# Challenges and techniques in RTS Game AIs development

In previous section {link} we showed that playing RTS games can be quite challenging, even for a human. In this chapter, we present the challenges in the development of RTS Game AIs and how game agents creators and researchers approach them.

### Challenges in RTS Game AIs

According to [1] current problems in RTS Game AIs can be grouped in 6 different areas as follows:

* **Planning**: As was mentioned in {link} due to the enormous size of action and state space, planning in this domain presents a problem where standard adversarial planning approaches are not directly applicable. A common approach is to use multiple levels of abstractions as was shown in {link}. However, this may not be enough as we explain on common techniques in the following section.
* **Learning**: due to limitations of usage of standard adversarial planning techniques researchers have been training to employ learning techniques to improve game AIs. In this area, most efforts were put into **Prior Learning** as a way how to learn appropriate strategy before game using for example replays and specific map information. Another are of focus is deploying online learning techniques allowing agents to improve their play while playing a game. It is called **In-game learning**. Researchers are also interested in **Inter-game learning** with the goal to increase the chance of victory in next game by learning from previous one.
* **Uncertainty**: what makes planning in this domain even harder is uncertainty as adversarial planning under uncertainty in domains of the size of RTS games is still an open question. Uncertainty comes from 2 sources – environment is partially observable, and a player cannot predict opponent's actions.
* **Spatial and Temporal Reasoning**: be able to position unit and building well in right time is a vital part of each game playing agent for strategy and tactic execution. Therefore, RTS AIs developers pay much attention to Spatial and Temporal Reasoning.
* **Domain Knowledge Exploitation**: in the case of RTS games (compare to for example board games) exploitation of domain knowledge remains quite an uncharted territory. There have been two main directions: in most common researchers are focusing on hard-coding strategies to agents, so agents have to only decide on an action from the predefined set of strategies. In other researchers are trying to learn plan, strategies, or trends from replays. However, how to learn any of this in games like StarCraft automatically is still uncertain.
* **Task Decomposition**: due to many challenges mentioned before and the fact that decomposition is natural even for humans (as is seen in {link}), a decomposition is a preferred approach for developing game AI. However, even it presents many challenges. The significant one is on design architecture which would enable individual AI techniques to work well together.

### Techniques for RTS AI development

To develop game playing agent, one needs to address most of the previously mentioned problems so classification according to this problems is hard. So [1] categorize techniques rather to 3 branches: strategy, tactic and reactive control. What each branch represents is described in detail in previous section {link}.

#### Strategy

Making decisions on strategy level presents still an open problem in the domain of RTS games because of the size of the search space. Solutions based on Markov Decision Processes (MDPs) and Partially Observable Markov Decision Processes (POMDPs) are hardly applicable here. Over the years there have been many other simpler approaches how to address those problems from planning based methods, hard-coded solutions to machine learning methods. We give a short description for just a few of them as more details can be found in [1]. Is worth to mention that most of the techniques mentioned assume complete information which may be problematic in an environment of imperfect information.

Most commonly used techniques are ones that are hard-coded. That especially holds in commercial RTS game industry. The most typical case is to use finite state machines (FSM) where AI behavior is decomposed to manageable states with conditions triggering transitions between them. Despite FSMs popularity and level of adoption by developers of game AIs, they are easily exploitable by opponents who can adapt, because they struggle to encode dynamic and adaptive behavior. Other approaches based on FSMs such as Hierarchical FSMs and Behavior trees (BT) remain exploitable too. Other well-explored techniques providing more flexibility are approaches employing planning techniques such as Case-based planning (CBP) and Hierarchical Task-Networks (HTN). Many researchers have been trying to approach the problem from machine learning point of view employing massive data sets of replays available. Very popular technique among researchers is case-based reasoning (CBR). Examples of CBR and other less common machine learning techniques such as Hidden Markov Models, Bayesian networks, and evolutionary algorithms, are listed in [2].

#### Tactics

Tactics involve two part: reasoning about tactical decisions in a battle and terrain analysis. When concerning those parts of tactics composed mainly from spatial and temporal reasoning for fighting battles, all the techniques mentioned in section {link} can be applied as well. On top of that literature such as [1] gives examples of other techniques such as Answer Set Programming (ASP) for walling (intentionally blocking the entrance to base), UCT algorithm (a Monte Carlo Tree Search algorithm) for tactical decisions, and so long. Most of the work done in Terrain analysis is usually performed off-line before a game as most of the information gathered about map holds for the whole match. Terrain analysis techniques are for example based on influence maps, application of Voronoi decomposition to detect regions and choke points, and others.

#### Reactive Control

There are many useful techniques to maximize the effectiveness of units, but Potential fields and Influence maps are the most prominent ones. Those techniques are used for things like obstacles avoidance or for staying in maximum shooting distance to minimize taken damage. There are other popular approaches based on usage of simple pathfinding algorithms represented in many cases by A\*. Many researchers have been exploring options to use machine learning techniques already mentioned in {link} and {link} for reactive control. On top of that, many of them have tried to employ reinforcement learning (RL) of some kind.

### Design of the state of the art game playing AI agents (in StarCraft)

Authors of [1] give a practical overview of architectures used in StarCraft bots participating in StarCraft AI competitions. The current situation on the field of complete game playing agents for StarCraft does not differ much from the one in 2013. Developer and researchers have been working to some extent on integrating many of the techniques introduced earlier to complete bots. The truth is that incorporating techniques alone to bot is not enough to match human ability to play RTS game, so designers use a lot of domain knowledge to improve the play of their agents. The typical approach is to divide the problem to subproblem which alone is technique how to handle such complex issues. Creators then can for each type of subproblem choose appropriate method how to deal with the problem. The real art of complete bot development is designing architecture which can integrate many techniques together to get an intelligent agent. By analyzing top bots’ structures [1] identifies following tools used by creators to help them achieve the goal:

* **Abstraction**: it is very common for AI agents in StarCraft to reason about the task on different levels of abstraction to make the problem easier to solve. For example playing a game can be seen from a high level as deploying strategy and from low-level spectrum as issuing commands to individual units. The usual practice is to develop a module for each level of abstraction and use outputs of reasoning on a high level of abstraction as input for the lower level. For example, the top-level module will select a strategy to execute, and the lower level module uses this to come up with build order which is then performed by the lowest level module.
* **Divide-and-conquer**: playing the game can be divided to separate task which can be under some assumptions handle relatively independently of each other so one module can, for example, concentrate on gathering resources and other on managing units in the battlefield.

Lots of bots use a combination of those two tools. Interesting fact about most current bots is that on the higher level they are usually scripted which usually involves using a set of predefined strategies to be executed by the agent. In better scenarios agents are mixing between those strategies. However, even this is not enough to match human player who can adapt to those strategies in worst case after few games as he can easily predict what will bot play based on previous games. On top of that agents lack adaptivity because once strategy is selected bot will follow it for the rest of the match.

# Bibliografie

|  |  |
| --- | --- |
| [1] | ONTANON, Santiago, Gabriel SYNNAEVE, Alberto URIARTE, Florian RICHOUX, David CHURCHILL a Mike PREUSS. A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft. *IEEE Transactions on Computational Intelligence and AI in Games* [online]. 2013, **5**(4), 293-311 [cit. 2017-05-05]. DOI: 10.1109/TCIAIG.2013.2286295. ISSN 1943-068x. Dostupné z: http://ieeexplore.ieee.org/document/6637024/ |
| [2] | SYNNAEVE, Gabriel a Pierre BESSIERE. A Dataset for StarCraft AI & an Example of Armies Clustering. *Artificial Intelligence in Adversarial Real-Time Games 2012* [online]. 2012, , 7 [cit. 2017-05-03]. Dostupné z: https://arxiv.org/abs/1211.4552 |
| [3] | PÉREZ, Alberto. *Multi-Reactive Planning for Real-Time Strategy Games*. Barcelona, 2011. Universitat Autònoma de Barcelona. |
| [4] | SHOHAM, Yoav. a Kevin LEYTON-BROWN. *Multiagent systems: algorithmic, game-theoretic, and logical foundations*. 2008. New York: Cambridge University Press, 2009. ISBN 05-218-9943-5. |