Capstone Flappy Bird Project

We began our capstone project by first deciding on what route to take. Even though we have struggled to deploy a successful model in the past, we felt it was important to at least attempt to run this model to improve our technical skill and demonstrate the advancements in our understanding of this applied science. Because we decided on this route, our first steps required us to do research to better understand the technical components of our project. For us this meant a combination of reviewing the recorded class lectures and module powerpoints, watching instructional and informational videos on youtube and reading through articles and webpages. The topics researched included understanding how to use Pygame, how to code a Flappy Bird clone, and a basic understanding of: Reinforcement/Deep Q Learning, AI agents, and how to deploy them to accomplish this task.

After doing some preliminary research we have a better understanding of what this project will require. We then decided to begin the coding process by following the Azure studio tutorial provided for this project. This is our first time using the Azure environment, so we expect there to be a bit of a learning curve. We begin by setting up our compute and Jupyter notebook as the environment to run the model. We install and import the required libraries. Then install the PyGame Learning Environment and clone the repository for PLE and the Flappy Bird game code. This takes a little bit of exploration and research as there are no directions specifying which repositories to clone. We do successfully find the correct repositories and use the assistance of AI to implement the coding portion of cloning the repositories and initializing the game environment. This is when we run into our first major hurdle. We receive an error that indicates some of the image assets cannot be accessed. Because there is more familiarity with how to problem solve errors in Colab, we decide to try to get the model running in Colab first and then transfer it back to Jupyter lab and debug whatever issues might arise from that. Using the help of Claude, we decide to solve the image asset problem by using a method that directly downloads the image assets from the individual URL sources where they are stored. This method is successful. If it hadn't been,

we were considering using gymnasium instead of pygame, but because we had success this seemed like an unnecessary leap. Once we get the PLE and Flappy Bird game running successfully we use the skeleton code provided to put in place the pre-trained MobileNetV2 with the appropriate training parameters, set up the DQN agent, set up the training stage and then the testing stage. Because we had to use a custom solution for asset loading, the skeleton code provided was put through Claude to be amended so that it could work with our existing code. This process took patience as we fed the errors into our AI resource, implemented its solutions and continued this process of trial and error until we had a functional model. Because we were still running on Colab at that point, the training process was clearly going to take many days running on that particular compute, so we moved the code over to Jupyter Notebook. This took an additional bit of trial-and-error code debugging to get it working, but we were successful.

At this point we start the process of observing the training to see if there are any obvious problems that need to be addressed. Because we are novices and have little experience running and observing models it was hard to imagine how long it would take to train the model when we began. When we start training, we realize it is going to take around 16 hours to train for all 500 iterations. We still have that time, so we monitor the first 5-10% of training before realizing our model is failing to train with positive results. We input our results for the AI to analyze and Claude lets us know the basic reward system isn't performing well and the epsilon is decaying too quickly, which is discouraging exploration of the agent. We alter the reward system to include positive rewards for staying alive, good positioning near pipes, successfully passing pipes and surviving for longer periods. We also adjust the agent hyperparameters to slow the epsilon decay and raise the epsilon minimum. We begin training again, at this time we decide to leave the model training overnight to see how it performs. When we wake up, we discover the runtime has somehow stopped and we have lost our training progress. Because this process of trial and error is becoming extremely time consuming, we ask Claude for suggestions to speed up the training process. We experiment with changing hyperparameters like learning rate, batch size, epsilon decay, and adjusting

the reward system. After that isn't showing improved results, we decide to implement a different model architecture at the behest of Claude. It is a Faster CNN network to replace the MobileNetV2 and Claude seems to believe this will quadruple the speed of training. We re-start training and at the time of writing this report, we are in the process of observing training. We have yet to see results beyond 20% of training because even after implementing the Faster CNN, the model is not training more quickly. Each iteration is taking between 1 to 2.5 minutes, still promising a total of approximately 16 hours of training before we will see results.

Because we will have to submit this report before this training ends, our ability to fully analyze the results are hindered. However, we can provide and analyze our training data so far and hypothesize about our future results. At the time of reporting the training has just completed its 88th iteration of 500, so we can observe 18% of our training results. So far, the average score hovers around 23, and the best score is 98. As far as we can figure, a perfect score in a game of flappy bird yields about 525 points. This doesn't necessarily bode well for our model unless we see a vast improvement in score average as it learns. Because this isn't even a quarter of the data, it is hard to know if the results we can see so far are indicative of trends to come or demonstrate the slower and less competent early stages of learning. Because I would like to see the model perform well, we are hoping the early stages of training produce a lot of "mistakes" so that the agent is able to gather a lot of data about what not to do so it can home in on successful strategies as training proceeds. However, if the training data is indicative of how the model will continue to perform for the rest of training and testing, we have low expectations of a high performing model. So far, the game score, which documents how far the bird makes it into the game, indicates the Flappy Bird object is rarely making it past the first of five pipes. It has never made it past the second pipe. The Epsilon reached its minimum around 54/500, and after that we saw an uptick in negative game scores. Based on what we understand about Epsilon decay, this could indicate that the model stopped experimenting too early and did not develop a sound method that will lead to successful runs. Even though we will not be able to report officially on our results we will

be monitoring them out of pure curiosity, to compare them to our current hypothesis and so that we can truly walk away from this project with a better understanding of how this technical process works.

Despite not being able to report on our model's final results, we still consider this a very successful experiment. In comparison to our group's midterm project, we have achieved significantly more. Just getting a model to run without code errors felt like an enormous victory. We also vastly improved our technical understanding of model architecture and the purpose of various functions and hyperparameters. The process of debugging with help and explanations from an AI assistant, despite being frustrating and time consuming, provided a hands-on learning experience that gave insight and context to the parts of the code that were hard to understand as novice programmers. Being able to read an analysis of the training data as it was being produced allowed us to understand and catch problematic patterns that we might otherwise have missed, or would have caught much later, delaying corrections and wasting precious time. Even the ability to navigate Claude, input the relevant data, parse the responses and implement the suggested changes is a skill that we strengthened, and will prove extremely valuable as we continue on our journey to complete this degree.

If we were to improve this model, which aims to teach an agent to play Flappy Bird successfully, we would begin by trying to figure out if it is possible to speed up the training process. We would either invest in the GPU compute or potentially run the model locally. Claude suggests that we might see 5-10x acceleration in the speed of training if we switched to GPU. One of the reasons this appears to be is that GPUs process batches of images simultaneously while CPU processes images sequentially. This would greatly reduce the time required for the model to assess all of the images that provide data to the model about its performance. We theorize this is the biggest hurdle to speed, as we have tried to implement various hyperparameter changes and even changed the model architecture to no avail. One could also theorize that 500 training iterations might be unnecessarily repetitious. Without the complete training data to observe, it is hard to know whether or not that is true.

There might be a reason for so many iterations and it might be that significant improvement requires this amount of trial and error. If, at the end of 500 plays, the model has not continued to improve steadily, we could experiment with the amount of training loops to shorten the training portion. Other potential changes we could make to the model that would improve its performance and learning speed might include changes to state representation. Implementing frame stacking, which uses a stack of frames or images instead of a single image to capture motion information and would add time information. This would give more accurate information to the agent about how it should move. Also we could give it more information about high-level features of the game such as distance to the next pipe, velocity of the bird and time since last flap. All of this data would hopefully give the agent enough information to play the game more successfully. One last suggestion made by Claude to improve the performance would be to exchange the DQN model for a Rainbow DQN model. This would allow a combination of types of DQN (Dueling, PER, Noisy Nets, Multi-step learning, Distributional DQN) to learn simultaneously, which greatly increases the potential for agents to learn successfully because it is using multiple learning strategies. One last addition that could potentially improve both the speed and performance of our model is implementing a process that would allow multiple games to be played simultaneously so that more data could be amassed more quickly.

In conclusion, this project was challenging in a great way because it forced us to grow. We grew as students, because we were confronted with daunting technical problems and worked hard to figure out how to overcome them. We grew as programmers because we were able to show significant improvement between our last attempt to run a model and this one. Most importantly, we grew as future AI professionals. We are demonstrably more knowledgeable when it comes to understanding the platforms we can do this work on, like Colab and Azure and Github, how to start running a model from scratch and how to use existing AI to most effectively assist us in advancing our understand.