Advances in Cognitive Systems: Papers from the 2011 AAAI Fall Symposium (FS-11-01)

Building Human-Level AI for Real-Time Strategy Games

Ben G. Weber
Expressive Intelligence Studio
UC Santa Cruz
bweber@soe.ucsc.edu

Michael Mateas
Expressive Intelligence Studio
UC Santa Cruz
michaelm@soe.ucsc.edu

Arnav Jhala
Computational Cinematics Studio
UC Santa Cruz
jhala@soe.ucsc.edu

Abstract

Video games are complex simulation environments with many real-world properties that need to be addressed in order to build robust intelligence. In particular, real-time strategy games provide a multi-scale challenge which requires both deliberative and reactive reasoning processes. Experts approach this task by studying a corpus of games, building models for anticipating opponent actions, and practicing within the game environment. We motivate the need for integrating heterogeneous approaches by enumerating a range of competencies involved in gameplay and discuss how they are being implemented in EISBot, a reactive planning agent that we have applied to the task of playing real-time strategy games at the same granularity as humans.

Introduction

Real-time strategy (RTS) games are a difficult domain for humans to master. Expert-level gameplay requires performing hundreds of actions per minute in partial information environments. Additionally, actions need to distributed across a range of in-game tasks including advancing research progress, sustaining an economy, and performing tactical assaults. Performing well in RTS games requires the ability to specialize in many competencies while at the same time working towards high-level goals. RTS gameplay requires both parallel goal pursuit across distinct competencies as well as coordination among these competencies.

Our work is motivated by the goal of building an agent capable of exhibiting these types of reasoning processes in a complex environment. Specifically, our goal is to develop an RTS agent capable of performing as well as human experts. Paramount to this effort is evaluating the agent in an environment which resembles the interface provided to human players as closely as possible. Specifically, the agent should not be allowed to utilize any game state information not available to a human player, such as the locations of non-visible units, and the set of actions provided to the agent should be identical to those provided through the game's user interface. Enforcing this constraint ensures that an agent which performs well in this domain is capable of integrating and specializing in several competencies.

Real-time strategy games demonstrate the need for heterogeneous agent architectures, because reducing the decision complexity of the domain is non-trivial. One possible approach is to split gameplay into isolated subproblems and to reason about each subproblem independently. This approach is problematic, because RTS gameplay is non-hierarchical resulting in contention for in-game resources across different subproblems. A second approach is to develop an abstraction of the state space and use it for reasoning about goals. This approach is also problematic, because different types of abstractions are necessary for the different tasks that need to be performed. RTS gameplay requires concurrent and coordinated goal pursuit across multiple abstractions. We refer to AI tasks that have these requirements as *multi-scale AI problems* (Weber et al. 2010).

Our domain of analysis is the real-time strategy game *StarCraft*. High-level StarCraft gameplay requires specializing in several different competencies of varying complexity. One such competency is micromanagement, which is the task of controlling individual units in combat scenarios in order to maximize the utility of a squad of units. This area of expertise focuses on reactive behavior, because it requires split-second timing of actions. Another competency is strategy selection, which is the task of determining which types of units to produce and which upgrades to research. This area of expertise focuses on planning and opponent modeling, and involves formulating strategies in order to counter opponent strategies. We also selected this game because it is played at a professional level, resulting in a large number of high-quality replays being available for analysis, and there is a large player base for evaluating our system.

Our approach for managing the complexity of StarCraft is to partition gameplay into domains of competence and develop interfaces between these competencies for resolving goal conflicts. It is based on the integrated agent framework proposed by McCoy and Mateas (2008), which partitions the behaviors in a reactive planning agent into managers which specialize in distinct aspects of gameplay. The decomposition of gameplay into managers is based on analysis of expert gameplay.

EISBot is a heterogeneous agent for playing StarCraft which incorporates many of these competencies. The core of the agent is a reactive planner that supports real-time action execution and concurrent goal pursuit. A large num-

ber of competencies are implemented in the ABL reactive planning language (Mateas and Stern 2002). The reactive planner interfaces with external components which perform prediction and planning tasks. These components are implemented using case-based reasoning, machine learning, and particle filters. The components are connected together by applying several integration idioms, including using working memory as a blackboard, instantiating new goals for the agent to pursue, and suspending and resuming the execution of tasks. We have performed evaluation of EISBot on a competitive StarCraft tournament ladder.

The remainder of this paper is structured as follows. In section 2 we discuss related work. Section 3 provides an overview of the EISBot architecture. Next, we discuss StarCraft gameplay and competencies in Section 4, and present the competencies in EISBot in Section 5. In section 6 we present an evaluation of EISBot. We provide conclusions in Section 7 and discuss future work in Section 8.

Related Work

Our agent builds on ideas from commercial game AI, cognitive architectures, and heterogeneous agent designs. It also continues the tradition of using games as an AI testbed.

Video games are ideal for AI research, because they provide a domain in which incremental and integrative advances can be made (Laird and VanLent 2001). RTS games in particular are interesting because they provide several challenges including decision making under uncertainty, spatial and temporal reasoning, and adversarial real-time planning (Buro 2003). Previous work has also shown that the decision complexity of RTS games is enormous (Aha, Molineaux, and Ponsen 2005).

Cognitive architectures provide many of the mechanisms required for integrating heterogeneous competencies. The ICARUS architecture is capable of reasoning about concurrent goals, which can be interrupted and resumed (Langley and Choi 2006). The system has components for perception, planning, and acting which communicate through the use of an active memory. One of the main differences from our approach is that our system works towards modeling gameplay actions performed by players, as opposed to modeling the cognitive processes of players.

SOAR is a cognitive architecture that performs state abstraction, planning, and multitasking (Wintermute, Xu, and Laird 2007). The system was applied to the task of playing real-time strategy games by specifying a middleware layer that serves as the perception system and gaming interface. RTS games provide a challenge for the architecture, because gameplay requires managing many cognitively distant tasks and there is a cost for switching between these tasks. The main difference from our approach is that we decompose gameplay into more distinct modules and have a larger focus on coordination across tasks.

Darmok is an online case-based planner that builds a case library by learning from demonstration (Ontañón et al. 2010). The system can be applied to RTS games by supplying a collection of game replays that are used during the planning process. The decision cycle in Darmok is similar to the one in EISBot, in which each update checks for finished steps, updates the world state, and picks the next step. Behaviors selected for execution by Darmok can be customized by specifying a behavior tree for performing actions (Palma et al. 2011). The main difference from our approach is that Darmok uses a case library to build a collection of behaviors, while EISBot uses a large collection of hand-authored behaviors.

Heterogeneous agent architectures have also been applied to the task of playing RTS games. Bakkes et al. (2008) present an approach for integrating an existing AI with case-based reasoning. The AI is able to adapt to opponents by using a case retriever which selects the most suitable parameter combination for the current situation, and modifying behavior based on these parameters. Baumgarten et al. (2009) present an RTS agent that uses case-based reasoning, simulated annealing, and decision tree learning. The system uses an initial case library generated from a set of randomly played games, and refines the library by applying an evaluation function to retrieved cases in order to evolve new tactics. Our system differs from this work, because components in EISBot interface using goals and plans rather than parameter selection.

Commercial game AI provides an excellent baseline for agent performance, because it must operate within a complex environment, as opposed to an abstraction of a game. However, the goal of commercial game AI is to provide the player with an engaging experience, as opposed to playing at the same granularity as a player. Both deliberative and reactive planning approaches have been applied to games. Behavior trees are an action selection technique for non-player characters (NPCs) in games (Isla 2005). In a behavior tree system, an agent has a collection of behaviors available for execution. Each decision cycle update, the behavior with the highest priority that can be activated is selected for execution. Behavior trees provide a mechanism for implementing reactive behavior in an agent, but require substantial domain engineering. Goal-oriented action planning (GOAP) addresses this problem by using a deliberative planning process (Orkin 2003). A GOAP agent has a set of goals, which upon activation trigger the agent to replan. The planning operators are similar to those in STRIPS (Fikes and Nilsson 1971), but the planner uses A* to guide the planning process. The main challenge in implementing a GOAP system is defining representations and operators that can be reasoned about in real time. One of the limitations of both of these approaches is that they are used for action selection for a single NPC and do not extend to multi-scale problems.

EISBot Architecture

EISBot is an extension of McCoy and Mateas's (2008) integrated agent framework. We have extended the framework in several ways. While the initial agent was applied to the task of playing *Wargus* (Ponsen et al. 2005), we have transferred many of the concepts to StarCraft. Additionally, we have implemented several of the competencies that were previously identified, but stubbed out in the original agent implementation. Finally, we have interfaced the architecture with external components in several ways.

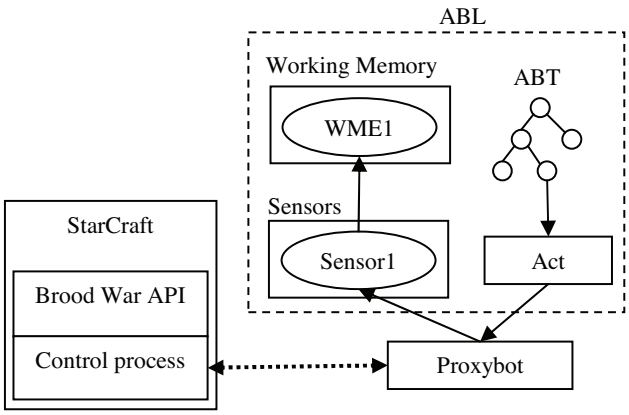

Figure 1: EISBot-StarCraft interface

The core of the agent is a reactive planner which interfaces with the game environment, manages active goals, and handles action execution. It is implemented in the ABL (A Behavior Language) reactive planning language (Mateas and Stern 2002). ABL is similar to BDI architectures such as PRS (Rao and Georgeff 1995), a hybrid architecture for planning and decision making. The main strength of reactive planning is support for acting on partial plans while pursuing goal-directed tasks and responding to changes in the environment (Josyula 2005).

EISBot interfaces with StarCraft through the use of sensors which perceive game state and actuators which enable the agent to send commands to units. The ABL interface and agent components are shown in Figure 1. The Brood War API, Control Process, and ProxyBot components provide a bridge between StarCraft and the agent. The agent also includes a working memory and an active behavior tree (ABT). The ABT pursues active goals by selecting behaviors for expansion from the behavior library, which is a collection of hand-authored behaviors.

Our agent is based on a conceptual partitioning of gameplay into areas of competence. It is composed of managers, each of which is responsible for a specific aspect of gameplay. A manager is implemented as a collection of behaviors in our system. We used several reactive planning idioms to decompose gameplay into subproblems, while maintaining the ability to handle cross-cutting concerns such as resource contention between managers. EISBot is composed of the following managers:

- **Strategy manager:** responsible for the strategy selection and attack timing competencies.
- **Income manager:** handles worker units, resource collection, and expanding.
- **Construction manager:** responsible for managing requests to build structures.
- **Tactics manager:** performs combat tasks and micromanagement behaviors.
- **Recon manager:** implements scouting behaviors.

Further discussion of the specific managers and planning idioms is available in previous work (McCoy and Mateas 2008; Weber et al. 2010).

Each of the managers implements an interface for interacting with other components. The interface defines how working memory elements (WMEs) are posted to and consumed from working memory. This approach enables a modular agent design, where new implementations of managers can be swapped into the agent. It also supports the decomposition of tasks, such as constructing a structure. In EISBot, the decision process for selecting which structures to construct is decoupled from the construction process, which involves selecting a construction site and monitoring execution of the task. This decoupling also enables specialized behavior to be specified in a manager, such as unit specific movement behaviors in the tactics manager.

Integration Approaches

We have integrated the reactive planner with external components. The following approaches are used to interface components with the reactive planner:

- Augmenting working memory
- External goal formulation
- External plan generation
- Behavior activation

These approaches are not specific to reactive planning and can be applied to other agent architectures.

The agent's working memory serves as a blackboard (Hayes-Roth 1985), enabling communication between different managers. Working memory is also used to store beliefs about the game environment. External components interface with EISBot by augmenting working memory with additional beliefs about the game environment. These beliefs include predictions about the opponents' strategy and unit locations. The agent incorporates these predictions in the preconditions of specific behaviors, such as selecting locations to attack.

Another way EISBot interfaces with components is by pursuing goals that are selected outside of the reactive planning process. Goals that are formulated external to the agent can be added to the reactive planner by modifying the structure of the active behavior tree, or by triggering ABL's spawngoal functionality, which causes the agent to pursue a new goal in parallel with the currently active goals. We have used this approach to integrate the reactive planner (Weber, Mateas, and Jhala 2010a) with an instantiation of the goal-driven autonomy conceptual model (Molineaux, Klenk, and Aha 2010).

We have also integrated the agent with an external planning process (Weber, Mateas, and Jhala 2010b). Plans generated by the planning component are executed in the reactive planner by adding elements to working memory which activate a sequence of actions to be executed. This approach is similar to the plan expansion mechanism. Interfacing with external planners enables the agent to incorporate deliberative reasoning components.

ABL agents can be interfaced with components which modify the activation conditions of behaviors. ABL behaviors contain a set of preconditions, which specify activation

Figure 2: A screenshot of StarCraft showing Protoss (green) and Terran (yellow) players engaged in combat.

conditions. These conditions can contain procedural preconditions (Orkin 2006), which query an external component about activation. We have explored the use of case-based reasoning for behavior activation (Weber and Ontañón 2010), while Simpkins et al. (2008) present a reinforcement learning approach for behavior activation.

StarCraft

StarCraft[1] is a real-time strategy game where the player takes the role of a commander in a military scenario. The single objective given to players is to destroy all opponents. Achieving this objective requires performing numerous tasks which at a minimum are gathering resources, producing structures and units, and attacking enemy forces. To gain an advantage over the opponent, a player may perform additional tasks including scouting and upgrading. A screenshot of combat in StarCraft is shown in Figure 2.

There are several properties that make StarCraft gameplay a challenging task. Actions are performed in real-time, requiring both deliberative and reactive reasoning. Also, the game enforces imperfect information through a fog-of-war, which limits visibility of the map to areas where the player controls units. There are simultaneous tasks to perform, and focusing too much attention on a single task is detrimental. Another challenge is issuing orders to hundreds of units, which may have distinct tasks.

Learning to play StarCraft is difficult for additional reasons. There is a steep learning curve to StarCraft, because making a mistake early in the game can have drastic consequences. However, it is difficult to evaluate the reward of individual commands due to the long game length. Also, inaction is extremely detrimental, because failing to issue orders results in opponents gaining a strategic or economic advantage.

[1]StarCraft and its expansion StarCraft: Brood War were developed by Blizzard Entertainment™.

There are three diverse races in StarCraft: Protoss, Terran, and Zerg. Each race has a unique technology graph (tech tree) which can be expanded to unlock new unit types, structures, and upgrades. Additionally, each race supports different styles of gameplay. Mastering a single race requires a great deal of practice, and experts in this domain focus on playing a single race.

Effective StarCraft gameplay requires specializing in several distinct competencies. Micromanagement is the task of controlling individual units in combat in order to maximize the utility of a unit. Terrain analysis is another area of competence which determines where to place units and structures to gain tactical advantages. Strategy selection is the task of determining which unit types to produce and upgrades to research, in order to counter to opponent strategies. Attack timing determines the ideal moment to launch an attack against the opponent.

StarCraft Gameplay

StarCraft gameplay involves performing tasks across multiple scales. At the strategic scale, StarCraft requires decision-making about long-term resource and technology management. For example, if the agent is able to control a large portion of the map, it gains access to more resources, which is useful in the long term. However, to gain map control, the agent must have a strong combat force, which requires more immediate spending on military units, and thus less spending on economic units in the short term.

At the economic scale, the agent must also consider how much to invest in various technologies. For example, to defeat cloaked units, advanced detection is required. But the resources invested in developing detection are wasted if the opponent does not develop cloaking technology.

At the tactical scale, effective StarCraft gameplay requires both micromanagement of individual units in small-scale combat scenarios and squad-based tactics such as formations. In micromanagement scenarios, units are controlled individually to maximize their utility in combat. For example, a common technique is to harass an opponent's melee units with fast ranged units that can outrun the opponent. In these scenarios, the main goal of a unit is self-preservation, which requires a quick reaction time.

Effective tactical gameplay also requires well coordinated group attacks and formations. For example, in some situations, cheap units should be positioned surrounding long-ranged and more expensive units to maximize the effectiveness of an army. One of the challenges in implementing formations in an agent is that the same units used in micromanagement tactics may be reused in squad-based attacks. In these different situations, a single unit has different goals: self-preservation in the micromanagement situation and a higher-level strategic goal in the squad situation.

Micromanagement

Expert players issue movement and attack commands to individual units to increase their effectiveness in combat. The motivation for performing these actions is to override the default low-level behavior of a unit. When a unit is given a command to attack a location, it will begin attacking the

first enemy unit that comes into range. The default behavior can lead to ineffective target selection, because there is no coordination between different units.

Target selection and damage avoidance are two forms of micromanagement applied to StarCraft. To increase the effectiveness of units, a player will manually select the targets for units to acquire. This technique enables units to focus fire on specific targets, which reduces the enemy unit's damage output. Another use of target selection is to select specific targets which are low in health or high-profile targets. Dancing is the process of commanding individual units to flee from battle while the squad remains engaged. Running away from enemy fire causes the opponent units to acquire new targets. Once the enemy units have acquired new targets, the fleeing unit is brought back into combat. Dancing increases the effectiveness of a squad, because damage is spread across multiple units, increasing the average lifespan of each unit.

Micromanagement is a highly reactive process, because it requires a large number of actions to be performed. During peak gameplay, expert players often perform over three hundred actions per minute (McCoy and Mateas 2008). Effective micromanagement requires a large amount of attention and focusing too much on micromanagement can be detrimental to other aspects of gameplay, since the overall advantage gained by micromanaging units is bounded. Due to the attention demands of micromanagement, it is more common earlier in the game when there are fewer tasks to manage and units to control.

Terrain Analysis

Expert players utilize terrain in a variety of ways to gain tactical advantages. These tasks include determining where to build structures to create bottlenecks for opponents, selecting locations for engaging enemy forces, and choosing routes for force movements. High ground plays an important role in StarCraft, because it provides two advantages: units on low ground do not have vision of units on high ground unless engaged, and units on low ground have a 70% chance of hitting targets on high ground. Players utilize high ground to increase the effectiveness of units when engaging and retreating from enemy units.

Terrain analysis in RTS games can be modeled as a qualitative spatial reasoning task (Forbus, Mahoney, and Dill 2002). While several of the terrain analysis tasks such as base layout are deliberative, a portion of these tasks can be performed offline using static map analysis. This competency also requires the ability to manage dynamic changes in terrain caused by unit creation and destruction.

Strategy Selection

Strategy selection is a competency for determining the order in which to expand the tech tree, produce units, and research upgrades. Expert players approach this task by developing build orders, which are sequences of actions to execute during the opening stage of a game. As a game unfolds, it is necessary to adapt the initial strategy to counter opponent strategies. In order to determine which strategy an opponent has selected, it is necessary to actively scout to determine which unit types and upgrades are being produced. Players also incorporate predictions of an opponent's strategy during the strategy selection process. Predictions can be built by studying the player's gameplay history and analyzing strategies favored on the selected map.

Strategy selection is a deliberative process, because it involves selecting sequences of actions to achieve a particular goal. Strategy planning can be performed both offline and online. Build order selection is an offline process performed before the start of a game, informed by the specific map and anticipated opponent strategies, while developing a counter strategy based on the opponent strategy during a game is an online task.

Attack Timing

Attack timing is a critical aspect of StarCraft gameplay, because there is a large commitment involved. There are several conditions that are used for triggering an attack. A timing attack is a planned attack based on a selected build order. Timing attacks are used to take advantage of the selected strategy, such as attacking as soon as an upgrade completes. An attack can also be triggered by observing an opportunity to damage the opponent such as scouting units out of place or an undefended base. There are several other conditions which are used to trigger attacks, such as the need to place pressure on the opponent.

The attack timing competency is deliberative and reactive. Timing attacks can be planned offline and incorporated into the strategy selection process. The online task of determining when to attack based on reactive conditions is complex, because a player has only partial observability.

Competencies in EISBot

EISBot incorporates several AI techniques for implementing the previously described gameplay competencies. We are using a variety of approaches for implementing these tasks and utilizing multiple sources of domain knowledge.

Micromanagement

Micromanagement is a complex task, because it requires precise timing and several unit-specific techniques. We have implemented micromanagement in EISBot by hand authoring several ABL behaviors which are specific to individual unit types. These additional behaviors augment the tactics manager's general attack behaviors with specialized combat behaviors. Implementation of these behaviors is discussed in previous work (Weber et al. 2010).

Hand authoring micromanagement behaviors requires a substantial amount of domain engineering. We selected this approach because creating emergent or explorative techniques capable of the necessary level of sophistication for micromanagement is an open challenge. To reduce this engineering effort, we focused on implementing micromanagement techniques that have been identified by the StarCraft gaming community[2].

[2]http://wiki.teamliquid.net/

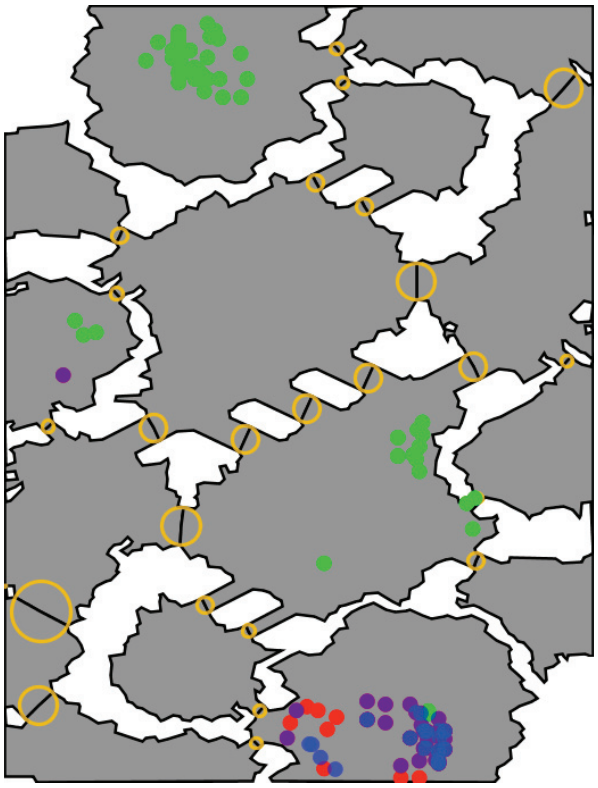

Figure 3: EISBot's perception of the game includes unit locations (green), estimations of opponent locations (blue), regions (gray), and chokepoints (orange). The locations of enemy units (red) are not observable by the agent.

Terrain Analysis

Terrain analysis is critical for effective tactical gameplay. Our system uses Perkin's (2010) terrain analysis library, which identifies qualitative spatial features in StarCraft maps including regions, chokepoints, and expansion locations. The features recognized by the library are added to EISBot's working memory, augmenting that agent's knowledge of the terrain. A portion of the agent's perception of the environment in an example scenario is shown in Figure 3.

The qualitative features identified by Perkin's library are used to specify activation conditions for several behaviors: the reconnaissance manager uses the list of expansion locations to select target locations for scouting opponents, the tactics manager uses the list of chokepoints to determine where to place forces for defending bases, and the construction manager uses the list of regions to determining where to place structures. The list of chokepoints is also used to improve the agent's estimation of the game state. EISBot uses a particle model to track the location of an encountered unit based on its trajectory and nearby chokepoints (Weber, Mateas, and Jhala 2011).

Strategy Selection

We have implemented strategy selection in EISBot using a combination of reactive and deliberative techniques. During the initial phase of the game, the agent selects and executes a build order, which is a sequential list of actions for pursuing a strategy. The agent reacts to unanticipated situations by incorporating a model for goal-driven autonomy (Molineaux, Klenk, and Aha 2010).

The strategy manager includes a collection of hand-authored behaviors for executing specific build orders. We are using goal-driven autonomy to decouple strategy selection from strategy execution in EISBot. In our system, each build order contains a list of expectations which must remain true throughout the execution of the strategy. If an expectation becomes violated, then a discrepancy is created and the agent selects a new strategy to pursue. This approach enables the agent to react to unforeseen game situations (Weber, Mateas, and Jhala 2010a).

After executing an initial strategy, the agent transitions to new strategies using a case-based planner. The case-based planner is used for both strategy selection and opponent modeling (Weber, Mateas, and Jhala 2010b). It selects actions to execute by retrieving the most similar situation from a library of examples, which are extracted from a corpus of expert replays. Plans include actions for producing units, building structures, and researching upgrades. The case-based planner interfaces with the reactive planner by adding sequences of actions to perform to working memory.

Attack Timing

There are several ways in which attacks are triggered in EISBot. A build order selected by the strategy selector can contain a timing attack annotation that specifies when to first assault the opponent. The strategy manager triggers a timing attack by adding an element to working memory, which is used in precondition checks for attack behaviors. Attacks can also be triggered based on reaching a specific army size. A collection of hand-authored behaviors are included in the agent to ensure that pressure is frequently placed on the opponent.

Evaluation

To determine the performance of EISBot, we have carried out an initial evaluation against human opponents on the International Cyber Cup (ICCup). ICCup is a competitive StarCraft tournament ladder with over ten thousand active StarCraft players. The skill of players on the ladder ranges from novice to professional, where the majority of players are amateur players with a strong understanding of the game.

ICCup assigns players a letter grade based on a point system, which is similar to the Elo rating system used in chess. Players start with a provisional 1000 points which is increased after wins and decreased after loses. Novices, or D- ranked players, quickly fall below the 1000 point barrier after a few matches, while amateurs, or D ranked players maintain a score in the range of 1000 to 2000 points. Scores of 2000 points and higher are achieved by highly competitive and expert players.

We have classified the performance of EISBot by running 250 games against human opponents on ICCup. The opponents had an average score of 1205 with a standard deviation

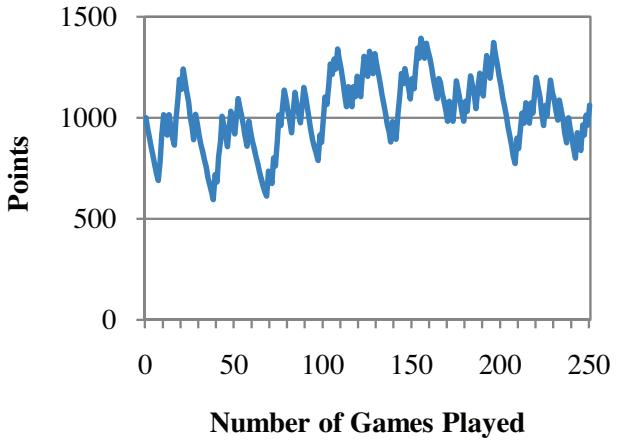

Figure 4: The point progression of EISBot on ICCup indicates that the bot is a D ranked (amateur) player.

of 1660. EISBot had an average win rate of 32% against human opponents and achieved a score of 1063 points, which outranks 33% of players on the ladder. The point progression of EISBot is shown in Figure 4. While there were several instances where EISBot's score dropped below 1000 points, the agent achieved an average score of 1027 points. Since EISBot was able to reach an average score above 1000 points after a large number of games, it is classified as a D ranked, or amateur StarCraft player.

Since we are working towards the goal of achieving human-level competency, our evaluation of EISBot is focused on performance against human opponents. In previous work, we report a win rate of 78% against the built-in AI of StarCraft and bots from the AIIDE 2010 StarCraft AI competition (Weber, Mateas, and Jhala 2011). To our knowledge, EISBot is the only system that has been evaluated against humans on ICCup.

Conclusion

We have presented an approach for building a human-level agent for StarCraft. We analyzed the domain and identified several of the competencies required for expert StarCraft gameplay. Our analysis motivated the need for a heterogeneous agent architecture, because gameplay requires a combination of reactive, deliberative, and predictive reasoning processes, which are performed both offline and online by players. We are developing an agent that interacts with RTS games at the same granularity as humans and exploring approaches for emulating these processes in an agent.

Our agent architecture is further motivated by the properties of the task environment. In addition to being real-time and partially observable environments with enormous decision complexities, RTS games provide a multi-scale AI challenge. Gameplay requires concurrent and coordinated goal pursuit across multiple abstractions. Additionally, RTS games are non-hierarchical and difficult to decompose into isolated subproblems due to cross cutting concerns between different aspect of gameplay.

EISBot builds on previous work, which conceptually partitioned RTS gameplay into distinct subproblems (McCoy and Mateas 2008). Additionally, we identified several competencies necessary for expert gameplay. These competencies include micromanagement of units in combat, terrain analysis for effective unit positioning, strategy selection for countering opponent strategies, and attack timing for identifying situations to assault the opponent.

The core of the agent is the ABL reactive planner which manages the active goals, interfaces with the game environment, and monitors the execution of actions. This component was implemented by hand-authoring a large number of behaviors, requiring a substantial amount of domain engineering. To reduce this effort, we utilized rules-of-thumb and specific techniques which have been identified by the StarCraft gaming community.

EISBot integrates with several heterogeneous components using multiple integration approaches. These include adding elements to working memory, formulating goals to pursue, generating plans to execute, and providing additional conditions for behavior activation. We have used these approaches to integrate the reactive planner with case-based reasoning, goal-driven autonomy, and machine learning components.

We have performed an initial evaluation of EISBot against human opponents on the ICCUP tournament ladder. The agent achieved an average win rate of 32% and outranked 33% of players on the ladder. While there is still a large gap between the performance of our system and expert players, EISBot achieved a D ranking which indicates that it's ability is above the novice level and on par with amateur competitive players.

Future Work

There are several directions for future work. The agent can be extended by identifying additional competencies in RTS games. These competencies could be implemented using reactive planning behaviors or an external component. Also, additional external components could be integrated into the agent for existing competencies. For example, Monte Carlo tree search could be used to implement the micromanagement competency (Balla and Fern 2009), and would require less knowledge engineering than our current approach.

Additional forms of evaluation could also be performed for EISBot. In our evaluation, EISBot played against human opponents in ladder matches, which consist of a single game. This form of analysis is insufficient for evaluating the adaptivity of the agent, because each game is a potentially different opponent. Another way to evaluate agents is in a tournament setting, where matches consists of multiple games against the same opponent. Performing well in this setting requires adapting to the opponent based on the outcome of previous games. Further evaluation could include ablation studies as well, to gain further understanding of the agent's performance against humans.

Another direction for future work is to further pursue the goal of interfacing with the game at the same level as a human player. In the current system, the agent's perception system is able to determine the locations and trajectories of

units by directly querying the game state. Future work could explore the use of computer vision techniques for extracting game state using screen captures (Lucas 2007). Currently, there is no restriction on the number of actions the agent can perform. To further mimic human actuation, future work could evaluate how limiting the number of allowed actions per minute impacts agent performance.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Number IIS-1018954. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Aha, D. W.; Molineaux, M.; and Ponsen, M. 2005. Learning to Win: Case-Based Plan Selection in a Real-Time Strategy Game. In *Proceedings of ICCBR*, 5–20. Springer.

Bakkes, S.; Spronck, P.; and van den Herik, J. 2008. Rapid Adaptation of Video Game AI. In *Proceedings of IEEE Symposium on Computational Intelligence and Games*.

Balla, R. K., and Fern, A. 2009. UCT for Tactical Assault Planning in Real-Time Strategy Games. In *Proceedings of IJCAI*, 40–45. Morgan Kaufmann.

Baumgarten, R.; Colton, S.; and Morris, M. 2009. Combining AI Methods for Learning Bots in a Real-Time Strategy Game. *International Journal on Computer Game Technologies*.

Buro, M. 2003. Real-Time Strategy Games: A New AI Research Challenge. In *Proceedings of IJCAI*, 1534–1535.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2:189–208.

Forbus, K.; Mahoney, J.; and Dill, K. 2002. How Qualitative Spatial Reasoning Can Improve Strategy Game AIs. *IEEE Intelligent Systems* 17(4):25–30.

Hayes-Roth, B. 1985. A Blackboard Architecture for Control. *Artificial intelligence* 26(3):251–321.

Isla, D. 2005. Handling Complexity in the Halo 2 AI. In *Proceedings of the Game Developers Conference*.

Josyula, D. 2005. *A Unified Theory of Acting and Agency for a Universal Interfacing Agent*. Ph.D. Dissertation, University of Maryland.

Laird, J., and VanLent, M. 2001. Human-Level AI's Killer Application: Interactive Computer Games. *AI magazine* 22(2):15–25.

Langley, P., and Choi, D. 2006. A Unified Cognitive Architecture for Physical Agents. In *Proceedings of AAAI*, 1469–1474. AAAI Press.

Lucas, S. 2007. Ms Pac-Man Competition. *SIGEVOlution* 2:37–38.

Mateas, M., and Stern, A. 2002. A Behavior Language for Story-Based Believable Agents. *IEEE Intelligent Systems* 17(4):39–47.

McCoy, J., and Mateas, M. 2008. An Integrated Agent for Playing Real-Time Strategy Games. In *Proceedings of AAAI*, 1313–1318. AAAI Press.

Molineaux, M.; Klenk, M.; and Aha, D. W. 2010. Goal-Driven Autonomy in a Navy Strategy Simulation. In *Proceedings of AAAI*, 1548–1553.

Ontañón, S.; Mishra, K.; Sugandh, N.; and Ram, A. 2010. On-Line Case-Based Planning. *Computational Intelligence* 26(1):84–119.

Orkin, J. 2003. Applying Goal-Oriented Action Planning to Games. In Rabin, S., ed., *AI Game Programming Wisdom 2*. Charles River Media. 217–228.

Orkin, J. 2006. Three States and a Plan: The AI of FEAR. In *Proceedings of the Game Developers Conference*.

Palma, R.; González-Calero, P. A.; Gómez-Martın, M. A.; and Gómez-Martın, P. P. 2011. Extending Case-Based Planning with Behavior Trees. In *Proceedings of FLAIRS*, 407–412.

Perkins, L. 2010. Terrain Analysis in Real-Time Strategy Games: Choke Point Detection and Region Decomposition. In *Proceedings of AIIDE*, 168–173. AAAI Press.

Ponsen, M.; Lee-Urban, S.; Muñoz-Avila, H.; Aha, D. W.; and Molineaux, M. 2005. Stratagus: An Open-Source Game Engine for Research in Real-Time Strategy Games. In *IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*.

Rao, A., and Georgeff, M. 1995. BDI Agents: From Theory to Practice. In *Proceedings of ICMAS*, 312–319.

Simpkins, C.; Bhat, S.; Isbell Jr, C.; and Mateas, M. 2008. Towards Adaptive Programming: Integrating Reinforcement Learning into a Programming Language. In *Proceedings of OOPSLA*, 603–614. ACM.

Weber, B. G., and Ontañón, S. 2010. Using Automated Replay Annotation for Case-Based Planning in Games. In *Proceedings of the ICCBR Workshop on Computer Games*.

Weber, B. G.; Mawhorter, P.; Mateas, M.; and Jhala, A. 2010. Reactive Planning Idioms for Multi-Scale Game AI. In *Proceedings of IEEE CIG*, 115–122. IEEE Press.

Weber, B. G.; Mateas, M.; and Jhala, A. 2010a. Applying Goal-Driven Autonomy to StarCraft. In *Proceedings of AIIDE*, 101–106.

Weber, B. G.; Mateas, M.; and Jhala, A. 2010b. Case-Based Goal Formulation. In *Proceedings of the AAAI Workshop on Goal-Driven Autonomy*.

Weber, B. G.; Mateas, M.; and Jhala, A. 2011. A Particle Model for State Estimation in Real-Time Strategy Games. In *Proceedings of AIIDE (To Appear)*.

Wintermute, S.; Xu, J.; and Laird, J. 2007. SORTS: A Human-Level Approach to Real-Time Strategy AI. In *Proceedings of AIIDE*, 55–60. AAAI Press.