; j < this.Layers[i].Neurons.Count; j++)
                {
                    Neuron n = this.Layers[i].Neurons[j];
                    n.Bias = n.Bias + (this.LearningRate * n.Delta);

                    for (int k = 0; k < n.Dendrites.Count; k++)
                        n.Dendrites[k].Weight = n.Dendrites[k].Weight +
(this.LearningRate * this.Layers[i - 1].Neurons[k].Value * n.Delta);
                }
            }

            return true;
        }

        /// <summary>
        /// Returns a text representation of the NeuralNetwork current
state
        /// </summary>
        /// <returns></returns>
        public override string ToString()
        {
            var sb = new StringBuilder("Name: " + this.Name);
            sb.AppendLine();

            for (int l = 0; l < this.Layers.Count; l++)
            {
                sb.AppendLine("\tLayer " + l.ToString() + " (" + this.Lay-
ers[l].NeuronCount.ToString() + " neurons)");

                for (int n = 0; n < this.Layers[l].NeuronCount; n++)
                {
                    sb.AppendLine("\t\tNeuron " + n.ToString() + " (" +
this.Layers[l].Neurons[n].DendriteCount.ToString() + " dendrites)");
                    sb.AppendLine("\t\t\tBias: " + this.Layers[l].Neu-
rons[n].Bias.ToString());
                    sb.AppendLine("\t\t\tDelta: " + this.Layers[l].Neu-
rons[n].Delta.ToString());
                    sb.AppendLine("\t\t\tValue: " + this.Layers[l].Neu-
rons[n].Value.ToString());
                    sb.AppendLine("\t\t\tDendrites");

                    for (int d = 0; d < this.Layers[l].Neurons[n].Den-
driteCount; d++)
                    {
                        sb.AppendLine("\t\t\t\tDendrite " + d.ToString() +
" weight: " + this.Layers[l].Neurons[n].Dendrites[d].Weight);
                    }
                }
            }

            return sb.ToString();
        }
    }
}
```

7.2 Unity MonoBehaviour Classes

7.2.1 Generic User Interface

```
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class ConsolePanelController : MonoBehaviour
{
    public ScrollRect ScrollRect;

    private Color DefaultLogColor = Color.white;

    private Dictionary<LogType, Color> LogTypeColors = new Diction-
ary<LogType, Color>()
    {
        {LogType.Warning, Color.yellow},
        {LogType.Error, Color.red},
        {LogType.Exception, Color.red},
        {LogType.Log, Color.white},
    };

    void OnEnable()
    {
        Application.logMessageReceived += HandleLog;
    }

    void OnDisable()
    {
        Application.logMessageReceived -= HandleLog;
    }

    void HandleLog(string logString, string stackTrace, LogType logType)
    {
        CreateLogEntryUI(logString, logType);
    }

    void CreateLogEntryUI(string log,LogType logType)
    {
        //create appropriate text mesh gameobject
       CreateTextMeshProGUIObject(log, logType);

        //forces canvas ui update so that the below command works properly
        Canvas.ForceUpdateCanvases();

        //scroll to bottom (last item)
        ScrollRect.verticalNormalizedPosition = 0f;
    }
    private GameObject CreateTextMeshProGUIObject(string log, LogType
logType)
    {
        var textGameObject = new GameObject
        {
            name = "log_" + log
        };

        textGameObject.transform.parent = ScrollRect.content;

        var textMeshPro = textGameObject.AddComponent<TextMeshProUGUI>();
        textMeshPro.text = log;
        textMeshPro.alignment = TextAlignmentOptions.Left;
        textMeshPro.enableAutoSizing = true;
        textMeshPro.autoSizeTextContainer = true;
        textMeshPro.fontSizeMax = 10;
        textMeshPro.fontSizeMin = 5;


        if (LogTypeColors.TryGetValue(logType,out var textColor))
        {
            textMeshPro.color = textColor;
        }
        else
        {
            textMeshPro.color = DefaultLogColor;
        }

        return textGameObject;
    }

}

using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.Events;

public class InputPanelManager : MonoBehaviour
{

    public GameObject InputPanel;

    private TMP_InputField _mainInputField;
    private TextMeshProUGUI _titleTextMesh;
    private TMP_Dropdown _dropdown;

    private UnityAction<string> _actionToTriggerOnSubmit;
    private UnityAction _actionToTriggerAfterHideOnSubmit;

    private bool _dropdownMode = false;

    private void Awake()
    {
        _mainInputField = InputPanel.transform.Find("MainInputField").Get-
Component<TMP_InputField>();
        _titleTextMesh = InputPanel.transform.Find("InputTextPanel").Get-
ComponentInChildren<TextMeshProUGUI>();
        _dropdown = InputPanel.transform.Find("Dropdown").GetCompo-
nent<TMP_Dropdown>();
    }
    public void ShowInputPanelWithDropdown(UnityAction<string> actionTo-
TriggerOnSubmit,List<string> dropdownValues, string messageBoxTitle =
null)
    {
        _actionToTriggerOnSubmit = actionToTriggerOnSubmit;

        _dropdown.AddOptions(dropdownValues);

        _dropdownMode = true;

        _mainInputField.gameObject.SetActive(!_dropdownMode);
        _dropdown.gameObject.SetActive(_dropdownMode);

        InputPanel.SetActive(true);
    }

    public void ShowInputPanel(UnityAction<string> actionToTriggerOnSub-
mit, string messageBoxTitle = null) => ShowInputPanel(actionToTriggerOn-
Submit,null,null,messageBoxTitle);

    public void ShowInputPanel(UnityAction<string> actionToTriggerOnSub-
mit, UnityAction actionToTriggerAfterHideOnSubmit = null, string over-
rideInput = null,string messageBoxTitle = null)
    {
        _actionToTriggerOnSubmit = actionToTriggerOnSubmit;
        _actionToTriggerAfterHideOnSubmit = actionToTriggerAfterHideOnSub-
mit;

        //if contains a value
        if (!string.IsNullOrWhiteSpace(overrideInput))
        {
            _mainInputField.text = overrideInput;
        }

        //if contains a value
        if (!string.IsNullOrWhiteSpace(messageBoxTitle))
        {
            _titleTextMesh.text = messageBoxTitle;
        }
        else
        {
            _titleTextMesh.text = "Enter value";
        }

        _dropdownMode = false;

        _mainInputField.gameObject.SetActive(!_dropdownMode);
        _dropdown.gameObject.SetActive(_dropdownMode);

        InputPanel.SetActive(true);

    }

    public void OnMainInputPanelButtonCancelClick() => CloseInput-
PanelAndClearMainInputFieldAndDropdown();

    public void OnMainInputPanelButtonSubmitClick()
    {
        string actionStringParam;

        if (!_dropdownMode)
        {
            var inputText = _mainInputField.text.Trim();

            //should implement throwing a message box to inform user of
validation error
            if (string.IsNullOrEmpty(inputText))
            {
                Debug.LogWarning("Input Panel Submit clicked without in-
put");
                return;
            }

            actionStringParam = inputText;
        }
        else
        {
            actionStringParam = _dropdown.options[_dropdown.value].text;
        }

        //invoke action and clear
        _actionToTriggerOnSubmit.Invoke(actionStringParam);
        _actionToTriggerOnSubmit = null;

        CloseInputPanelAndClearMainInputFieldAndDropdown();

        if (_actionToTriggerAfterHideOnSubmit != null)
        {
            _actionToTriggerAfterHideOnSubmit.Invoke();
            _actionToTriggerAfterHideOnSubmit = null;
        }

    }

    private void CloseInputPanelAndClearMainInputFieldAndDropdown()
    {
        InputPanel.SetActive(false);
        _dropdown.ClearOptions();
        _mainInputField.text = string.Empty;
    }

}
```

7.2.2 Generic Helpers

```
using UnityEngine;

public class CameraMovement : MonoBehaviour
{
    public float MoveSpeed = 30;
    public float ZoomSpeed = 10;

    public float ZoomSpeedOrthographic = 0.5f;

    public float mouseSensitivity  = 01.0f;

    private Vector3 lastPosition;
    private Camera _camera;

    private void Awake()
    {
        _camera = GetComponent<Camera>();
    }
    void Update()
    {
        //keyboard movement
        float xAxisValue = Input.GetAxis("Horizontal") * MoveSpeed *
Time.deltaTime;
        float yAxisValue = Input.GetAxis("Vertical") * MoveSpeed *
Time.deltaTime;
        float zAxisValue = Input.mouseScrollDelta.y * ZoomSpeed *
Time.deltaTime;

        if (_camera.orthographic == true)
        {
            var newSize = _camera.orthographicSize - ZoomSpeedOrthographic
* zAxisValue;
            if (newSize > 2)
            {
                _camera.orthographicSize = newSize;
            }
        }
        else
        {
            transform.position = new Vector3(transform.position.x + xAxis-
Value, transform.position.y + yAxisValue, transform.position.z + zAxis-
Value);
        }

        //mouse panning
        if (Input.GetMouseButtonDown(2))
        {
            lastPosition = Input.mousePosition;
        }

        if (Input.GetMouseButton(2))
        {
            var delta = Input.mousePosition - lastPosition;
            transform.Translate(-delta.x * mouseSensitivity, -delta.y *
mouseSensitivity, 0);
            lastPosition = Input.mousePosition;
        }

    }
}


using cakeslice;
using UnityEngine;

public class ObjectSelector : MonoBehaviour
{
    public Color SelectedObjectColor = Color.blue;
    public GameObject SelectedObject;
    private bool _isMouseDragging = false;

    // Update is called once per frame
    void Update()
    {

        bool isOverUI = UnityEngine.EventSystems.EventSystem.cur-
rent.IsPointerOverGameObject();

        if (isOverUI)
        {
            return;
        }

        if (Input.GetKeyDown(KeyCode.Delete))
        {
            if (SelectedObject != null)
            {
                FindObjectOfType<DecisionTreeDesignerController>().Remove-
SelectedNode();
                SelectObject(null);
            }
        }

        if (Input.GetMouseButtonDown(0))
        {

        }

        //right click - deselect or link nodes
        if (Input.GetMouseButtonDown(1))
        {
            bool hit = Physics.Raycast(Camera.main.ScreenPointToRay(In-
put.mousePosition), out var hitInfo);

            if (hit && hitInfo.transform.gameObject.tag.Equals("Se-
lectable"))
            {
                var targetTreeNodeScript = hitInfo.transform.gameOb-
ject.GetComponent<DecisionTreeNodeMono>();

                if (SelectedObject != null && targetTreeNodeScript !=
null)
                {
                    var sourceTreeNodeScript = SelectedObject.GetCompo-
nent<DecisionTreeNodeMono>();

                    if (sourceTreeNodeScript != null)
                    {
                        FindObjectOfType<DecisionTreeDesignerControl-
ler>().ConnectNodes(sourceTreeNodeScript, targetTreeNodeScript);
                    }
                }
            }

            SelectObject(null);

        }

        //left click - main selection
        if (Input.GetMouseButtonDown(0))
        {
            bool hit = Physics.Raycast(Camera.main.ScreenPointToRay(In-
put.mousePosition),out var hitInfo);

            if (hit)
            {
                if (!hitInfo.transform.gameObject.tag.Equals( "Se-
lectable"))
                {
                    return;
                }
                SelectObject(hitInfo.transform.gameObject);
                _isMouseDragging = true;
            }
            else
            {
                SelectObject(null);
            }

        }

        if (Input.GetMouseButtonUp(0))
        {
            _isMouseDragging = false;
        }

        if (_isMouseDragging && SelectedObject != null)
        {
            var mousePosition = Camera.main.ScreenToWorldPoint(In-
put.mousePosition);
            SelectedObject.transform.position = new Vector3(mousePosi-
tion.x, mousePosition.y, SelectedObject.transform.position.z);
        }

    }

    void SelectObject(GameObject selectedGameObject)
    {
        if (SelectedObject == selectedGameObject)
        {
            return;
        }

        //deselect previous object if any
        if (SelectedObject != null)
        {
            //old way that changes object's material
            //change color of prev selected item to default state
            //_currentSelectedObject.GetComponent<Renderer>().mate-
rial.SetColor("_Color", _currentSelectedObjectColor);

            //since outline component is used, just destroy it
            Destroy(SelectedObject.GetComponent<Outline>());


            SelectAllLineRenderers(SelectedObject, false);

            //if same object deselect
            if (SelectedObject == selectedGameObject)
            {
                SelectedObject = null;
                return;
            }
        }

        SelectedObject = selectedGameObject;

        if (SelectedObject == null)
        {
            return;
        }
        //old way that changes object's material
        // Get the Renderer component from the new cube
        //var renderer = _currentSelectedObject.GetComponent<Renderer>();

        //_currentSelectedObjectColor = renderer.material.color;

        //// Call SetColor using the shader property name "_Color" and
setting the color to red
        //renderer.material.SetColor("_Color",SelectedObjectColor);

        //since outline component is used, add it
        SelectedObject.AddComponent<Outline>();

        SelectAllLineRenderers(SelectedObject, true);
    }

    void SelectAllLineRenderers(GameObject gameObject,bool selected)
    {
        var lineRenderers = gameObject.GetComponentsInChildren<LineRen-
derer>();

        foreach (var lineRenderer in lineRenderers)
        {
            var lineColor = selected ? Color.red : Color.yellow;

            lineRenderer.material.color = lineColor;
            lineRenderer.material.color = lineColor;
        }
    }

}
```

7.2.3 Decision Tree

```
using Modules.DecisionTree;
using TMPro;
using UnityEngine;

public class DecisionTreeLine : MonoBehaviour
{

    public DecisionTreeBranch TreeBranch { get; set; }

    public DecisionTreeNodeMono SourceNode { get; set; }

    public DecisionTreeNodeMono TargetNode { get; set; }

    private LineRenderer _lineRenderer;

    private Transform _textMeshProTransform;

    private void Awake()
    {
        _lineRenderer = gameObject.AddComponent<LineRenderer>();

        _lineRenderer.material = (Material)Resources.Load("Materials/Den-
driteMat", typeof(Material));
        _lineRenderer.widthMultiplier = 0.2f;
        _lineRenderer.positionCount = 2;
        ResetColors();
    }
    public void Initialize(Vector3 posA, Vector3 posB, DecisionTreeBranch
branch, DecisionTreeNodeMono sourceNodeScript)
    {
        SourceNode = sourceNodeScript;
        name = "branch_"+ branch.Name;
        transform.position = posA;

        TreeBranch = branch;

        _lineRenderer.SetPosition(0, posA);
        _lineRenderer.SetPosition(1, posB);


        var textPosition = (posB - posA) / 2 + posA;

        _textMeshProTransform = CreateTextMeshProObject(
            textPosition,
            transform,
            branch.Name).transform;
    }

    // Update is called once per frame
    void Update()
    {
        if (SourceNode != null)
        {
            _lineRenderer.SetPosition(0, SourceNode.transform.position);
        }

        if (TargetNode != null)
        {
            _lineRenderer.SetPosition(1, TargetNode.transform.position);
        }

        if (SourceNode == null || TargetNode == null)
        {
            return;
        }

        _textMeshProTransform.position = (TargetNode.transform.position -
SourceNode.transform.position) / 2 + SourceNode.transform.position;

    }

    public void ResetColors()
    {
        _lineRenderer.startColor = Color.white;
        _lineRenderer.endColor = Color.red;
    }

    private GameObject CreateTextMeshProObject(Vector3 position, Transform
parent, string branchName)
    {
        var textGameObject = new GameObject();
        textGameObject.name = "branch_text" + branchName;
        textGameObject.transform.position = position;
        textGameObject.transform.parent = parent;

        var textMeshPro = textGameObject.AddComponent<TextMeshPro>();
        textMeshPro.text = branchName;
        textMeshPro.color = Color.black;
        textMeshPro.alignment = TextAlignmentOptions.Center;
        textMeshPro.enableAutoSizing = true;
        textMeshPro.autoSizeTextContainer = true;
        textMeshPro.fontSizeMax = 10;
        textMeshPro.fontSizeMin = 5;

        return textGameObject;
    }
}


using Modules.DecisionTree;
using System.Collections.Generic;
using System.Linq;
using TMPro;
using UnityEngine;

public class DecisionTreeInputDataController : MonoBehaviour
{
    public List<string> InputList { get; private set; } = new
List<string>();

    List<GameObject> _inputListGameObjects = new List<GameObject>();

    private DecisionTreeDesignerController _designerController;

    private InputPanelManager _inputPanelManager;

    private GameObject _inputDataContent;


    // Start is called before the first frame update
    void Start()
    {
        _designerController = GetComponent<DecisionTreeDesignerControl-
ler>();
        _inputPanelManager = FindObjectOfType<InputPanelManager>();
        _inputDataContent = GameObject.Find("InputDataContent");
    }

    public void OnGenerateRandomInputDataButtonClick()
    {
        if (_designerController.DecisionTree == null)
        {
            //add error handling
            Debug.LogWarning("Decision Tree cannot be empty.");
            return;
        }

        var inputList = DecisionTreeUtil.CreateRandomInputForDecision-
Tree(_designerController.DecisionTree);

        ClearUp();

        foreach (var input in inputList)
        {
            InputList.Add(input);
            _inputListGameObjects.Add(CreateTextMeshProGUIObject("input_"
+ input, _inputDataContent.transform, input));
        }
    }

    public void OnClearAllInputDataButtonClick() => ClearUp();
    public void OnClearLastInputDataButtonClick()
    {
        if (InputList.IsNullOrEmpty())
        {
            Debug.LogWarning("No input data found!");
            return;
        }

        InputList.RemoveAt(InputList.Count - 1);
        Destroy(_inputListGameObjects[_inputListGameObjects.Count - 1]);
        _inputListGameObjects.RemoveAt(_inputListGameObjects.Count - 1);

    }

    public void OnAddInputDataButtonClick()
    {
        if (_designerController.DecisionTree == null)
        {
            //add error handling
            Debug.LogWarning("Decision Tree cannot be empty.");
            return;
        }

        var availableInputs = GetAvailableInputs();

        if (availableInputs.IsNullOrEmpty())
        {
            Debug.LogWarning("No available inputs found!");
            return;
        }

        _inputPanelManager.ShowInputPanelWithDropdown(
            selectedInput =>
            {
                Debug.Log("Input " + selectedInput + " was added");
                InputList.Add(selectedInput);
                _inputListGameObjects.Add(CreateTextMeshProGUIObject("in-
put_" + selectedInput, _inputDataContent.transform, selectedInput));
            },
            availableInputs,
            "Select an input (branch)"
            );
    }

    public void ClearUp()
    {
        if (InputList.Count <= 0)
        {
            return;
        }

        Debug.Log("Clearing up Input Data List......");

        _inputListGameObjects.ForEach(go => Destroy(go));
        _inputListGameObjects.Clear();

        InputList.Clear();
    }

    private List<string> GetAvailableInputs()
    {
        var availableInputs = new List<string>();

        var decisionTree = _designerController.DecisionTree;

        //if list empty, then show branches of root node
        if (InputList.IsNullOrEmpty())
        {
            return _designerController.DecisionTree.RootNode.Branches.Se-
lect(b=> b.Name).ToList();
        }

        //if more than one input, we need to iterate each node of tree to
find current node and availabe options(brances)

        DecisionTreeNode currentNode = _designerController.Decision-
Tree.RootNode;

        for (int i=0;i<InputList.Count;i++)
        {
            var branch = currentNode.Branches.FirstOrDefault(b => b.Name
== InputList[i]);

            //if branch exists based on input, change current iterated
node
            if (branch != null)
            {
                currentNode = branch.Node;

                if (i == InputList.Count - 1) //if this is the last added
element of list
                {
                    return currentNode?.Branches?.Select(b =>
b.Name).ToList();
                }

                continue;
            }

        }

        return availableInputs;
    }

    private GameObject CreateTextMeshProGUIObject(string objectName,
Transform parent, string text)
    {
        var textGameObject = new GameObject();
        textGameObject.name = objectName;
        textGameObject.transform.parent = parent;

        var textMeshPro = textGameObject.AddComponent<TextMeshProUGUI>();
        textMeshPro.text = text;
        textMeshPro.color = Color.white;
        textMeshPro.alignment = TextAlignmentOptions.Center;
        textMeshPro.enableAutoSizing = true;
        textMeshPro.autoSizeTextContainer = true;
        textMeshPro.fontSizeMax = 30;
        textMeshPro.fontSizeMin = 5;

        return textGameObject;
    }
```

```
}


using Modules.DecisionTree;
using TMPro;
using UnityEngine;
using System.Collections.Specialized;
using System.Collections.ObjectModel;
using System.Linq;

public class DecisionTreeNodeMono : MonoBehaviour
{
    public DecisionTreeNode TreeNode { get; set; }
    public ObservableCollection<DecisionTreeLine> OutgoingBranchScripts {
get; set; } = new();
    public DecisionTreeNodeMono ParentNodeScript;

    public Color RootNodeColor = Color.red;
    public Color DecisionNodeColor = Color.yellow;
    public Color LeafNodeColor = Color.green;


    private void Awake()
    {
        OutgoingBranchScripts.CollectionChanged += OnDecisionTree-
BranchScriptsModified;
    }

    public void Initialize(DecisionTreeNode node)
    {
        TreeNode = node;
        UpdateNodeName(node.Name);
        UpdateColorStatus();
    }

    private void OnDecisionTreeBranchScriptsModified(object sender, Noti-
fyCollectionChangedEventArgs e) => UpdateColorStatus();

    public void UpdateColorStatus()
    {
        if (TreeNode.IsLeafNode)
        {
            SetColor(LeafNodeColor);
        }
        else if (TreeNode.IsRootNode)
        {
            SetColor(RootNodeColor);
        }
        else if (TreeNode.IsDecisionNode)
        {
            SetColor(DecisionNodeColor);
        }
        else
        {
            SetColor(Color.white);
        }

    }

    public void UpdateNodeName(string newNodeName)
    {
        TreeNode.Name = newNodeName;
        gameObject.name = "node_" + newNodeName;
        GetComponentInChildren<TextMeshPro>().text = newNodeName;

    }

    /// <summary>
    /// Manually call destroy; Should not use OnDestroy event since it's
triggered more times than wanted
    /// </summary>
    public void Destroy()
    {
        Debug.Log("Destroying node: " + TreeNode.Name);

        //clean up parent node
        if (ParentNodeScript != null)
        {
            //remove connection with parent
            ParentNodeScript.DeconnectNode(this);
        }

        //cleanup all sub children and branches
        foreach (var branchScript in OutgoingBranchScripts.ToList())
        {
            branchScript.TargetNode.Destroy();
        }

        Destroy(gameObject);
    }

    /// <summary>
    /// Removes connection with specified node
    /// </summary>
    /// <param name="targetNode"></param>
    public void DeconnectNode(DecisionTreeNodeMono targetNode)
    {
        if (targetNode == null)
        {
            Debug.LogWarning("Cannot deconnect null node!");
        }

        var branchScript = OutgoingBranchScripts.FirstOrDefault(l =>
l.TargetNode == targetNode);

        if (branchScript == null)
        {
            Debug.Log("No branch connected on node: " + TreeNode.Name);
            return;
        }

        Debug.Log("Destroying connected branch: " + branchScript.Tree-
Branch.Name);

        //cleanup memory branch
        TreeNode.Branches.Remove(branchScript.TreeBranch);

        //remove branch script from parent node
        OutgoingBranchScripts.Remove(branchScript);

        targetNode.ParentNodeScript = null;

        //destroy branch gameobject
        Destroy(branchScript.gameObject);
    }

    private void SetColor(Color color)
    {
        // Get the Renderer component from the new cube
        var renderer = GetComponent<Renderer>();

        // Call SetColor using the shader property name "_Color" and set-
ting the color to red
        renderer.material.SetColor("_Color", color);
    }

}


using cakeslice;
using Modules.DecisionTree;
using System.Collections.Generic;
using System.Linq;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

/// <summary>
/// Used to control all simulation related actions including generated in-
put
/// </summary>
public class DecisionTreeSimulationController : MonoBehaviour
{
    public ScrollRect SimulationScrollRect;

    private DecisionTreeDesignerController _designerController;
    private DecisionTreeInputDataController _inputDataController;
    private DecisionTree DecisionTree => _designerController.DecisionTree;

    private bool _simulationMode = false;

    private DecisionTreeNode _currentNode = null;
    private int _currentInputValueIndex;
    private GameObject _currentNodeGameObject = null;

    private List<GameObject> _simulatedGameObjects = new List<GameOb-
ject>();

    // Start is called before the first frame update
    void Start()
    {
        _designerController = GetComponent<DecisionTreeDesignerControl-
ler>();
        _inputDataController = GetComponent<DecisionTreeInputDataControl-
ler>();
    }

    public void OnSimulateButtonClick()
    {
        if (_simulationMode)
        {
            Debug.LogWarning("Reset simulation in order to start it
again.");
            return;
        }
        if (DecisionTree == null)
        {
            Debug.LogWarning("Decision Tree cannot be empty.");
            return;
        }

        //if empty input list
        if (_inputDataController.InputList.IsNullOrEmpty())
        {
            Debug.LogWarning("Input list cannot be empty.");
            return;
        }

        _simulatedGameObjects.Clear();//should adjust to clear components
        _currentInputValueIndex = 0;

        UpdateCurrentNode(DecisionTree.RootNode, _designerController.Deci-
sionTreeGameObject.transform.GetChild(0).gameObject);

        CreateTextMeshProGUIObject("sim_input_text_" + _currentNode.Name,
SimulationScrollRect.content.transform, $"{_currentNode.GetNodeType()}:
{_currentNode.Name}");

        _simulationMode = true;

    }

    public void OnResetSimulationClick()
    {

        if (!_simulationMode)
        {
            Debug.LogWarning("Must be in simulation mode in order to stop
it.");
            return;
        }

        //clean up ui
        foreach (Transform child in SimulationScrollRect.content.trans-
form)
        {
            Destroy(child.gameObject);
        }

        foreach (var go in _simulatedGameObjects)
        {
            var treeLineScript = go.GetComponent<DecisionTreeLine>();

            if (treeLineScript != null)
            {
                treeLineScript.ResetColors();
                continue;
            }

            Destroy(go.GetComponent<cakeslice.Outline>());
        }

        _simulatedGameObjects.Clear();//should adjust to clear components
        _currentInputValueIndex = 0;

        _simulationMode = false;
    }
    public void OnSimulatePreviousStepButtonClick()
    {
        if (!_simulationMode)
        {
            //add error handling
            Debug.LogWarning("Must be in simulation mode.");
            return;
        }

        if (_currentInputValueIndex == 0) //first element exit
        {
            //add error handling
            Debug.LogWarning("This is the first step cannot go further
back!");
            return;
        }

        var treeLineScript = _currentNodeGameObject.GetComponent<Decision-
TreeLine>();

        if (treeLineScript == null) //if last selected object is node ob-
ject
        {
            var treeNodeScript = _currentNodeGameObject.GetComponent<Deci-
sionTreeNodeMono>();

            Destroy(treeNodeScript.GetComponent<cakeslice.Outline>());

            _simulatedGameObjects.RemoveAt(_simulatedGameObjects.Count -
1);
            _currentNodeGameObject = _simulatedGameObjects.Last();

        }
        else //if last selected object is line object
        {
            treeLineScript.ResetColors();

            _simulatedGameObjects.RemoveAt(_simulatedGameObjects.Count -
1);
            _currentNodeGameObject = _simulatedGameObjects.Last();

            _currentInputValueIndex--;

        }

        Destroy(SimulationScrollRect.content.transform.GetChild(Simula-
tionScrollRect.content.transform.childCount - 1).gameObject);

    }

    public void OnSimulateNextStepButtonClick()
    {
        if (!_simulationMode)
        {
            //add error handling
            Debug.LogWarning("Must be in simulation mode.");
            return;
        }

        if (_currentInputValueIndex == _inputDataController.Input-
List.Count) //last element exit
```

```
        {
            //add error handling
            Debug.LogWarning("This is the final step cannot go further!");
            return;
        }

        var inputValue = _inputDataController.InputList[_currentInput-
ValueIndex]; //branch value

        var treeLineScript = _currentNodeGameObject.GetComponent<Decision-
TreeLine>();

        if (treeLineScript == null) //if last selected object is node ob-
ject
        {
            var treeNodeScript = _currentNodeGameObject.GetComponent<Deci-
sionTreeNodeMono>();

            if (!treeNodeScript.TreeNode.IsLeafNode)
            {
                var nextBranchGameObject = treeNodeScript.Out-
goingBranchScripts.First(s => s.TreeBranch.Name == inputValue);
                AddBranchLineToSimulationList(nextBranchGameObject.gameOb-
ject);
                CreateTextMeshProGUIObject("sim_input_text_"+ next-
BranchGameObject.TreeBranch.Name, SimulationScrollRect.content.trans-
form,"Branch: "+ nextBranchGameObject.TreeBranch.Name);
            }
        }
        else //if last selected object is line object
        {
            UpdateCurrentNode(treeLineScript.TreeBranch.Node, treeLine-
Script.TargetNode.gameObject);
            CreateTextMeshProGUIObject("sim_input_text_" + _current-
Node.Name, SimulationScrollRect.content.transform, $"{_current-
Node.GetNodeType()}: {_currentNode.Name}");
            _currentInputValueIndex++;

        }

    }

    private void AddBranchLineToSimulationList(GameObject lineGameObject)
    {
        lineGameObject.GetComponent<LineRenderer>().startColor =
Color.red;
        lineGameObject.GetComponent<LineRenderer>().endColor = Color.red;
        _currentNodeGameObject = lineGameObject;
        _simulatedGameObjects.Add(lineGameObject);
    }


    private void UpdateCurrentNode(DecisionTreeNode treeNode, GameObject
treeNodeGameObject)
    {
        _currentNode = treeNode;
        _currentNodeGameObject = treeNodeGameObject;

        _currentNodeGameObject.AddComponent<cakeslice.Outline>();

        _simulatedGameObjects.Add(_currentNodeGameObject);
    }

    private GameObject CreateTextMeshProGUIObject(string objectName,
Transform parent, string text)
    {
        var textGameObject = new GameObject();
        textGameObject.name = objectName;
        textGameObject.transform.parent = parent;

        var textMeshPro = textGameObject.AddComponent<TextMeshProUGUI>();
        textMeshPro.text = text;
        textMeshPro.color = Color.white;
        textMeshPro.alignment = TextAlignmentOptions.Left;
        textMeshPro.enableAutoSizing = true;
        textMeshPro.autoSizeTextContainer = true;
        textMeshPro.fontSizeMax = 15;
        textMeshPro.fontSizeMin = 5;

        return textGameObject;
    }


}

using Modules.DecisionTree;
using System.Collections.Generic;
using UnityEngine;

[RequireComponent(typeof(DecisionTreeSimulationController))]
[RequireComponent(typeof(DecisionTreeInputDataController))]
public class DecisionTreeDesignerController : MonoBehaviour
{
    public DecisionTree DecisionTree { get; private set; }

    public GameObject DecisionTreeGameObject { get; private set; } = null;

    public GameObject NodePrefab;

    public float XDistance = 6f;
    public float YDistance = 6f;

    private InputPanelManager _inputPanelManager;

    private DecisionTreeNodeMono _rootNodeObject;
    private DecisionTreeInputDataController _treeInputDataController;

    // Start is called before the first frame update
    void Start()
    {
        _inputPanelManager = FindObjectOfType<InputPanelManager>();
        _treeInputDataController = GetComponent<DecisionTreeInputDataCon-
troller>();
    }

    public void OnSaveTreeButtonClick()
    {
        //show new node input box and add node
        _inputPanelManager.ShowInputPanel(
            inputTreeName =>
            {
                DecisionTree.Name = inputTreeName;
                DecisionTreeRepository.SaveToFile(DecisionTree);
                Debug.Log("Tree saved with name: " + inputTreeName);
            },
            null,
            DecisionTree.Name,
            "Enter tree name (don't modify if you want to override exist-
ing tree file)"
            );
    }

    public void OnLoadTreeButtonClick()
    {

        var treeNames = DecisionTreeRepository.GetAllNames();

        if (treeNames.IsNullOrEmpty())
        {
            Debug.LogWarning("No saved trees found!");
            return;
        }

        //show new node input box and add node
        _inputPanelManager.ShowInputPanelWithDropdown(
            enteredTreeName =>
            {
                var loadedTree = DecisionTreeRepository.LoadFromFile(en-
teredTreeName);

                if (loadedTree == null)
                {
                    Debug.LogError("Error loading decision tree with name
" + enteredTreeName);
                    return;
                }

                Debug.Log("Tree loaded with name: " + enteredTreeName);

                ClearAll();

                DecisionTree = loadedTree;

                DrawTree();
            },
            treeNames,
            "Select a tree to load"
            );
    }

    public void OnClearAllButtonClick() => ClearAll();

    public void OnRedrawButtonClick()
    {
        if ( DecisionTree == null )
        {
            Debug.Log("No tree found!");
            return;
        }

        ClearAll(true); //remove only gameobjects
        DrawTree();

    }
    /// <summary>
    /// Removes all tree related game objects
    /// </summary>
    private void ClearAll(bool onlyGameObjects = false)
    {
        if (DecisionTreeGameObject != null)
        {
            Debug.Log("Removing Decision Tree GameObjects...");
            Destroy(DecisionTreeGameObject);
        }

        _treeInputDataController.ClearUp();

        if (!onlyGameObjects)
        {
            Debug.Log("Removing memory Decision Tree...");
            DecisionTree = null;
        }
    }

    private void DrawTree()
    {
        CreateDecisionTreeGameObject();

        Debug.Log("Drawing tree...");
        Vector3 rootNodePosition = Vector3.zero;
        DrawNodes(DecisionTree.RootNode, rootNodePosition);
    }

    private void CreateDecisionTreeGameObject()
    {
        DecisionTreeGameObject = new GameObject("DecisionTree");
        DecisionTreeGameObject.transform.position = Vector3.zero;
    }

    public void OnGenerateDemoButtonClick()
    {
        ClearAll();

        DecisionTree = DecisionTreeFactory.CreateDemoHeartAttackRiskDeci-
sionTree();

        DrawTree();

    }

    private void DrawNodes(DecisionTreeNode node, Vector3 currentPosition,
Transform parentTransform = null,DecisionTreeLine prevLine = null)
    {

        var nodeGameObject = Instantiate(NodePrefab);

        nodeGameObject.transform.position = currentPosition;

        var nodeScript = nodeGameObject.GetComponent<Decision-
TreeNodeMono>();

        nodeScript.Initialize(node);

        nodeGameObject.transform.parent = DecisionTreeGameObject.trans-
form;
```

```
        if (node.IsRootNode)
        {
            _rootNodeObject = nodeScript;
        }

        if (prevLine != null)
        {
            nodeScript.ParentNodeScript = prevLine.SourceNode;
            prevLine.TargetNode = nodeScript;
        }

        if (node.IsLeafNode)
        {
            return;
        }

        for (int i = 0; i < node.Branches.Count; i++)
        {
            var newPos = currentPosition + new Vector3(-
((float)(node.Branches.Count - 1)/ 2)*XDistance + i* XDistance, -YDis-
tance, 0);

            var lineGameObject = new GameObject();
            var lineScript = lineGameObject.AddComponent<Decision-
TreeLine>();

            lineScript.Initialize(currentPosition, newPos,
node.Branches[i], nodeScript);

            lineGameObject.transform.parent = DecisionTreeGameOb-
ject.transform;

            nodeScript.OutgoingBranchScripts.Add(lineScript);

            DrawNodes(node.Branches[i].Node, newPos, nodeGameObject.trans-
form, lineScript);
        }
    }

    #region NodeActions

    private DecisionTreeNodeMono AddNode(string nodeName)
    {

        //generate empty decision tree if none
        if (DecisionTreeGameObject == null)
        {
            CreateDecisionTreeGameObject();
        }

        DecisionTree ??= new DecisionTree();

        var nodeGameObject = Instantiate(NodePrefab);
        nodeGameObject.transform.position = Vector3.zero;
        var nodeScript = nodeGameObject.GetComponent<Decision-
TreeNodeMono>();

        var treeNode = new DecisionTreeNode(nodeName);

        //if tree has no root yet, make the new node root
        if (DecisionTree.RootNode == null)
        {
            DecisionTree.RootNode = treeNode;
            treeNode.IsRootNode = true;
        }

        nodeScript.Initialize(treeNode);

        nodeGameObject.transform.parent = DecisionTreeGameObject.trans-
form;

        return nodeScript;
    }

    private void ConnectNodeToSelectedNode(DecisionTreeNodeMono node)
    {
        // if a node is selected, place new node below it and connect the
nodes
        var selectedNode = GetSelectedNode();

        if (selectedNode == null)
        {
            return;
        }

        ConnectNodes(selectedNode, node);

        node.transform.position += selectedNode.transform.position + new
Vector3(0, -YDistance, 0);
    }

    public void OnAddNodeButtonClick()
    {
        DecisionTreeNodeMono addedNodeScript = null;

        //show new node input box and add node
        _inputPanelManager.ShowInputPanel(
            newNodeName =>
            {
                addedNodeScript = AddNode(newNodeName);
            },
            () => //action todo after node insert
            {
                ConnectNodeToSelectedNode(addedNodeScript);
            },
            null,
            "Enter node name"
            );

    }

    public void OnRenameNodeButtonClick()
    {

        var selectedNode = GetSelectedNode();

        if (!selectedNode)
        {
            Debug.LogWarning("Rename node button was clicked without se-
lecting node.");
            return;
        }

        var selectedDecisionTreeNodeName = selectedNode?.TreeNode.Name;
        _inputPanelManager.ShowInputPanel(
            newNodeName =>
            {
                var selectedNode = GetSelectedNode();
                selectedNode.UpdateNodeName(newNodeName);
            },
            null,
            selectedDecisionTreeNodeName,
            "Enter node name");
    }

    public void OnRemoveNodeButtonClick()
    {
        RemoveSelectedNode();
    }

    public void RemoveSelectedNode()
    {
        var selectedNode = GetSelectedNode();

        if (selectedNode == null)
        {
            Debug.Log("No node is selected.");
            return;
        }

        if (selectedNode.TreeNode == DecisionTree.RootNode)
        {
            Debug.Log("Cannot delete root node");
            return;
        }

        Debug.Log($"Removing selected node: {selectedNode.name}.");

        selectedNode.Destroy();
    }

    public void OnSetRootNodeButtonClick()
    {
        var selectedNode = GetSelectedNode();

        if (GetSelectedNode() == null)
        {
            return;
        }

        //reset old root node (if any) to normal
        if (_rootNodeObject)
        {
            _rootNodeObject.TreeNode.IsRootNode = false;
            _rootNodeObject.UpdateColorStatus();
        }

        //convert to new root node
        selectedNode.TreeNode.IsRootNode = true;
        DecisionTree.RootNode = selectedNode.TreeNode;
        _rootNodeObject = selectedNode;
        _rootNodeObject.UpdateColorStatus();
    }

    private DecisionTreeNodeMono GetSelectedNode()
```

```
    {
        var objectSelector = FindAnyObjectByType<ObjectSelector>();

        if (objectSelector == null || objectSelector.SelectedObject ==
null)
        {
            return null;
        }

        var selectedNode = objectSelector.SelectedObject.GetComponent<De-
cisionTreeNodeMono>();

        return selectedNode;

    }

    public void ConnectNodes(DecisionTreeNodeMono sourceNode, Decision-
TreeNodeMono targetNode)
    {

        var validationResult = DecisionTreeNodeConnectionValidator.Vali-
date(sourceNode.TreeNode, targetNode.TreeNode, targetNode.ParentNodeScript
!= null);

        if (!validationResult.Success)
        {
            Debug.LogWarning(validationResult.ErrorMessage);
            return;
        }

        //if source node already connected to target node deconnect them
        if (targetNode.ParentNodeScript == sourceNode)
        {
            sourceNode.DeconnectNode(targetNode);
            return;
        }

        _inputPanelManager.ShowInputPanel(
            newBranchName =>
            {

                var newBranch = new DecisionTreeBranch(newBranchName);

                if (sourceNode.TreeNode.Branches == null)
                {
                    sourceNode.TreeNode.Branches = new List<DecisionTree-
Branch>();
                }

                newBranch.Node = targetNode.TreeNode;

                sourceNode.TreeNode.Branches.Add(newBranch);

                var lineScript = new GameObject().AddComponent<Decision-
TreeLine>();
                lineScript.Initialize(sourceNode.transform.position, tar-
getNode.transform.position, newBranch, sourceNode);

                lineScript.transform.parent = DecisionTreeGameOb-
ject.transform;

                lineScript.TargetNode = targetNode;
                sourceNode.OutgoingBranchScripts.Add(lineScript);

                targetNode.ParentNodeScript = sourceNode;
            },null,null, "Enter branch value");
    }

    #endregion
}
```

References

[1] https://www.ibm.com/topics/neural-networks.

[2] https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/artificial-intelligence-vs-machine-learning/#introduction.

[3] https://www.v7labs.com/blog/neural-networks-activation-functions.

[4] https://en.wikipedia.org/wiki/Unified_Modeling_Language.

[5] https://medium.com/@smagid_allThings/uml-class-diagrams-tutorial-step-by-step-520fd83b300b.

[6] https://en.wikipedia.org/wiki/Game_engine.

[7] Unity, “Unity3d,” https://unity.com/.

[8] https://en.wikipedia.org/wiki/Unity_(game_engine).

[9] https://www.unrealengine.com/en-US/unreal-engine-5.

[10] https://github.com/lutzroeder/netron.

[11] https://tensorspace.org/.

[12] https://gephi.org/.

[13] https://playground.tensorflow.org.

[14] https://alexlenail.me/NN-SVG/index.html.

[15] https://en.wikipedia.org/wiki/Decision_tree.

[16] https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design.

[17] https://en.wikipedia.org/wiki/Argus_Panoptes.

[18] "Github," https://github.com/JamesNK/Newtonsoft.Json.