Mario Agent Turing Test Writeup

Grant Ferguson and Joseph Petitti

December 5, 2019

1 Design and Behavior Goals

Our overall design and behavior goal was to create an AI agent that can navigate Mario levels successfully while also behaving like a human being to the best of its ability. More specifically, we decided that in pursuit of these goals the more detail oriented goals would be:

1. To have the AI agent exhibit traits that many of players demonstrated
2. To have the agent exhibit distinctly human perceived thoughts (i.e. stopping to think before acting)
3. To have the agent be able to do seemingly random movements for a non-perceivable reason, like a fallible human player

We chose these specific goals for a few reasons, the first being that these were tasks that we believed would lead to the most human like agent possible, while also being able to be built using the framework. Another key reason that we thought that these would be proper goals to have for our agent was that with these goals, most of the user testing that we did would have a similar trait or traits to these rules. By the end of the project, most of these specific goals we feel were met, and in doing so the agent was somewhat human-like in its behaviors.

2 Implementation

We implemented a decision tree to drive our AI. We created a node interface (`INode`) that requires an `execute()` method. There are two types of nodes that implement `INode`, decision nodes and action nodes. Decision nodes override `execute()` to execute one of their various `INode` branches based on some condition, and action nodes override `execute()` to return a specific action. This way we can recursively iterate down the decision tree by `execute()` on the root decision node. We implemented various decision nodes based on conditions such as an enemy being nearby, Mario standing on a ledge, or simply randomly selecting a child branch. We also made several action nodes, such as walking left, jumping right, or doing nothing. The complete decision tree is shown in Figure 1.

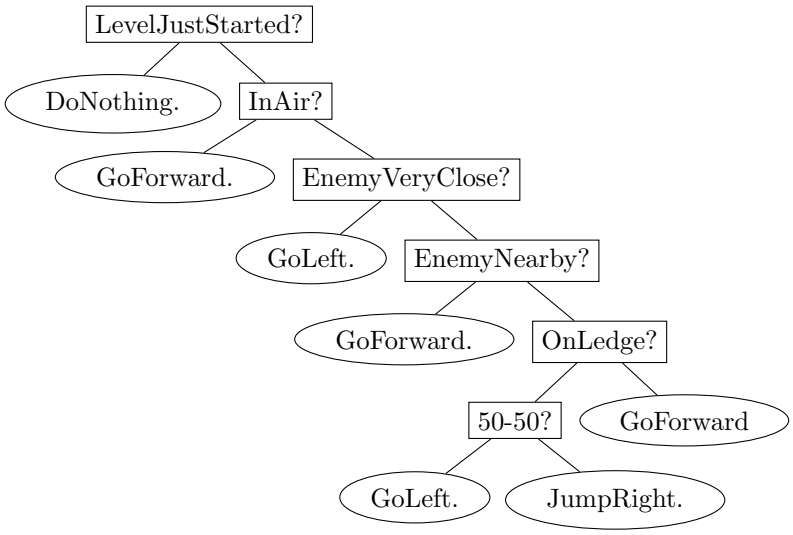

Figure 1: Left branches represent "yes", right branches represent "no"

3 Testing Process

For testing our agent to see what the actual results were of each of the nodes (questions, actions, etc.) we decided that the best approach would be to run through actual original Mario levels and see particular edge cases where either the agent got stuck or the agent had some interactions that we did not previously account for in the decision tree itself. In doing so we would hopefully discover situations that would help us identify places to add additional nodes into the tree where the tree itself was lacking proper interactions with certain level elements built into certain levels. Another aspect of testing the agent has to do with how the agent was initially created. Since we used the A* methods from the `robinBaumgarten` agent inside of the Mario framework, part of the testing was seeing the limitations of that system and to what extent we could use it to achieve parts of the tree and other actions. Upon testing through those classes, we discovered that there was not a way to move to a specific location in the level, and that that would have to be done manually.

4 Pros and Cons

The pros and cons of a decision tree are fairly apparent once it is implemented in any agent, and though there are situations where they can be mitigated, they are still interesting and important to look at when deciding on what type of system to use for an AI agent. To start, the benefits of using a decision tree are fairly straightforward when using the Mario Framework, especially if using it for the agent setup inside a package. When you use a decision tree, it is fairly simple to build ones the pieces are set up, and it is also fairly easy to explain to other users or developers of the system. Another advantage to using decisions trees is the low processing time for each decision, as each decision node is a singular boolean expression in code, and getting to an action requires at most

the amount of levels in the tree expressions. So in a binary decision tree, which would have the minimum amount of nodes per layer, you could have 16 possible actions in 4 layers, and that is with only using binary layers.

Some disadvantages are also apparent when using a decision tree over other methods in the Mario framework. One of these disadvantages is that when you use a decision tree, it can be limited in the framework presented based on the amount of decisions you can make in a frame to frame basis based off of no previously knowledge of the level compared to other systems you can use to make the agent seem more human. Decision trees can't store a state, so it's difficult to make complex actions that rely on the previous actions.

Some things that we would change if we were to do the project again would be to try and find a way to get specific input locations from the A* methods to be able to move Mario with more precision when making decisions, as well as being able to show that when Mario moves there would be a specific function to go to a spot using A* instead of in the general forward direction. It may have been a poor decision to use a decision tree in the first place, and a more flexible method such as a utility table may have been better suited for this task.