**A brief description of the game:**

The Kalah game needs two players, each player has 6 normal holes, each normal initially has 6 stones and one Kalah hole. In each turn one player picks one of their own normal holes(stones in the hole need > 0) and distributes them in each consecutive hole CCW including their own Kalah hole but not the opponent's Kalah. If the last stone lands in their own Kalah, then this player gets one more move. If the last stone lands in their own normal hole and this hole is empty, then this player takes all the stones in the opponent's opposite hole and the one in the empty hole and puts them into their own Kalah. If one player has 6 empty normal holes, then the opponent takes all the stones left on his side and puts them in his Kalah. When one player's Kalah has > 36 stones, then this player owns.

**The definition of the heuristic function and its justification:**

My heuristic function will evaluate the current state based on its attribution and whether it can move again or not. The basic evaluation function is simple, just equal to "state.a_fin - state.b_fin". After that, if this state is called by minimizing value and this state can move again, then the evaluation score of this state will get 6 penalty scores. And if this state is called by maximizing value and this state can move again, then the evaluation score of this state will get 6 bonus scores.

I chose "state.a_fin - state.b_fin" to be the basic evaluation function because the winner always has more stones in their Kalah than the loser. Thus, to achieve the max possible stones in the Kalah each turn will be the first privilege of this game. However, the only exception is that the current state has a chance to move again. If the player gets a chance to move again, then the player is more likely to get "combos". In this case, the value of this state will get a bonus or penalty based on its attribution.

**Illustrate how it works with example moves it chose and explain why it chose them in terms of the function:**

For example, in the initial state, each player has 6 stones in their normal holes and 0 in Kalah, then for all of its successor states, state.a_fin = 1 and state.b_fin = 0 no matter which hole player_a picks. In this case, the evaluation score of the successor states will be "7, 1, 1, 1, 1, 1". This because if player_a picks the first hole, then the last stone will land in the Kalah and get an addition move, so the score of first state will be "1 - 0 + 6 = 7". The last stone of other hole didn't land in Kalah, so their score will be "1 - 0 = 1". Thus, in this example, the heuristic function will choose the first hole.

**How long it takes to make a move at different search depths:**

I tested the code on my computer with a 3.7GHz CPU. The search time for the first few steps were slow, but as the game continuons fewer and fewer stones and available holes left, the search time will become very fast, so I will only show the search time of the first 5 searches with different depth.

| depth: 5 | depth: 7 | depth: 9 | depth: 11 | depth: 13 |
|---|---|---|---|---|
| 0.0100286 | 0.1274207 | 1.2622833 | 9.8226737 | 57.722357 |
| 0.0070245 | 0.0818351 | 0.6230591 | 5.3054493 | 30.390520 |
| 0.0090158 | 0.0491591 | 0.1665511 | 1.8031604 | 6.2655427 |
| 0.0140467 | 0.0165711 | 0.1745789 | 1.1574790 | 5.0785415 |
| 0.0040133 | 0.0290956 | 0.0461538 | 0.3656728 | 2.5893936 |

**How each heuristic function affect the performance:**

I compare the previous one with the most basic heuristic function, just return “state.a_fin - state.b_fin” without considering moving again. The difference is during the game, the one that considered moving again was usually moved many (3-5)times in one turn, but the other one didn’t.