Graph and the World of Brains

Edwin Lunando/13509024
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
edwinlunando@gmail.com

***Abstract*—the paper is analyzing about the massive use of graph that contribute significantly to the improvement of artificial intelligence. The paper starts with the AI behavior structure that most of them are visualized with graph. When a case became so sophisticated because there are so many sub-cases in the implementation of AI, the graph itself could simplify the design of its case. Moreover, we could design such fuzzy logic into a simple one. Then this paper will talk about the logic of path finding in AI. There are many ways to find the best way to reach the destination but this paper would explain the importance of using graph to obtain the best way.**

***Index Terms*—behavior, conceptual graph, path finding, artificial intelligence**

I. INTRODUCTION

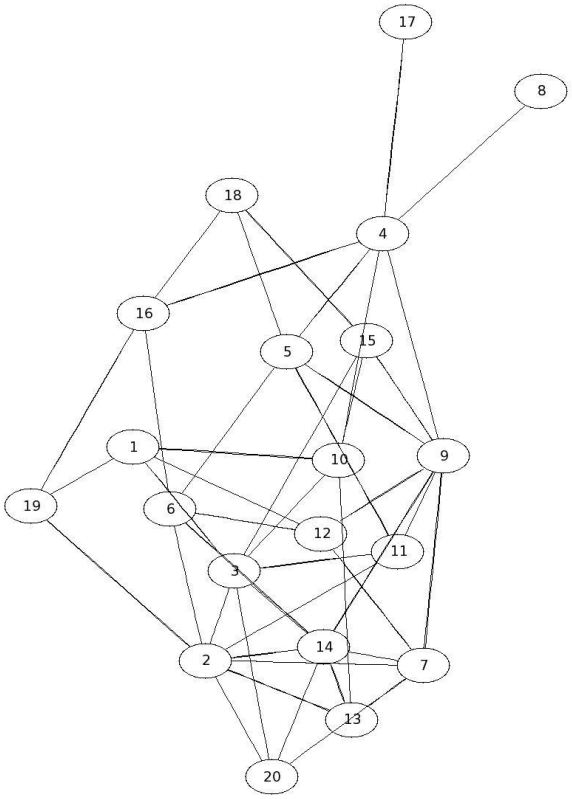
I.I Introduction to Graph

Graph is one of many mathematical structures used to model pairing relations between objects from a certain collection of objects. A "graph" in this context refers to a collection of vertices or 'nodes' and a collection of *edges* that connect and link pairs of vertices. A graph could be *undirected*, meaning that there is no distinction between the two vertices associated with each edge, or its edges may be *directed* from one vertex to another adjacent vertex.

$$G = \{V, E\}$$

Graphs are represented graphically by drawing a dot for every vertex, and drawing an arc between two vertices if they are connected by an edge. If the graph is directed, the direction is indicated by drawing an arrow.

A graph drawing should not be confused with the graph itself (the abstract, non-visual structure) as there are several ways to structure the graph drawing. All that matters is which the vertices are connected to which others by how many edges and not the exact layout. In practice it is often difficult to decide if two drawings represent the same graph. Depending on the problem domain some layouts may be better suited and easier to understand than others.



Picture 1 Graph example

I.II Introduction to Artificial Intelligence

Artificial intelligence (**AI**) is renowned as the intelligence or brain of machines and the branch of computer science that have an objective to create it. AI textbooks define the vigorous field as "the study and design of intelligent agents where an intelligent agent is a system that perceives its environment and takes actions that maximize its chances of success. John McCarthy, who coined the term in 1956, defines it as "the science and engineering of implementing intelligent machines.

The field was founded on the claim that the central property of humans, intelligence—the sapience of *Homo sapiens*—can be so precisely described that it can be simulated by a machine. This raises philosophical issues about the nature of the mindset and the ethics of creating artificial beings, issues which have been addressed by

myth, fiction and philosophy since antiquity. Artificial intelligence has been the subject of optimism, but has also suffered setbacks and, today, has become an essential part of the technology industry, providing the heavy lifting for many of the most difficult problems in computer science.

Picture 2 Robot ASIMO

AI research is highly technical and specialized, deeply divided into subfields that often fail to communicate with each other that causing the field become very sophisticated. Most subfields have grown up around particular institutions, the work of individual researchers, the solution of specific problems, longstanding differences of opinion about how AI should be done and the application of widely differing tools as this field complexity growth quickly in the mean time. The central problems of AI include such traits as reasoning, knowledge, planning, learning, communication, perception and the ability to move and manipulate objects. General intelligence (or "strong AI") is still among the field's long term goals.

II the Use of Graph for decision-making of one behavior

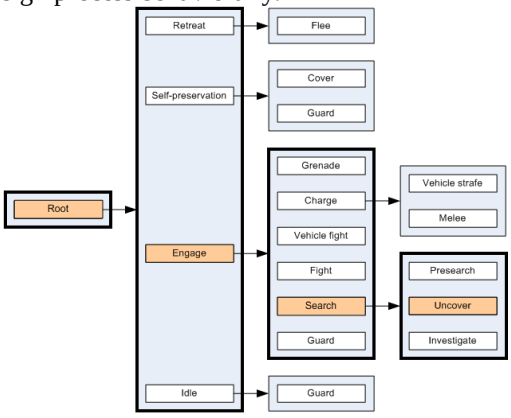
First of all, in order to simplify the design of the AI's behavior we need to visualize it. There are two methods that will be explained in this paper, Behavioral and Structural. Each approach has its own advantage and drawback. Both will be explained in order to compare which one is more appropriate within the case.

II.I Behavioral Graph

Behavioral means we can analyze every single act upon the AI itself and then programmed it into its core. The good side of this method is simple and no need of much complex algorithm to implement the method but, for some big cases this method will consume a lot of memory and turn out into spaghetti code as it would create huge amount of possible act as behavioral is known as brute force method for implementing AI but we can use this method effectively by drawing a good design of behavior tree.

The tree data structure can be generalized to represent directed graphs by removing the constraints that a node may have at most one parent, and that no cycles are allowed. Edges are still abstractly considered as pairs of nodes, however, the terms *parent* and *child* are usually replaced by different terminology (for example, *source* and *target*). Different implementation strategies exist, for example adjacency lists.

First, we can use the Behavior tree to visualize the design process behaviorally.

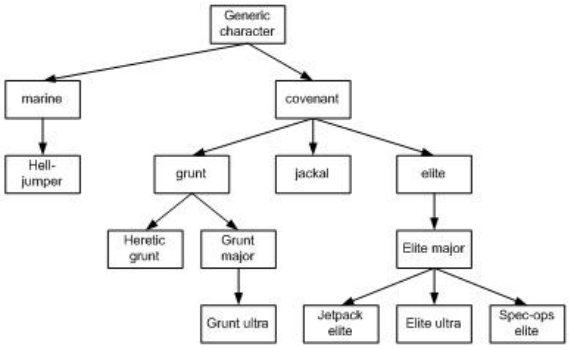


Picture 3 Behavior Tree Example

The behavior tree can be very simple or the most complicated graph you'll ever see. For some simple behavior this method is recommended as for some reason:

1. They employ a single, simple notation for analysis, specification and to represent the behavior design of a system.
2. They represent the system behavior as an executable integrated whole.
3. They can be understood by people without the need for formal methods training. By strictly retaining the vocabulary of the original requirements this eases the burden of understanding.

After we use the behavior tree to classify every single behavior we want to implement into the AI, we can see that we can add or remove any behavior that we want easily. There will be no overlapping or error based on the linked behavior.



Picture 4 Behavior Tree Example

A behavior tree is to be executed each time an entity needs to decide upon an action. The tree evaluates from the root evaluating each chosen child node. Each node returns a Boolean to indicate the success or failure of that node. Therefore, the interface requires an execute method that returns a Boolean. As it is intended that the behavior tree may make use of the refectories cell Parameters class, this method must receive an argument of this type.

To connect nodes and leaves of type Behavior Tree Node, each node must maintain an array of children and provide methods for adding to this array.

To use a behavior tree, the user must first create each node/leaf and connect them. Once constructed, an entity needing to choose an action will execute the root node which in turn, upon deciding the correct child, will execute the chosen child node and pass on the parameters.

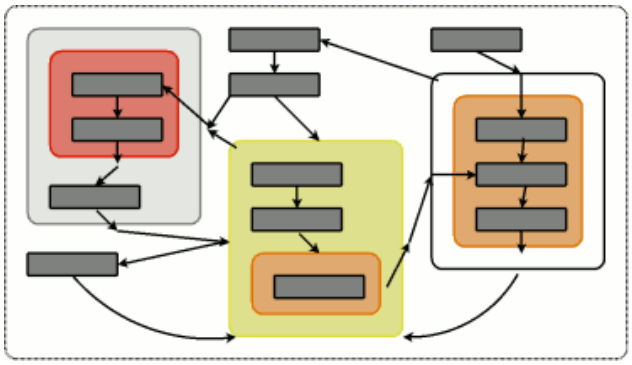
Eventually the root node will receive a Boolean value indicating the success or failure of its child and, presuming its own execution is complete, will return its own success or failure to the user. The interpretation of this final return status will be implementation specific in so much that a return of false is not necessarily a negative result dependent on the design of the tree itself.

II.II Structural Graph

Structural means we need create an organ that built from relations of behavior and put it into our AI's brain. The word organ means a group of behavior that has link into each other behavior so that in order to go to different behavior the AI don't have to restart classifying the behavior from the beginning but they only need to jump to another behavior.

The organ that we will build is known as Finite State Machine. A **finite-state machine** (**FSM**) or **finite-state automaton** (plural: *automata*), or simply a **state machine**, is a mathematical abstraction sometimes used to design digital logic or computer programs. It is a behavior model composed of a finite number of states, transitions between those states, and actions, similar to a flow graph in which one can inspect the way logic runs when certain conditions are met. It has finite internal memory, an input feature that reads symbols in a sequence, one at a time without going backward; and an output feature, which may be in the form of a user interface, once the model is implemented. The operation of an FSM begins from one of the states (called a *start state*), goes through transitions depending on input to different states and can end in any of those available, however only a certain set of states mark a successful flow of operation (called *accept states*).

Finite-state machines can solve a large number of problems, among which are electronic design automation, communication protocol design, parsing and other engineering applications. In biology and artificial intelligence research, state machines or hierarchies of state machines are sometimes used to describe neurological systems and in linguistics—to describe the grammars of natural languages.


Picture 5 HFSM Example

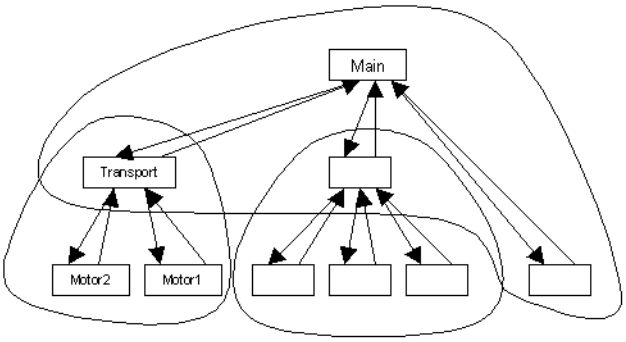
In our case we'll use the hierarchical finite state machine. So, the design will look like a tree but still using the same concept of finite state machine.

The design of its organ is basically one state master that control of all state in its structure and then all master state can be controlled by another state master. The goodness of this design method is that you'll never need to loop back into your first command of the master. You'll just have to jump to you master and then you can jump to another master. Eventually the AI will be more effective.

The recommendation to use a hierarchy of state machines is based on experience and some suggestions of a theoretical nature. When designing state machines we can make logical errors which result for instance in infinite loops or deadlocks. The more state machines are in the system, the higher is the probability of such errors and finding of those errors is more difficult.

Imagine a system of 100 state machines where each state machine may send a command to any other state machine and uses states of any state machine for

definition of its behavior. It is very difficult to understand how such a system truly behaves and a structure of such a system corresponds to non-structured software.



Picture 6 HFSM Example

The diagram shows a hierarchical system with 3 levels: the upper level containing a single Master state machine (Main), the second level containing two state machines and the third, lowest level with 6 state machines.

A design of any state machine requires always an analysis of a limited part of the system. A design of the Motor1 state machine covers the outside signals and commands from the Transport state machine. While designing the Transport state machine we are interested only in the states of state machines Motor1 and Motor2 and commands from the Main state machine.

While designing the Main state machine we take into account only states of its Slaves (Transport, etc.). In a correctly designed hierarchical system any state machine "sees" only the states of its slaves which represent an abstracted but all the same complete set of the control signals which are relevant for the state machine. The only state machines that "communicate" with the outside signals are the Main state machine and the state machines on the lowest hierarchy level.

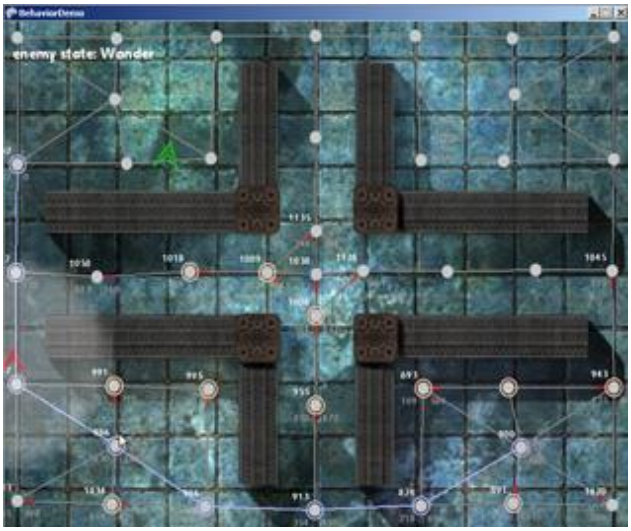
The main drawback while using the structural approach is that there are possibilities that there will be infinite looping jump as we can jump into another state easily.

III the Use of Graph for path-finding of one destination

III.I Visualize the terrain with node graph

In this case we need to use the node graph to visualize the terrain/map that surrounds the AI. Path finding is really just a way to tell an AI system to find the most efficient way to navigate from one point to another. However, before you can achieve this, you need to establish the points themselves. In path finding systems, these points are generally called "**nodes**."

These nodes are all connected together in some fashion to create a network, which we refer to as a "node graph." You can think of this node graph as a series of interconnected waypoints. The AI search algorithm will navigate throughout these points to move from one node to another, by way of the most efficient path of neighboring nodes.



Picture 7 Node Graph Example

After building all this nodes, now we go to the method that will escort us to our destination. There are many methods to obtain a path between two nodes, like the DFS(Deep First Search), or BFS(Broad First Search), but sometimes you don't just want to get a path that connect the nodes, you also want to get the "best" path, which most of the time will mean the one that takes less time or the closest, depending on what you want.

III.I Path finding by Dijkstra Algorithm

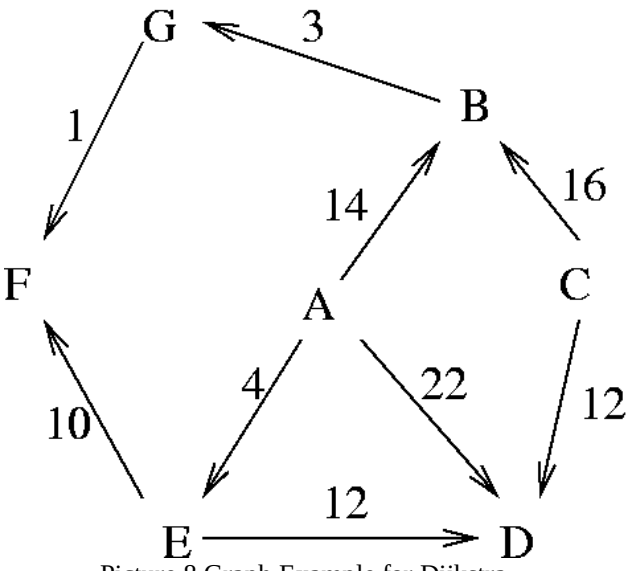
In this paper I will just refer to it as the one that *costs* less to follow. (In our case, remember "cost" means distance. In a game, you may wish to find the least dangerous path, or the one that uses least fuel, or the least visible).

There are algorithms that work depending on the cost, the most important ones being the Dijkstra Algorithm and the A* (A-Star) Algorithm. Lets talk first about the **Dijkstra Algorithm**.

This algorithm was created by Edsger Dijkstra in 1959. It advances depending on the cost that each edge represents, the edges with lower cost will always be chosen first so when you reach the target, we will be sure that the path is the lowest costing path available.

Because this algorithm works with costs, we must have a place where we will be storing the cost of moving to each node from the source (the starting node); call this a cost vector.

For example if the source is the node 0, when we access the cost vector at position 3, this will return the lowest cost until now of moving from 0 to 3. We will also need a way of storing the shortest path: I'll explain the Shortest Path Tree in the next step.



Picture 8 Graph Example for Dijkstra

The basic steps of the algorithm are the following:

1. Take the closest node not yet analyzed.
2. Add its best edge to the Shortest Path Tree.
3. If it is the target node, finish the search.
4. Retrieve all the edges of this node.
5. For each edge calculate the cost of moving from the source node to the arrival Node.
6. If the total cost of this edge is less than the cost of the arrival node until now, then update the node cost with the new one.
7. Take the next closest node not yet analyzed.
8. Repeat this until the target is reached, or there are no more nodes available.

V. CONCLUSION

In short, there are many use of graph in the field of Artificial intelligence in which simplifies the work process of the brain itself.

In order to control its behavior we can use two different approaches. It was behavioral or structural. Each approach has different advantage compared to the other. While behavioral give us full control of the brain behavior so that we can easily add or remove any behavior we want, structural approach enable us to make more efficient algorithm so that it would reduce the work that the brain has to take.

When it comes to path finding, graph assistance will help to implement the best way a lot. In theory graph has many efficient way to find the best path with best cost or etc. Moreover, graph has huge amount of search algorithm that would significantly help in the process of path finding in artificial intelligence.

REFERENCES

[1] http://en.wikipedia.org/wiki/Behavior_Trees. Tanggal akses 14/12/2010.
[2] Munir, Rinaldi, "Matematika Diskrit", Edisi Ketiga, Penerbit Informatika: Bandung, 2005.
[3] http://www.ai-blog.net/ Tanggal akses 14/12/2010.
[4] Rosen, Kenneth H, Discrete Mathematics and its applications", 4th edition,McGraw-Hill Science : English, 1998.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 29 April 2010

ttd

Edwin Lunando/13509024