Sean Hasse

Ethan Hunt

Christian Cervantes

Senior Design Project Dota II

## Design of the Project

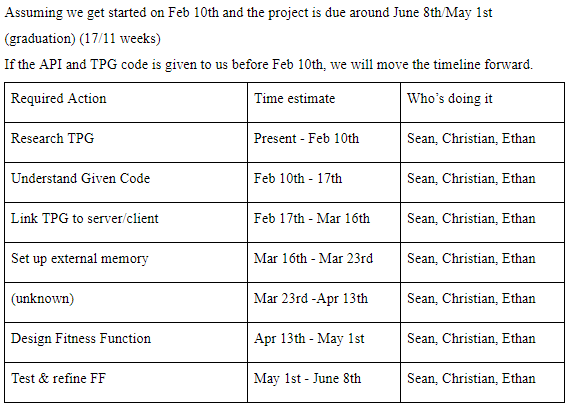
Our senior design project was to create an entry for the GECCO 2020 Competition, ‘Dota 2 1-on-1 Shadow Fiend Laning Competition’. The purpose of the contest is to build an A.I. capable of beating the built-in Hard A.I. in a symmetric 1 vs 1 environment. The contest organizers provided code for a TPG, or a Tangled Programming Graph. This will be our method of building a self-learning A.I. and we will explain more about it later on. We were also provided code snippets of what a self-learning Agent would look like. Our goal for the event was to make a self-learning Agent based on the code snippets using a TPG, and train it until it can consistently beat the in-game Hard A.I. For our project, however, we took a more modest goal of creating the code for the self-learning Agent and implementing a Fitness Function that can teach our agent how to play the game.

The game Dota II is a popular online game, the basic idea of Dota II is that there are two teams of five players that compete against each other to destroy the others towers. The game is presented on a single map, and the players can choose from a selection of 119 different characters each having their own unique abilities. Over the course of the game the characters level up becoming stronger, players have to choose the best plan of action to take to defeat the enemies. On the map there are multiple lanes that players can take to get to the enemies tower, and along the way there are other towers that can be destroyed. There are also in game bots called creeps that travel along paths to attempt to attack any opposing heroes, creeps, and buildings in their way. The game mode that our bot will be competing in is known as 1v1 Mid, meaning that there is only one player per team, and they both fight for control over the center path. To win 1v1 Mid, either player must kill the opponent twice, or destroy their first tower. Some of the mechanics that our agent will be required to utilize are kills, last hits, denies, tower health, levels, net worth, wins, game time, and deaths. A last hit is when our agent lands a last hit on one of the enemy’s creeps, which will give our agent some experience so he can level up and he will also receive gold for it. Denies are sort of a complex thing to do, it is when one of our agent’s creeps is near death, our agent can kill him so the enemy does not gain experience or gold for the kill. The in game currency is gold, which is acquired passively over time and through killing enemy creeps. The game does have a timer, but for our purposes it is sped up ten times so our games do not take long. For our project the hero that we will use is the Shadow Fiend, and he will be against an opponent Shadow Fiend. He can use different tactics along the way to get the best possible outcome and help him win. The way our project will be graded when we submit it to the organizers is our agents ability to defeat the enemy bot.

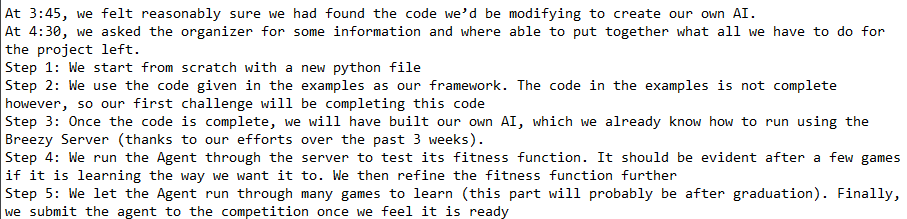
The TPG is the main tool used to get our agent to learn and evolve. The basic functionality of the TPG is it will create a collection of programs all of which will return a bid, which is just a number that will be associated with an action that our agent can take. The program with the highest bid is the action our agent will execute, if the action is useful in any way the gene is mutated and passed to the new generation of agents. The TPG has six python files which are, agent, learner, program, team, trainer, and utils. The agent class is essentially a wrapper class for a team, the agent class is just an interface that we can use. The agent class can perform a few functions like reward an agent at the end of the game, or select an action based on the world state. The learner class is the class that is in charge of the training, it does all the tasks of an evolutionary algorithm. Each learner has a program that will return it a bid to select an action to take, then once the action is performed it can choose what agent to evolve.. The program class does not really do much; it only obtains a bid to pass to the learner class. The team class is the main building block of the TPG, each team has multiple learners which decide the action to take. In the class we can add and remove learners, and also mutate the learners to help the progress of our agent. The trainer class is used in the functionality of the TPG, we create trainers to store various evolutionary parameters. One thing the trainer class does that is important is sorting the best agents to mutate over to the new generation. The only thing the utils class does is return a probability that is used throughout the TPG to perform some functions like choosing actions for our bot to do.

## Creation and Implementation of the Agent

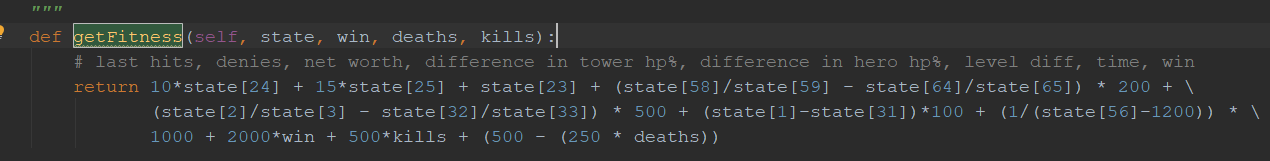
At the start of the semester we created a timeline for our project, given that not all of the code necessary to begin was available to us at the start.



We went forward with researching TPGs, trying to understand how they work and what would be required of us when we were eventually programming with one. We did receive the code (via a GitLab release) for the TPG on February 17th, and for the server on March 1st. The code for the TPG contained all the files needed to implement a TPG using Python 8. We spent the next two weeks reading through it and finding what each class did. The TPG was fairly complicated, so it took longer than originally expected. Since we did not have the server code, this seemed reasonable however. When we got the server code on March 1st, we were able to quickly accelerate our progress. The server code included instructions for how to connect to the server that will control the Shadowfiend character in-game, as well as code snippets of what an Agent would look like. We spent a week going through the instructions for setting up a connection to the server, until we were finally able to run a game of Dota II using the sample Agent on our computers.

It was not long after this point, however, when we were forced to change our schedule to account for the Covid-19 response. After our group had settled down, we started meeting again by March 25th. We moved to online meetings via Discord, and we recorded our meeting goals and accomplishments via notepad documents and shared them with Professor Zhao via Github. On the 25th, we re-evaluated our current standing, goals and schedule. 

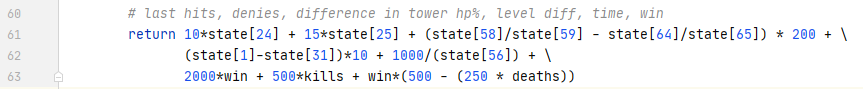
We began meeting with Professor Zhao via Zoom on Thursdays, and meeting on Mondays and Wednesdays regularly. We started following our new schedule and made quick progress. We identified which sections of code from the sample Agent would need to be worked on and improved, and ported over the working parts of code to our Agent. By April 17th, we had successfully built our own Agent, and gotten it to run on the Breezy Server, where it played Dota II. At this point, we had crossed one of our most difficult hurdles of creating our own Agent that could play the game based on the sample Agents. On April 20th, we created our own Fitness Function, which is what rewards the Agent for performing well in-game. We can tune the Fitness Function to reward certain behaviors we desire more heavily than others, and punish behavior we don’t want the Agent to learn. Our initial Fitness Function is shown below.



The criteria we used to create our fitness function were last hits, denies, net worth, the health differences between the towers of each character, difference in health between each character, level differences between characters, the game time, winning, kills and deaths. We chose kills because it is an important aspect of the game if our agent killed the enemy bot twice this would ultimately give him the most points because he would get points for killing the enemy, and a massive boost for winning the game. Denies and last hits are important for the agent because this helps our character gain experience to level up, and also to gain gold which would help his net worth giving him a higher fitness. We decided to add health differences to our function because it would give our agent the incentive to attack the enemy tower, if the enemy tower was ever destroyed he would win the game, thus giving him a higher fitness. We also added level differences so that our agent would get the incentive to level up and try to be a higher level than the enemy bot. Finally we added the game time to the function because if the game ended quicker and he won he should be rewarded slightly for it. Once we implemented our fitness function we started to test our agent.

In the beginning of our testing phase we never really knew where anything was being saved to, we would get a message saying that features were saved to a database. At first we just ran a few games just to see if we would see any slight improvements. We had tested our agent against one of the given agents, and it seemed to be working. Eventually we realized that none of our progress was being saved, so we had to implement a way to save our agents fitness, and load it. Once we figured out how to save our agents, and get the progress to load we went back and continued to test our agent for progress. Initially we would simply watch to see if there was any progress being made, and then we started looking at the statistics of each game to see what was changing. After running a few games we noticed that our agent fitness would sometimes give a negative value. We had to go back and refine our fitness function to see what we should really be focused on. After reviewing our function we revised it, and went back to testing. In total we ran about 200 or more games, and in general our agent learned a lot of ways to play the game. In the beginning he would simply just run straight past the enemy and eventually die. Then he would run north east until he ran into a tree and sat there the whole time, eventually losing. It was evident that our fitness function required fine tuning.

To accomplish this, we continuously ran games using our Agent and watching his actions to see if he was performing actions we wanted him to perform. We compared that with the score he was receiving for performing those actions and fine-tuned the Fitness Function to the one shown below.



We changed a few things from our initial Fitness Function. Firstly, we removed the comparison between bot health totals. We found that this was encouraging the Agent to play passively, when we needed it to learn aggression first. We also removed the value of net worth, as it led to the Agent trying to drag games out, also encouraging passivity. We tweaked many values as well, particularly for the level difference and time. We found that the level difference was often too pronounced, and overshadowed the positive actions it was taking, so we reduced the negative impact of having worse levels. We also found that when it was being aggressive, the game ended quicker than we expected, and so the Agent was punished instead of rewarded. Now the Agent is rewarded for finishing games quickly to encourage aggression, but it will not be punished for letting games continue longer either. We also added a condition to the reward for not dying, that it will only be rewarded for not dying if it wins. This was another problem of encouraging passivity.

To show the progress we made, we recorded several games to compare the progress made by the Agent. First we measured the Agent not using any Fitness Function (essentially always a blank slate), then we measured after training it with the first Fitness Function, and again after training it with the second one.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Agent | Level | Last Hits | Denies | Tower HP | Game Time (s) | Deaths | Kills | Wins |
| Base Agent, no FF | 2 | 0 | 0 | 1605 | 265 | 2 | 0 | 0 |
| --- | 2 | 0 | 0 | 0 | 1121 | 0 | 0 | 0 |
| Agent w/ 1st FF | 5 | 2 | 0 | 1725 | 534 | 2 | 0 | 0 |
| --- | 6 | 6 | 0 | 1427 | 567 | 2 | 0 | 0 |
| Agent after new FF | 5 | 5 | 0 | 1745 | 445 | 2 | 0 | 0 |
| --- | 4 | 7 | 0 | 1688 | 447 | 2 | 0 | 0 |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

As you can see, the Agent with no FF is pretty pathetic. It died very quickly, or remained entirely passive without accomplishing anything. The first FF was significantly better, as it learned to be more aggressive and gain levels and last hits, but it was still very inefficient. These were two of its best games after a very long time training. The new FF was able to get similar results in terms of levels and numbers of last hits but in less in-game time. We are also able to find a higher correlation between the Fitness Score and the Agent’s actions, leading to the Agent becoming more consistent as it continued to train.

## Postmortem

Working on this project together we all have learned a lot. One of the biggest things we learned was python, originally the organizers said that this project was written in C, but once everything was uploaded it was written in python. This was one of the biggest challenges because none of us had any experience working with python. The fact that all of the python was really high level, made it very difficult to code. One thing that we had to code on our own were the save/load functions in python. We did have a few examples from some of the files in the TPG, so we used those as references. We decided that we would save at the end of each game session, and load at the beginning of each group of games. One thing we found was that the files the organizers gave us were not indented correctly, because in python indentation and white spaces matter for some things. After fixing this problem we had working save\load methods that we implemented on our own. Some other things we got to work with were tangled programming graphs which are a fairly new tool used in genetic programming to help AI learn. Another thing we worked with was client/server communication, which is very helpful for us to know how to work with. This whole project is really beneficial to us because AI is a big topic in our field as computer scientists, and having some working experience will definitely help us in our future. At this point in the project we will continue to run our agent so it can learn. The competition deadline is in June so we have some time to continue to work on the project before we submit it. Overall we are very proud of all the progress we made, it was very nerve wracking in the beginning of the semester because they uploaded everything fairly late. We do think if we had those extra two weeks we could have possibly accomplished a bit more. Either way we are happy with our results.