Extraction of Interaction Events for Learning Reasonable Behavior in an Open-World Survival Game

Emmett Tomai

University of Texas Rio Grande Valley, 1201 W. University Dr., Edinburg, TX 78539
emmett.tomai@utrgv.edu

Abstract

Extracting event knowledge from open-world survival video games is a promising domain to investigate the application of Machine Learning techniques to routine human decision making. This contrasts with and builds upon typical game-based decision making work that focuses on optimal behavior. We propose an *Interaction Graph* data structure that can be trained from game play to enable hybrid reasoning and statistical estimation about what events can happen in the world. This enables an agent to exhibit increasingly more reasonable behavior after low numbers of training runs. An implementation and initial experimental validation are presented.

Introduction

The problem of agents that can make intelligent decisions is a long-standing AI challenge, which has seen fruitful work with both classic board games and video games. In classic board games, the decision making is characterized by selecting the optimal move from a deceptively simple set of possible moves. There are only a handful of pieces and actions in games like Chess or Go, but the interdependency of one move on another creates a combinatorial explosion of possible states. Recently, Deep Learning with Monte Carlo simulation has proven highly successful in surpassing the highest levels of human performance in Go (Silver et al. 2016). Machine Learning has also been successfully applied to a range of video game playing challenges (cf. Galway et al., 2008), notably the recent success of Deep Reinforcement Learning with Atari games (Minh et al. 2015). In most of this work, the decision making also selects from a small set of possible actions, with an explosive number of resulting configurations (here due to fine-grained spatiotemporal state). Considering both cases, Machine Learning for games has advanced the state-of-the-art in both notably deliberative and reactive decision-making. And in both cases, the type of decision making strongly applies to expert, task-specific performance.

In contrast, routine human decision making is not as notable for optimality as it is for robustness in the face of irrelevant state, adaptability to different contexts, and quick learning. Here too, video games can provide a useful domain for investigation. *Open-world survival games* are highly exploratory in nature, and involve a wider range of tasks repeated in an ever-evolving context. The goal is less to find the optimal behavior to win the game, and more to explore the range of behaviors that meet the criteria of surviving to explore further. Players must decide whether and how to respond to a variety of opportunities and threats as they are discovered. *Reasonable behavior* in this context is not optimal, but should (1) make choices consistent with pursuing some set of (possibly changing) goals, (2) not choose actions that are clearly detrimental or inferior in the short-term to other options, and (3) not require re-learning applicable knowledge in a new situation. The broad goal of this work is to explore how Machine Learning techniques can be applied to learning this reasonable behavior.

We hypothesize that this challenge requires an AI system to be able to identify the notable short-term outcomes afforded by the current situation, regardless of whether those outcomes are relevant to a specific task. The abstraction of *durative events* provides a composable structure to enable the system to accumulate and generalize operational knowledge around. In essence, we want to learn the characteristics of events in the environment, rather than a global policy to achieve a specific goal in the environment. We propose to automatically segment events in terms of unique interaction configurations between agents and other entities. The result is a novel *Interaction Graph (IGraph)* where nodes are types of interaction events, and edges are transitions between them. We then use each node and edge as context for probabilistic models that learn to predict features of the events and transitions (e.g. how long will it last, will it result in this or that outcome, will it lead to another type of interaction). Extracting and using this knowledge is a specialized type of reinforcement learning (Sutton & Barto 1998), where the IGraph fills the role of the MDP, enabling algorithms that can predict possible paths through the transition space.

In this paper, we present the IGraph concept and implementation details to explain how it extracts event models from game play. We also present a validation experiment providing evidence that it is capable of learning how to make intelligent decisions about considering choices, outcomes and how to reach goals in the world.

Related Work

For Machine Learning in modern video games, much prior work has focused on optimizing agents' low-level, real-time movements and actions using neural networks, evolutionary computing and reinforcement learning (cf. Galway et al., 2008). These techniques have also been applied to tactical and strategic decision-making, by isolating those elements and creating appropriate abstractions of the game state for the models to work with.

The most prolific genre for work in tactical and strategic decision-making in video games has been the Real-Time Strategy (RTS) genre. RTS work is a good analogue for decision-making in open-world survival games, since RTS popularized the mechanics for gathering, building and crafting, as well as the movement, control and combat models used in most 2d survival games. The prior work in RTS games suggests that learning in a complex game environment requires isolating specific abstractions of state and action. Several projects have used reinforcement learning with neural network q-value approximation to learn combat micro-management (Micic et al., 2011; Shantia et al. 2011; Wender & Watson, 2012). These approaches succeed by greatly simplifying the action space to fight or retreat scripts, and simplifying the feature space using manual abstractions such as the closest enemy or aggregate enemy health within range. These abstractions allow the learning model to work with relevant, fixed-size input. (Jaidee & Munoz-Avlia, 2012) presented a q-learning algorithm capable of playing complete, simple RTS scenarios by training on each class of unit and building separately. The state space and action space could therefore be tailored to each class, greatly reducing the size. Again, various useful abstractions were used in the state space and the action scripts (e.g. count of units stronger than x, attack all units weaker than attacker). (Sharma et al., 2007) used a three-layered architecture with a scripted planner on top, hybrid case-based reasoning and reinforcement learning (CBR/RL) for tactical decisions, and reactive planning at the bottom to show transfer learning in a simplified RTS environment. The CBR/RL component replaces the typical MDP by storing the learned transitions in cases and retrieving them in new scenarios. The inputs are global abstractions of game state (e.g. overall unit count, territories held) and the action space is simplified to Attack, Explore, Retreat and Conquer goals that are carried out by the reactive layer. (Synnaeve & Bessiere, 2012) used Bayesian inference to predict the outcome of attacks over the abstractions of regions and army strengths. In this work, we embrace the need for modular, local abstractions to learn over, but seek to move away from hand-made models towards a more general framework of events and outcomes.

The nodes of the IGraph provide context to train, validate and utilize regression and classification models for reasoning tasks. These models have become very mature in recent years, with a number of stable and accessible libraries providing a wide variety of off-the-shelf implementations. Many models can be found supporting continuous and categorical input and outputs, probabilistic predictions and dimensionality reduction. For this project, we are working in the *Scientific Python*[1] environment with easy access to linear and polynomial regression, as well as a wide range of trainable classifiers and regressors including Naïve Bayes, Decision Tree ensembles, SVM, Gaussian Processes and Discriminant Analysis.

Selecting from a pool of models and parameters is a fundamental problem in any data analysis field. This work follows the established view of model selection as an exhaustive search over the quality of results of the available models. (Linhart & Zucchini, 1986) formalized this using n-fold cross-validation for each model, and (Schaffer, 1993) applied it specifically to selecting a machine learning classifier for a given data set. Model parameters can be viewed as a recursive extension of that search. Significant work has also been done on improving that search by leveraging heuristic knowledge about the models (cf. Brodley, 1993) and better measuring the *fit* of a model (cf. Browne & Cudeck, 1992; Kohavi, 1995). Model parameter tuning and feature selection can be broadly viewed as recursive extensions of that search, and again, considerable work has gone into those areas both for general-purpose and model-specific techniques (cf. Guyon & Elisseeff, 2003; Yu & Liu, 2004; Snoek et al., 2012).

Open-World Survival Games

In an open-world survival game, the player is free to move throughout the game world, collecting resources from node entities such as trees, ponds and rocks. These resources are used to craft useful items such as tools and weapons, as well as structures that provide benefits such as shelter and storage. Roaming enemies (*mobs*) must be avoided or defeated in combat or else they will kill the player. Typically,

[1] https://www.scipy.org/about.html

there are additional environmental features such as hunger, thirst and exposure that can also end the game. While the general goal of the genre is to not die, the player experience is centered around exploration. The open world means that the player can go anywhere at any time, and more advanced enemies and resources are found together in different areas. The craftable items (and structures) are arranged in an advancing tree, such that creating later items requires creating earlier items, either because they are required for the crafting, required for gathering more advanced resources, or because the earlier items are necessary to survive to get to the later items. This gives the player an immediate progress system, while still allowing freedom to explore different branches, areas and secondary goals. These games are often paired with secondary goals such as building creative structures, defeating specific enemies or following stories laid out in the world.

Open-world survival games can have fast-paced action, such as numerous recent AAA "sandbox" shooters, but the genre is not specifically based on that interface. This work takes place at the level of decision-making about *behaviors*, such as walking to a pond and drinking from it or moving to a new region to explore. This maps conveniently to 2d survival games using a *click-to-move* interface, which we consider here. Recognizing such behaviors in a continuous control environment is outside the scope of this project, but given that actions like gathering and attacking are discrete and clearly observable, many of the noise problems inherent in continuous movement could be factored out.

Interaction Graph

The IGraph is a set of nodes and edges where each node abstracts an *interaction*: a set of behaviors being performed together over an interval of time that share at least one entity. This does not mean that the behaviors start and end at the same time, only that they fully cover the interval of the node. The IGraph represents transitions between these interaction states. For example, as shown in Figure 1, the red agent begins by performing a gather behavior targeting the flower bush in part (a). Both entities are part of the interaction. The green agent then begins to attack the red agent in part (b), joining the interaction. This interaction node exists for as long as both the green and red agents continue these behaviors and no other behaviors are performed involving red, green or the bush. From there, the red agent might choose to attack the green agent back, as in part (c). This transitions to a new interaction node, where the bush is no longer part of the interaction. Alternatively, the red agent might choose instead to attack innocent passerby blue, who is idle, and is part of the interaction node in part (d). Each node in the IGraph has a primary agent entity, which is the point of view of the transitions. Each node is unique to the set of behaviors and arguments in the interaction. For example, all cases in this world where *`(gathers e1 e2)` `and (attacks e3 e1)`* are grouped into one node with the agent in role *`e1`* as primary and a second node with the agent in role *`e3`* as primary.

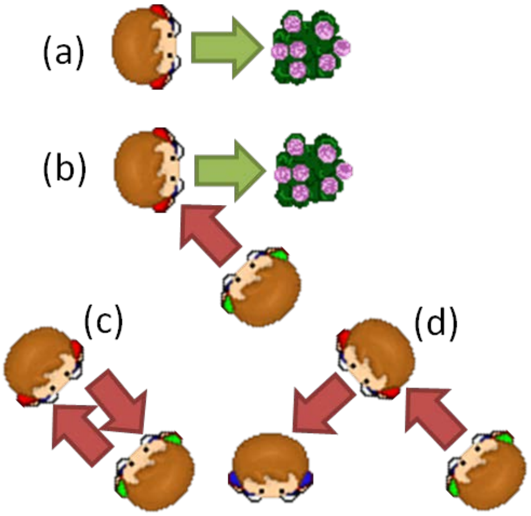

Figure 1. Example interaction states.

Transition edges within the IGraph are of four types, relative to the primary agent:

1) A *choice* transition involves the primary agent deciding to change the behavior they are performing. A choice transition may stochastically lead to more than one node, but is initiated as part of the decision process. In Figure 1, (b)=>(c) is a choice.

2) An *external interrupt* transition involves an entity not in the interaction starting an overlapping behavior to become part of the interaction in the destination node. In Figure 1, (a)=>(b) is an external interrupt.

3) An *internal interrupt* transition involves an entity in the interaction starting a new behavior. If the green agent in (c) ran away, this would be an internal interrupt. A special case of this is the primary agent dying.

4) A *completion* transition happens when the primary agent behavior completes, either by succeeding or by failing. If the red agent in (b) finished gathering from the bush, this would be a completion.

The IGraph provides a learnable, inspectable framework for generalizing predictions and estimations about what can happen in the game world. Formally, each node consists of:

`B:` the set of behaviors in the interaction.

`L:` a set of entity variables generalizing the entities in *`B`*.

`C:` a set of choices that have been observed, where each choice is a behavior type and bindings to `L`.

`O:` a set of outcome effects observed on completion.

`T:` a set of observed interrupt transitions, with bindings to `L` and open bindings to external entities.

Predictors (classifiers and regressors) specific to each node are trained to predict time-to-completion and the probabilities for each outcome in `O`, the probabilities of each transition in `T` (including the notable probability of death), and the probabilities of each transition following from each choice in `C`. Both interrupts and outcomes are treated as independent probabilities for simplicity, such that the posterior probability of an outcome (assuming that the agent doesn't chooses to continue the behavior) is its estimated probability multiplied by (1.0 - the probability that none of the interrupts happen).

Fundamental Reasoning Abstractions

Identifying the right state and action abstractions for each predictor is critical to good performance. Rather than hand making those abstractions, our training system starts with a set of reusable *fundamental reasoning abstractions*. These are concept models that abstract details of the world such as entities having positions in the world, behaviors taking time, behaviors having outcomes, gaining an item being when something is in an agent's inventory that was not there before, or the definition of the distance between two entities. These abstractions are not linked to any particular prediction, but it is up to the learning process to determine where and when they are appropriate to use.

Training the Interaction Graph

The IGraph is built by playing the game and recording traces of entity behaviors. It can be easily updated and retrained as more data becomes available. A sequence of exemplar nodes is created from a game trace by starting with the sequence of behaviors for the primary agent entity. In Figure 2, part (a), the green, orange and blue rectangles represent a sequence of primary agent behaviors on a timeline. Every other behavior in the trace is then compared to that sequence, such as the purple behavior shown in part (a). If it has common entities with the green and orange behaviors (which it overlaps), it splits the sequence into five nodes, as shown in part (b). The second and third nodes in part (b) involve both the primary agent and the entities involved in the purple behavior. As shown in parts (c) and (d), if the second purple node has overlapping entities only with blue (and not orange), then there are still only five nodes.

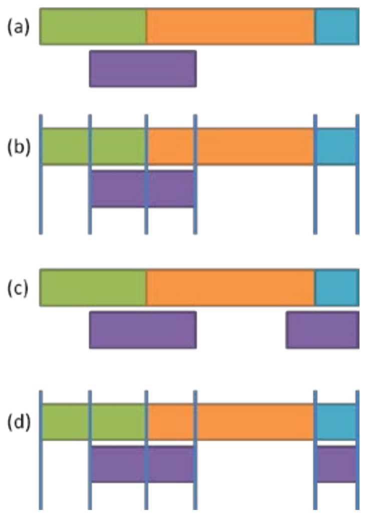

Figure 2. Segmenting trace behaviors into exemplar nodes.

Each exemplar sequence is fed into the training IGraph, and each exemplar node unifies its *behavior signature* (the behaviors with specific entity bindings) against `B(L)` from the existing graph nodes. If there is a match, the exemplar node is added as an exemplar to that node, to be used to add choices, outcomes and transitions, and to train the predictors. Otherwise a new node is created.

The set of Choices `C` for a node are identified by exemplars where the primary agent behavior does not complete, but is different in the next node in the exemplar sequence. Outcomes `O` are identified for completed behaviors by taking a state delta between the world state at the start of the exemplar node *s0* and the end *s1*. The delta is taken by applying a generic set of fundamental abstractions, which can be expanded and left for the system to sift through. For this experiment, the effect models were:

`obtain(entity_id, item_type_id, ct):` the specified entity has *ct* more of the specified item type in their inventory in *s1* than in *s0*.

`lose(entity_id, item_type_id, ct):` the specified entity has *ct* less of the specified item type in their inventory in *s1* than in *s0*.

`die(entity_id):` the specified entity, which is a decision-making entity (player, agent or mob), exists in *s0* and not in *s1*.

`remove(entity_id):` the specified entity, which is not a decision-making entity, exists in *s0* and not in *s1*.

The outcomes are sets of always co-occurring effects observed. For example gathering from a bush might always give leaves and flowers (one outcome) but only sometimes twigs (another, independent outcome). The exemplars stored in the node for the completion cases are marked as positive or negative examples for each of the outcomes. Those exemplars are also used to train the time-to-completion predictor.

Internal and external interrupt transitions are identified from the sequence of exemplars as all those that are not choices or completions. Interrupts are mutually exclusive, so each exemplar is stored as a positive example of only one interrupt transition. The exemplar behavior signature for the destination of the transition is compared against the source signature to identify open entity bindings in the former (e.g. "the entity who attacked"). In generating positive and negative training data, the open entity bindings are bound against each potential entity in the world (with only type and range filters). So in the case of an external attack, the entity who did attack in the exemplar sequence is a positive example, while all the entities who did not attack (in that exemplar or any other) are negative. Alternative negative exemplar generation strategies are one of many settings that the learning process can automatically search and validate to find the best predictions and estimates.

Once the available exemplars have been stored in the training IGraph, the predictors are trained. For a continuous value such as time-to-completion, a set of regression models are automatically evaluated, while for categorical values a set of classifiers are automatically evaluated. For binary categorical values, regression to probability between 0 and 1 is also considered. The feature vectors used as input to each candidate predictor are generated from all the fundamental abstractions that apply to the entities involved in the interaction. This includes entity types and quantities as well as attributes (both type-level values such as the movement speed of a bear, and instance-level values such as an entity's current health). If a behavior binds more than one entity, than all the fundamental abstractions of relationships between entities are also included. Spatial abstractions are particularly useful here, such as distance, path distance, distance to a path and topological grouping. The training process includes all available relationships and uses simple dimensionality reduction and verification techniques to figure out what is predictive.

For a given predictor, a set of learning models are tried. The training feature vectors are filtered to remove categorical values if they are not supported, and to bin continuous values if they are not supported. Each model is wrapped (if necessary) to provide normalization of continuous values based on the training data and dimensionality reduction if possible. N-fold cross-validation is also wrapped around each learning model. Based on the output of the validation, the predictors can be compared for effectiveness, and/or additional volume of exemplars can be generated by the system. An accepted learning model is retrained on the entire set, subject to dimensionality reduction, then retrained on only the applicable features.

In order to support quick evaluations of the threat of death, the value of *dread* is calculated for each node in the graph, analogously to reward in standard MDP-based reinforcement learning. Instead of reward for a specific goal, dread estimates how much death has come from passing through that node. This mechanism should be extensible to other generally positive or negative concerns.

Finally, the training IGraph exports itself for run-time use, removing exemplars and other unnecessary intermediate data.

Run-Time Interaction Graph Agent

In order to use the information extracted by the IGraph, a run-time agent can be assigned to an agent entity in the game world and given a set of (possibly changing) goals to attempt to reach. Importantly, the IGraph does not have to be trained on those particular goals (although it should speed up training). The goals available are determined by the game and the agent must be able to evaluate against the game state to determine when they are met.

The agent monitors the game state by generating the behavior signature for itself each frame. Whenever the signature changes, it retrieves the corresponding node from the IGraph. When in an idle state, the agent retrieves all choice transitions from that state, gets the destination IGraph nodes, generates valid bindings to the entities in the world, and evaluates the resulting candidate states. The evaluation calculates three values: expected reward, expected cost and what we refer to as *concern*. The expected reward is a straightforward utility calculation of the estimated probability of each outcome, given completion, by its value to the agent's goals and the estimated probability of completion. Likewise, the expected cost is simply the estimated time to reach completion. In considering each candidate choice, the agent uses the *value ratio*, which is the expected reward over the expected cost. Concern is an estimate of the risk of death (losing) for each choice. The destination node has its own predictor for the probability of a death outcome in the absence of any transition to another node. This is added to the sum of dread for each possible interrupt transition out of that state, multiplied by the probability of that transition.

The candidate choices are sorted according to their value ratio and concern. Choices with no value are discarded, as a random movement would be preferable. The remaining choices are separated into low, medium and high concern bins and sorted by value ratio. The highest valued choice in the lowest non-empty bin is chosen for execution.

For non-idle states, the only difference is that the current state is also evaluated and sorted with the rest to see if the agent should stick with the current behavior.

Experimental Setup

Game Setup

As an initial validation of the IGraph to extract useful knowledge about events and outcomes, we have created a testbed 2d open-world survival game. This experimental game simplifies the out-of-scope behavior recognition problem as click-to-move behaviors are directly identifiable as the commands given by the experimental agent. The mobs in the game use the same behavioral system, so their behaviors are easily traced as well. The game uses a standard *Component-Entity-System* architecture (Boreal Games, 2013), where all state data is contained in plain data arrays. Every agent decision creates a behavior component which uses *Behavior Tree* semantics (Simpson, 2014) including status codes RUNNING, SUCCESS and FAILURE. In this way, standard game architecture enables data collection, with minimal extra effort in the game engine itself. Entity attributes and relationships are likewise observable, but for this work we have simplified the process by making those values directly available from the components. This is similar to data that is made available through systems like the Brood War API[2] for StarCraft AI work. Also for experimental convenience, we have a non-interactive Python build of the game that runs agents either headless or with a minimalist visualization for debugging.

To generate initial data, the game is played by an *Exploration Agent* that chooses random behaviors to execute. At any time that no behavior is in progress, the agent binds all possible behaviors and randomly selects one. Due to the very high branching factor, movement to all possible empty locations is not included. Instead, movement to a single, random location is included as a possibility. During execution of a behavior, the agent may randomly interrupt with a certain probability, and select a different behavior. A run ends when goals set for the agent are fulfilled, the agent dies, or a time limit is reached. The attributes used for training predictors are determined by the game engine (e.g. hp, attack speed, awareness distance) while all relationship abstractions and effect models that can be evaluated against the game state are included.

Experimental Design and Results

The initial testing is focused on its ability to quickly learn to play the basic game by playing. For each test, 100 scenarios were played by the run-time agent and scored for success rate and time to win. Each scenario involves meeting a set of random gathering goals from randomly placed resource nodes while avoiding or defeating randomly placed mobs. The first test was performed with an empty IGraph (0 training games). After each test, the IGraph was trained with 50 more training games and tested again, up to 500. The success rates are shown in Figure 3, and the average time to success (among the successful runs only) are shown in Figure 4.

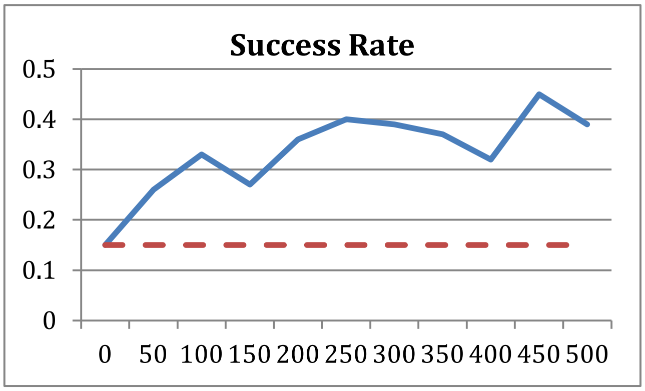

Figure 3. Success rate over 100 testing runs after training on 0-500 random sample runs.

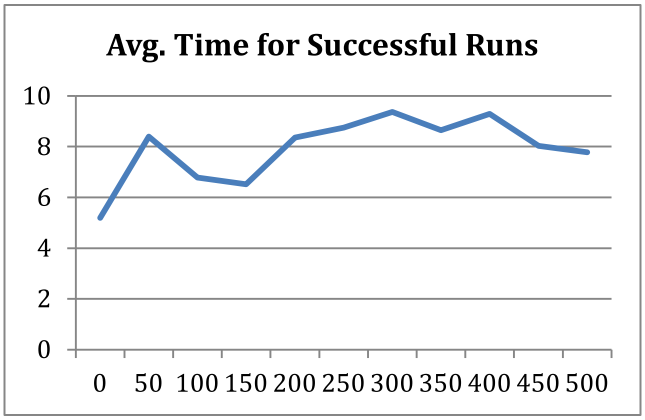

Figure 3. Average time spent completing the successful runs after training on 0-500 random sample runs.

As shown in Figure 3, the untrained success rate (random behavior) is around 15%. The system very quickly improves, although it also flattens out rather quickly. Because of the transparency of the extracted events, we can see that the improvement is due to learning to predict 1) which types of resources nodes give which types of items, 2) the time cost to gather from a given node, and 3) which behaviors will directly (i.e. attack) or indirectly (i.e. moving to close to a patrolling mob) lead into unwinnable fights. The stochastic nature of resource drops and fights does mean that the system could never be right all the time, and the simplicity of these initial scenarios does limit the creative responses available to the agent.

The average time spent completing the goals is roughly stable, although it does increase with more training. To clarify, this has nothing to do with processing time, as the times are in world clock, which runs on a fixed tick. It is possible that since the more trained agents win more often,

[2] https://bwapi.github.io/

they are winning harder/longer scenarios. The scenarios are all short gathering cycles, as we saw no real difference in longer or more spread out scenarios except that they took longer to process.

Conclusion and Future Work

We have proposed a novel knowledge structure, the Interaction Graph, which generalizes over interactions between entities, presented the implementation details, and done initial testing to verify that it can learn the basic game set up. The IGraph learns from playing, enables reasoning about all known possibilities in the state space and provides context for task-specific predictors to perform hybrid symbolic/statistical reasoning. We have shown that as the IGraph is trained, the agent behavior becomes more reasonable in going after the right resource nodes and avoiding detrimental combat.

We are continuing to add more features to the game and expand the model to handle them, including planning ahead (crafting), memory for exploring, more complex combat, environmental threats and multi-agent interactions (cooperation and antagonism). Along with this incremental development will be additional fundamental abstractions. A key question we are exploring is how the IGraph will scale, particularly at run-time, with the increase in complexity of the game.

We have also implemented a real-time Monte-Carlo Tree Search component for focused training, allowing the system to "rewind" and try alternative paths to quickly refine its predictors. At this time we do not have experimental validation of that system. We also ran a comparative Convolutional Neural Network solution to the basic game runs, but performance was so poor that we believe there must be an implementation error to fix.

References

Boreal Games. (2013). *Understanding Component-Entity-Systems.* https://www.gamedev.net/articles/programming/general-and-gameplay-programming/understanding-component-entity-systems-r3013. Retrieved July 10, 2017.

Brodley, C. E. (1993). Addressing the selective superiority problem: Automatic algorithm/model class selection. In Proceedings of the tenth international conference on machine learning (pp. 17-24).

Browne, M. W., & Cudeck, R. (1992). Alternative ways of assessing model fit. Sociological Methods & Research, 21(2), 230-258.

Galway, L., Charles, D. and Black, M. (2008). Machine learning in digital games: a survey. Artificial Intelligence Review. Volume 29, Number 2, 123-161.

Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. Journal of machine learning research, 3(Mar), 1157-1182.

Jaidee, U., & Muñoz-Avila, H. (2012, October). Classq-l: A q-learning algorithm for adversarial real-time strategy games. In Eighth Artificial Intelligence and Interactive Digital Entertainment Conference.

Kohavi, R. (1995, August). A study of cross-validation and bootstrap for accuracy estimation and model selection. In Ijcai (Vol. 14, No. 2, pp. 1137-1145).

Linhart, H., & Zucchini, W. (1986). Finite sample selection criteria for multinomial models. Statistische Hefte, 27(1), 173-178.

Micić, A., Arnarsson, D., & Jónsson, V. (2011). Developing game AI for the real-time strategy game StarCraft. Technical report, Reykjavik University.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.

Schaffer, C. (1993). Selecting a classification method by cross-validation. Machine Learning, 13(1), 135-143.

Shantia, A., Begue, E., & Wiering, M. (2011, July). Connectionist reinforcement learning for intelligent unit micro management in starcraft. In Neural Networks (IJCNN), The 2011 International Joint Conference on (pp. 1794-1801). IEEE.

Sharma, M., Holmes, M. P., Santamaría, J. C., Irani, A., Isbell Jr, C. L., & Ram, A. (2007, January). Transfer Learning in Real-Time Strategy Games Using Hybrid CBR/RL. In IJCAI (Vol. 7, pp. 1041-1046).

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Dieleman, S. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484-489.

Simpson, C. (2014). *Behavior trees for AI: How they work.* http://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php. Retrieved July 10, 2017.

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In Advances in neural information processing systems (pp. 2951-2959).

Sutton, R. S., & Barto, A. G. (1998). Reinforcement learning: An introduction (Vol. 1, No. 1). Cambridge: MIT press.

Synnaeve, G., & Bessiere, P. (2012, September). Special tactics: A bayesian approach to tactical decision-making. In Computational Intelligence and Games (CIG), 2012 IEEE Conference on (pp. 409-416). IEEE.

Wender, S., & Watson, I. (2012, September). 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 (pp. 402-408). IEEE.

Yu, L., & Liu, H. (2004). Efficient feature selection via analysis of relevance and redundancy. Journal of machine learning research, 5(Oct), 1205-1224.