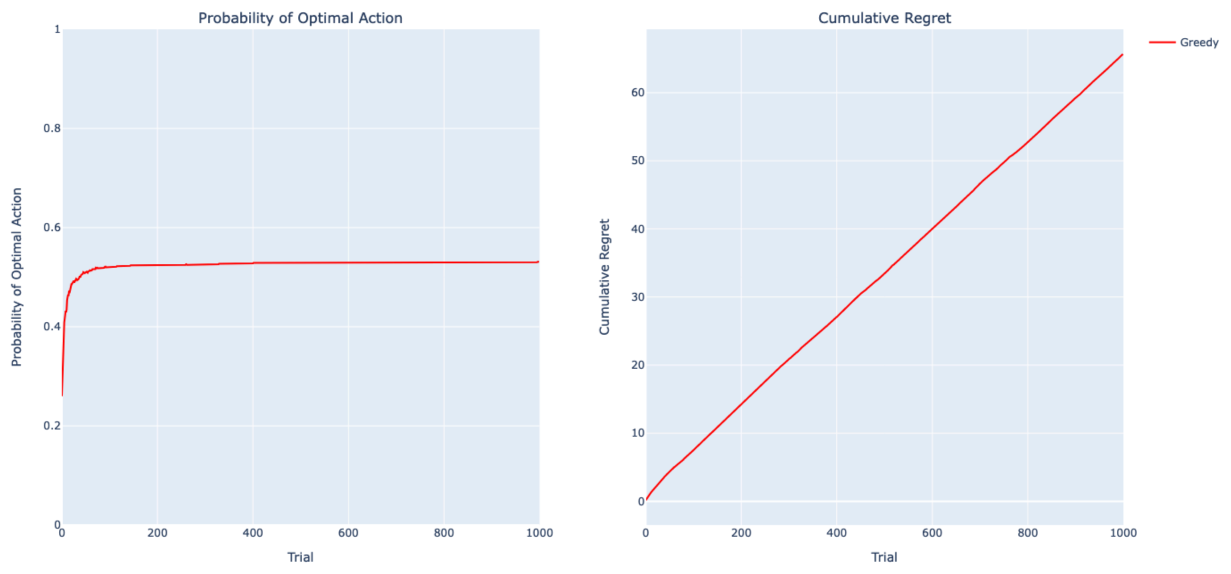
Hw#1 Report - Viv, Diego, Eylul, Aiden

Problem 2: The Greedy Player

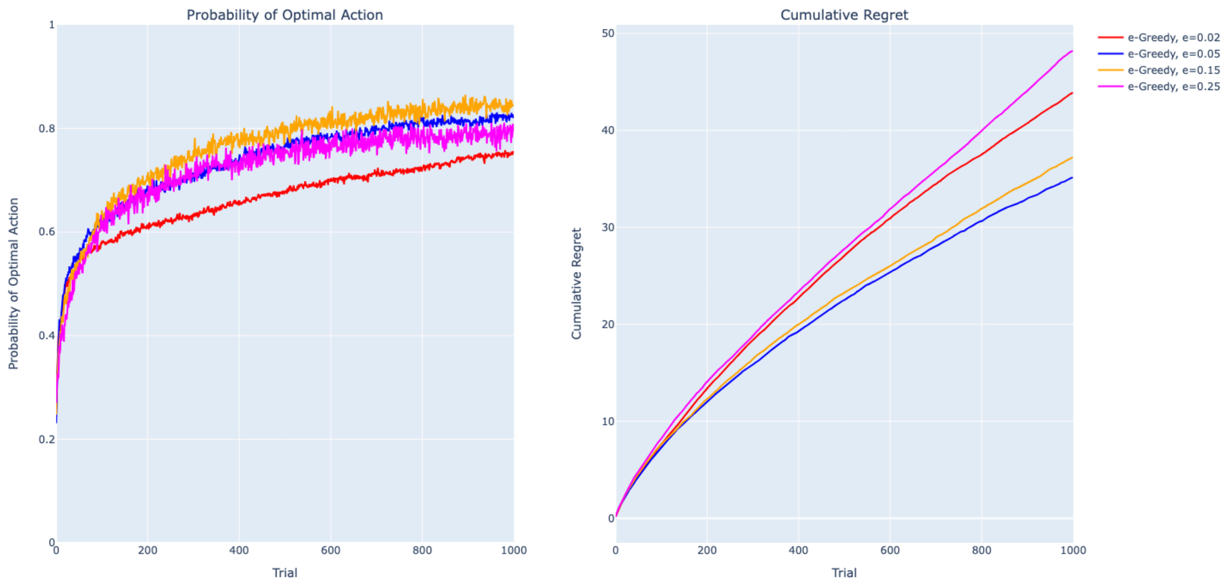
1. **Expectations:**
    - We expect our agent to pick the action with the highest quality value, however, we don't expect it to be optimal, since we're not exploring we're just exploiting.
2. **Sim Results:**



3. **Interpretation:** OPT(T=1000) = 0.53
    - The probability of optimal action never increases over time because it starts exploiting immediately. As we can see, it's not much higher than chance since the agent doesn't have enough samples.

Problem 3: The e-Greedy Player

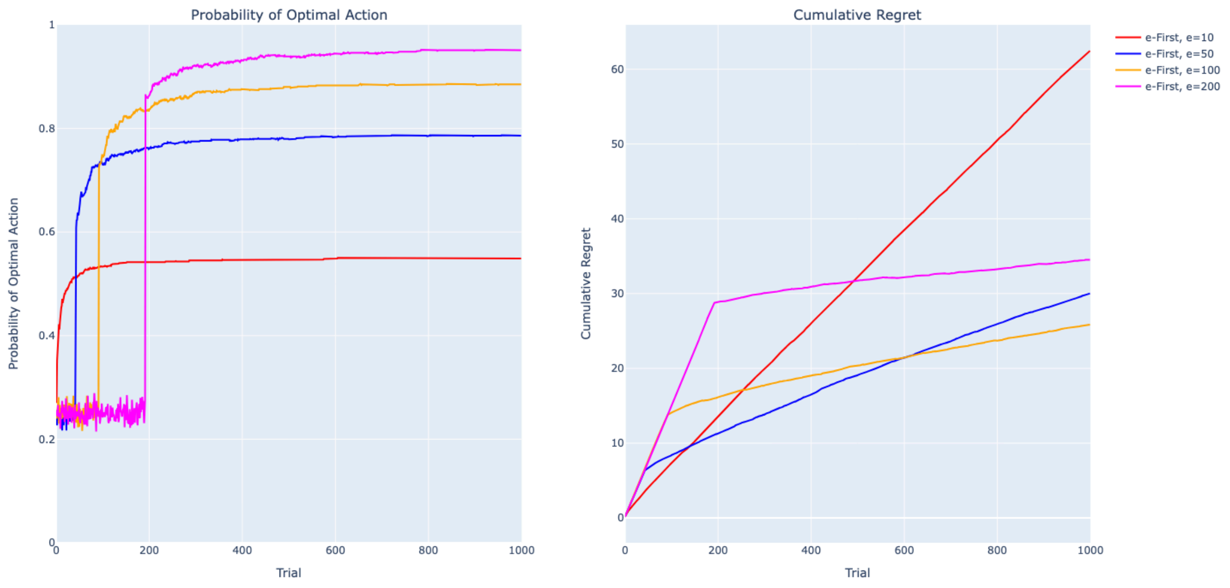
1. **Expectations:** We expect our agent to perform worse with the lower epsilon values since there wouldn't be enough exploration. For the higher values, we expect our agent to perform better when it comes to picking the optimal action but have different amounts of cumulative regret, potentially resulting from exploring too much the higher the value gets.
2. **Sim Results:**



3. **Interpretation:**
   - e =0.15 OPT(T=1000) = 0.834
   - e =0.05 OPT(T=1000) = 0.829
   - e =0.25 OPT(T=1000) = 0.788
   - e =0.02 OPT(T=1000) = 0.743
   - The one with the least cumulative regret e = 0.05 which is not the same as the one with the highest OPT (e = 0.15). It only spent 5% of its time exploring so it didn't build up that much regret because the probability of the optimal action was still pretty high.

Problem 4: The e-First Player

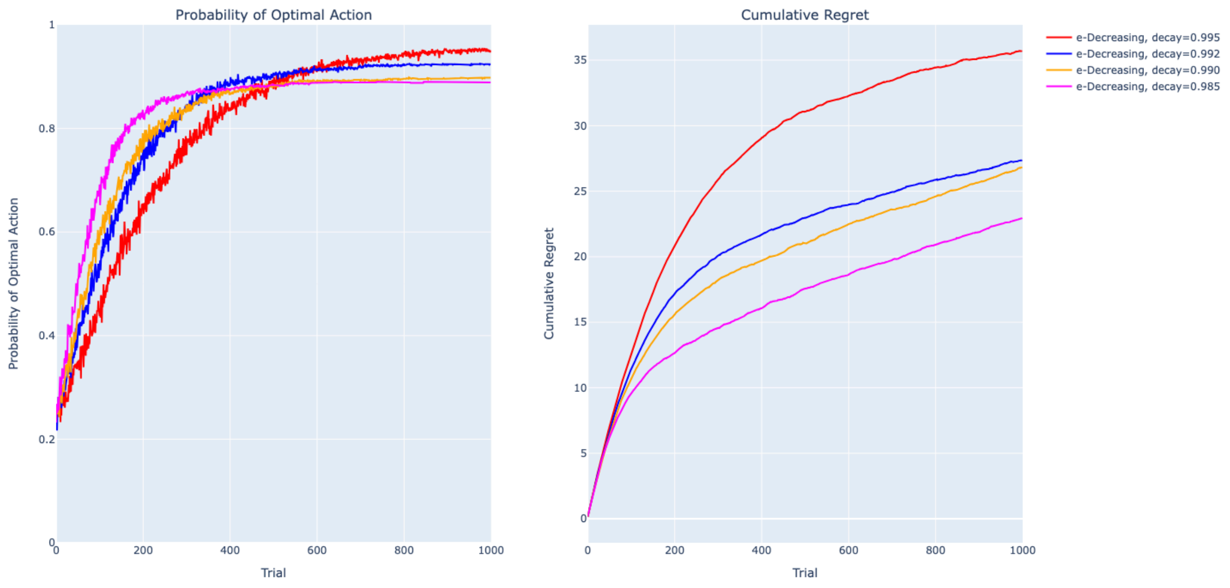
1. **Expectations:** We expect our agent to perform better with values that are more in the middle such as 50 and 100, but expect it to perform worse with more extreme values such as 10 and 200 since 10 explores too little and 200 may explore too much.
2. **Sim Results:**



3. **Interpretation:**
   - e = 200 OPT(T=1000) = 0.949
   - e = 100 OPT(T=1000) = 0.88
   - e = 50 OPT(T=1000) = 0.803
   - e = 10 OPT(T=1000) = 0.549
   - The one with the least cumulative regret was e = 100 which is not the same as the one with the highest OPT (e = 200). The OPT for e = 200 is higher since the agent explores more than e = 100 which means it keeps accumulating regret for another 100 trials.

Problem 5: The e-Decreasing Player

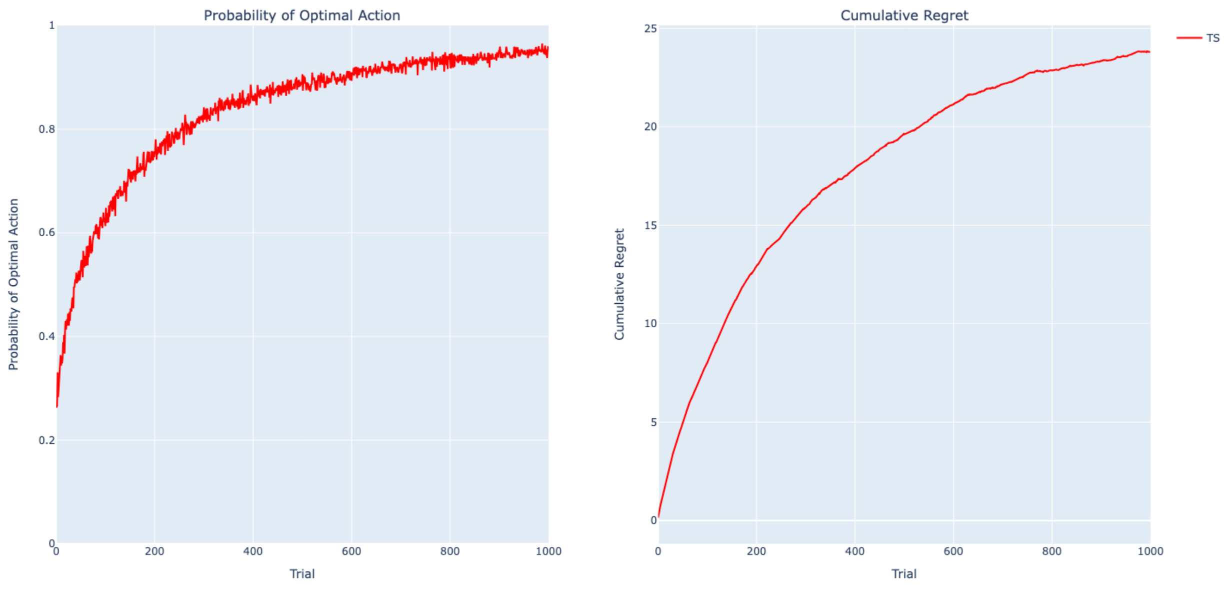
1. **Expectations:** We expect the higher values to perform better with the lower decay rates since it will "cool" epsilon slower and give the agent a better chance to find the best action.
2. **Sim Results:**



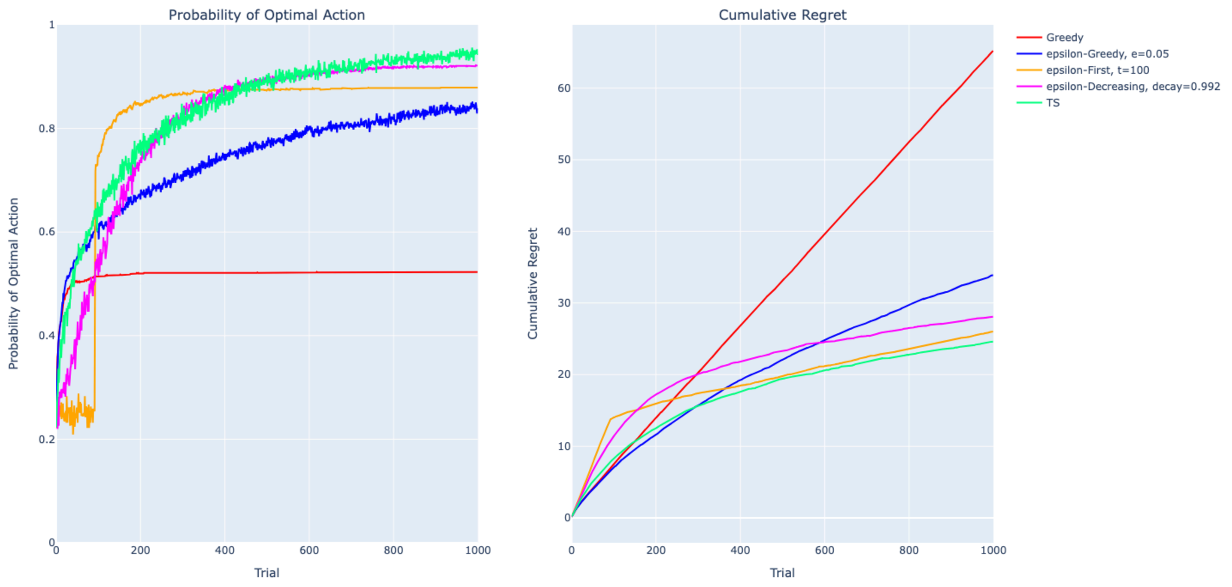
3. **Interpretation:**
    - e = 0.995 OPT(T=1000) = 0.959
    - e = 0.992 OPT(T=1000) = 0.93
    - e = 0.990 OPT(T=1000) = 0.908
    - e = 0.985 OPT(T=1000) = 0.887
    - The one with the least cumulative regret was e = 0.985 which is not the same as the one with the highest OPT (e = 0.995). This is because the agent with less cumulative regret picked the best action sooner than the one with the OPT. However, if there were more than 1000 trials, the agent with the highest OPT would eventually have lower regret than the current agent with the least amount of regret.

Problem 6: The Thompson Sampling Player

1. **Expectations:** The Thompson Sampling bandit player was implemented by taking a random beta distribution (based on PDFs) with the parameters being the sum of rewards (wins) and the times when the action missed out on the reward (losses).
2. **Sim Results:**



3. **Sim Comparison:**

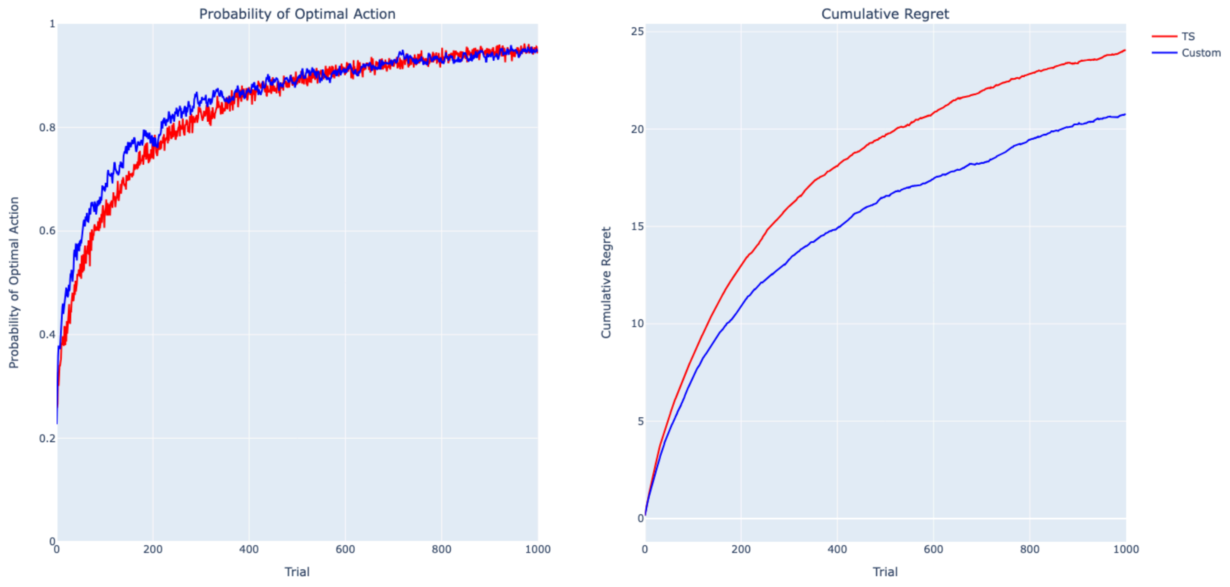


4. **Reflect:** The Thompson Sampling agent performed similarly to our epsilon-decreasing agent in terms of OPT. However, TS ended with the highest OPT and the lowest cumulative regret which means that this is the best-performing agent so far. Our Thompson Sampler does not need to rely on choosing some parameters whereas the other

ASRs require choosing some parameter(s) (e.g. epsilon) before the trials begin that we don't even know are right.

Problem 7: They Grow Up So Fast

1. **Expectations:** We followed the simple Upper-Confidence Bound (UCB) formula given in the specs. We experimented with different hyperparameter values (c) and in the end found out that 0.55 gave the best result.
2. **Sim Results:**



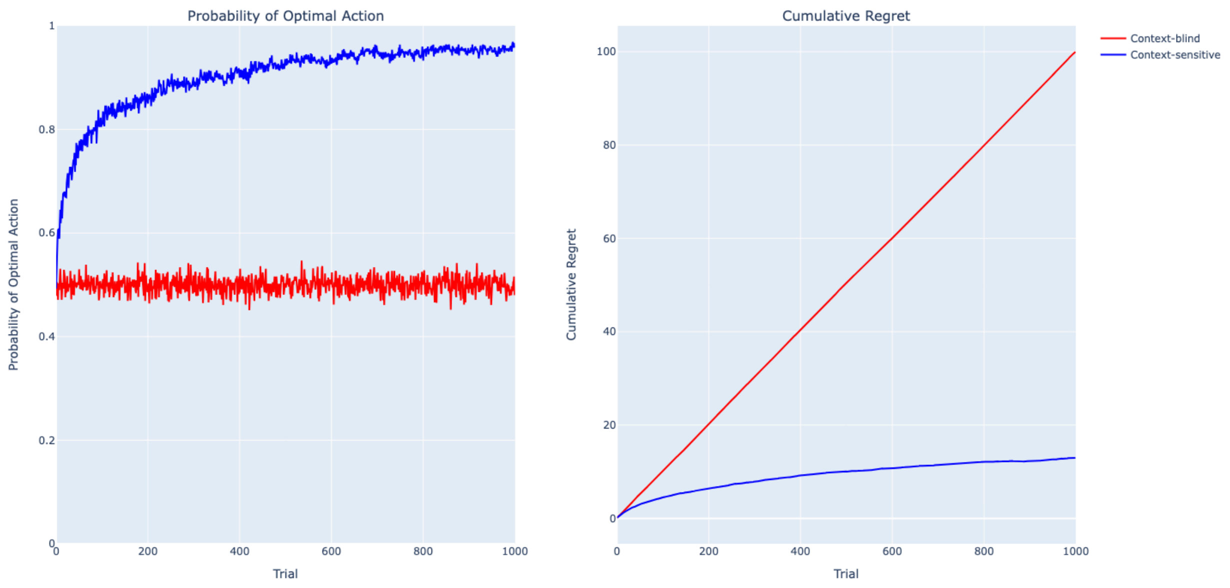
Problem 8: The Contextual Bandit Player

1. **Expectations:** Since the only discerning factor between the contextual agent and the context-blind agent were non-action variables associated with the action at time t, I realized that memoizing the context variables and its associated cardinality in conjunction with the action we took (with its decision variables and associated cardinality), there would be more reward that an action got with certain contexts as opposed to others. When making our choice (where we used the UCB ASR), we filtered out our history with the context of the action and then chose the highest UCB value for an optimal action. It

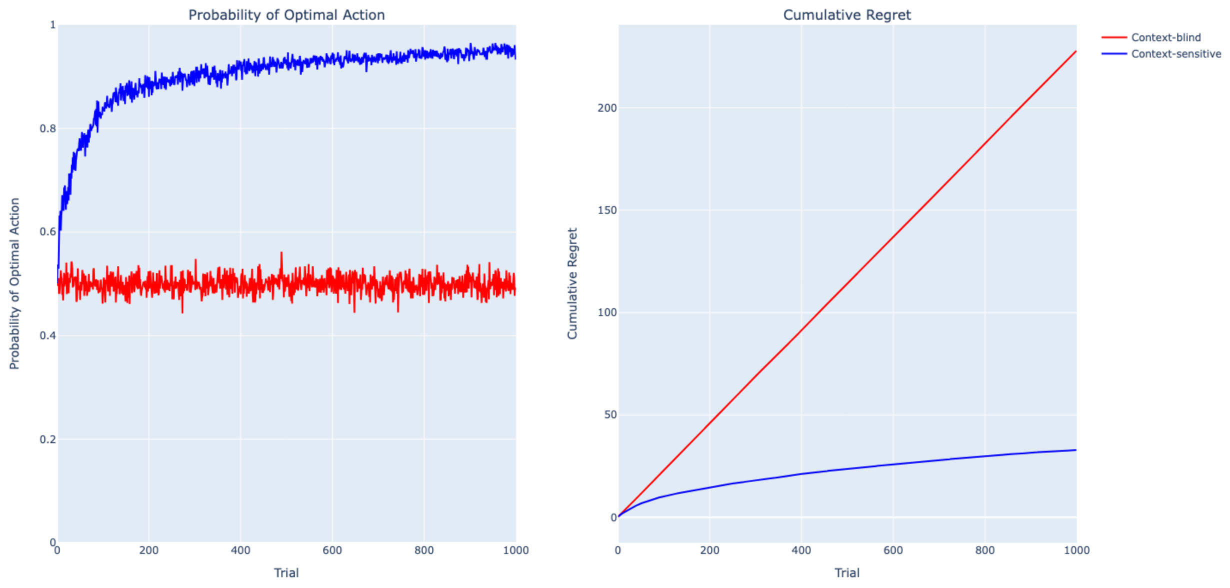
would ideally perform better than agents without context because our agent can associate certain actions leading to rewards with certain contexts due to the history. We stored the context associated with an action in the history combined with the action so that the reward would associate the two.

2. **Sim Results:**

Basic:



Advanced:



3. **Sim Comparison:** There are more context variables to work with, and we combined our action and context dictionaries. So there was much more processing to be done to store the rewards of all the actions with every possible combination of contexts in the advanced problem as opposed to the basic one.

Problem 9:

1. We chose the Dueling Bandit variant because it sounded cool.
2. In the Dueling Bandit variant, instead of having an independent reward for each arm, the focus is on comparing the rewards of two arms at a time. The algorithm receives feedback in the form of a pairwise comparison, where the user or system indicates which of the two chosen arms has a higher perceived reward. The goal is to iteratively refine the knowledge about the arms' relative performances through these comparisons, ultimately identifying the arm with the highest reward.
3. A notable strategy to address the challenges of Dueling Bandit is the Aggregated Pairwise Comparison (APC) approach. This approach combines pairwise comparisons to construct a ranking of arms based on their relative merits. The idea is to aggregate the comparisons in a way that captures the underlying preferences of the decision-maker. The APC strategy involves modeling the preferences in a linear order and updating the model based on the new comparisons.
4. In game development, Dueling Bandits could be applied to the optimization of game features. For example, if a game designer needs to experiment with different variations of a game mechanic, such as two different character movement algorithms. Each algorithm variation is treated as an "arm" in the Dueling Bandit framework. In lieu of just depending on user feedback, the designer could use pairwise comparisons to determine which one is preferred in terms of user experience and performance. This could be especially relevant when assessing subtle differences that might not be obvious through standalone evaluations.