

Artifical intelligence in Real-Time strategy games

By Kerezsi Alex



Contents

[1. Introduction 2](#_Toc452474091)

[1.1 Real-Time Strategy Games 2](#_Toc452474092)

[1.2 Motivation 4](#_Toc452474093)

[1.3 RTS AI Competitions 4](#_Toc452474094)

[2. Application Programming Interface (BWAPI) 6](#_Toc452474095)

[2.1 API Capabilities 6](#_Toc452474096)

[3. RTS Sub-Problems 7](#_Toc452474097)

[3.1 Strategy 7](#_Toc452474098)

[3.1.1 Learning and data collecting 8](#_Toc452474099)

[3.1.2 Enemy modeling 8](#_Toc452474100)

[3.1.3 Strategic stance 9](#_Toc452474101)

[3.1.4 Army composition 9](#_Toc452474102)

[3.1.5 Build order 10](#_Toc452474103)

[3.2 Tactics 10](#_Toc452474104)

[3.1.6 Scouting 11](#_Toc452474105)

[3.1.7 Combat timing 12](#_Toc452474106)

[3.1.8 Building placement 12](#_Toc452474107)

[3.2 Reactive control 13](#_Toc452474108)

[3.2.1 Unit micro 14](#_Toc452474109)

[3.2.2 Multi-agent pathfinding 14](#_Toc452474110)

[4. KBOT 17](#_Toc452474111)

# Introduction

Introduced in the early 1990, real-time strategy games have recently become popular for artificial intelligence research. In the past couple of years significant advancements have been made in RTS games with contribution from many fields within computer science and engineering. Motivation for RTS AI has also grown rapidly with the emergence of competition such as Google AI Challenge, StarCraft AI Competitions (organized by SSCAI and AIIDE).

## Real-Time Strategy Games

Real-Time strategy games can be classified as strategic video games which simulate military warfare on various scales. Players assume the role of a military commander in charge of a group of forces which must build an economy, construct bases (buildings and defenses), and create a combat army (train units and research new technologies/abilities in order to increase the capabilities for their units) and defeat the enemies by taking control of their command centers or destroying their armies and bases. RTS games vary in size and complexity. However they share some common traits from the traditional strategy games:

**Real-Time**: RTS games are played in real-time, meaning that each player can perform several actions as fast as the game is executed (an average of 30 frames per second), and the game progresses normally even if no actions are given. This is unlike traditional games like ***chess*** or ***go***where players have several minutes to perform an action, and the game cannot progress until a player has acted or the turn’s time has reached the permitted limit. For example StarCraft runs on 24 frames/second, in other words, a player can perform an action once every 42 milliseconds.

**Simultaneous Moves**: Arises when players have to make their strategy choices simultaneously, without knowing the strategies that have been chosen by the other player(s). Additionally, these actions may be durative, in other words, they require some time to complete.

**Imperfect information:** Players in RTS cannot see their opponent’s units and actions unless they are actively scouting them. Typically, a map is initially covered by a fog-of-war (a layer covering the maps locations that are not explored or the player has no vision over it).

**Non-Determinism**: Some RTS games have non-determinism in their actions. This characteristic refers to a random event occurring while performing a constant action. For example, in StarCraft, units that are positioned on different heights, have a small change of missing their target.

**Multi-Unit Control**: RTS games allow users to control multiple units at the same time, with each able to be given individual actions. This means that at any given state there may be an exponential number of possible actions with respect to the number of units the player controls.

**Complexity**: The complexity of RTS games is much higher than traditional games, in terms of state space size, the number of actions can be performed at any time step, and the number of actions required to reach the end of a game. For example the number of possible states in ***chess*** is approximately 1050, Go has around 10170, while StarCraft has shown to have at least 101000 as a lower bound [1].

## Motivation

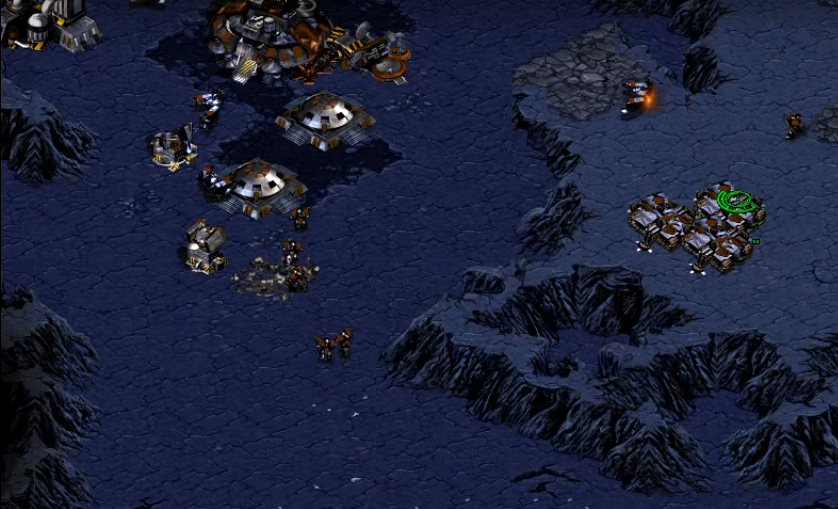
When **StarCraft** was released in 1998, it captured the video game world in a way never seen before, with millions of players playing competitively over LAN and on Blizzard’s battle.net servers. The game has been sold in millions of copies and it became one of the most popular games in the world. In South Korea, so many peoples were playing StarCraft that the Ministry of Culture, Sports and Tourism formed KeSPA, the Korean e-Sports association to manage and promote the professional play of the game in their country. StarCraft has been played professionally in Korea and around the world ever since, with millions of dollars in prize money being awarded annually. With such an established industry and competitive RTS gaming we can motivate to develop better agents that can play against players not only for competitive purposes but also to enhance the player’s multi-tasking abilities.

**Creating better AI agents:** Recent advances in were made in the gaming industry for traditional games. AI agents have been created that are capable of defeating human world champions at several games like: *Chess*, *Go*, *Texas* *Holdem* poker. [5] Competitions, whether it is Human vs. Human, Human vs. AI or AI vs. AI, has always been a motivating factor to research on because it combines elements from mathematics, multi-unit control and prediction.

In my opinion, artificial intelligence will provide a key aspect to our future not only in the gaming industry, but also in other domains such as: medicine, biology, chemistry, image processing and many other domains we can think of.

## RTS AI Competitions

Recently AI competitions became popular due to their challenging properties, with the goal of beating a professional human player at popular RTS games such as StarCraft. These competitions had several categories focusing on sub-problems in RTS games such as providing a good build order and resource management in order to survive attacks and avoid resource exhaustion [2].



# Application Programming Interface (BWAPI)

The Brood War Application Programming Interface is a free and open source C++/Java framework that is used to interact with the popular real-time strategy game **StarCraft: BroodWar**. Using BWAPI, students, researchers and hobbyist can create Artificial Intelligence agents that can play the game without any human interaction [3].

BWAPI reveals the visible parts of the game state to AI modules by default. Information on units that have gone back into the fog of war is denied to the AI. This enables programmers to write competitive non-cheating AI’s that must plan and operate under partial information conditions. BWAPI also denies user input by default, ensuring the user cannot take control of the game units while the AI is playing. These defaults can be changed for flexibility unless enforced by the Tournament Module.

## API Capabilities

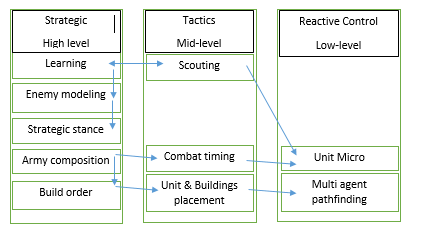
* Write competitive AI’s for StarCraft by controlling individual units.
* Read all relevant aspects of the game state.
* Analyze replays frame-by-frame, and extract trends, build orders, and common strategies.
* Get comprehensive information on the unit types, upgrades, technologies, weapons and more.
* Study and research real-time AI algorithms in a robust commercial RTS environment

# RTS Sub-Problems

Real time strategy games are incredibly complex, even for professional player. In order to manage the games complexity we break it into sub-problems. The sub-problems are not necessarily independent. This approach requires a level of abstraction in order to be trackable by humans. Researchers have adapted the divide and conquer strategy in order to treat each sub-problem and reduce the overall complexity.

We can categorize the RTS sub-problems based on the time scale that we deal with and the level of abstraction the problem requires in order be approachable. These categories are **Strategy**, **Tactics** and **Reactive Control**.

These categories mimic a military command hierarchy, both in terms of command as well as information processing.



## Strategy

At the highest level of abstraction comes the strategy and corresponds to the decision making of the agent/player in an RTS game. Each decision is crucial because it can influence the games odds in the favor of the player that takes appropriate decisions in appropriate times. In order to get a better overview of the applied strategy in RTS games, I’ve pointed out some of the main keys on which I will be focusing on:

* Learning and data collecting
* Enemy modeling
* Strategic stance
* Army composition
* Build order

## Learning and data collecting

In order to act, a player/agent needs to collect as much information as possible from the map and from the enemy players regarding their buildings, army and positioning. Examples of this game data collected can be rules, unit, properties actions etc. In RTS games, players who have more information about resource locations and enemies, have a significantly higher change to win the game by bringing the appropriate decisions for the states in which they are in.

In order to collect data and process it, a player usually sacrifices one or more of his units, sending them to explore and find where the enemy’s bases are located. It is important to know that that the more time with respect to the space explored is obtained, the higher changes are for the player to win.

## Enemy modeling

In the beginning, the players have no vision or information about their enemies, but as the game progresses, more and more information is obtained such as race, buildings, number of units or map positioning. A player must attempt to learn as much as possible to get a better predictions of what the next actions might be. An example of a proper enemy modeling method would be the hidden Markov model (HMM) in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states [4]. In simpler Markov models (like Markov chain), the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters. In a hidden Markov model, the state is not directly visible, but the output, dependent on the state, is visible. HMM’s are also known for their application in temporal pattern recognition such as speech handwriting, gesture recognition etc.

## Strategic stance

The player’s strategic stance determines the style of the play in an RTS game which corresponds to a balance between aggression and economic state. When a player has a large number of resources, his units production is at an exponential rate, he may adapt an offensive style whereas a player with low economy and small army might adapt a defensive stance in order to hold up to the enemy forces. There are various styles that can be adapted also depending on the race and positioning of the player.

## Army composition

Army composition is decided by the strategic stance, with special consideration to the predicted opponent’s army composition. Each unit type in an RTS game has its own unique properties such as attack style, attack range, movement speed, movement type, hit points and mana points. These complex interactions between units make an army composition difficult. Most of the time, the enemy’s stance determines the army’s composition.



## Build order

Once the army composition has been chosen, the army units must be built by the player using one of his workers to build structures that allow creating supply and units using the gathered resources. The sequence of actions taken to arrive at a given set of goal units is called build order. Build order can be described as a resource allocation problem which features concurrent actions. As important as it is for developing the strategy, build orders often get hardcoded due to their efficiency issues. An alternative for hardcoding would be to implement a priority based system where each build would have assigned a priority value. Based on this value, the agent would know what action follows the current one.

## 3.2 Tactics

One step down in the abstraction level, we can find the tactics category, which is responsible for obtaining strategic goals. Tactics are specially focused on actions that take a small amount of time to perfume and obtain. For human players tactics involve much more flexibility due to the quick issuing commands that can be done during the game over units, builds and other elements that are involved in the gameplay process.



## Scouting

The act of gathering information in RTS games is recognized as **Scouting**. Games like StarCraft have their maps covered by the fog of war, disallowing the player to predict or act with respect to his opponent’s strategy. In order to get information about the map, the enemy positioning and build progression, a random worker is assigned to explore the map. Each unit has a different range view, and enemy units can be seen only in the vision radius of the players units. Some units are invisible (Ghosts, Dark Templar or any burrowed Zerg unit), so they can be detected only with special equipment or detector units. For example the Zerg’s Overlord can detect any close invisible units [6].

## Combat timing

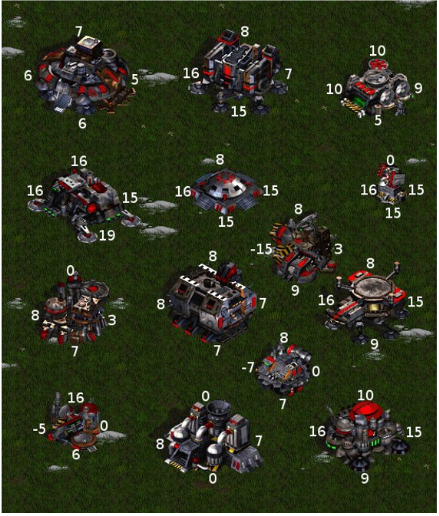
Once an army has been built, the player must decide where and when to land the next attack. In RTS games, attack timings are very important due to the rock-paper-scissors nature of army composition (i.e. the unknown state or the lack of information regarding the enemy’s army determines a random decision making system where a player has to make an assumption and act accordingly without knowing any environment data).

Most combat related data are focused on low-level abstraction, rather than the more abstract task of planning when and where squads should attack in RTS games.

## Building placement

After deciding what the next required building is, the player should consider placing the building in such a way that does not block the players own units in moving, and increases the difficulty for the enemy to invade the base. At the moment, creating a perfect build placement is rather difficult to obtain because the base should fit and take advantage of its current environment. For example ***Chokepoints*** are tight passing areas that allow players to form defensive strategies by blocking them and creating strong anti-ground structures to defend the entrance in the base. Therefore a good building placement would be as near as possible to the chokepoint.

A player must also consider the width and height of a building. Some building also require special placement (Example: Refineries require geysers in order to perform the build). Each building has defined its own length measured in tiles. A tile is a 32 x 32 pixel space. A building cannot be built on occupied tiles or on tiles that have units walking or taking any kind of action on them. One must always consider to check the tiles availability whenever planning the next build.



*The following pictures describes the Terran and Protos building structures (width and height)*

## Reactive control

Reactive control problems involve creating unit actions that involve achieving unit specific goals (Example: *Scouting the enemy base or engaging an invasion over a group of enemy units*). We can list two types of unit controls:

* Unit Micro
* Multi-agent path finding

## Unit micro

Unit micro refers to the unit’s action management mainly in combat performed by the player on a frame. World champion at StarCraft (Jaedong) once said that he can perform 400+ actions/minute and that unit micro lead him in becoming world champion at Broodwar.

There are numerous successful strategies that apply unit micro. For example a Liquipedia article [7] explains the advantages of mutalisk micro on harassing the enemy units. It prevents the targeting AI to pick different mutalisk every the units move out and come back in the range of the defending towers.

## Multi-agent pathfinding

Pathﬁnding and terrain analysis are an important aspect of most video games, and is especially so in RTS games where a player may control a high number of units at once. RTS pathﬁnding typically consists of guiding multiple units on a 2-D map, with units having various properties such as size, speed, and acceleration. In most games, pathﬁnding focuses on shortest-path optimization, whereas RTS games may involve more complex optimizations involving unit damage, keeping units in formations, or avoiding enemy vision.

The default path finding algorithm implemented in StarCraft is based on the popular A\* path finding algorithm and is executed every time a unit or a group of units receive a moving/attack/construct action.

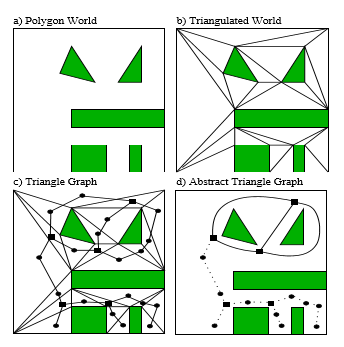
Due to the large computational needs of A\*, it has been modiﬁed in many ways in order to produce pathﬁnding systems for real-time games. Hagelback [8] combined potential ﬁelds with A\* pathﬁnding as a means of unit navigation in StarCraft, concluding that it was preferable to naive A\* for StarCraft navigation.



Later there have been other researches regarding pathfinding and a new method called **triangulation method** has been discovered that is much faster and reliable than Hegelback method dividing the map in triangles and iterating.

Given an environment represented with a polygonal description — one that has barriers between traversable terrain and obstacles described as line segments (see on the figure bellow) — we wish to create a triangulation. This is done by inserting edges between these line segments endpoints until all spaces are divided into triangles [9].

In its simplest form, pathﬁnding in a constrained triangulation is done by hopping from triangle to triangle



How to reduce triangulation. Graph is constructed from polygonal world description

In its simplest form, pathﬁnding in a constrained triangulation is done by hopping from triangle to triangle. This presents a challenge because we do not know the path the object will take through each triangle.

# KBot

KBot is a StarCraft AI agent created with the purpose of analyzing better methods to optimize agents by improving cyclical computations and asynchronous frame management. Also the agent uses data collection and action ordering systems to perform several tasks and receive info from the environmental output. With the gathered data, a processing is made in order to cache de important outputs and use them in the upcoming frames.

## Application structure

In order to create accessibility in the whole project, the bot had to be arranged on different abstraction levels. Each level is computed on a separate frame and provides full access to all other related levels.

There are 3 abstraction levels:

* + Unit level abstraction
  + Action level abstraction
  + Coordinator level abstraction

Unit level abstraction provides control over certain units and provides the movement, training, attack and gathering actions. Each

* 1. Frame management

On each frame a separate computation is made.

* 1. Level of abstractions
  2. Algorithms workflow
  3. Data analysis

Bibliography

[1] Santiago Ontano´n, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. A survey of real-time strategy game AI research and competition in StarCraft. TCIAIG, 2013. Available at: https://hal.inria.fr/file/index/docid/871001/filename/survey.pdf

[2] StarCraft AI competitions (AIIDE). Available at: https://webdocs.cs.ualberta.ca/~cdavid/starcraftaicomp/

[3] Brood War API: Available at: https://github.com/bwapi/bwapi

[4] The Hidden Markov model (HMM). Available at: https://en.wikipedia.org/wiki/Hidden\_Markov\_model

[5] Advances in AI making games smarter .Available at: http://theinstitute.ieee.org/technology-focus/technology-topic/advances-in-artificial-intelligence-make-games-smarter571

[6] Overlord overview: http://starcraft.wikia.com/wiki/Overlord

[7] Mutalisk Harassment: http://wiki.teamliquid.net/starcraft/Mutalisk\_Harassment#Grouping

[8] Johan Hagelback. Potential-ﬁeld based navigation in StarCraft. In CIG (IEEE), 2012.

https://www.researchgate.net/publication/261280564\_Potential-field\_based\_navigation\_in\_StarCraft

[9] Douglas Demyen and Michael Buro. Eﬃcient triangulation-based pathﬁnding.

https://skatgame.net/mburo/ps/tra.pdf