# Presentation of the bot and its Experimental Evaluation

In the previous chapter, we gave an overview of the architecture of our bot with description how we integrated Inverse Reinforcement Learning to it. In this chapter, we present examples of our implementation with an overview of agents, and they desires. We also discuss the behavior of our bot and results against built-in AI. As our bot is still more prototype than any serious threat to current the state of the art competition bots we give analysis to explain his current performance to help us in future development.

## Realization of bot

We decide for our bot to play Zerg. We like the idea of the hive and seeing our bot as one is compelling to us. In figure {link} we present high-level overview of the implementation of agents with their desires and relations. Agents representing units and agents to handle the economic aspect of the game or build orders are standard even in other MAS systems such as those defined in [1] or [2]. We introduce a new type of agent representing the location where a base can be built. In our opinion using this kind of agent could be better than the central solution when coordinating multiple attacks.

{figure - agents}

Each base location has the same set of desires for others. Those are desires such as build static anti-ground or anti-air defense, or to send ground or air units to hold this base. A base decides on a commitment to those desires using decision modules learn trough IRL. When a base makes a commitment, it shares desire with system together with its location as a parameter. This desire then propagates to units’ “HeapOfPlans” to let them decide if they want to realize the plan they have for this kind of situation. For example, realization in the case of worker and desire to build static ground defense has a form of abstract plan. A worker commits to it when it thinks it is nearest to this location. Then it starts to execute individual steps of the abstract plan by moving to the site, selecting a suitable place for building and finally building it. In our implementation, each worker operates with some time limit to meet this intention. After that, it may stop pursuing it and let others try it. To describe states for a base location with our base on it, we use features related to units (buildings, army – for the enemy and us) on the site, and economic value (workers mining resources). On top of that everything is compared to global values by introducing another set of features. For the enemy base, we use a similar set of features.

Using MAS has other benefits expect decomposition. For example, workers can easily cooperate on another shared desire to gather resources by making reservations on the resource they are currently gathering. Other workers can use this knowledge to decide on mining other resources they choose one they believe is free and nearest to them.

Trough IRL we are currently learning decisions for the ECO Manager. It uses learned decision modules to decide when to expand, build another worker, increase population capacity or build another gas extractor. So far, we have added support only for three military type of units – melee and ranged ground, and air. Evan though, we omit research and other unit types, it should provide bot with some level of flexibility to start with. Decisions, when to build any unit or infrastructures, are made by Unit Order Manager and Building Order Manager. Those are also learned through IRL by replay observation. Rest of the stuff is currently hard-coded.

To learn decision-making trough observation, we composed dataset of roughly 500 replays. We download those replays from forum thread {link} where the users upload hand curated packages of interesting plays. There are many large datasets {link} available. However, we currently restricted ourselves to hand curated replays as we did not have the capacity to parse larger datasets.

## Analysis of bot’s behavior and its performance

To show our bots current capabilities, we put videos of our bot beating default Zerg AI on 1v1 maps {link}. In our setting, we experiment with a dataset of various sizes; we let our bot to see some replays ranging from 250 to 500. For each dataset, we use all replays for the definition of MDPs, and at most 35 replays for learning reward function using IRL. On one machine employing more replays is very time-consuming. To speed things up and complexity restriction, we limit our MDPs to 1500 states. Sadly, 1500 as K for clustering is in most cases not sufficient enough to get the best partitioning as we discovered when we did some data exploration.

After each learning, we tested our bot against built-in AIs on different competitive maps varying in size. It is fascinating to observe that the bot does different build orders against various opponents. In many cases, the build order is also influenced by map size. Transitions are also different and may depend on the situation. The acting is in many cases amusing. However, using the right set of default replays for learning and the whole set of replays for MDPs, bot learned a strategy able to beat zerg opponent in 1on1 maps in few scenarios. When comparing two provided videos illustrating this strategy transitions are different – in one setting bot build mid-game infrastructure in other it does the massive expansion.

By testing our bot, we analyzed following issues which in our opinion undermine the performance of our bot most:

* Bot builds basic infrastructure at the beginning but sometimes is idle until it has another vision on the enemy. To reduce the number of states, we eliminated time from our features. We based them on the current situation only. Together with insufficient scouting micro, it presents a big problem for bot ability to survive. Slow transition in most cases leads to certain death.
* The composed dataset may be very opinionated and too small. With a narrow set of situations as most of it are games from professional players. Some of the features may not describe the problem well. Mentioned time is one of the most prominent examples.
* There are many bugs in our bot, and our bot is currently lacking any serious micro. We also omit many capabilities of the full-scale player. Many capabilities are interrelated, and their absence can degrade other ones.
* There are many parameters settings we have not optimized well yet due to the state of the development.
* Learning single decision-making module for all setting shows to be too optimistic. A number of examples limit us when training decision modules to generalize to all situations.

We identify many problems crippling performance of our bot. The positive thing is that we can do something about them. Concrete steps to correct those issues are presented in the final chapter as part of future work.