PROJECT: DYNAMIC ARMY PREDICTION

ECS170 Spring 2018

GROUP MEMBERS

Jiehong Jiang, jjhjiang@ucdavis.edu, 914974552
Colin Beardshear, crbeardshear@ucdavis.edu, 914385840
Malini Pathakota, mmpathakota@ucdavis.edu, 913241367
Hoseung Lee, hoslee@ucdavis.edu, 914001975
Lacey Campbell, lbcampbell@ucdavis.edu, 912170808
Sri Kavya Dindu, sdindu@ucdavis.edu, 913022557

CHANGES IN GROUP MEMBERSHIP

None.

I. PROBLEM

StarCraft II is a RTS game where players gather resources and strategically spend them on units, buildings and research in order to achieve the ultimate goal of overpowering the opponent and winning. Because this game relies heavily on choosing the right strategy early in the game, being able to predict the enemy's army formation becomes a very valuable skill. Current army prediction algorithms learn from data gathered from gameplays on build orders, amount of resources, and current army formation to predict which strategy the enemy is using. However, in addition to determining the strategy the enemy is using, being able to predict every specific build the enemy makes during certain periods of time during the game drastically increases the player's chance of victory. If the player can correctly predict the exact units that are coming for them at specific times during the game, they can make better decisions on how to counter the enemy. Our objective is to create an algorithm that predicts the enemy's army composition given the information that's been gathered by scouts, including construction units, buildings, and current army formation. We call this *Dynamic Army Prediction*.

There are currently many algorithms that predict the strategy the enemy will utilize based on their army composition and information gathered from the scouts, however, there are no algorithms specifically aiming to predict what the enemy will build next based on the units they already possess, thus, making our solution to the problem of army prediction unique.

One such algorithm that predicts the enemy's overall strategy rather than immediate builds is a Bayesian Model for Opening Prediction. This project predicts the enemy's opening strategy and its possible technology trees; however, it differs from our approach in two main ways. First, the model predicts a strategy whereas we focus on current army composition and ultimately future army composition. Secondly, it predicts the strategy at the beginning of the game whereas as our solution is a dynamic/ real time implementation. Another similar project is Combat Prediction based upon the Lanchester Attribution Laws. This project uses the above-

mentioned model to hypothesize about the enemy army's combat strategies, technology, and resources. It addresses a much wider scope than our solution and unlike the previously stated project, Combat Prediction aims to work in real time. Link to enemy strategy prediction algorithm: https://hal.archives-ouvertes.fr/hal-00607277/file/OpeningPrediction.pdf.

Dynamic army prediction is a very interesting problem to solve because, to our knowledge, not many attempts have already been made to solve this specifically. Most prediction models focus on the overall strategy the enemy will utilize based on their opening build order while we want to focus on immediate builds at any point during the game. This problem is also technically interesting because it is difficult to determine the relationships between every unit created as well determine how these relationships evolve as time progresses within the game. This requires a large amount of training and implementing approaches such as linear regression.

CHANGES FROM ORIGINAL PROBLEM

Our main objective of creating a model that predicts the enemy's build order based on the information that has been gathered from scouts has stayed the same. However, we added a component to this goal: determining the result of the game using the amount and total health of a specific unit type that both sides have near the start of the game. We call this Win Loss Prediction. Being able to dynamically predict the enemy's build order based on scouted information is vital to winning the game, however, also being able to determine the player's chance at victory based on their starting strategy and initial scouted information further increases their chance of success by allowing them to edit their approach. This new component of our objective further utilizes the data we had already extracted on the number of units and times they were created and allows us to explore different statistical models.

II. POSSIBLE AI APPROACHES

A possible AI approach to this problem is creating a neural network to learn relationships between all the units created during different periods of the game. A neural network would not only allow us to find linear relationships but also more complex higher order relationships between units. However, creating a neural network wouldn't be feasible for this project because of the large data sets and extensive training they require. Another problem with using a neural network is its "black box" nature where it's unclear how and why a prediction was determined given certain input, making the logic hard to follow and evaluate.

Another solution is creating a Bayesian network where each node, which in this case would be unit type, would have a probability associated with it depending on how far into the game the player was to determine what units were most likely to be created next. This method would work better than the neural network as they are less sensitive to smaller data sets and are more resistant to overfitting the data, therefore making them more efficient in rapidly changing environments such as StarCraft II. However, a Bayesian network would also be a difficult approach for our project because determining the conditional probabilities between nodes requires a large amount of training and computational power, neither of which we can accurately do within our timeframe.

III. SOLUTION

We will mainly use a statistical approach to analyze the relationships between two units. We will extract data from the replay files regarding the amount of units made and what time they were made at and analyze every pair of different units to evaluate whether the units even have a relationship and then determine the regression model that best fits the pair. We will also evaluate how each pair of unit's relationship changes as the game progresses. Then we will use the multiple regression models determined for each pair to predict what units will be created in the future, given what units had already been created in the game. Moreover, since the method only bases on the amount of units that are created throughout the match, We may also use machine learning methods process the data to dig hidden relationships. Although the goals of the projects aren't completely similar, we can use the model mentioned earlier that predicts an opponent's opening build order using a Bayesian Model to provide a baseline for how the accuracy of our prediction model compares to existing models.

Neural and Bayesian networks would be able to solve our problem, however, because many units seem to have linear relationships with each other, the simpler solution of linear regression can be utilized. Linear regression is the most easily understandable and approachable among all we solutions we have thought of so far. Linear regression is also the most doable solution for this problem within our restricted time frame. Most of the other workable machine learning approaches that we thought of are supervised and obviously, accurately labeling all our data within this timeframe seemed unfeasible.

This algorithm will contribute to solving the problem of dynamic army prediction as no similar prediction algorithms exist thus far. However, many improvements can still be made to this algorithm to make it more accurate. The algorithm can be expanded to not just learn from data involving units being created, but also learn from data involving the layout of the map being played on and data on which resources were being gathered. It can also be further trained with replays with different factions, because the algorithm's observation of features is based on the general concept of StarCraft II rather than specifics, such as, knowing the tech tree of one faction first. This algorithm can ultimately be utilized by other gameplay bots to make predictions in real time in order to assist the player. Our algorithm would provide a starting point for others to add to in order to eventually develop a highly accurate army prediction model.

CHANGES FROM ORIGINAL SOLUTION

Our previous solution to the Dynamic Army Predictor of using Linear Regression models proved to be insufficient due to the complexity of the relationships between the different units. We were not able to accurately fit any pre-existing statistical model to the relationships between units so we developed our own model to encompass these complex relationships. Our new solution combines two ideas that we call "First Encounter" and "Build Time Density." Both of these will be further detailed in the section below.

For the Win Loss Predictor, our original approach sought to explore Bayesian networks, but given the short time frame we decided instead to explore logistic Regression. logistic Regression, we decided, was quite appropriate for the situation at hand because when units are

first created, the growth of the number of units accelerates quickly and as the actual battle begins, the number start to incline. With a logistic fitting, we can then go on to predict the win loss probabilities for future units of the same type using their health values and quantities. However, this prediction model requires both player to be the same race as it compares the amount and total health a specific unit that both sides possess. The coefficients derived from the logistic regression can also be applied to the Lanchester model referenced in the introduction in order to dynamically find the win loss probabilities rather than statically as we currently do. However, due to time constraints, we were not able to finish its implementation of Dynamic Win Loss Prediction. At its current state, Win Loss Prediction can only determine the chance of a player's victory if they are playing as the same race as the opponent.

IV. DESIGN AND TECHNICAL APPROACH

We propose using Linear Regression or some other statistical model to predict the amounts of complementing units over time. For instance, Mutalisks and Zerglings are commonly used together, so we will form a regression model which predicts their exact correlation. In addition, we plan to use another model to dynamically predict the enemy's army composition from the available data (obtained by scouting).

In general, Linear Regression is effective for highly correlated data members. Thus, we believe using it for selected complementary units would result in accurate predictions of future unit builds. However, it may fall short in dynamic army prediction, since game unit compositions vary more widely during the later parts of the game. For instance, predicting based on buildings would be ineffective because both players are likely to have all of the buildings; late game scenarios diverge from generic build orders, creating a much larger problem space. As later in the game a single model's accuracy would decrease significantly, we propose using several separately calibrated models for the different partitions of game time. For example, minute 0-10, 10-20, 20-30 would have their own unique linear regression models for each unit pair. The partition boundaries may vary as we experiment with different partition sizes.

CHANGES FROM ORIGINAL DESIGN

We attempted to develop a prediction model using Linear Regression alone, however as shown in Figure 1 which showcases the graphs of relationships between Roaches and all the other units, fitting a Linear Regression model was not be an option. We explored other statistical models such as a Logistic Model, however none of them could accurately fit relationships as complex as these due to the sheer number of factors that influence build orders.

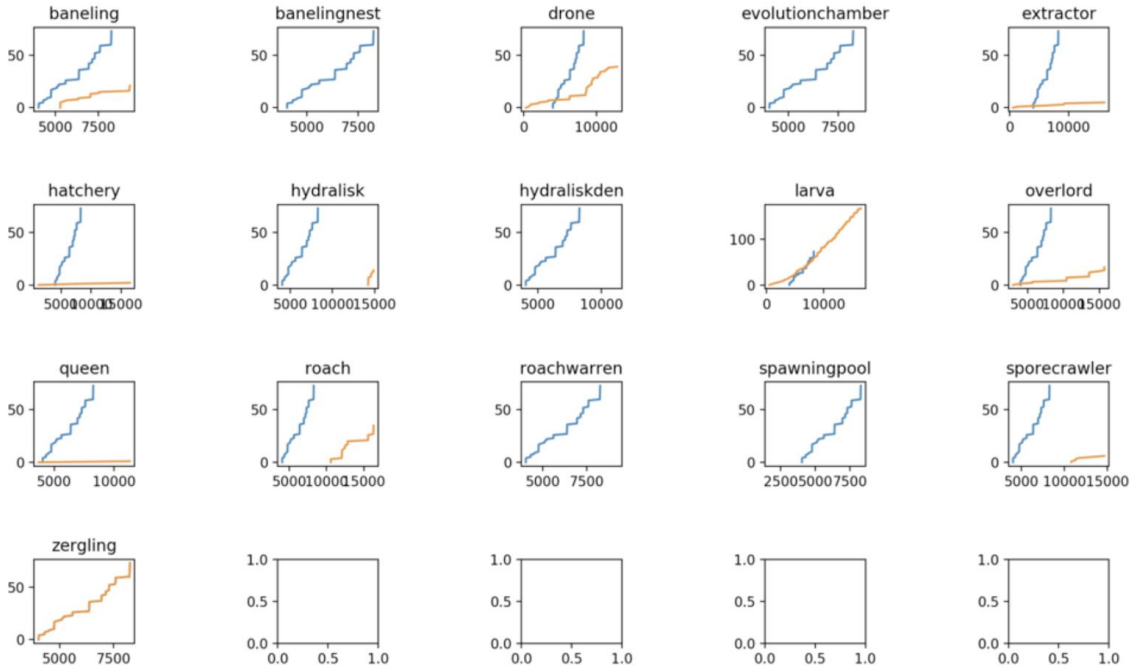


Figure 1. Graphs Showing the Relationships between Roaches (blue) and the other Units (orange)

Because Linear Regression was too simplistic of a model to fit onto the complex relationships between the units, we developed our own model using two ideas we call “First Encounter” and “Build Time Density.”

“First Encounter” is designed to calculate the “relative strength” relationship between any units, buildings and technologies in an unsupervised manner. We define this term as the difference between the time of creations of units. We calculate these values for each unit and then create an array of the “first encounter” values in the following format:

```
data['race']['observedUnit']['comparedUnit'] = [array of first
                                encounters]
```

After creating the first encounter array for all units, we take the variance of each array and define this term as its “relative strength.” The main idea behind this is that among all StarCraft games, there is a high probability of a set of units being created following the creation of other specific units. The relative strength relationships mentioned earlier are represented by the variance that is calculated by the combination of all data. The intuition is that if we have low first encounter variance between two units, then we can say that these two units also have a strong relative strength relationship. In short, the lower the variance, the greater the chance of that unit being created. For example, take the relative strength relationship between Spires and other units:

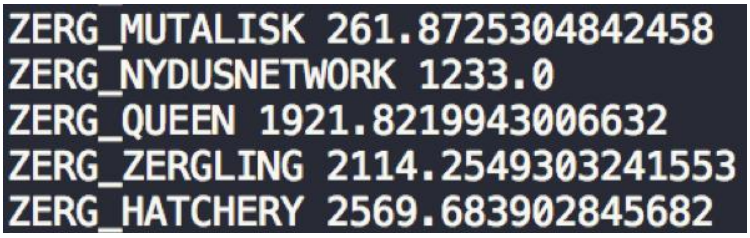


Figure 2. Relative Strength Relationship

This output contains the "Relative Strength Relationship" between Spires and other units. The numerical values are calculated from the first encounter array mentioned above. Each numerical value corresponds to a score representing the possibility of the related unit being created. The lower the score, the higher the probability of the enemy building that specific unit. From this output, we can see that the score for Mutalisk is very low in comparison to the others. This means that these units have the greatest possibility of being built following the creation of Spires. Because we only extracted data from 70 games, our prediction model is inaccurate in predicting build orders late in the game, however, the model performs somewhat accurately for early game predictions.

The relative strength and first encounter calculations alone do not take into account the influences the game clock and the number of created units has on build orders. In order to solve this problem, we weight the relative strength values by the normalized "Build Time Density." The intuition behind this is that specific units are more likely to be created during specific times of the game. For example, take the graphs of the Build Time and Build Density for Hydralisks:

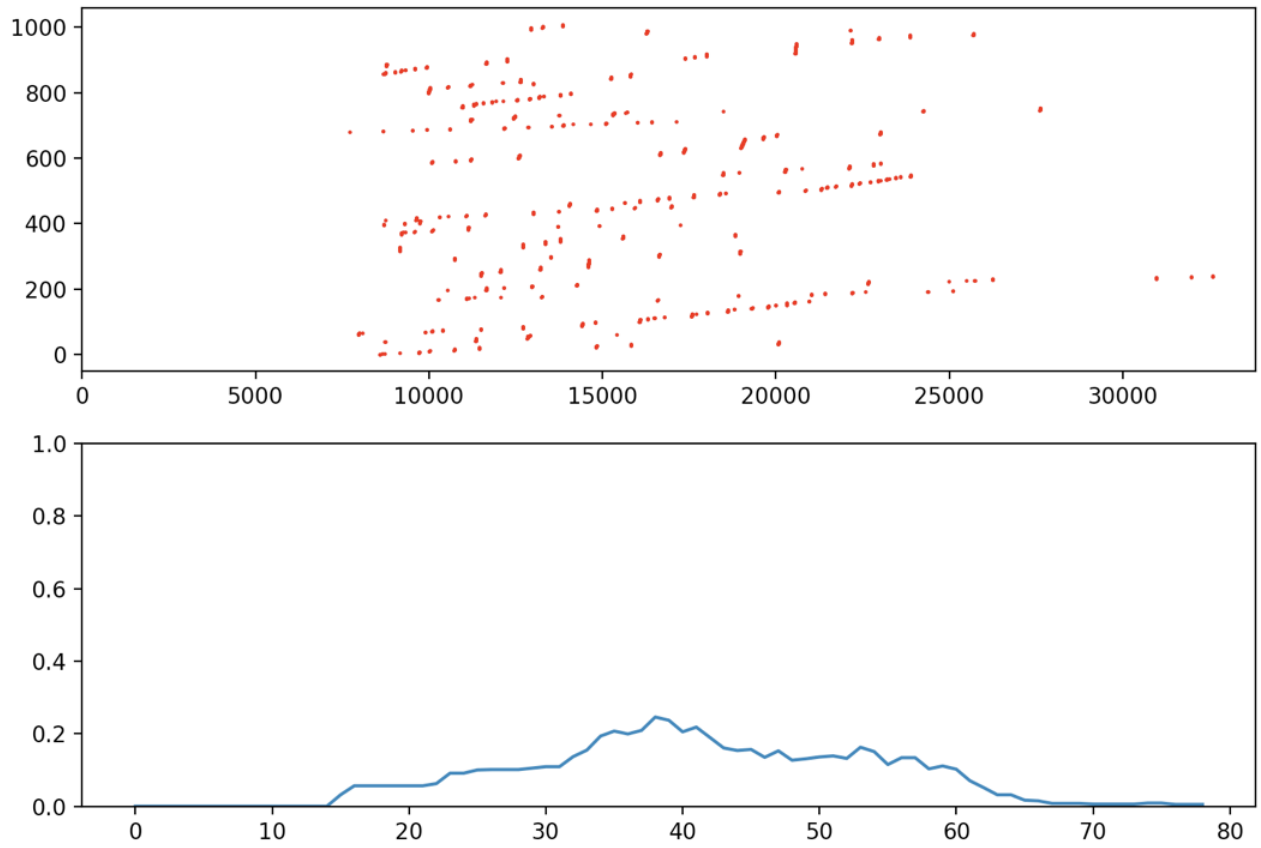


Figure 3. Build Time and Build Time Density Graphs for Hydralisks from 70 Games

From these graphs, we can clearly see that Hydralisks are more likely to be built in the middle of the game rather than right at the start or closer to the end. By weighing the relative strength with the normalized unit density, we are able to increase and decrease the scores shown in Figure 2 based on the probability of these units being created during specific times of the game. At its current state, the Dynamic Army Predictor does not take into account the number of units seen by the player- it creates its predictions based on the unit type and game clock alone. The final prediction values for the Dynamic Army Prediction are given below in the Section VII.

As stated above, the logistic Regression model is fitted to our data in predictor.py. Instead of using just the number of units, we are also examining the total health of the units as well. The health is a good indication of the strength of a certain unit type because the more health a unit has, the longer it is able to withstand attacks and survive. Our version of the logistic Regression utilizes the number of units and total health. Then it scales the effect across multiple attribution values in order to come up with an accurate fitting. We then take the given fit and use the inbuilt python PANDAS predict mechanism to we predict the win loss probability for the player.

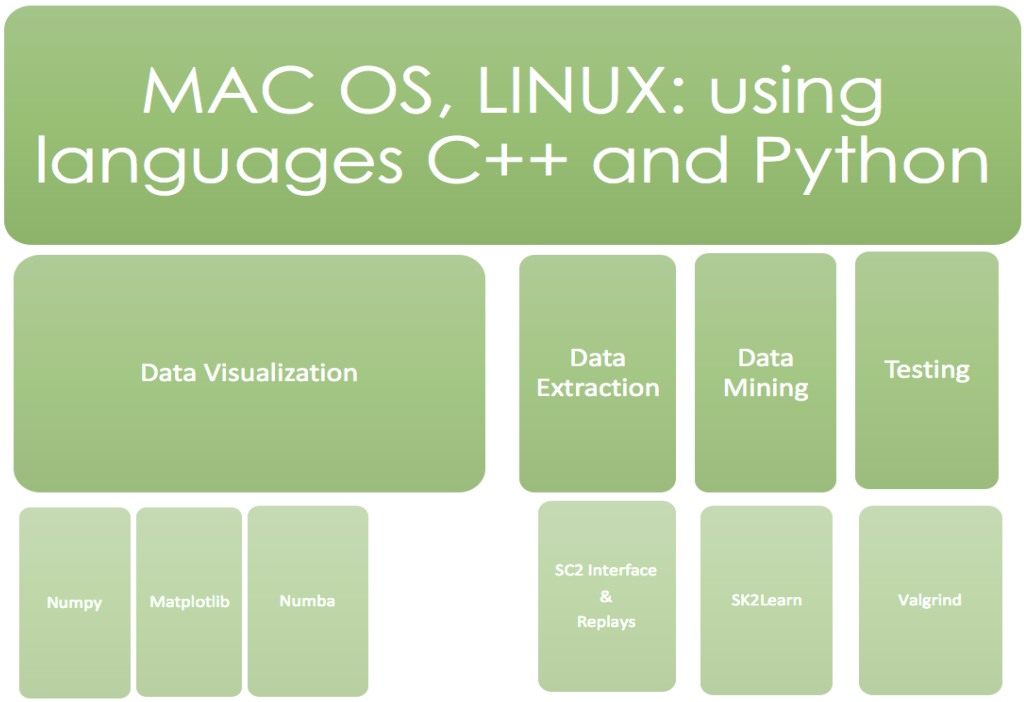
```
See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated
  # 220.000000 1           0.282462
    Optimization terminated successfully.
            Current function value: 0.532474
            Iterations 6
                           Logit Regression Results
    ==============================================================================
    Dep. Variable:                    win   No. Observations:                   63
    Model:                          Logit   Df Residuals:                       57
    Method:                           MLE   Df Model:                           5
    Date:                Wed, 06 Jun 2018   Pseudo R-squ.:                 0.2073
    Time:                        19:29:57   Log-Likelihood:                -33.546
    converged:                       True   LL-Null:                       -42.317
                                            LLR p-value:                 0.003577
    ==============================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
    ------------------------------------------------------------------------------
    UnitA_Health      0.0048      0.003      1.734      0.083      -0.001       0.010
    UnitB_Health      0.9373      0.960      0.976      0.329      -0.945       2.819
    Attribution_2    -0.2292      0.895     -0.256      0.798      -1.984       1.525
    Attribution_3    -1.2420      0.864     -1.437      0.151      -2.936       0.452
    Attribution_4    -2.4005      1.079     -2.225      0.026      -4.515      -0.286
    intercept        -5.5363      3.040     -1.821      0.069     -11.494       0.422
    ==============================================================================
                           0         1        OR
    UnitA_Health    0.999378  1.010204  1.004777
    UnitB_Health    0.388867 16.761180  2.553011
    Attribution_2   0.137536  4.596948  0.795139
    Attribution_3   0.053056  1.571997  0.288796
    Attribution_4   0.010943  0.751329  0.090675
    intercept       0.000010  1.524823  0.003941
    UnitA_Health  UnitB_Health  Attribution  intercept  Attribution_2  \
    0         7235.0      7927.0          1.0        1.0              0
    1         7552.0      7930.0          2.0        1.0              1
    2         7643.0      7743.0          3.0        1.0              1
    3         7845.0      7629.0          4.0        1.0              0
    4         8237.0      7614.0          1.0        1.0              1

    Attribution_3  Attribution_4  win_pred
    0              0              0  0.174712
    1              0              0  0.144077
    2              1              0  0.057615
    3              0              1  0.018834
    4              0              0  0.199719
```

Figure 4. Numerical Output on Canopy

From the above generated data we can see that the strength of unit (coefficient of unit health) correlates with a give higher health value. We can also see that units were able to win more games when they had a higher total health, therefore resulting in the win probability for that player to be higher. Here, the bottom most chart represents the different type of units in the game and depicts the probability of winning.

V. TECHNOLOGY STACK



Our team will be work on both Linux and the MAC OS. Since we have chosen the SC2 API and numpy/mathplotlib as our mathematical computing package the main languages we will use are C++ and Python.

We will first extract data using the s2client-api from the provided replay files in a JSON format containing information on the type of unit created, the amount of unit created, and what times in seconds they were created at for both the player and the enemy. The extracted data will be in this form: “Zerg” : [1, 10, 43, 89].

Once the JSON file has been created, we will use Matplotlib and numpy (a scientific computing package based in Python) to identify the specific relationships between every pair of units. We will also form different linear regression models based on which partition of the game the player is in using a similar approach. Finally, we will design the final mathematical model based on these previous models to be able to dynamically predict the enemy’s army formation.

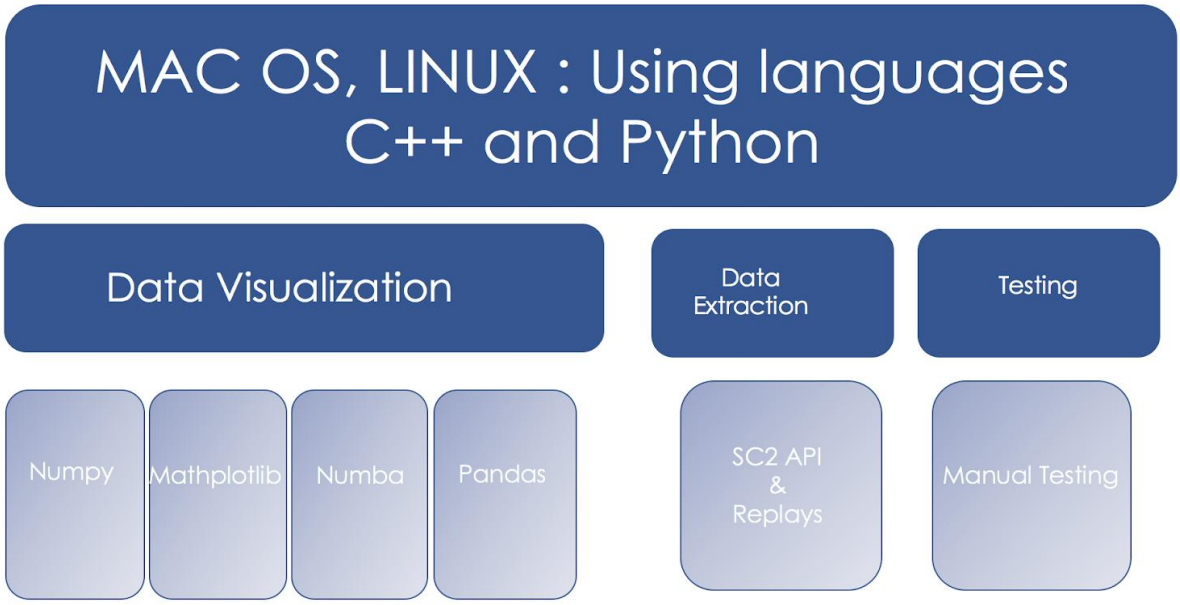
Since our solution requires training from large sets of data in order to find the best possible correlation, we will use Numba (a numpy aware compiler) to reduce execution time by implementing numpy arrays. If time permits, we will use SK-Learn for a more machine learning approach to this problem. We will use this to data mine and create linear regression models as well as more complex mathematical models.

Finally, for testing we’ll be using Valgrind for catching memory issues. Since we are working with a large data set and aim for a dynamically productive solution, Valgrind allows us

to automatically detect any issues connected to memory management and treading bugs and profile our programs in detail to help us s where exactly we went wrong. We will also be implementing c++ unit testing as we develop to ensure each part is working as it should separately and can be integrated into the larger solution smoothly.

Although not officially part of our technology stack, Github plays a major role in the development of our solution. Many of the APIS we are using are sourced from Github (i.e SC2 and Numpy).We will primarily be using sc2-api to carry out our goals. The sc2-api provides access to in-game state observation and unit control. Other Starcraft II AI resources are on github as well, so if we plan to use something other than the CommandCenter, github will be essential for carrying out our project. Also, we plan to use github to document our own code. At the end of our project, as our timeline details, all our code will be committed to detailed along with a detailed explanation of the solution as well as an implementation guide. Last but not least, Github will be a major part of ow our team divides the tasks and communicates about each other's code contributions.

CHANGES FROM ORIGINAL STACK



There were no radical changes to our technology stack. If we didn't use something in the original stack, it was already mentioned that we might not use it in our final project. As we already discussed in our first draft of the project proposal, there were two methods of data extraction that we were considering apart from the s2client-api: pysc2, and CommandCenter.

Pysc2 is a python-based Starcraft II Learning Environment developed by Deepmind. CommandCenter is a C++-based AI bot for Starcraft Broodwar and Starcraft II. The team tested these two methods of data extraction and determined that they were too inefficient for our use. While pysc2 offered data extraction such as unit count and build time, the Starcraft II world was represented as pixels (Figure 1). For instance, a SCV would be represented as a circular blob of

14 pixels. So if we were to get the total amount of SCVs on the screen, we would have to add up the total amount of "SCV pixels" then we would divide it by 14 to get the total amount of SCVs. This is highly inconvenient as SCVs frequently overlap each other, which reduced the total SCV pixels on the screen. This method of representation was inferior to sc2-api since the Blizzard api would directly count the amount of SCVs instead.

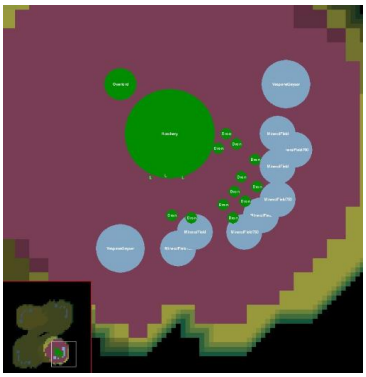


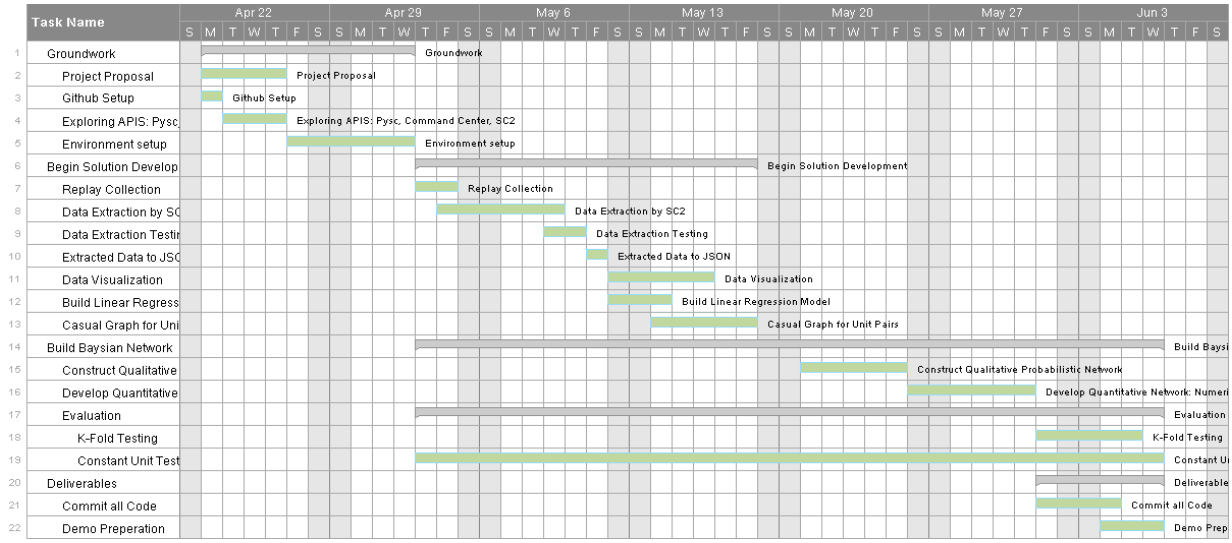
Figure 5. Blobs of Pixels. Drones (green) and Minerals (blue)

CommandCenter is a bot based on the construction of UAlbertaBot, a StarCraft BroodWar bot written in C++ as well. CommandCenter turned out to be too specific for our needs. The creator of CommandCenter describes it as "based on the architecture of UAlbertaBot, and is intended to be an easy to use architecture for you to quickly modify, play with, and build your own bot." As such, it was more suited for specific build orders and actually playing the Starcraft II game than for data extraction.

For data mining, we ultimately decided against using SK-Learn and we decided to create an additional win prediction model instead. So, the final change in the technology stack was our use of pandas, a python-based library for data manipulation and analysis, for win prediction data visualization.

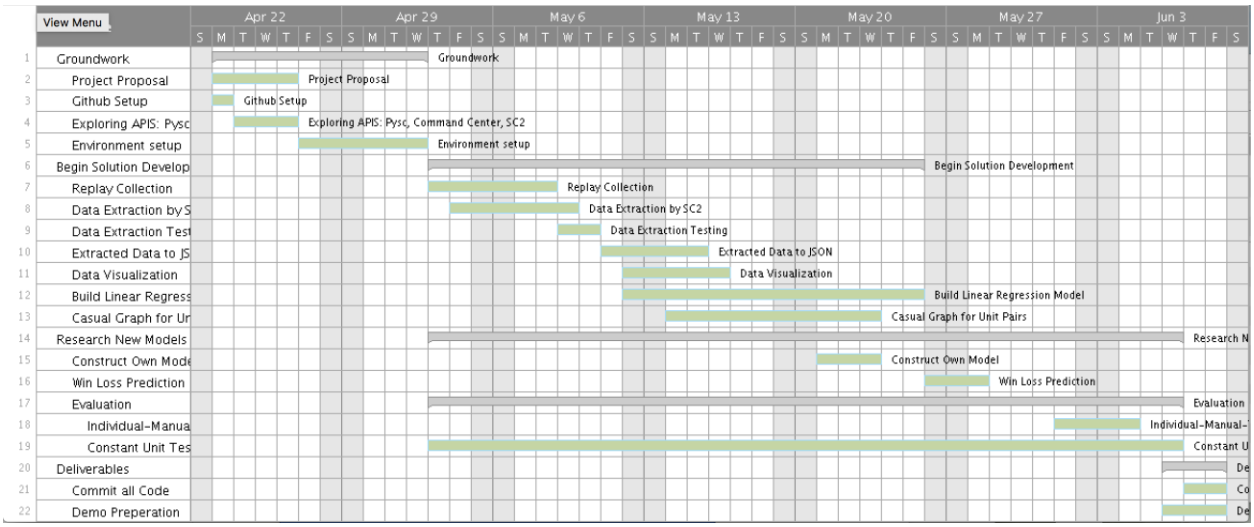
VI. SCOPE AND TIMELINE

Dynamic army prediction ideally should delve deeper then a linear correlation between unit pairs and successful strategies. However, given the short amount of time, we hope to take the first couple steps down the path of a more complicated army prediction solution. A full solution to this problem would be founded on a probabilistic statistical model which expresses the conditional dependency of all variables on each other. Our first goal was to implement such a model, but we soon found that while planning such a mapping of a multitude of variables would prove to time costly and infeasible.



CHANGES FROM ORIGINAL SCOPE

Our scope changed drastically. We had to deviate from our original plan of fitting a linear regression model since it was too simple for the complex data. Instead as you will see down below, we choose to research more easily applicable models. We choose to observe the interlinks between the properties of the units in the form of First Encounter and Build Time Density. In addition, we also tried our hand at win loss prediction based of logistic regression. Although a simple approach, it proves able to predict although not quite accurately. We took much longer than expected for data collection. Our initial estimate was around four days, but in reality, it was seven days due to large amounts of data we were extracting. The JSON conversion also threw a wrench in our initial timeline since we weren't able to get the conversion to that format working in time. Linear regression also ended up taking up more time than expected since we weren't quite able to figure out how to execute the model and pushed our time bounds. Since the deadline was approaching rather than exploring Probabilistic Networks we chose to implement our own model and logistic regression for a relationship based and health based predictions.



VII. DOCUMENTATION AND ACCESS

We will use a GitHub repository for our code and documentation. The README file will include information on how to set up the environment and run the code in order to reproduce our army predictions. It will also include a thorough description of our project and the mathematical model we utilized. In addition, we will also create a GitHub Page with a link to our GitHub repo as well as a complete written and graphical analysis on the success of our dynamic army prediction algorithm.

GitHub Repo: https://github.com/jjiangweilan/Dynamic-Army-Prediction
GitHub Web Page: https://jjiangweilan.github.io/Dynamic-Army-Prediction/

VIII. EVALUATION

We will use k-fold cross-validation testing in other to evaluate the success of our dynamic army prediction algorithm. We will randomly split the data we extracted from the replay files into 10 equal subsets. 9/10th's of this data will be used for training the army predictor while the leftover data is held as the validation set for the model. The predictions the model makes will be compared to the data and patterns within the validation set in order to determine the model's accuracy. This process will be repeated 10 times so that each piece of data is used in the validation set exactly once and then finally we will average the accuracy determined from each test to obtain a final percentage of the accuracy of our model.

CHANGES FROM ORIGINAL EVALUATION

Because we had to develop our own statistical model after fitting a Linear Regression model failed, we were short on time and not able to use k-fold cross-validation for testing. Instead, we manually tested the Dynamic Army Predictor by extracting data from a replay file at a given time and running the prediction model based on the information gathered by scouting.

The Dynamic Army Predictor then outputs the probability of certain units appearing based on the scouted information. For example, if scouts saw that the enemy had Zerglings, Roaches, and a Hydralisk Den at minute 14 of the game, the Dynamic Army Predictor would output the following:

```
'Zergling','Hydraliskden','Roach' at 14:
ZERG_HYDRALISK 866.6295873238444
ZERG_MUTALISK 1184.0044180563987
ZERG_ULTRALISK 1631.941048553335
ZERG_ZERGLING 1667.0600759384356
ZERG_ROACH 2246.7335243029793
ZERG_HATCHERY 2256.7266629475075
ZERG_QUEEN 2388.042075678204
ZERG_EXTRACTOR 2493.9337879525756
ZERG_BANELING 2745.264887498538
ZERG_CORRUPTOR 2875.204016145028
ZERG_SPORECRAWLER 3512.5562124820585
ZERG_INFESTOR 3634.2377559254724
ZERG_SWARMHOSTMP 4851.190094561914
ZERG_SPINECRAWLER 4906.350895150753
ZERG_HYDRALISKDEN 5938.999009239683
ZERG_NYDUSCANAL 7064.234813927637
ZERG_ROACHWARREN 7499.128491785
ZERG_SPIRE 9954.722072007273
ZERG_BANELINGNEST 10049.895016369863
ZERG_EVOLUTIONCHAMBER 10477.45197594935
ZERG_INFESTATIONPIT 12011.998827522799
ZERG_ULTRALISKCAVERN 12975.540372735573
ZERG_SPAWNINGPOOL 19491.723274555447
```

This lists all the units the enemy's chosen race could have and the probability of them creating that specific unit. More prediction results are shown in the GitHub linked below. As mentioned in Section IV, the lower the score, the more likely the enemy will create that unit. We looked at the units with the lowest scores and then analyzed the replay the data was initially extracted from to see if these units were created. We repeated this process at varied times with different replay files and reached the conclusion that the Dynamic Model Predictor was mostly accurate early on during the game and as the game progresses, becomes more and more inaccurate. With extra time, we would increase the accuracy of our model by introducing a new variable into it- the amount of each unit seen by the player's scouts. We would also more thoroughly test our prediction model in order to get an estimate of how accurate it is.

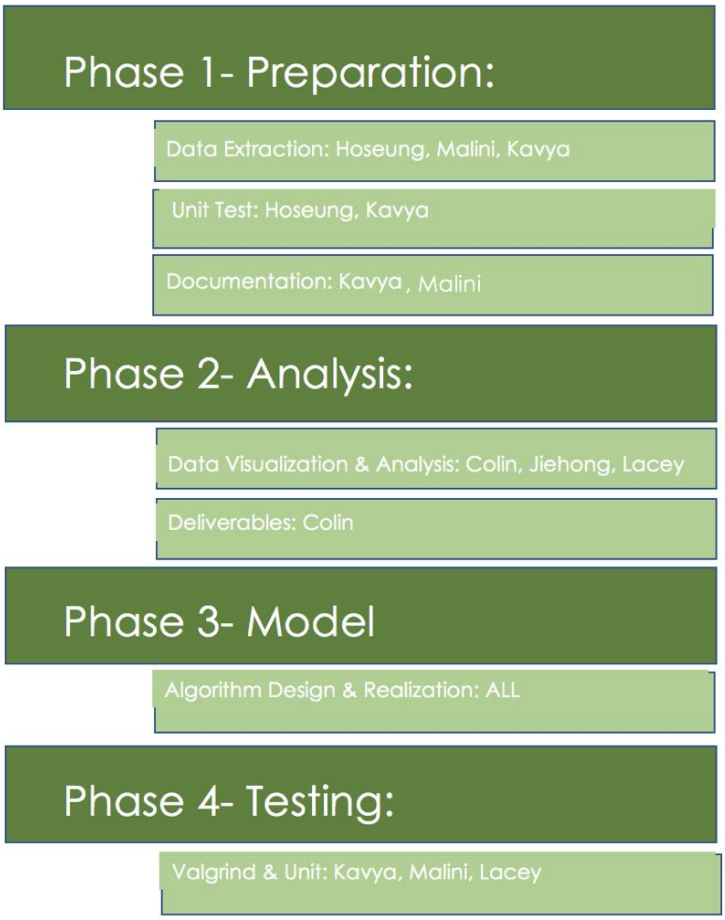
Due to the time restrictions and our last-minute addition of the win loss prediction feature we were unable to perform Valgrind Testing as an assessment of accuracy. However, based of the outcomes of the validation set and predictions from our predictor, we believe predictor still needs more work. Its slightly accurate when used within the first three or minutes in the game after which it began to scale largely too fast. We manually cross checked the likelihood of our predicted probabilities coming true manually. Based on just this basic level of evaluation we were able to tell that our predictor wasn't very accurate since it would predict low rates of

winning for both players--which is not possible. We believe that the relationship we assumed to be present between a unit's health and the outcome of the games was to simple and nearsighted assumption to make. Much like our problem with linear regression being too simple to encompass the complexity of the data, we don't think the variables we choose as to represent the probability of winning were sufficient enough to produce good predictions. In essence, with our Win Loss Predictor, we had the reverse issue from the linear regression model. We boiled our data down to the minimum bare to make a clean-cut prediction but instead it became an improbable and unrealistic prediction.

IX. DELIVERABLES

After completing our project, we will make our GitHub repository public for others to reference and create a GitHub Page detailing the entire project and its results. We won't be submitting our code to tournaments, however, we will write a conference paper on our results and conclusions. The links the public GitHub repository and GitHub Page are linked above.

X. SEPARATION OF TASKS FOR TEAM



CHANGES IN SEPARATION OF TASKS

XI. INDIVIDUAL TASK SUMMARIES

Lacey: Documentation

Colin: Wrote the first version of the data process tool

Jiehong: Proposed and implemented dynamic prediction algorithm. Creating visualization tool for data analysis. Helped debugging data extraction. Maintaining github commit.

Malini: Data Extraction for Win Loss, helped Design and Implement Win Loss Prediction Model. Worked on documentation and pulling together the initial proposal, project report, and readme for github.

Kavya: Data Extraction of health/ enemy data for Win Loss. Worked with Malini to design and implement logistic regression for WinLoss. Also worked on compiling documentation for the project: proposal, project report, and github readme/website.

Hoseung: Data Extraction Implementation, Build Order Creation Implementation, debugging Data Visualization, minor role in debugging Win Rate Implementation