9th International Conference on Software, Knowledge, Information Management and Applications
(SKIMA), 15-17 December, 2015, Kathmandu, Nepal.

# A Bidding System For StarCraft II Planning Agents

***Abstract*—We propose an auction system as a way to achieve consensus between agents covering the high-level planning tasks of the Real-Time Strategy game StarCraft II. We will specify the problem domain and how we mapped the task of planning in StarCraft II on a multi-agent system. From here, we will pin down how user tasks are represented and how the agents receive information of the game world, including the introduction of a blackboard to synchronize information on dynamic events, such as attacks. Finally, the auction system takes care of distributing tasks originating from a global utility system to the different agents, where each agent can value its capability to fulfill the goals and side-constraints of a task on its own.**

***Keywords—RTS, StarCraft, StarCraft II, Reasoning, Bidding Systems, Utility Systems***

## I. Introduction

The Real-Time Strategy (RTS) game StarCraft is a popular research platform [1]. RTS games offer an array of interesting challenges for developing player-level Artificial Intelligence (AI), such as tactical and strategic reasoning, opponent modelling and planning in uncertain, dynamic and adversarial environments [2]. Many of these challenges are, in particular high-level planning, are still active research topics [3]. Our research focuses on StarCrafts successor, StarCraft II. While different in some details, the game play and game mechanics are comparable and research done based on StarCraft II should transfer to StarCraft and potentially other, similar RTS games.

We present work on a non-cheating StarCraft II player-level AI, called Eunoia. The bot is a multi-agent system and solves the reasoning tasks via a loosely coupled array of Command Agents, which each supervise a group of Profession Agents taking care of the actual unit control. This paper will present how an Auction System - which bears similarities to Utility Systems - is put into use to distribute tasks issued by Command Agents to their respective Profession Agents. We will further explain how a blackboard-like approach is used to gather information from the StarCraft II game and distill it into AI world knowledge. We will also take a look on the data format and constraints that accompany tasks in such a player-level AI.

## II. Problem Domain

The state space in StarCraft II is huge and cannot be broken down into exact atomic tasks like in board games such as chess. Sensory information in the game can give a bot a perfect information scenario but the innumerable actions make it unfeasible trying to check all possibilities. As our bot implementation targets developing a non-cheating bot for StarCraft II, one has to reduce the number of sensors to a human-like minimum. This naturally means taking the rules of the game, such as Fog of War, into account. Yet, further, human awareness and retentiveness should be imitated. For these reasons the world state representation is a mix between different established techniques. StarCraft II provides an easy to access event system and its data module allows for retrieval of almost all unit-specific values. Responsibility is to choose the right information and store it purposefully.

The prototype described in this paper only covers planning for all economical functions, such as expanding, and the first military production building of the Terran race, the Barracks. The architecture can be expanded to cover the other two buildings, Factory and Starport, but for the sake of simplicity, we opted for a simpler prototype. The Barracks, with its attachments and research buildings, already offer a selection of different units with very different game mechanics.

## III. Tasks

Tasks are the smallest building blocks of behaviors and need to be based on the facts in the memory system. Tasks for the system are mostly low-level with only few abstract commands. Abstract commands have to be concretized to get atomic tasks for the agent to execute. This may result in many tasks executed in superhumanly speeds. Another reason to choose atomic tasks is the need to foresee their impact to the game state. The more complex a task the harder it is to discern the influence. The following tasks were chosen. They are sufficient for a simple agent playing a whole game. Additional tasks can easily be added, due to the loose system architecture.

- Train SCV
- Train Marine
- Train Marauder
- Train Reaper
- Build Supply Depot
- Build Barracks
- Build Engineering Bay
- Expand
- Defend
- Attack

## IV. World-State Representation

In order to make decisions an agent needs to be able to capture the current state of the world around them. A player needs to react an attacks or issue attacks themselves depending on military strength. For Eunoia several methods for sensing the world and storing information about the world are combined. The information is divided into static knowledge, which does not change once the game has started and dynamic knowledge, variable bits of information changing as the game progresses.

Most of the static knowledge is already stored in StarCraft II. Player locations, map topography such as nav-meshes and possible expansion points are stored in the map files. Unit data and other domain knowledge is stored in the unit data files and is easily accessible through interface functions. Remaining static knowledge is simply stored in global variables once the game has started. Those being for example races and player start locations.

For sensing dynamic knowledge sensors to capture in-game events and extract relevant data are utilized. Besides sensors a polling system, which gathers information at each AI loop cycle is established. This way the dynamic knowledge can periodically be updated and a consistent world view to base decisions on is present. Following are some of the dynamic information bits being captured.

- Current resource count
- Allied unit count of specific unit type
- Allied expansions count
- Enemy unit count of encountered unit types
- Enemy base count

Most dynamic knowledge is stored in a simple database with predefined keys for easy access. For arbitrary data and occasionally occurring events a blackboard architecture is used.

### A. *Sensors*

So-called Triggers from the Trigger Editor. These sensors react to predefined events from the game by executing certain actions. Sensors expose information which is associated with the event through variables. Due to this sensors can be reused for multiple purposes. Once a sensor is triggered the associated information for preconditions can be checked and action-sequences executed accordingly. Some in-game events used are:

- Unit training progress started / completed to assign units to attack waves.
- Unit construction progress Completed for agent spawning.
- Unit takes damage to react to attacks and defend accordingly.
- Unit dies to react to structure destruction.

## V. Data Considerations

Comparing and contrasting concrete values can be difficult. Some variables in StarCraft II have no predefined maximum value and variables may rely on one another. Dave Mark wrote about such issues in his book “Behavioral mathematics for game AI” [4]. He writes about issues stemming from using measurable factors to decide on rational and irrational behavior.

Gathering the data in a computer game is usually very simple and one of the main reasons researchers choose games for AI research. Most world state variables are linked to concrete values, which can be accessed. But numerical, concrete values are not always useful for decision-making. The perception of a value may change throughout the game. It is, for example, difficult to decide whether a player has maximum minerals or the strongest military force. These considerations are depending on in-game factors like game-time, or how strong the military force of an opponent is.

For these reasons artificial limits on many of the variables had to be imposed and relative weights for related values had to be given. Eunoia features two parallel world views to handle most of the calculations. On the one hand there is a simple numeric world state, where every value is stored as retrieved. No capping and no weights are applied there.

On the other hand there is a normalized and capped world view (see fig. 1). All values are normalized between 0 and 1. Variables with discretionary or infinite maximum values are capped at values founded on StarCraft II domain knowledge or expert knowledge. Both views are used for either calculating utility of tasks or issuing bids.

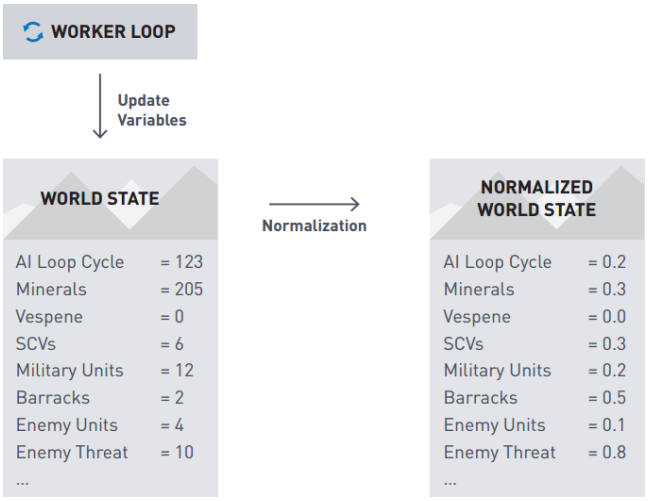


Figure 1: Worker Loop gathering information from Sensors, WorldState and Normalized WorldState Variables.

## VI. Blackboard Architecture

Aside from regularly updated dynamic knowledge and constant information in the form of static knowledge sometimes random information has to be stored. Blackboard Architectures [5, 6] allow several individual and separate systems (called experts) to access information in a centralized space. Experts can post new data onto the blackboard, simply read or even erase data from the blackboard. Usually an arbiter is introduced to avoid access issues. It controls which expert is allowed to access the board based on their insistence, a value displaying how important it is for an expert to have access. The arbiter could be left out because only a few systems access the information on the board and neither of them can get into conflicts. Sensors write information onto the board and agents read and remove this information once they are permitted to execute a task. Amplifying the classical architecture a basic garbage collection mechanism is introduced. All entries on the blackboard have a time-to-live (ttl) counter. Every loop this counter will be decreased for all entries. Once an entry's ttl-counter reaches zero, the entry will be removed. Special entries can have an infinite time-to-life and will stay on the board until they are treated with or the game ends, whichever comes first.

If, for example, a sensor registers an attack at the base a new blackboard entry for said attack will be created. The entry will consist of the location of the attacked unit, the number of attacking units and the number of loops this information should be valid. If then a defend-task has the highest utility an agent will read the information and use it to execute the task.

Cutting down unnecessary features and having decided on the world view of the agent leads to a controllable environment. With keeping all vital features, the results can later be projected onto a bigger scope if necessary. The next step is to translate the goals and first findings into a functioning agent architecture.

## VII. Auction System and Task Treatment

Tasks function as atomic actions to be executed by their respective agents. Whenever a task is created by the utility system, it has to be delegated to an appropriate agent if possible. The system adapts in size and shape to the current game state. This is why a delegation system, which works with this level of flexibility and versatility, is needed. Kolp et al. [7, 8] researched common Multi Agent Systems and scored the different approaches against different requirements. According to their paper, bidding-architectures sufficiently inherit the quality of adaptivity and modularity. They involve competitivity with participating agents bidding for items. An auctioneer creates and organizes auctions. It displays the item to be auctioned and accepts bids from all agents. The auctioneer is then responsible for closing the auction and awarding the item to the winning agent.

This architecture enables connecting the two hierarchical levels loosely and issue task delegation easily (see fig. 2). Bidding systems work regardless of the number of agents. The more agents the higher the chance of finding a perfectly fitting agent and gaining maximum utility. On the other hand items are only awarded if any bid is issued at all. This way lower level agents are able to veto on items if they seem inappropriate or unworthy. If that happens the next task in the queue of high utility tasks is fetched.

Bidding architectures can be interpreted as reversed utility systems. Instead of scoring available actions and assigning the winning action to an agent, one item is scored by different agents and only executed if profitable. No centralized logic is doing the decision-making but rather small agents assess their local environment and internal state and bid accordingly.

Kolp et al. [8] mention only one additional architecture which performs adaptive and rather modular. Joint venture style functions by delegating authority to a joint management actor which coordinates tasks and resources. Each principal partner can manage itself locally and interact with other partners directly. However, strategic operations and decisions on a global scale are carried out by the joint management actor. For this system it would mean that task delegation would be decided by this centralized actor. This actor had to decide on an abstract worldview with heavy organizational overhead if the system changes. The connection between local actor and the centralized joint management actor is inflexible and would limit the organic structure. This is why the auction system is used.

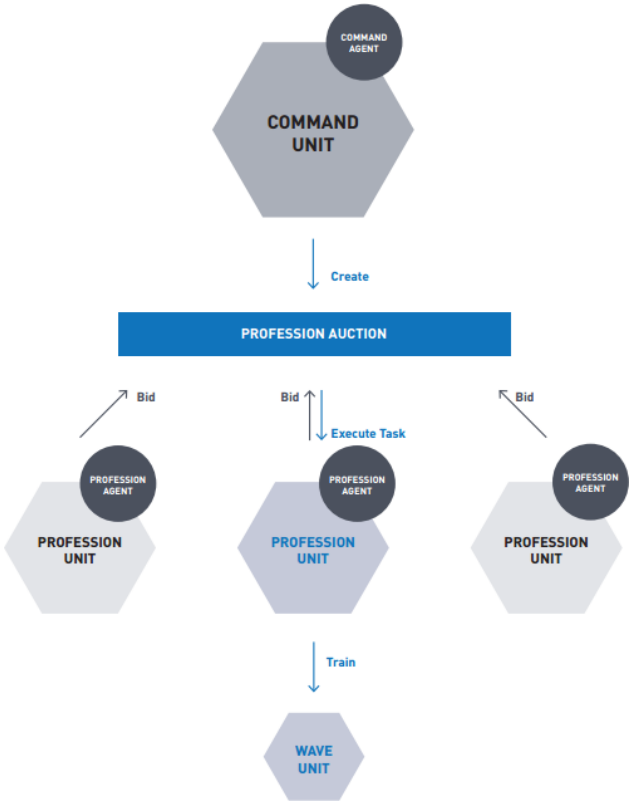


Figure 2: Profession Agent bidding in Eunoia. A Command Agent issues a Profession Auction - for example building a new unit - and each Profession Agent issues a bid, depending on static and dynamic knowledge of the game (enemy start location, current build queue, distance to base …)

### A. *Auctions*

Bidding architecture in Eunoia are used to distribute tasks to command and profession agents. The flow of the procedure is depicted in fig. 3. Command and profession agents share no tasks, so two different auctions are held. On command auctions, only command agents bid and there is no clear auctioneer. The auction is started by a third party auctioneer which is spawned especially for that case and handles the bidding and task delegation. After the auction the auctioneer is destroyed. For profession auctions the parenting command agent functions as the auctioneer and only descending profession agents are allowed to bid.

Auction bidding is where the logic component in Eunoia is located. It is the part where most of the reasoning is happening and where the biggest potential for future projects lays. The greatest benefit is that while bidding the agents only need to reason about their immediate surrounding and internal state. No strong level of abstraction is needed as much of the world view is unnecessary due to the limited reach of the agent calculating the bid. Knowledge for the bot is still centralized and any agent can access any information of the world state if necessary. Otherwise the low level agents would not be able to come to rational results when compared to centralized systems.

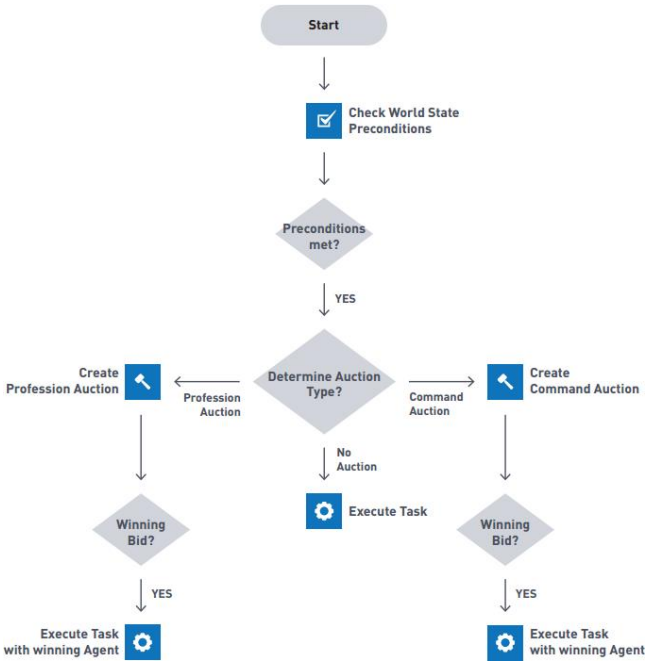


Figure 3: Task Treatment in Eunoia.

For Eunoia the bidding logic is a two-step process. An agent evaluates its agent preconditions, which reflect the internal state. This could for example be the existence of a special building-attachment, which accelerates training of units. Some of those internal preconditions can prevent any bid at all. For example, if the agent is not able to execute the task at all, because it is lacking a special building-attachment. Preconditions are stored as numerical variables attached to each agent, comparable to symbols from Jeff Orkin's GOAP approach [9]. Agent preconditions are as followed:

- Queue Spots $\epsilon$ N (0-5), stores the number of available training spots.
- Energy $\epsilon$ N (0-200), is a special resource for Command Centers and other casters.
- Is Orbital Command $\epsilon$ {0,1}, translates into a boolean value for a Command Center attachment.
- Number of Profession Units $\epsilon$ N, stores the number of profession units a command center commands.
- Can Train $\epsilon$ {0,1}, translates into a boolean value to distinguish training from research-facilities.
- Can Research $\epsilon$ {0,1}, translates into a boolean value to distinguish training from research-facilities.
- Has Reactor $\epsilon$ {0,1}, translates into a boolean value for an attachment allowing simultaneous training of two units for production buildings.
- Has Techlab $\epsilon$ {0,1}, translates into a boolean value for an attachment allowing training of special units and research of unit enhancements for production buildings.

After preconditions checks the agent calculates a bid for the task at hand. For each task a particular formula is utilized. These pre-baked formulas are comparable to utility functions and inherit the same dynamic. Every agent uses the same formulas to calculate the bid. This modular system results in agent bids only differentiating due to internal symbol or state differences. Spatial and temporal reasoning is done based on these differences. Agents could plan ahead and disregard preconditions because they would be true in a foreseeable future. For example the building attachment, needed to train a certain unit, is already being constructed. After the auctioneer has gathered all bids the winning agent will be commanded to execute the task.

### B. *Task Execution*

Winning agents should execute the task immediately after the auction. While it is technically and conceptually possible to withhold execution for internal planning, this would alter the current world state and change future decision-making. Withholding the execution could cause the same task being picked for execution next, because the bot decides that the problem at hand has not been dealt with. In Eunoia all tasks are executed immediately upon delegation (see fig. 4).

Agents will gather all information necessary to execute the tasks via the centralized world state database, or from

information bits stored on the blackboard. Defend tasks, for example, need a position to send the units to. Once a message has been retrieved from the blackboard, it has to be invalidated there to prevent multiple defense-actions from using the same information. World state variable do not need to be changed, as they will be refreshed continuously and before the next task-execution.

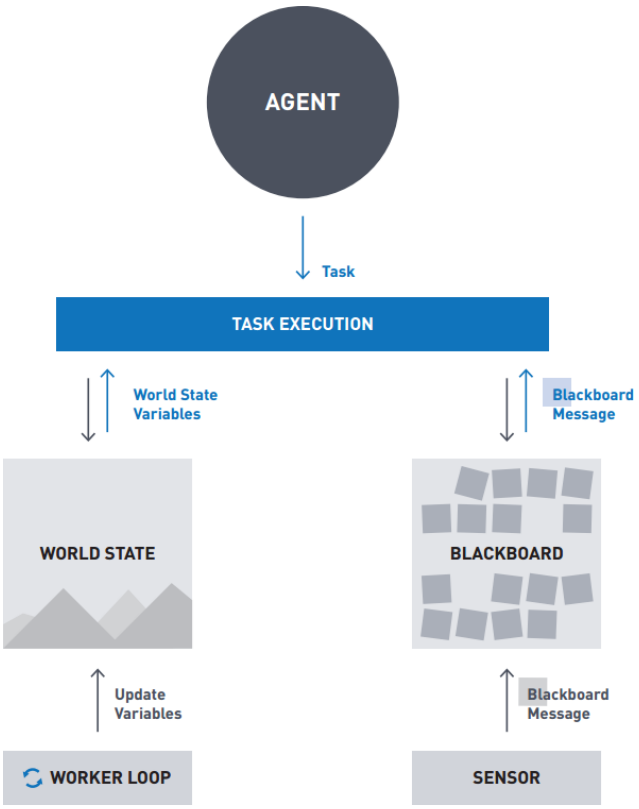


Figure 4: Task Execution in Eunoia.

## VIII. Conclusion

We presented how Command Agents and Profession Agents work together via an Auction System to distribute tasks among them. To execute tasks, world knowledge and agent communication had to be solved, which was done via an abstract world-space representation achieved through a sensory layer and a blackboard to store information on dynamic events such as attacks. We explained how all these systems interact to form a player-level reasoning system for StarCraft II.

Further evaluation will focus on two central aspects of player-level AIs in RTS games: Effectivity and believability. Effectivity is measured by a current test series which fields Eunoia against the AI routines with which StarCraft II is shipped. The study is currently underway, but preliminary results indicate that Eunoia is capable to beat even more difficult AI opponents (difficulty level 5/7 shows good win rates for Eunoia).

Yet, playing a game effectively does not necessarily mean that the AI plays in a natural, human-like way. Further evaluation will therefore target believability, in the form of pseudo-Turing tests, where human test participants encounter an opponent, of which they don't know if it's Eunoia or a human player. After a game round, they will state whether they believe to have faced an AI or a human.

## References

[1] Ontanón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., & Preuss, M. (2013). A survey of real-time strategy game ai research and competition in starcraft. Computational Intelligence and AI in Games, IEEE Transactions on, 5(4), 293-311.

[2] Weber, B. G., Mateas, M., & Jhala, A. (2011, November). Building Human-Level AI for Real-Time Strategy Games. In AAAI Fall Symposium: Advances in Cognitive Systems (Vol. 11, p. 01).

[3] Yannakakis, G. N. (2012, May). Game AI revisited. In Proceedings of the 9th conference on Computing Frontiers (pp. 285-292). ACM.

[4] Mark, D. (2009). Behavioral mathematics for game AI. Boston, MA; Singapore, Course Technology Cengage Learning.

[5] Millington, I., & Funge, J. (2012). Artificial intelligence for games. CRC Press.

[6] Downie, R. B. D. I. M., & Blumberg, Y. I. B. (2001). Creature smarts: The art and architecture of a virtual brain.

[7] Kolp, M., Castro, J., & Mylopoulos, J. (2001, May). A social organization perspective on software architectures. In Proc. of the 1st Int. Workshop From Software Requirements to Architectures. STRAW (Vol. 1, pp. 5-12).

[8] Carbonell J. G. et al. (2002), eds.: Revised papers: Seattle, WA, USA, August 1 - 3, 2001. Springer. Page 128 ff.

[9] Orkin, Jeff. "Symbolic representation of game world state: Toward real-time planning in games." Proceedings of the AAAI Workshop on Challenges in Game Artificial Intelligence. Vol. 5. 2004.