NAME – SHARATH SREENIVASAN
STUDENT ID - 200109355

CSC 584 BUILDING GAME AI HOMEWORK 3 REPORT

The main task of the assignment was to implement and explore decision and behavioral trees. This was done with the help of Processing, an open source flexible software sketchbook and a language to code within the context of visual arts and java. Processing 3 was used to implement the different trees, each of which have been described in detail in the parts below. I have implemented all the 3 sections by making use of the processing plug in for eclipse

Decision Tree

A decision-tree is a tool that uses a tree-like graph or model of decisions and the possible consequences, including the chance and event outcomes. My decision tree was constructed based on the position of the character in the screen. So, I basically divided my environment into 4 parts and performed actions depending on the position of the character in the screen. The 4 parts included the following actions

1. Top Left – Seek the mouse pressed location with an increased velocity.
2. Top right – Wander in the top right section of the environment
3. Bottom Left – Seek a particular on the point
4. Bottom Right – Changes color.

The position of the character is checked at each instance and depending on the x and y co-ordinate the corresponding action is called. This was implemented using a DecisionNode. Every node in the graph is of either a leaf node or an internal node. If the node is an internal node, then it performs a check of the co-ordinates of x and y. If the node is a leaf node, the function sets an attribute called leaf_type which bascally determines which leaf node is to be executed is set to a value to determine the type of action to be executed.

The conditions of the decision tree is as follows: -

The root node checks the position of the character. If the x co-ordinate of the character is lesser than 300, the function returns the left child. However, if the position is greater than 300, it returns the right node. In the left node, the y co-ordinate value is checked. If the value of y is lesser than 400, then it sets of the type of leaf node to a particular variable. Similarly, the types are set based on the decision tree. At each instance, the tree is traversed and the type of the leaf node set is checked to call the particular action. The decision tree is constructed in the setup function.
The structure of the decision tree is shown below –

NAME – SHARATH SREENIVASAN
STUDENT ID - 200109355

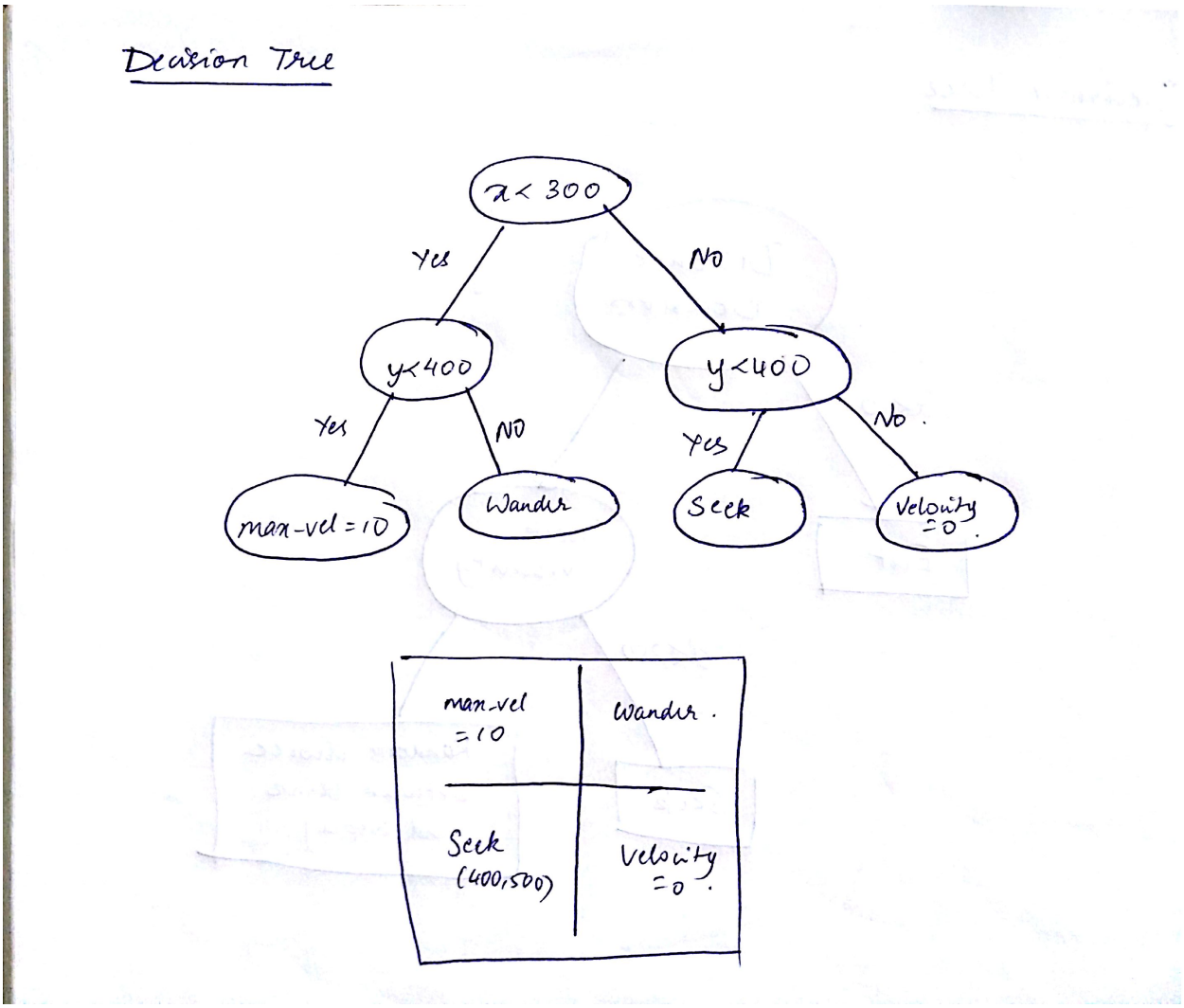

Figure 1 – Decision Tree

Behavior Tree

A behavior tree is a mathematical model of plan of execution code that describes the plan of execution that describe the switching between a finite set of tasks in a modular fashion. The three types of nodes I have implemented in my behavior tree is as follows

1. Sequence
2. Selector
3. Random Selector

A sequencer is an internal composite that executes the actions in the children from left to right in sequential order till the first order returns a 'false'. A selector executes the actions in the children till one of them returns the first 'true'. A random selector executes the actions in the children, chosen randomly, till one of them returns a true.

NAME – SHARATH SREENIVASAN
STUDENT ID - 200109355

The following actions have been implemented for the monster
1 Checking the vicinity
2. Seeking the position of the character
3. Eating the character.

The monster has some properties that set it apart from the character. The monster is more like a ghost. The walls and obstacles have no effect on the monster. It passes across them like they were not there at all. When the monster is not hunting, it keeps changing color and there's a blinking ability for it. When the monster senses the character is nearby, the monster changes color and seeks the position of the character. The monster I have implemented is basically another shape in the environment. Initially, the monster is scanning to check if the character is in its vicinity. When the scanning stage is executing, a ring is displayed around the monster The monster also changes color repeatedly. These two actions are chosen at random from the behavior tree. At each instance, the distance between the monster and character is checked. If the distance is below a certain threshold, the monster seeks the character.

The character seeks the position of the mouse click. The function localizes the character position and the position of the mouse click. Both the points are localized in the graph and the character seeks the mouse click position based on the path determined by the A star algorithm.

If the character is within a certain distance of the monster, the monster would seek the character's position and would therefore invoke the seek function. If the monster is extremely close to the character [difference between the monster's position and the character's position is less than 10] the eat function is called which resets the position of the character and the monster to an initial point from which all the actions can be implemented. The user is notified with a pop up which indicates the character has been eaten by the monster.

A screenshot of the behavior tree is as shown below: -

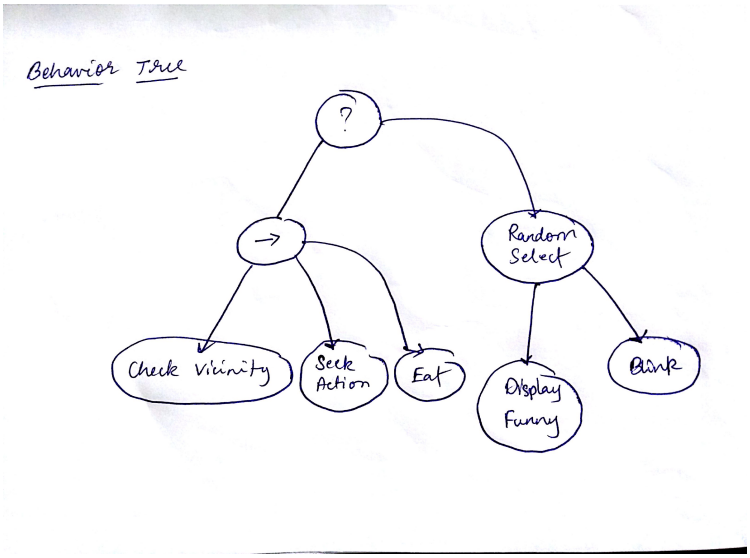

Figure 2 – Behavior Tree

NAME – SHARATH SREENIVASAN
STUDENT ID - 200109355

I found the behavior tree to implement much harder to implement than compared to the decision tree. Some of the problems I faced include the fact that when the monster would seek the position of the character, the path of the character was reset and the character would seek the position of the mouse click without adhering to the obstacles in the environment. As a result, I was required to implement different functions for the monster to separate the monster from the character. The behavior tree was implemented using a DecisionBehavior class which has implementations for the node. If the node is a leaf node, then the corresponding action is called. Another problem I faced was that every time the positions of the monster and character is reset, the character would seek the position of the last mouse click. To solve this, I initialized the path to null each time the positions were changed to the initial point. The monster behaved as expected.

Decision Tree Learning

The main task in this section is to implement a self-learning decision tree for the monster's behavior. The attributes I chose for the construction of the decision tree include whether the monster is within the range of the character, the distance of the monster from the character and the action implemented by the monster. I collected data for about 5000 instances. I made sure that there were enough instances to document each of the actions of the monster namely seek, blink, display and eat. Each of these values were written into a file [Output.txt] which was used to construct the decision tree.

Entropy was used as a measure to calculate the gain from splitting of an attribute at each level of the tree. The tree contains a maximum of 3 levels.

A screenshot of the decision tree obtained through the data is as shown below: -

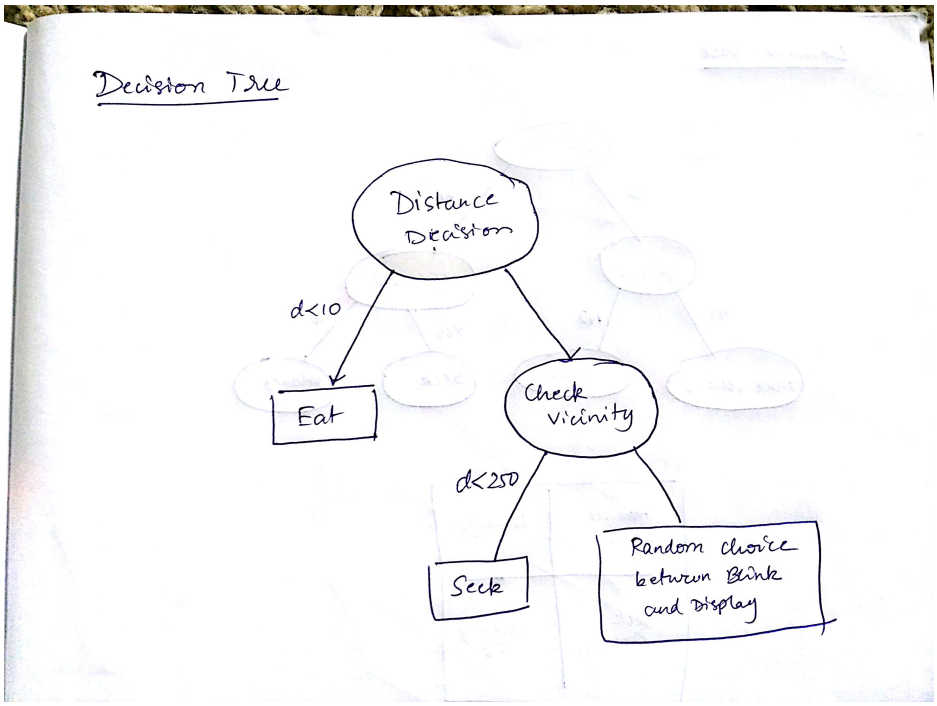

Figure 3 – Decision Tree Obtained from Learning

At each instance, the distance of the monster from the character is checked. If the distance of the monster is lesser than 10 units, the monster would eat the unit. If the distance was greater then the monster would check if the character was in its vicinity. If the character was in its vicinity, the monster would seek the character. If not, it would randomly choose to either change color or shape.

The decision tree constructed performed the same as the behavior tree constructed in the previous section. Since the actions implemented by both trees are the same and the number of actions is low, the tree performs the same as was expected. If the environment was more complex, more data would be needed to learn the tree and the chances of non-identical behavior increases.

Screenshots are included below

APPENDIX

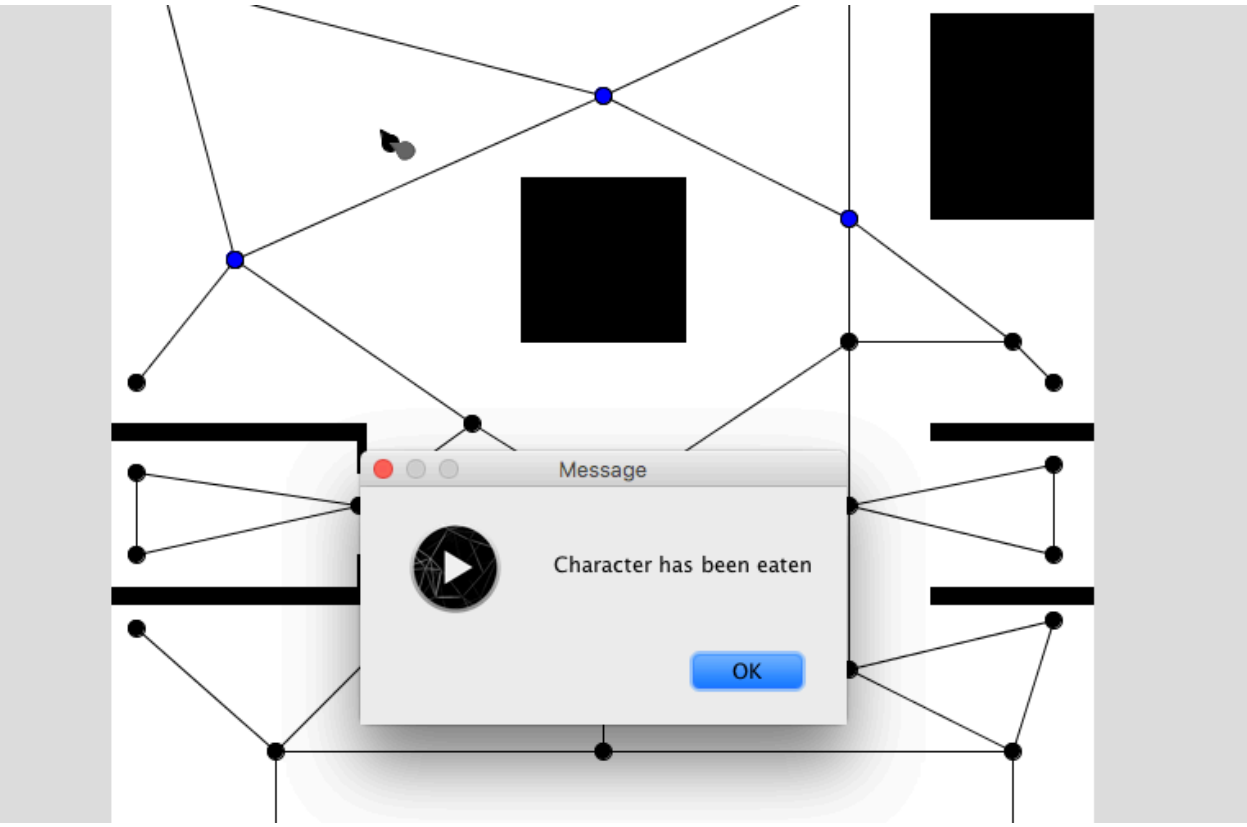

Figure 4 – Monster eating Character

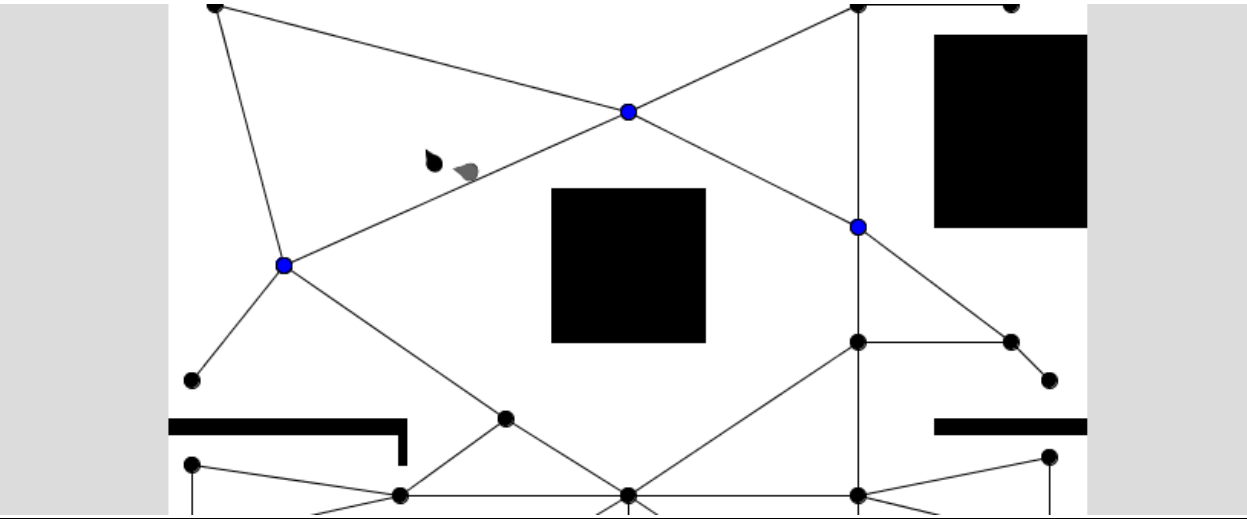

Figure 5 – Monster seeking character

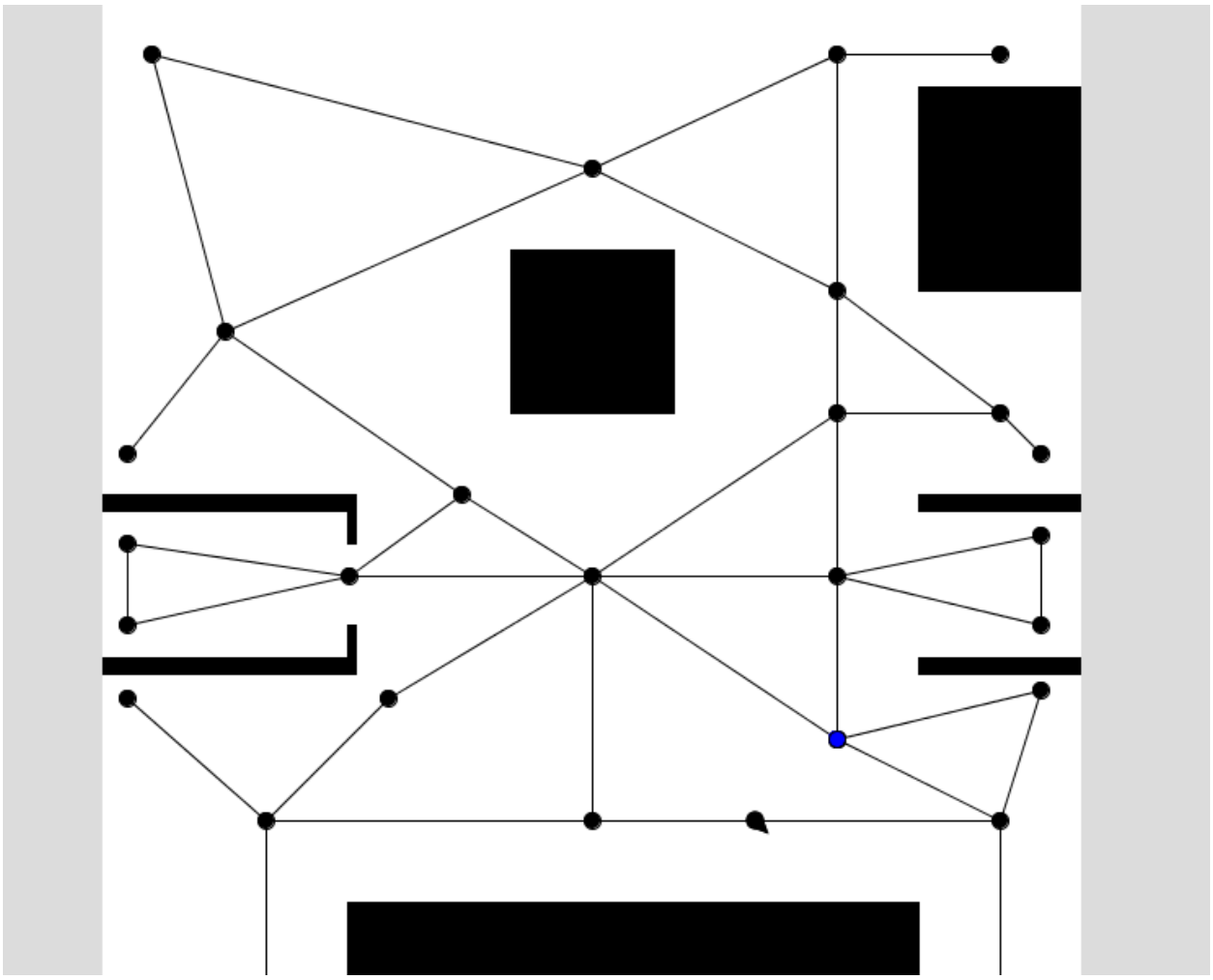

Figure 6 – Character seeking (400, 500)

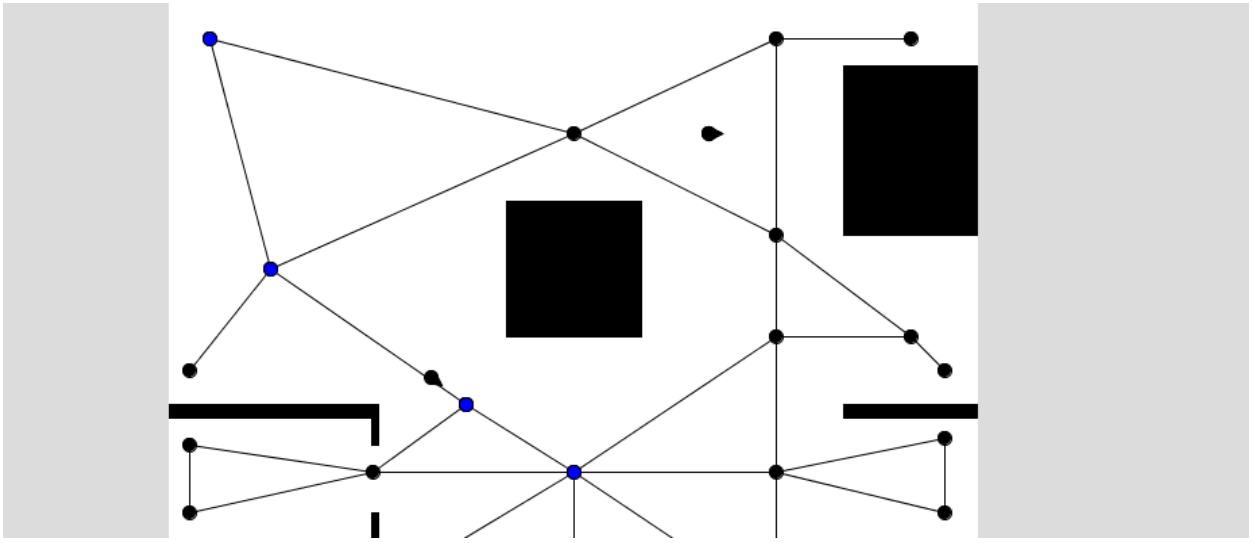

Figure 7 – Character seeking with increased velocity and monster scanning