Describe your approach.

My program uses a minimax search with alpha-beta pruning to evaluate positions and selects the best action based on a simple forward progress heuristic. The algorithm I used was taken from the sample code provided in *Artificial Intelligence: Foundations of Computational Agents* by David L. Poole and Alan K. Mackworth, without any modifications. During development, I experimented with other heuristics in addition to forward progress, including prioritising longer hops, mobility (counting total moves available on each side of the frog), availability of lily pads one jump away from the final position, and having frogs nearer or closer away from the other player's frogs. Unfortunately, because my strategy for tuning the weights of these different factors was just manually playing agents against one another a handful of times and adjusting based on my best guess, I couldn't find a heuristic that reliably worked better than simple forward progress. Given more time, I would look at adding code to play agents against one another and report results in a way that was more efficient and reliable and optimise my heuristic in that way. I did find that my final program performed consistently better than a simple greedy approach or a randomised agent, using the manual playing method.

Another customisation/parameter I considered was the depth of my minimax search. The final implementation has a maximum depth fixed at only 3 plies. Again, if I were to find the optimal depth, this would involve knowing how much increasing depth would affect computational time. My current implementation does not have time management and therefore I am not sure when it begins to approach the maximum CPU time available for my agent instance. If I had integrated this into my code, I might have found that I could have increased to 4 or more plies, or even have found a way to implement an iterative or even dynamic deepening based on how much clock I had burned compared to what I expected. As it is, I found that 3 plies was a strong improvement on previous implementations and I wanted to err on the side of caution.

My code does not have any explicit move ordering heuristic, but generates children in the order: moves, jumps, then grow. My plan was to try longer jumps first, so that the alpha-beta cuts would happen earlier, but I didn't get round to it.

Performance evaluation

As briefly discussed above, I evaluated my program's performance by manually playing it against earlier versions of my agent. I started with a purely random implementation, then a greedy version based on a combined forward progress and hop-length evaluation function, and then finally minimax. I found that the current version consistently performed better than previous implementations.

Supporting work

The only other significant work I can think of that I did was creating a debugging print_board function to make sure my program state was in step with the referee. I am actually not even sure if this was necessary or if there was an inbuilt function I could have used, but there you go.