CS7632 Term Project - Haxball

Jayanta Bhowmick, Divyanshu Goyal, James Hahn
Harsh Krupo KPS, Saifil Nizar Ali Momin, Code

December 1, 2020

1 Introduction

Haxball is a top-down, two-dimensional, closed-world, multiplayer soccer game with simple mechanics as shown in Figure 1. Players choose to compete in a 1v1, 2v2, or 3v3 setting and attempt to score goals on the opposing team with five buttons: move left (A), move right (D), move up (W), move down (S), or shoot (spacebar). Unfortunately, as the game exists currently, there is no option to play against a computer. As such, this project's goal is to simulate a bot that replicates a human's actions while ensuring the bot thinks/acts realistically, but still thinking/acting like a human. We accomplish this primarily through the creation of a behaviour tree shown in Figure 2 (1v1). Our demo illustrates its success in accomplishing our goals of building a bot for the 1v1 setting and expanding it to the 2v2 setting. We hope this framework will allow easy expansion to the 3v3 scenario in the future.

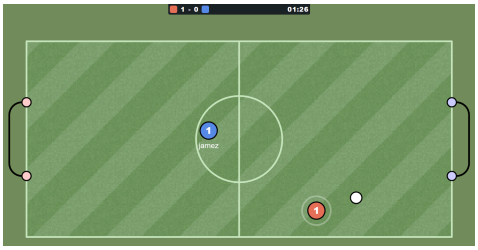

Figure 1: Example game screenshot of Haxball

2 Algorithm, Goals, and Accomplishments

As mentioned above, the sole approach to tackle 1v1 and 2v2 in this project is a behaviour tree. Figure 2 perfectly illustrates the initial tree created for our 1v1 task. While a modified tree was created for the 2v2 task, there is not enough space in this report to accommodate it. We used a mix of daemon, selector, sequence, and action nodes to accomplish our two aforementioned goals. Essentially, we broke the logic down into many unique scenarios. In general, if the bot has the ball and the goal is clear, the bot should immediately align with the ball and goal and shoot. If the bot has the ball and the goal is not clear, just shoot the ball at a wall to hopefully entice the opponent to chase the ball and move them out of position. If the bot does not have the ball and the opponent is not close to the ball, chase to gain possession. Finally, if the bot does not have the ball and the opponent is in a good position to shoot at goal, stay about 75% of the way between the ball and goal, restricting the opponent's space to shoot while also maintaining enough space to react to a shot, recover, and gain possession of the ball.

To create bots of varying difficult and aggressiveness, at each game state update, a uniform random variable is sampled from [0, 1] and if the value is less than σ, the bot kicks the ball if it has the opportunity. The bots can be 'easy' ($\sigma = 0.4$, non-aggressive), 'medium' ($\sigma = 0.7$), or 'hard' ($\sigma = 1.0$, aggressive) difficulty.

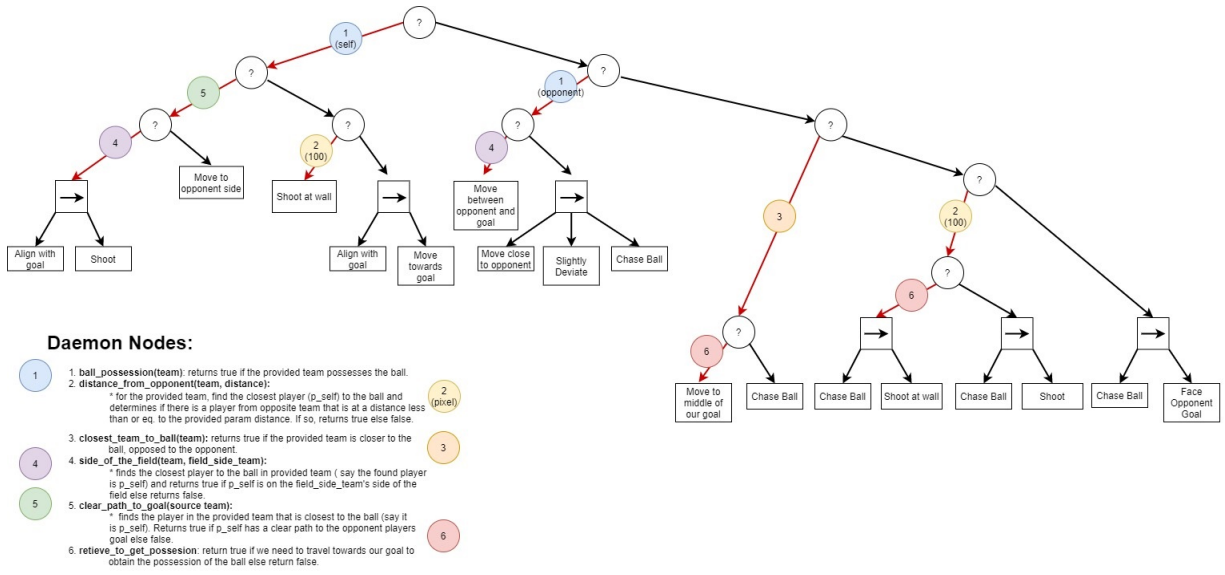

Figure 2: 1v1 Behaviour tree

3 Challenges

We encountered two significant issues, outside of bugs, while implementing this project. First, we used an existing codebase (haxball-chameleon) which enables our client to send game state and input action requests to the Haxball API. While this saved time on development, it also required weeks of reading code and figuring out the functionalities at our disposal. Second, while the initial behaviour tree in Figure 2 works in theory, we found handling and validating the tree was intractable. This led to the bot often being out of position. At times we also noticed that the bot was in a infinite loop among the tree nodes. In addition to these challenges, the physics engine adds momentum to players which makes accurate positioning/movement difficult. As a result, the bot's location changed too often to effectively carry out a given strategy in the behaviour tree. To remedy this, our new approach was to minimize the tree and merge a few individual nodes to ensure operations were atomic, rather than convoluted with a series of dependencies on numerous parent nodes. The final product was a simplified tree containing four nodes: 1) chase the ball, 2) align with the ball when possession is gained, 3) shoot when close to the ball, and 4) position halfway between the ball and goal when the opponent is in an attacking position. In a 1v1 scenario, the 'hybrid' bot uses all 4 nodes. In the 2v2 scenario, an 'attack' bot uses nodes 1, 2, and 3, while a 'defend' bot solely utilizes node 4. Surprisingly enough, this bot simulates our version of a 'beginner' human player we observed while watching other online games. It can compete with many players, but when it encounters a skilled player, it will lose most of the time. We believe this is a lesson to be learned about building an AI: try to simplify as much as possible while maintaining some semblance of human thought to it. Attempts at reconstructing the rest of the tree never quite replicated the fluent nature of this simplified bot.

4 Conclusion

The primary lesson from this project is that a simple AI may be better than an over-engineered AI. While in theory a meticulously designed AI looks better, an AI with a concise list of actions gives less room for the bot to encounter bugs in its own code. Despite this, as will be shown in our demo, we successfully accomplished our goal of creating 1v1 and 2v2 bots capable of simulating human nature while maintaining the naivety of a bot.