Hide and Freak

Andreas Ekberg[1], Gustav Johansson[2], Lucas Hägg[3]

Abstract
This report describes a project focused on developing an AI system capable of navigating and making decisions within an environment. The implementation used behavior trees, multi-agent system and a blackboard model to produce coordinated decision-making between the agents. There was three agents using different senses to find the hiding player.

Source code: https://github.com/andreas-ekberg/HideNSeek

Authors
[1] *Media Technology Student at Linköping University, andek016@student.liu.se*
[2] *Media Technology Student at Linköping University, gusjo626@student.liu.se*
[2] *Media Technology Student at Linköping University, lucha567@student.liu.se*

Keywords: Behaviour Trees — Multi agent system — Blackboard model

Contents

1. Introduction

Behavior trees are a common feature in modern gaming and frequently used to design complex agents. It is a branch-like structure where a situation is broken down to smaller nodes and followed consequently. Even with this simple idea the possibilities are endless.

The goal of the project is to create this behaviour tree for multiple agents and making the different agents able to communicate with each other to find the player.

2. Theory

The different theories are described in this section.

2.1 Pathfinding

There are many different pathfinding algorithms that are used but one of them is A* pathfinding. For the algorithm to work the world is divided into squares called nodes, there is a starting node A and a target node B shown in figure 1.

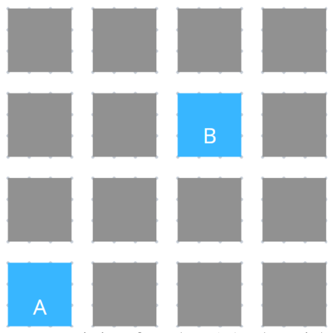


Figure 1. The prerequisite for the A* algorithm, a starting node A and a target node B

The way the algorithm determines the shortest path between two points is by evaluating the cost of going to each node. This consists of two different costs: the H-cost, representing the distance to the target node and the G-cost, representing the distance from the start node. and if added together it is called the F-cost. A horizontal or vertical movement costs 10 while a diagonal cost 14.

One iteration will start at a node and look at the cost of all surrounding nodes, then the cheapest node will be the next focus node. This will repeat until the algorithm has found a path to the target node. An example is shown in figure 2.

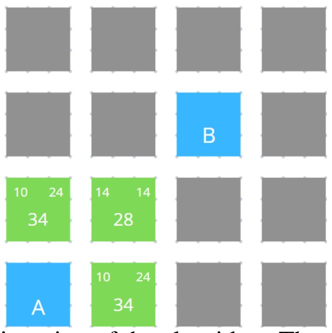


Figure 2. One iteration of the algorithm. The top left number represent the G-cost and the right the H-cost. The middle number is the F-cost.

For A* to guarantee finding the shortest path, the heuristic function must be admissible. An admissible heuristic ensures that it never overestimates the cost to reach the goal. In other words, the heuristic provides a lower estimate of the real cost. The criteria to see if the heuristic h is admissible is seen in equation 1.

$$h(n) < \textit{the true cost to reach the target node} \tag{1}$$

2.2 Behaviour trees

Behaviour trees are used in different areas. For example video games, robotics and computer science. Behaviour trees is a model of a system and a plan of executions. Behaviour trees are often used to create a complex behaviour, built on simple tasks. This leads to an easier programming experience because behaviour trees makes it possible for the user to only implement the actual tasks and then attach it to the tree as a node. The structure of the tree then creates the behaviour.

Behaviour trees are usually represented graphically as a tree of nodes. The different nodes are state-, selector- and sequence nodes. The outgoing node is the parent node while the incoming node is a child node. The state nodes are these simple tasks that is structured as the child nodes of both selector- and sequence nodes.

2.2.1 Sequence node

A sequence node contains of several child nodes. When a sequence node is entered, it will always enter the left-most child. This type of node is structured a way that when the first child node returns true, the program will enter the next child node of the sequence. Figure 3 presents a simple example of how a sequence node could be used. Entering the node named *Door unlocked*, the sequence will keep track of what the node returns. If the child node returns true, the sequence will then enter the other child node named *Go into room*.

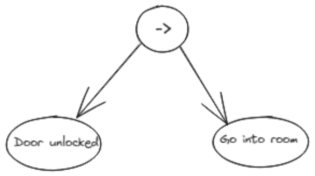


Figure 3. Graphical representation of an example of a sequence node. The sequence node is represented as the parent node in figure above.

2.2.2 Selector node

A selector node is structured similar to the sequence node. The difference between them is that the selector will move on to the next child node if the the first child node returns false. Figure 4 shows an example of how a graphical representation of a selector could look like. The selector will enter the first left-most child node named *Do i see player*. If the agent doesn't see the player, the node will return false and then enter the next child node called *Look for player*.

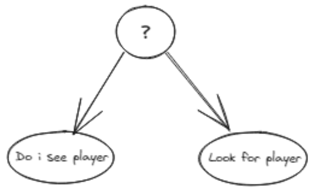


Figure 4. Graphical representation of an example of a selector node. The selector node is represented as the parent node in figure above.

2.3 Blackboard model

The blackboard model is a framework for collaboration and information sharing. It is a common database storing information that has been gathered and accessing it.

3. Method

This project was created using the Unity editor. [1]

3.1 Environment

The implementation of the environment was not a prioritize since the goal of the project was to create a behaviour tree and a multi-agent system. Therefor assets was downloaded from the *Unity Store*. The assets downloaded were sprites for the different characters and the *tilemap* for the map. When these assets were downloaded, they were imported to Unity. The sprites were set up by placing one of the sprites on the map, then connecting the other sprites to an animation attached to the game object. The map were created by using a tool in Unity called *Tilemap Editor*. The Tilemap Editor allowed editing of the imported tilemap such as adding new pixels, color pixels differently, removing certain pixels and so on. The key factor of the Tilemap Editor was the copy tool which were used to create our own tilemap in the actual game. After the tilemap was implemented in the game, a *Tilemap Collider* attribute were attached to the map for the reason that the characters wouldn't move through walls. Doors were also included in the tilemap but were differently implemented. The doors had a script attached to them which had the purpose of when the player were close to the door, they could press the button *K* and then the door would open and the Tilemap Collider removed. [2]

3.2 Agents

The Agents in the game are enemies to the player. They were the characters that had their behaviour decided by the behaviour trees. The game contained 3 different agents. Each agent had a different key behaviour. One *Sniffer*, one *Listener* and one *Watcher*.

The sniffer had the ability to smell a part of the path that the player was walking. The player left behind a trail of nodes that the sniffer could enter to register the position of the node. This was implemented by attaching a script to the sniffer that could identify these nodes and also a script attached to the player that made the player drop nodes in a specific frequency.

The listener had the ability to listen for the player. This was implemented by attaching a script to the agent that notices if the players position were inside a radius from the agent.

The watcher had the ability to look for the player. The script that were attached to the agent included a check if the player were inside a box collider that represented the vision span of the agent. This box collider were implemented so that it could rotate according to the agents moving direction.

3.3 Pathfinding

After creating the environment the agents needed a way to walk around. This was done by using the previous mentioned A* pathfinding algorithm. The position of the agent is known and becomes the A node, the B node is either a randomly chosen point on the map or a position from the blackboard. The map is divided into nodes which is shown in figure 5.

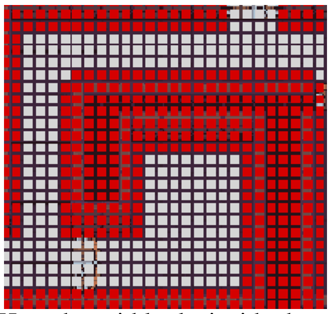


Figure 5. How the grid looks inside the unity editor

When both of the positions are sent to the algorithm the path is calculated and the agent will move to the nodes that represent the shortest path. This whole process is called a path request which an agent can call for.

3.4 Behaviour tree

The initial phase of implementing the behaviour tree consisted of coding classes. First was the creation of the Node class, which served as the base class for various types of nodes in the Behavior Tree. This class contained fields for the state of the node, a list to manage child nodes, and a method to attach child nodes. Following this, the enumeration NodeState was created as a member variable, defining the possible states of a node: RUNNING, SUCCESS, or FAILURE. These states provided a means to track and manage the state transitions of each node during the tree's evaluation process. Two specific node types, Selector and Sequence, were then implemented. The Selector class focused on evaluating child nodes until it encounters a successful or running child, representing an ”OR” behaviour. Conversely, the Sequence class, also extending class Node, assessed child nodes in sequence, ensuring an ”AND” behavior by returning success only if all children succeeded. Furthermore, the Behaviour tree class acted as the control center, containing the root node and a method to trigger the evaluation process. This class initialized the Behavior Tree and executed the tree by calling the evaluate method of the root node.

After implementing pathfinding and finishing the code for behaviour trees, the agents behaviour needed to be more complex. By describing their wanted behaviour in simple situations a basic tree was created which is shown in figure 6.

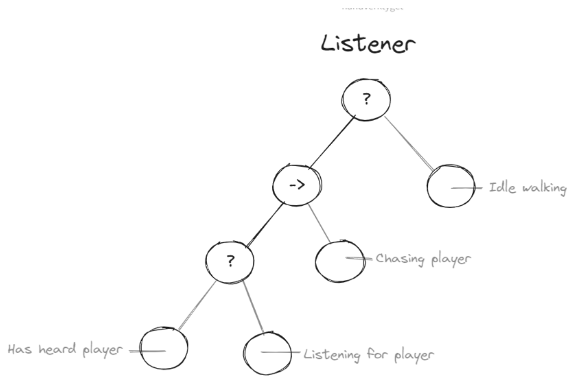


Figure 6. First draft of the listener tree

This behaviour tree made the listener agent able to walk around in the environment and listen for the player. While this tree worked in the beginning it needed some changes to work with the other agents and that three is shown in figure 7. This was the final design of the behaviour tree for the listener.

3.5 Communication

Inspired by *Multi-Agent systems*, the blackboard model were used to create a communication link between the agents. A script called *AIBrain* were created which were a shared script between all the agents. This script stored the position of the player. Each agent had the ability to upload new granted information about the players position to this script and the ability to read new found information that was uploaded from

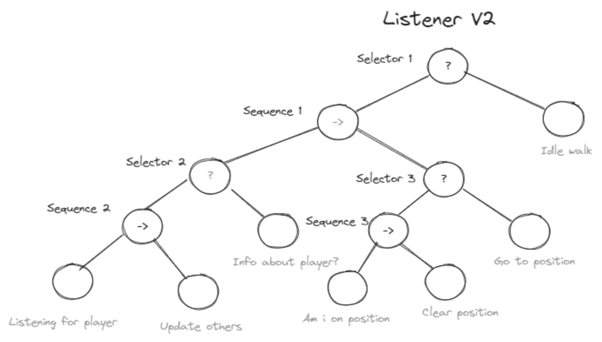
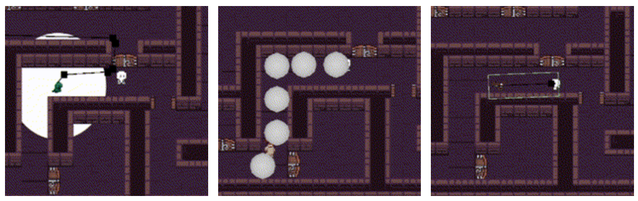


Figure 7. Final structure of the listener tree

the other agents. When new information were found, the agents together would approach the last known position of the player. [3]

4. Result

This is the result of the project.



(a) Listener **(b)** Sniffer **(c)** Watcher

Figure 8. These are images of each agent and their unique abilities.

Figure 8 visualises the different agents and their abilities. Picture 8a shows the listener-agent and its hearing radius that is set with a fixed radius. Picture 8b shows the nodes which the player drops behind its trail. The sniffer enter these nodes and updates the player position. Picture 8c shows the watcher-agent and the collider box which when the player enters, the agent starts to chase.

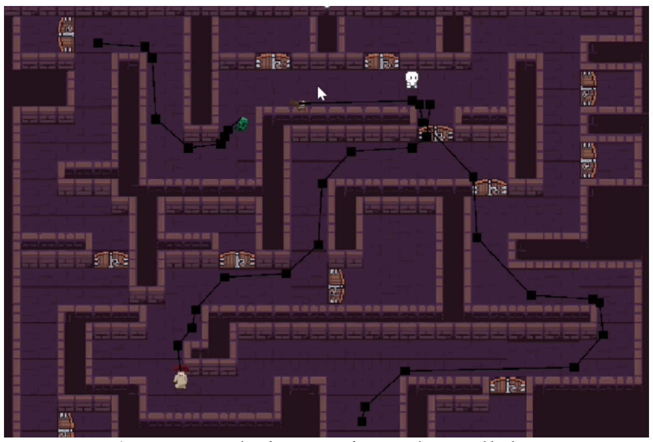


Figure 9. A moment during runtime where all the agents are looking for the player.

Figure 9 and figure 10 is the view of the game from the Unity Editor and is the result of the project. Figure 9 shows all agents and the player. This figure is a snapshot of the game while the agents are in their idle state. They are currently walking to randomly preassigned positions around the map. The black squares that is connected by lines, in both figures, represents the path of each agent.

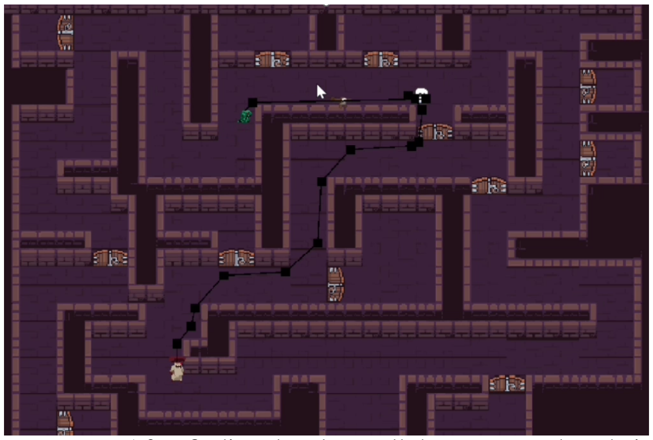


Figure 10. After finding the player all the agents update their current path to the players position.

Figure 10 shows a snapshot of all agents when they've recently updated the path through the AIBrain script. Moments before the snapshot were taken, the watcher agent detected the player withing its box collider.

5. Discussion

The main goal of the project was to develop multiple agents to be able to communicate with each other. The usage of both behaviour trees and a blackboard model was an interesting approach that worked well together.

5.1 Issues

The main issue with the current edition of the game is the pathfinding system. When multiple agents request a path each frame the system fails. A solution that was implemented was that each turn the agent takes the system request a new path which reduces the amount of updates but did not solve the whole problem. The agents will switch between the ”chase state” and the ”idle state” which results in a sporadic movement across the whole map. Therefore the agents will all proceed to the player for one turn but after that they will move away from the target.

5.2 Improvements

Some improvements that could be made is that around the map there could be interactive objects that the player could use against the agents. This would in turn make the agents forced to react to the players decisions and make their behaviour more complex. Another interesting improvement is to make the agents react to the player in more different ways. Currently all agents will walk to the players position when they sense the player. A more interesting approach would be to make each of the agents have more personality by for example making the sniffer only send information and not chase. Or that the watcher would charge in small sprints. This would create more diverse situation and expand the different behaviour trees making them more unique and interesting.

6. Conclusion

The project successfully implemented a multi-agent system using behavior trees and a blackboard model for communication. The agents managed to locate and chase the player, showcasing the effectiveness of behavior trees in defining diverse agent behaviors. There were challenges with the pathfinding system that prevented the agents to stay on correct paths. Potential improvements to the game include introducing interactive objects and diversifying agent reactions to make the gameplay more engaging. Despite the challenges, the team had fun creating the game and learned the potential of behavior trees and multi-agent systems.

References

[1] Unity, *Unity - Manual*, Retrieved: 2023-11-05,
Available: https://docs.unity3d.com/Manual/index.html

[2] Unity Asset Store,*Unity Asset Store*, w.a, Retrieved: 2023-09-20,
Available: https://assetstore.unity.com/

[3] Stefania Costantini,*Intelligent Agents*, w.a, Retrieved: 2023-09-20,
Available: https://shorturl.at/dxA36