**Project Design Clarification**

Monte Carlo tree search: A tree search algorithm like minimax or alpha-beta pruning requires a large chunk of the tree to be constructed. Moreover, if the tree size is large, it needs a good heuristic function to evaluate a position to avoid searching the whole tree. So Monte Carlo trees prevent the need for large chunk tree construction by introducing a random rollout concept

Usage of UCB enhances Monte Carlo trees to add an element of exploration over exploitation that already exists in standard Monte Carlo tree search.

**State per node**: ( Ball number, wickets\_left, runs\_scored\_so\_far, feature\_of\_current\_batter, feature\_of\_current\_bowler, whose\_perspective\_does\_node\_represent )

**Action in each node:**  Either action\_ball or action\_bowl (depending on whether a play is from the perspective of batter or bowler).

**Reward in each node:** The runs scored at end of innings (i.e no balls left or no wickets left) obtained by random rollout phase of MCTS. If the node was in the perspective of the batting team, then reward is +ve runs, UCB will be calculated and the highest UCB value will be selected in the selection phase of UCB. If the node was in the perspective of the bowling team, then reward is –ve runs(this is because the bowling team needs to minimize the final runs of innings). Here, UCB = Q – Root term. Hence, eventhough we select the maximum UCB value, it acts as minimizing the runs.

**Computing V(S) needs to construct a whole tree. But in MCTS, that is not the idea. Hence, the V(S) and Q(S) concepts are not used as defined in the class.**

**Step-by-step procedure of my design**

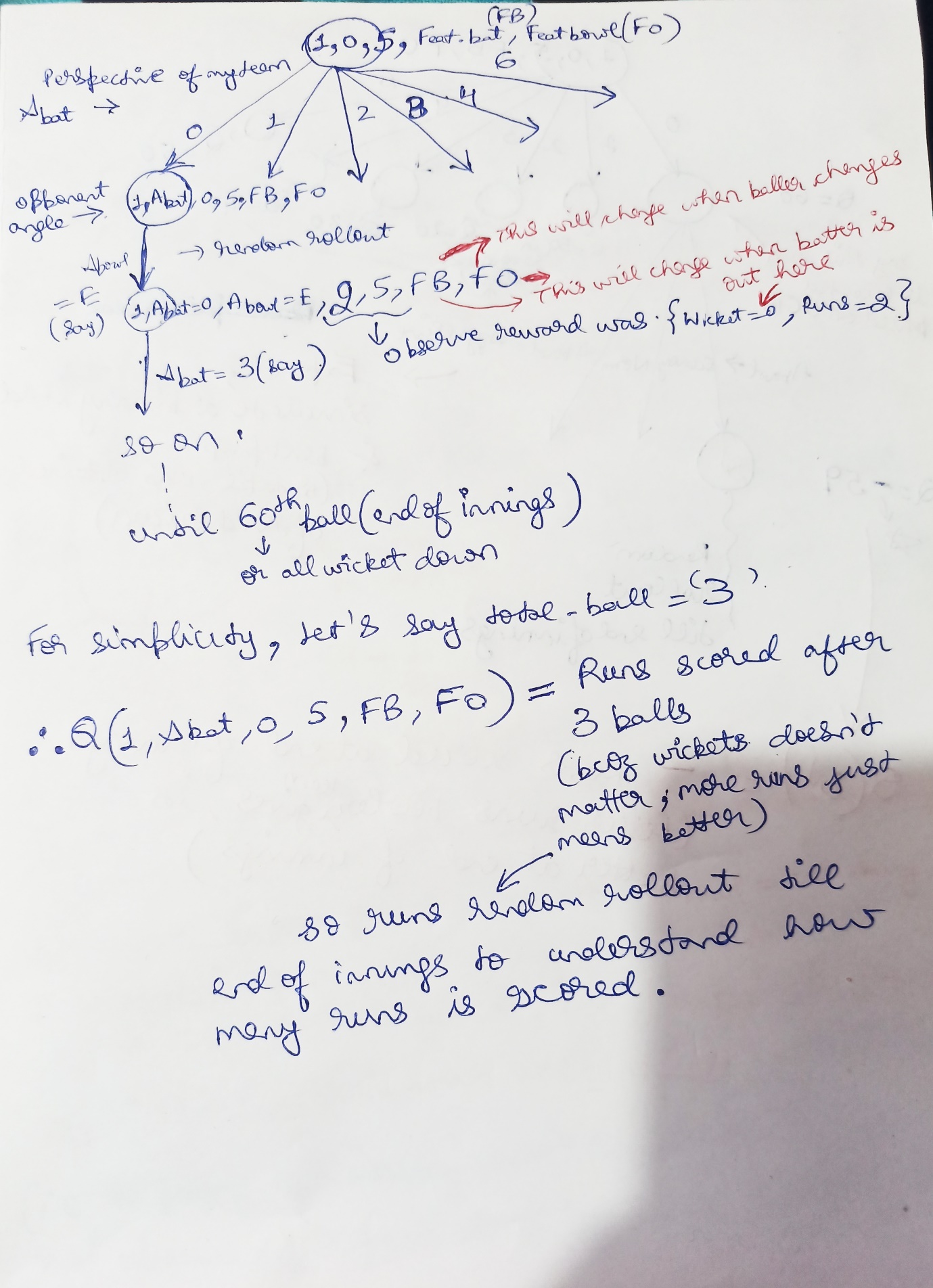
The MCTS tree will start with current state as root state node. From there, roles keep alternating per action. First level we play an action(arm) in our perspective(action\_bat) and next level we play an action(arm) in opponent’s perspective(action\_bowl).

“Feature\_bowler” changes in the state of a node when bowler change(after an over).

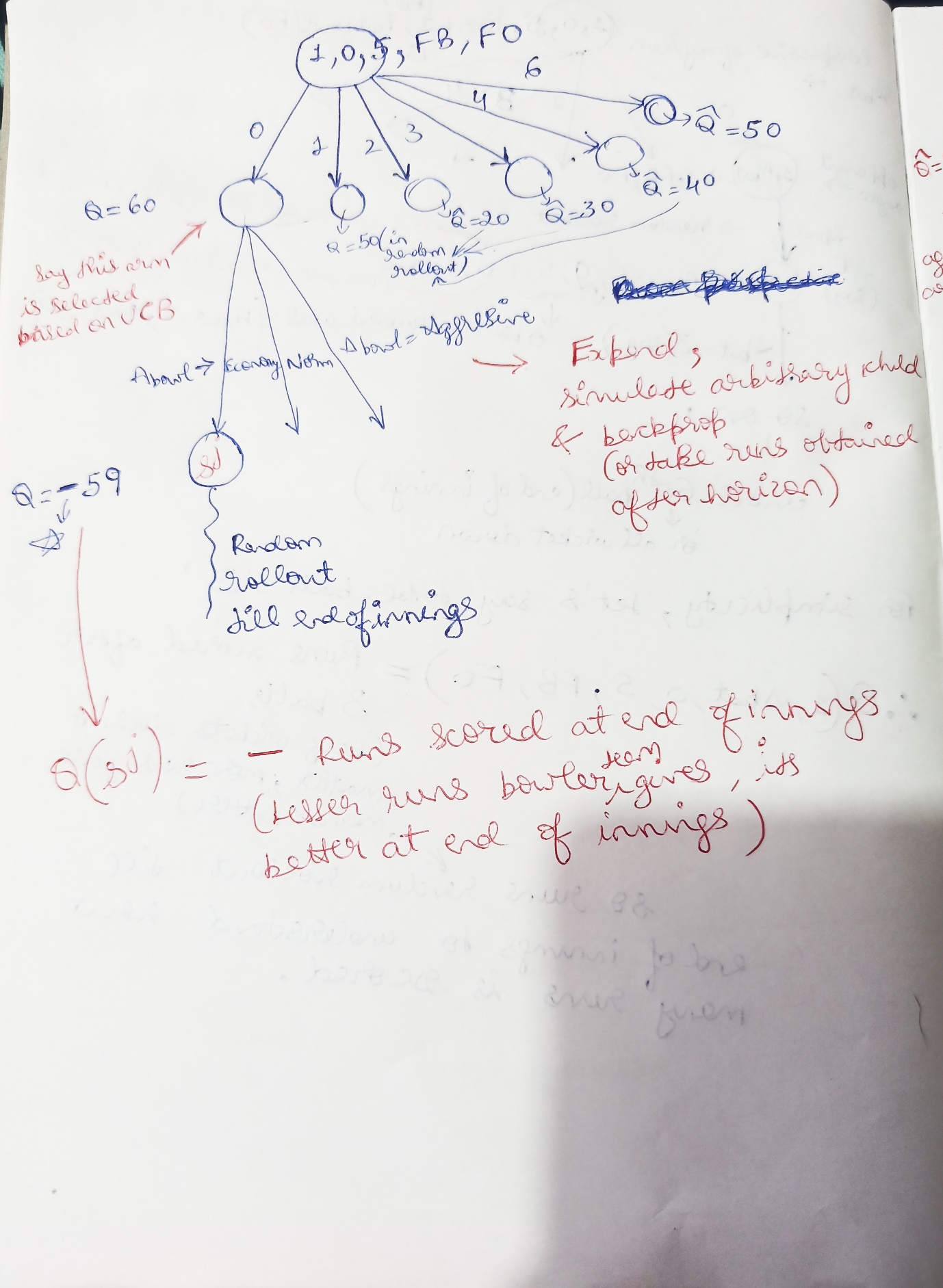
“Feature\_batter” changes in state of node when batsmen get out

Either feature\_bowler or feature\_batter changes only after action\_bowl is played(see below image). This is because “wicket,runs” are obtained only after knowing both action\_bat and action\_bowl.

The following image shows the first round of run of MCTS:



Let’s say, after the above procedure, we rerun MCTS for five more rounds. This means all first-level action\_bat is explored fully and valued. After this, we restart from the selection phase. Without loss of generality, let’s say action\_bat= 0(arm 1) was selected(due to highest UCB value). This selected node is then expanded, and an arbitrary child action is played. **Note that, the state resulting from action\_bowl is stochastic.** During the expansion phase, the obtained state is simply believed to be deterministic and random rollout is done. The reward is –ve of runs scored at the end of innings. The following shows these:

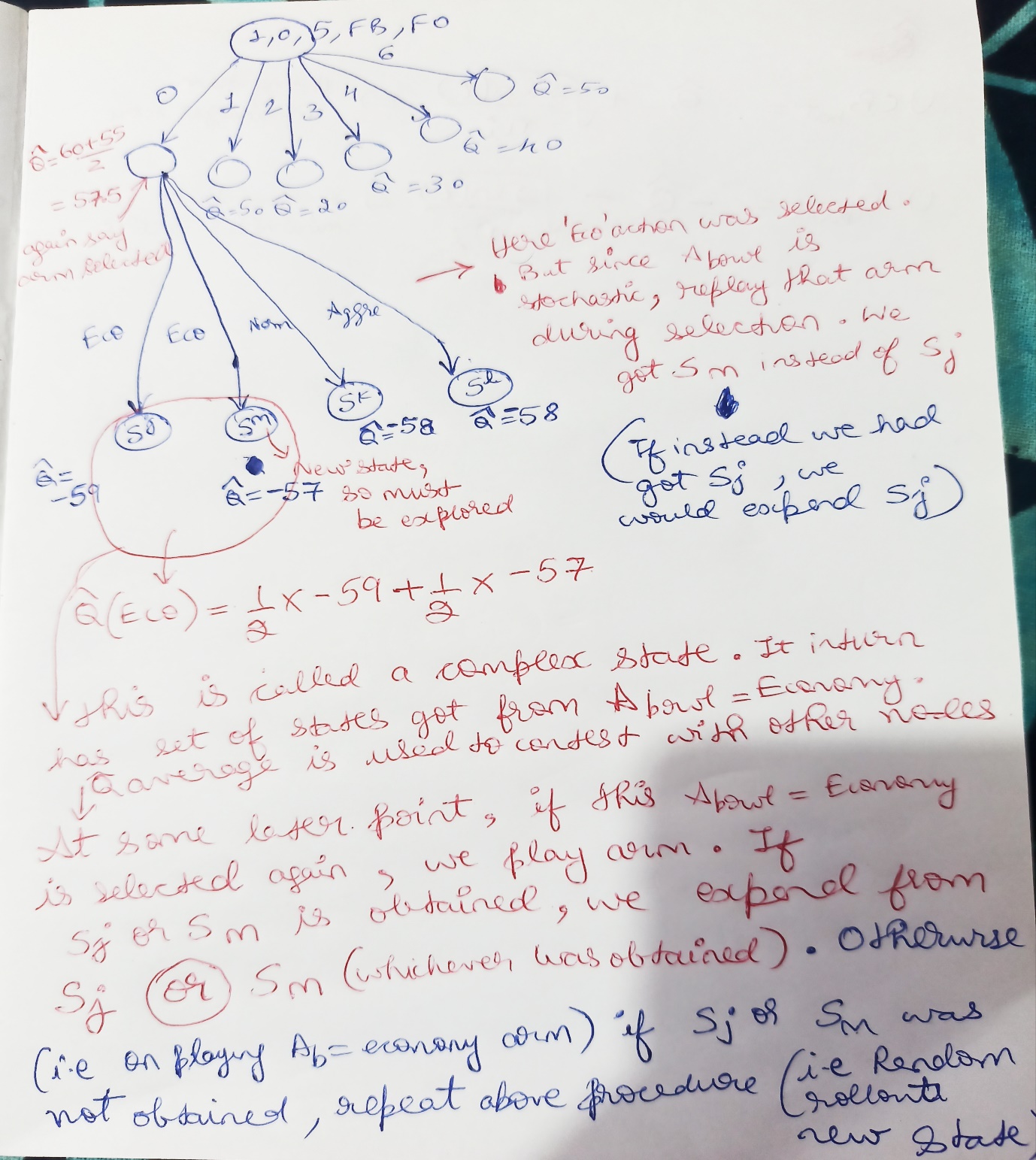


When an action\_bowl is selected in the selection phase, we don’t directly move to the state and continue. Instead, we play that stochastic arm and get to the state(say S\_m). If this state is **not the same** as any previously encountered states, then we add this new state as part of the **complex state**. This complex state has all states obtained by playing a particular action\_bowl arm. Then, the new state(S\_m) which was obtained now, is simulated with the random rollout and valued.

Suppose, if selecting action\_bowl did not produce new states other than those inside the complex state, then the algorithm moves to the obtained state itself and continues from there.

The value of a complex state is the expected value over all states in it. That is, the value of a complex node is the probability weighted average of all states in it.

Above explained details are shown in the image below:



**When contesting for the selection phase at the non-deterministic layer, we consider the complex node’s expected value and not its inside state’s value**. Once a complex state is selected, the action is played, and if the obtained state is within the complex state, then probabilities of occurrence of all states in that complex node are updated and the algorithm continues over the obtained state.

**The entire design above assumes that the simulator can be called and the “wicket,runs” can be obtained at any state. It also assumes that we are given ordering of bowlers and their features.**

**What to do in explore phase?**

As far as I have understood, since Pout and PRuns are not needed in the above design, explore phase is just used to determine an optimal batting order for a fixed given bowling order.

This can be done in the following ways:

1. When the number of samples is significantly less

For each batsman, consider two cases. One case is very bad bowler(all bowling features=1), Action\_bowl = agressive, Action\_bat = 6; this is the best possible run-scoring performance case for any possible bowler. Another case is, consider an extraordinary good bowler(all bowling features=5), action\_bowl = economical, Action\_bat = 0; this is the worst possible run-scoring performance case for any possible bowler.

Take a few samples for each case to estimate PRuns for each batsman.

Rank each batsman based on the median of Pruns and greedily arrange the batting order. That is, the best run-scoring player plays first

1. When the number of samples is large

Similar to the above case, pair each batter with each opponent bowler and take the best and worst-case performance against that bowler.

Rank each batsman based on the median of PRuns and greedily arrange the batting order. That is, the best run-scoring player plays first.

**Doubt in code**

* In the code, we are not given the current opponent bowler’s feature(self.feature\_bowler is passed in get\_team\_batting\_action function), which I have assumed as a mistake in code. **The entire design above is based on the understanding that opponent features in the current state of the game are known, the bowling order is fixed and known(atleast in phase 1) before innings starts. Otherwise, our MCTS building cannot account for changing the ordering of bowlers in the current design)**.
* In the class video, sir told that the explore\_compute\_value function is used to compute Q(s),V(s) etc..(If it is possible). Seeing that, I feel my thinking is either completely different or completely wrong. This is because the above design deviates from calculating Q(s) completely. Q(s) computation needs Pout and PRuns. Given so less number of samples in explore phase, coming up with Pout and PRuns for all possibilities is hard. Moreover, we are given the freedom to simulate random rollout directly (without self-play). **So I have not used Q(s), V(s) concepts as defined in the class. As I have understood Q(s),V(s) can be computed if we build an entire tree(minimax algo or alpha-beta). This is what is illustrated in sir’s toy example of compute value since state was very simple and we didn’t face combinatoric explosion. When we face such curse of dimensionality issues and use MCTS to solve it, I think Q(s) as defined in class in terms of V(s) doesn’t apply since we simply cannot find V(s) until horizon. Maybe we can use k-ply search to limit horizon, but that needs usage of heuristic evaluation function which is whole different issue to solve. So is my understanding correct sir?**
* In phase 1 of project batting team should know bowling team order. This is not passed to team class.
* Wickets\_left variable in match class doesn’t seem to be updated when wicket falls.
* Simulator access is not given to team class to be used in MCTS.

I am sorry if I overlooked some missing things in code above, since it is in match class, I have not control over it. Hence informed you sir.