**STATEMENT OF WORK V2:**

1. **Rationale Statement:**

Since Alpha Go project and the IBM Watson Jeopardy competition, there’s been a lot of progress with Reinforcement learning. What it shows is that there’s a lot of potentials for Machines to perform better than humans even in tasks which we do not completely understand.  
  
In our situation, there’s been a lot of interest in the T-Rex game and we’re here to prove this same concept of how we can train machine learning model to perform better than the average human while playing this game. Although this is a simple application, we believe its application is very scalable and could go way beyond just the T Rex game.

1. **Refine Problem Statement:**The problem here and with most reinforcement learning projects is being able to teach a ML model how to perform task which even we humans don’t have concrete rules to perform. An example of such tasks would be teaching a computer how to walk; the best approach to do this would be the same way babies learn i.e., they get punished for their mistakes by falling when they miss their steps and get better with the more steps which they take.   
     
   That is our proposed solution here. We plan to implement a reinforcement learning model which will get rewarded per each correct move and gets punished when it makes a mistake. So, for instance if it makes a mistake, that sequence of action gets scored lower and the probability of that action being repeated becomes lower and hence we come up with a model that is self-learning.
2. **Identify Data Requirements & Data Sources:**Unlike supervised or unsupervised ML models, with reinforcement learning it works in self-learning manner i.e., all data is self-generated from the environment which it works in. So, in our situation the inputs and data which we will be working with include the inputs of the environment such as the height of the nearest obstacle, distance to the nearest obstacle, speed of the game, the list of actions for our model to choose from, and the outputs are the estimated optimal actions to take based on input, as predicted by the model. Because our data is self-generated and the number of data points, we have depended on the number of times the model makes a decision about our next move, we require that the game be run for many episodes in order to get proper coverage of the domain of our inputs and guarantee an accurate model.
3. **Data Assumptions, Limitations & Constraints:**When it comes to assumptions, we are usually referring to guesses which we make for missing data in most situations. In the case, because of the nature of reinforcement learning and the fact that the data is self-generated based on the needs of the algorithm, we do not need to make any assumptions about how we can handle missing/improper values. In our model, our assumption is that the input variables i.e., the details about the current game state will be independent of one another, there will not be much covariance in our data. Our dataset is limited by the fact that the data is generated in real time and is partially random, meaning that we cannot insure near-complete coverage of the domain of our inputs and some game states may not be accurately accounted for by our model if similar states have not been seen before. Our data is constrained to numbers, of which speed is always positive, heights are always below a certain value (the height of the floor), and distance to nearest obstacle is allows non-negative.
4. **Test Process:**

In our situation, we don’t have an actual test dataset to work with, instead we have the game to test our model with. We will continuously save our models after a game is played (a game is over when the model hits an obstacle). So as time goes by, we compare each model’s performance with that of the previous model, if it performs better, we override the current model with the better performing model.   
  
The testing phase comes when we load the model from memory and allow it to freely play the game according to its accumulated data and knowledge based on the training it’s undergone. We will check the fully trained model by hand for proper understanding of how the game functions and check the scores that it achieves while playing based on the knowledge it has gained, without updating the model as done during the training phase.

1. **Reinforcement Learning Architecture:**

For our T-Rex Dino bot, we will be using an off-policy, model-free architecture. Model-free algorithms are often described as “trial and error” because they do not consider the probability that choosing an action *a* while in state *s* will result in that agent being in the next state *s’*, referred to as the transition probability distribution. Off-policy algorithms do not always follow the policy that they are learning (hence the name off-policy), and instead sometimes choose an action that is not predicted to give the best reward.

1. **Algorithm:**

We will be using a deep Q-learning algorithm (off-policy and model-free) that will be able to learn to play the game. Q-learning is a machine learning algorithm that uses the Q-function, which predicts the reward given by taking action *a* in state *s*, to predict the best next action. Traditional Q-learning uses a Q-table, which is represented by a matrix with states as rows, actions as columns, and the estimated reward for taking action *a* in state *s* as entry *s*,*a* in the matrix. Because the state space is not discrete, a neural network is used to represent the Q-function, which is deep Q-learning.