Starlytics: Strategy Analytics Web App for Starcraft II

Adam Wild
Georgia Institute of Technology
Atlanta, Georgia
awild6@gatech.edu

Patrick Hackett
Georgia Institute of Technology
Atlanta, Georgia
patrick.d.hackett@gatech.edu

Irfan Al-Hussaini
Georgia Institute of Technology
Atlanta, Georgia
alhussaini.irfan@gatech.edu

Anvesh Reddy Chennuru
Georgia Institute of Technology
Atlanta, Georgia
anvesh@gatech.edu

KEYWORDS

Real-Time Strategy, Artificial Intelligence, Neural Network, Starcraft 2

1 INTRODUCTION

1.1 AI and Professional Gaming

Building Artificial Intelligence (AI) for games that can successfully compete with professional players has been a compelling measuring stick for the progress of AI since IBM's Deep Blue beat Garry Kasparov in 1996. Google's DeepMind and OpenAI have led the field, but are unable to compete in games with incomplete information and an exponentially larger number of changing variables. [8]

For Starcraft II, DeepMind applied their Alphazero algorithm and it failed to perform the most basic tasks. In addition, AIs in the original Starcraft (SC1) have been unable to beat any semi-pro player.

1.2 Starlytics

Stacraft II (SC2) is a highly competitive popular real-time strategy game played across the world. Players choose one out of three races to play as: Protoss (P), Terran (T), and Zerg (Z), mining minerals to use as resources to build buildings and armies in real time, with the goal of destroying an opponent's armies and buildings. [Figure 1]

Using a database of 36,000 SC2 game replays from human players parsed into game state data by an API developed by Blizzard and Google, we built a tool that analyzes game state data to provide strategic analysis to players after their games. This is unique in that all current strategic guidance to players comes from watching pro players, practicing, or being involved in online communities. By leveraging machine learning to automate prediction models and analysis of gameplay, we also have a pathway to providing more effective models to guide AI to compete with professional players in much more complex environments.

2 LITERATURE SURVEY

2.1 SC1 Discrete Strategies

Usunier uses a deep neural network on features to propose micromanagement tasks: a limited scenario of low-level unit control. [11] Ontanon shows top-performing AIs use hard-coded strategies with subdivision of tasks. [5] Another study showed usage of temporal difference algorithms in a 1,000 game dataset to manage combat units. [14]

Figure 1: Gameplay photo showing a standard Protoss player's base. Includes a primary building called a Nexus, units mining minerals that are used to purchase new buildings and units, as well as a small cluster of army units and defensive structures. The minimap is shown in bottom left.

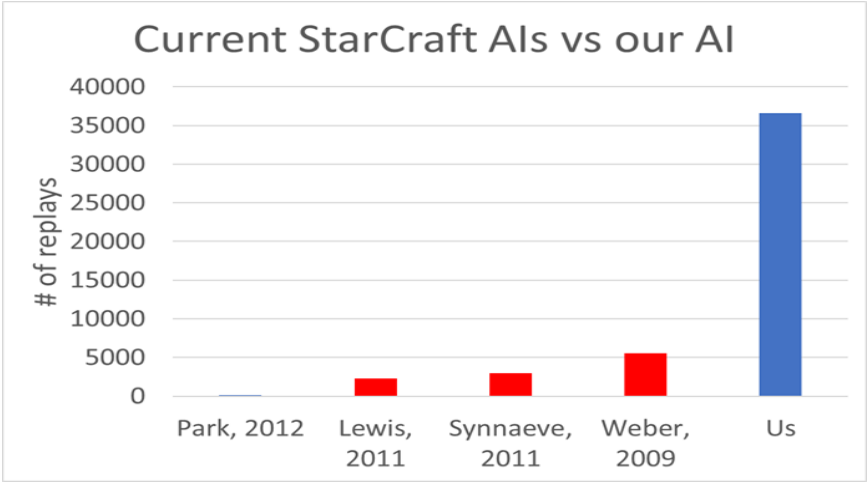

Figure 2: Number of replays used in our analysis compared to previous research

These perspectives show the success in breaking game states down to discrete interactions to draw conclusions, but are limited to very restricted situations. We will proceed with the intention of integrating these tactical analyses with grand strategy.

2.2 SC1 Grand Strategies

Micic analyzes current in-game AI strategies that use additional information beyond that available to the human player. Despite this advantage, they are unable to adapt to changes. They use reinforcement learning (RL) to micromanage tasks, which becomes harder as complexity increases. [4] Peng

implements RL concepts with a deep multiagent reinforcement learning which can learn multiple strategies for coordination between the units. [7] Lewis presents a large-scale analysis showing players with more actions per minute are more likely to win. [3] Synnaeve used a Bayesian model trained on a large number of replays (3,000) to predict the build order used by opponent. This approach is limited because it only analyzes buildings, and requires labeled data. [9] Weber also uses a Bayesian network which gives a probability distribution for a given observation with 5,500 replays. Incomplete information is simulated through noise or missing features. [13] Park claims that adding random noise is not a good simulation of the fog of war. Their AI uses an in-game scouting unit to build a count of buildings but did not perform well due to limited dataset (100 replays). [6]. Vinyals introduced the SC2 Learning Environment, a RL environment acting every 8 game frames to mimic human actors. [12]

While these models are limited in their ability to perform on a discrete level and to succeed across all game states, they show the benefits of using Bayesian stochastic models incorporating state-dependent predictions to guide overall strategic adjustments and predictions.

2.3 General Machine Learning (ML) Theory

Silver et al. reached an exciting benchmark in AlphaGo defeating a professional player. Their AI evaluates the plays through a network to avoid unnecessary tree search. [8] Jordan details how progress in ML has been driven by new theory, the availability of data, and low-cost computation. [2] One high-impact area of progress is deep networks, which can be implemented via TensorFlow, a deep-learning framework that can utilize a large vector space. [1] Tipping provides background for Bayesian networks that can evaluate probable strategies given a partial observation of the current game state. [10]

2.4 Macro-Management Dataset for SC2

Wu, et al. use the SC2 API to parse a large dataset of replays focusing primarily on macro-management game state data. The data is preprocessed in order to assure replays have active players. High level actions such as in-game resource collection, unit count, idle workers are normalized. They performed baseline predictions of win and loss for different in game match-ups using Recurrent Neural Networks (RNNs) reaching a mean accuracy of approximately 59 percent. This dataset located at https://github.com/wuhuikai/MSC was used in our analyses in order to integrate higher accuracy predictions with a post-game strategy analysis web app.[15]

3 METHODS

3.1 State of the Art Value

Few AIs have been built using the release of the SC2 API in August 2017, and none specifically to help players improve their game play. Our large dataset (36,619) is much larger to prior training attempts (5,500); this, combined with a narrow focus on predicting the win probability based on current game state, will help refine our strategic analysis to a competitive level. [Figure 2] The success of our model and platform can be used to help AI to make strategic choices at a professional level.
Furthering the success in the area of game AI would have large implications in ability of AI to successfully measure and make real time decisions based on large number of variables, leading to possible improvements to fields ranging from self-driving cars to real-time battlefield decisions.

3.2 Data

The SC2 API enables the analysis of labeled game state data focusing on visible units. In the attached figure, the running API visualizes game state information. [Figure 3]
Our data has been pulled from 36,619 game replays,

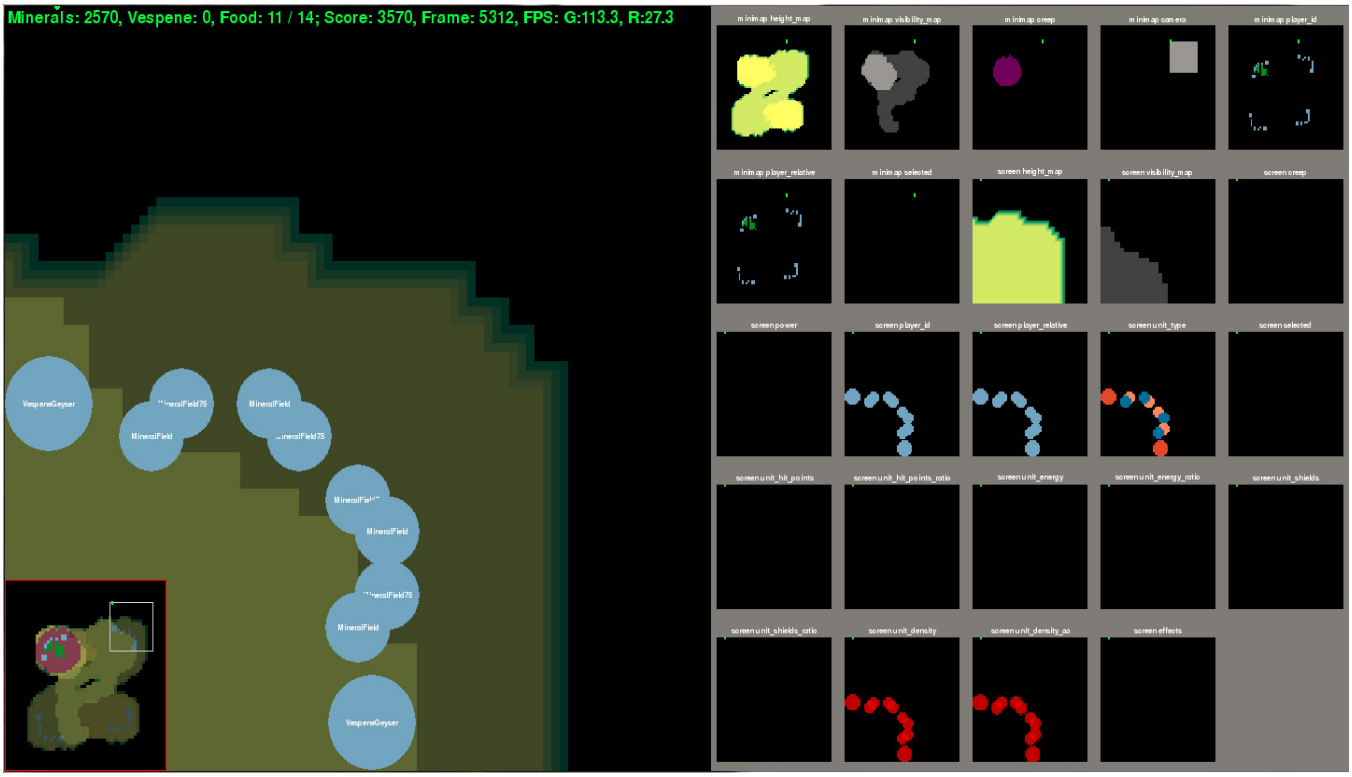

Figure 3: API Agent processing replay game state data

restricted to games with an MMR (Match-making ratio) above 1000, APM (player Actions Per Minute) above 10, and frames above 10000. This ensures game data is from valid matches with both players of decent skill level and participating. A normalized and parsed dataset has been provided by Wu, Huikai, and Zang. Game data is collected every 2.57 seconds and includes resource collection rate, army size, building count, unit density, and map features at each point in the game. The game state player data is contained in Global Feature Vector files, whereas the map features are contained in Spatial Feature Tensor files.

The total size of dataset is approximately 100 GB in the compressed tensorized dense matrix. Once converted into a sparse matrix for training or evaluation the size expands to many times the compressed size. The largest part of the dataset, the Spatial Tensors, were only used for visualization purposes in the shape of the arena. The Global Feature Vector consisting of game state information rather than pixel-wise information was used for building the models to predict win or loss at different time points.

3.3 Model

Initially, we performed multiple model analyses on the full dataset at certain time points in the game to visualize model performance and investigate match-up differences in model performance. We split the full dataset into training and validation (70%-30%), extracting a vector of 10 features representative of game state at certain points in the game. By compiling these vectors across all replays of a given match-up we were able to construct a database for classifiers to be trained on. The main models trained and tested across all 36,619 replays were Linear Regression (LR), Artificial Neural Network (ANN), and Random Forest Classifier (RFC). The current smaller model running in the backend of the Starlytics Web App is trained on PvP. The

size of the resulting model for the two tasks were ~200MB and ~5.6GB respectively for result and action prediction.The model was trained on a subset of the data spanning all time frames of 1000 replays (1.6 GB) for approximately 200,000 datapoints. The model was chosen for local hosting computational speed.

3.4 Web App and Visualization

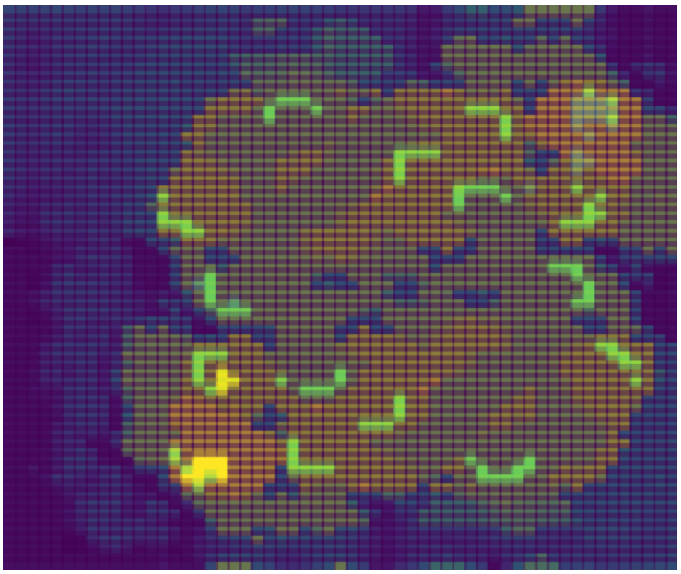

Figure 4: A sample visualization of the minimap compiled from replay spatial feature tensor data

Starlytics allows the user to upload a file, use a slider to move across time points, obtain a prediction based on that frame, and look at the condition of the minimap at that point. The minimap is a reconstruction of the in-game map representing units, visibility, and topography constructed using overlapping visual representations of spatial feature tensor data. [Figure 4] This enables players to view time points where their actions changed the likelihood of winning. We designed the app in such a way that uploading or selecting from the database could be done in a single page. An invalid user input is thresholded to the number of files. The page also displays the average accuracy of the current replay file using each frame and the actual winner of the game and allows the user to reset to the default which is the first file in the dataset.
The web app was developed using a Django framework implementing a python backend consisting of a RFC with 50 trees with bootstrapping and features and depth not restricted. [Figure 6]

4 EXPERIMENTS

4.1 Results

The ANN and LR classifiers have similar performances and outperform the Random Forest for every frame we tested. As shown in Figure 5 the accuracy grows linearly with regard to time. When the time frame is close to 0, any classifier trained will have accuracy comparable to a random classifier (i.e. close to 50%). Which makes sense since determining the winner when little to no move has been played is difficult. As time progresses, the accuracy grows, and the predictions become more accurate. In addition, the sensitivity and the specificity of all models were comparable with accuracy, indicating the model consistently predicted both wins and losses.

For some frames, some models will underperform. By keeping the best performing classifier for each frame, we are able to give a more precise prediction of who is expected to win in future predictions in specific match-ups. [Figure 7]

4.2 Starlytics Predictions

Our algorithm was evaluated using the dataset provided by Wu et al [15]. It consists of a set of parsed replays with JSON keys for a training and testing dataset. Our data was trained for 1000 replays in the training dataset and evaluated using the entire testing dataset.
The web app displays the prediction of the winner of the game based on the current frame. The model was trained for both action prediction and result prediction for PvP with an accuracy of ~ 58.5% on

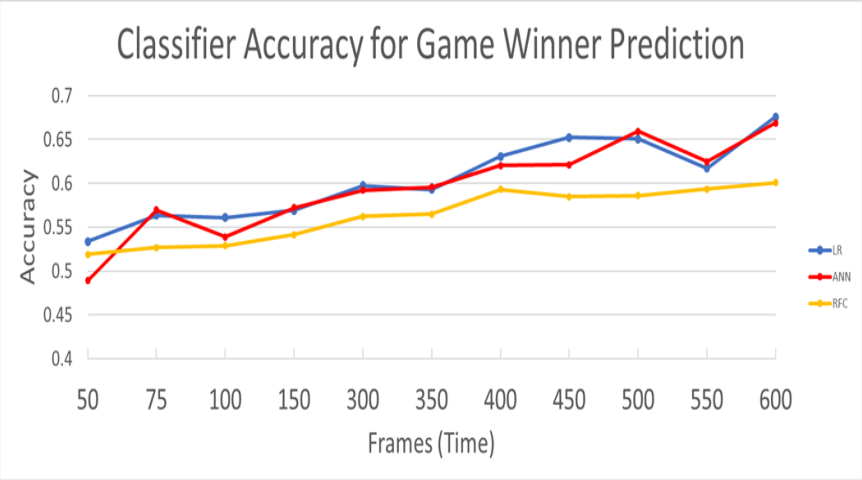

Figure 5: Accuracy of models across time.

result prediction and ~ 78% on action prediction using 1000 replays for evaluation which are higher than the baseline accuracy reached by Wu et al [15] was ~ 51.4% on action prediction and 76.3% on build order prediction.

4.3 Discussion

In Figure 7 the specific match-up comparison graph shows that PvZ is the easiest to predict and PvT is the hardest. This implies that in PvZ, a player builds his edge over time which ultimately results in a win whereas in PvT, the game remains undetermined until one big battle at the end of the game decides the winner.

In PvT, players traditionally are known to engage in small skirmishes to gain small edges while building up their main army. A big clash between these decides the winner at the end. PvZ is more of a battle of resources, with massive army engagements. If the Zerg player can control as many bases as he wants and expand freely, without being contained by the Protoss opponent, he is more likely to win. This situation of monopoly on the resources available on the map slowly builds up and is a clear indicator of who is winning. This information shows up in the minerals and vespene collection rate used as features for training our models.

Figure 6: The Starlytics Web App. Users can upload recently parsed replay data and view their win prediction rate at different points in the game, using the information to determine when their actions may have influenced the outcome of the game

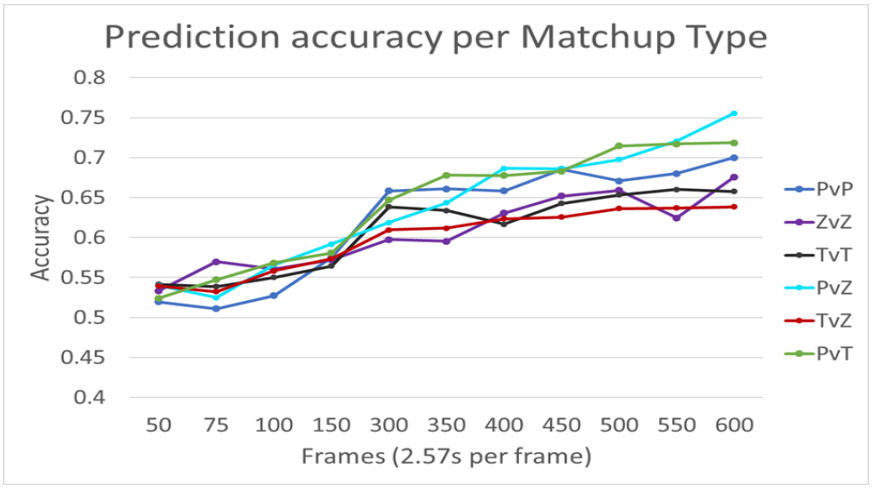

Figure 7: Accuracy of models in predicting outcome of different matchups at different time-points in a game.

5 FUTURE DIRECTIONS

Starlytics brings a combination of AI and post-hoc game analysis to provide gamers with strategic guidance on gameplay, using the Starcraft II API in a new and publicly consumable manner. In addition, the high volume of analyses can be used to help guide future AI development decision making, shifting the window for AI processing real-time decisions in complex feature spaces.

This web app provides a foundation for both strategic game analysis for players, and a foundation for improving AI performance in more complex scenarios. In order to surpass local hosting limitations in the proof of concept, future models could be hosted and trained on AWS EC2 instances in order to incorporate full 36,619 replays used in preliminary model comparisons and allow highest-performing model for each comparison. Models and the web app range of strategic insight could be expanded and improved as an open source repository, and, for this purpose, will be posted publicly to https://github.com/patrickdhackett. A possible future project could integrate the real time modeling and prediction into an AI bot to increase performance at the professional level.

5.1 Team effort and lessons learned

All team members have contributed equally in effort and time. The initial proposal was to develop real-time strategy guidance for AI using the SC2 API, but as we progressed in our analyses and exploration of the data and the problem we reached two key insights. First, as a data modeling and visualization project, the immediate impact of creating a strategy analytics web application allowed for the full pipeline from data to presentation to a customer in a novel way. Second, we could focus more on the problem of predicting and less on that of training a bot with the API integration, which made more sense under our original goal to specialize on game winning prediction, not to address the broad needs of a real-time AI bot. The path from conception to web application over the course of the project allowed us to narrow our focus on prediction while transforming our results into a new prescriptive analytics resource for a gamers with future implications for the progress of AI.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning.. In *OSDI*, Vol. 16. 265–283.

[2] Michael I Jordan and Tom M Mitchell. 2015. Machine learning: Trends, perspectives, and prospects. *Science* 349, 6245 (2015), 255–260.

[3] Joshua Lewis, Patrick Trinh, and David Kirsh. 2011. A corpus analysis of strategy video game play in starcraft: Brood war. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, Vol. 33.

[4] Aleksandar Micic, Davíð Arnarsson, and Vignir Jónsson. 2011. *Developing game AI for the real-time strategy game StarCraft.* Ph.D. Dissertation.

[5] Santiago Ontanón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. 2013. A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in games* 5, 4 (2013), 293–311.

[6] Hyunsoo Park, Ho-Chul Cho, KwangYeol Lee, and Kyung-Joong Kim. 2012. Prediction of early stage opponents strategy for StarCraft AI using scouting and machine learning. In *Proceedings of the Workshop at SIGGRAPH Asia.* ACM, 7–12.

[7] Peng Peng, Quan Yuan, Ying Wen, Yaodong Yang, Zhenkun Tang, Haitao Long, and Jun Wang. 2017. Multiagent Bidirectionally-Coordinated nets for learning to play StarCraft combat games. *arXiv preprint arXiv:1703.10069* (2017).

[8] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

[9] Gabriel Synnaeve, Pierre Bessiere, et al. 2011. A Bayesian Model for Plan Recognition in RTS Games Applied to StarCraft.. In *AIIDE*.

[10] Michael E Tipping. 2001. Sparse Bayesian learning and the relevance vector machine. *Journal of machine learning research* 1, Jun (2001), 211–244.

[11] Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. 2016. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *arXiv preprint arXiv:1609.02993* (2016).

[12] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. 2017. StarCraft II: a new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782* (2017).

[13] B. G. Weber and M. Mateas. 2009. A data mining approach to strategy prediction. In *2009 IEEE Symposium on Computational Intelligence and Games*. 140–147. https://doi.org/10.1109/CIG.2009.5286483

[14] Stefan Wender and Ian Watson. 2012. Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft: Broodwar. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE, 402–408.

[15] Huikai Wu, Junge Zhang, and Kaiqi Huang. 2017. MSC: A Dataset for Macro-Management in StarCraft II. *arXiv preprint arXiv:1710.03131* (2017).