University of Tehran

School of ECE

**Report**

Ameer Hussain Birjandi – Nima Zamanpour

فهرست

[StarCraft Multi-Agent Challenge (SMAC) 3](#_Toc140645893)

[StarCraft 3](#_Toc140645894)

[SMAC 3](#_Toc140645895)

[State and Observations 4](#_Toc140645896)

[Actions 5](#_Toc140645897)

[Reward 5](#_Toc140645898)

[Qmix Algorithm 6](#_Toc140645899)

[Dec-POMDP 6](#_Toc140645900)

[Deep Q-Learning 6](#_Toc140645901)

[Deep Reccurent Q-Learning 6](#_Toc140645902)

[Value Decomposition Networks 7](#_Toc140645903)

[QMIX 7](#_Toc140645904)

[Proposed Method 9](#_Toc140645905)

[Results so far 10](#_Toc140645906)

# StarCraft Multi-Agent Challenge (SMAC)

## StarCraft

In a regular full game of StarCraft, one or more humans compete against each other or against a built-in game AI. Each player needs to gather resources, construct buildings, and build armies of units to defeat their opponent by conquering its territories and destroying its bases.

Similar to most RTSs, StarCraft has two main gameplay components:

* **Macromanagement** (macro) refers to high-level strategic considerations, such as the economy and resource management.
* **Micromanagement** (micro) refers to fine-grained control of individual units.

Generally, the player with the better macro will have a larger and stronger army, as well as a stronger defensive scheme. Micro is also a vital aspect of StarCraft gameplay with a high skill ceiling, and is practiced in isolation by professional players.

StarCraft has already been used as a research platform for AI, and more recently, RL. Typically, the game is framed as a competitive task: an agent takes the role of a human player, making macro decisions and performing micro as a puppeteer that issues orders to individual units from a centralised controller.

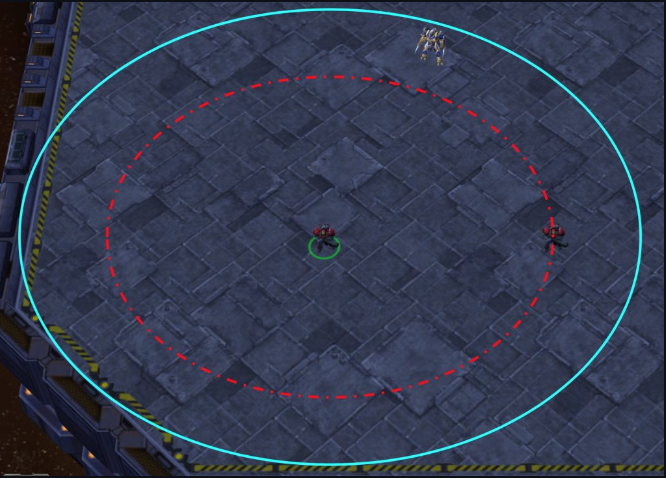
StarCraft II, the second version of the game, was recently introduced to the research community through the release of Blizzard’s [StarCraft II API](https://github.com/Blizzard/s2client-proto), an interface that provides full external control of the game, and DeepMind’s [PySC2](https://github.com/deepmind/pysc2/), an open source toolset that exposes the former as an environment for RL. PySC2 was used to train [AlphaStar](https://www.deepmind.com/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii), the first AI to defeat a professional human player in the full game of StarCraft II.

## SMAC

We introduce the StarCraft Multi-Agent Challenge (SMAC) as a benchmark for research in cooperative MARL.

In order to build a rich multi-agent testbed, SMAC focuses solely on unit micromanagement. We leverage the natural multi-agent structure of micro by proposing a modified version of the problem designed specifically for decentralised control. In particular, we require that each unit be controlled by an independent RL agent that conditions only on local observations restricted to a limited field of view centred on that unit. Groups of these agents must be trained to solve challenging combat scenarios, battling an opposing army under the centralised control of the game’s built-in scripted AI.

## State and Observations



The local observations of agents include the following information about both allied and enemy units which are within the sight range:

* distance
* relative x
* relative y
* health
* shield
* unit type
* last action (only for allied units)

Additional state information about all units on the map is also available during the training, which allows training the decentralised policies in a centralised fashion. This includes the unit features present in the observations, as well as the following attributes:

* coordinates of all agents relative to the map centre
* cooldown / energy
* The last actions of all agents Note that the global state should only be used during the training and must not be used during the decentralised execution.

The shape of the observations is fixed and equal to . However, if the agent is standing somewhere which there are not any agents in its sight, its observation will be filled with zeros.

## Actions

Agents can take the following discrete actions:

* move[direction] (four directions: north, south, east, or west)
* attack [enemy id]
* heal [agent id] (only for Medivacs)
* stop

An agent is permitted to perform an attack/heal action only towards enemies/allies that are within the shooting range.

In the game of StarCraft II, whenever an idle unit is under attack, it automatically starts a reply attack towards the attacking enemy units without being explicitly ordered. We limited such influences of the game on our agents by disabling the automatic reply towards the enemy attacks and enemy units that are located closely.

The maximum number of actions an agent can take ranges between 7 and 70, depending on the scenario.

## Reward

Agents only receive a shared team reward and need to deduce their own contribution to the team’s success. SMAC provides a default reward scheme which can be configured using a set of flags. Specifically, rewards can be sparse, +1/-1 for winning/losing an episode, or dense, intermediate rewards after the following events:

* dealing health/shield damage
* receiving health/shield damage
* killing an enemy unit
* having an allied unit killed
* winning the episode
* losing the episode
* Nonetheless, we strongly discourage disingenuous engineering of the reward function (e.g. tuning different reward functions for different scenarios).

# Qmix Algorithm

## Dec-POMDP

A fully cooperative multi-agent task can be described as a Dec-POMDP which consists of the following properties:

* describes the true state of the environment.
* Individual actions:
* State transitions:
* Shared team reward:
* Observation function:
* Action-observation history:
* Decentralized policies:

Although training is centralised, execution is decentralised, i.e., the learning algorithm has access to all local action-observation histories and global state , but each agent’s learnt policy can condition only on its own action-observation history .

## Deep Q-Learning

Deep Q-learning represents the action-value function with a deep neural network parameterised by θ. Deep Q-networks (DQNs) use a replay memory to store the transition tuple i, where the state is observed after taking the action u in state s and receiving reward r. θ is learnt by sampling batches of b transitions from the replay memory and minimising the squared TD error:

Where . θare the parameters of a target network that are periodically copied from θ and kept constant for a number of iterations.

## Deep Reccurent Q-Learning

In partially observable settings, agents can benefit from conditioning on their entire action-observation history. Deep Recurrent Q-networks (DRQN) make use of recurrent neural networks.

## Value Decomposition Networks

VDNs aim to learn a joint action-value function , where is a joint action-value history and is a joint action.

The loss function for VDN is as follows:

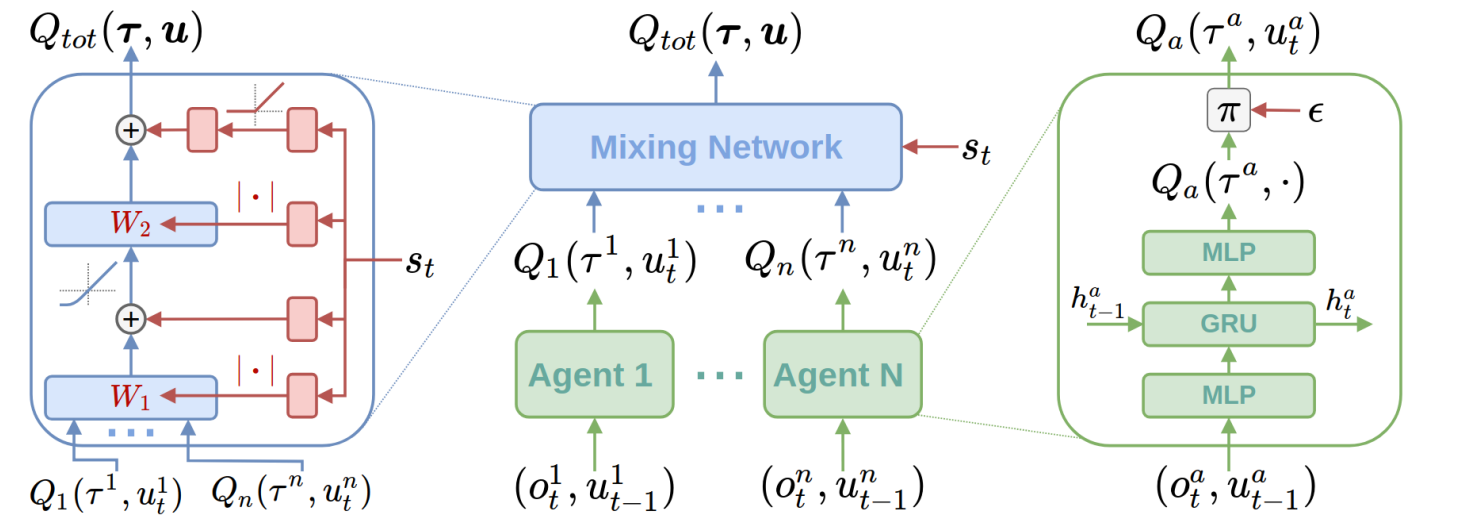
## QMIX

Monotonicity can be enforced through a constraint on the relationship between and each :

For each agent a, there is one agent network that represents its individual value function We represent agent networks as DRQNs that receive the current individual observation and the last action as input at each time step.

The mixing network is a feed-forward neural network that takes the agent network outputs as input and mixes them monotonically, producing the values of .To enforce the monotonicity constraint, the weights (but not the biases) of the mixing network are restricted to be non-negative. This allows the mixing network to approximate any monotonic function arbitrarily closely.

The weights of the mixing network are produced by separate hypernetworks. Each hypernetwork takes the state s as input and generates the weights of one layer of the mixing network. Each hypernetwork consists of a single linear layer, followed by an absolute activation function, to ensure that the mixing network weights are non-negative. The output of the hypernetwork is then a vector, which is reshaped into a matrix of appropriate size. The biases are produced in the same manner but are not restricted to being non-negative. The final bias is produced by a 2-layer hypernetwork with a ReLU non-linearity.



The loss function for VDN is as follows:

,

where .

# Proposed Method

In this part we want to add an agent that we call the *captain* which aims to control the sight range(attention) of the agents. The *captain* state consist of all the observations of the agents. The default sight range for agents is 9 and so the *captain* must choose between the actions below:

* Change the sight range to 6
* Keep the sight range at 9
* Change the sight range to 12

Important note is that default shoot range for all units is 6. So, even with the lowest sight range for allies, it still won't give enemy first see first shoot.

The Reward of the *captain* is calculated with the below formula:

,

where is the normal agents reward (usually about 0.6 at its peak) and is the kill-sight ratio.

By adding the *captain* the training process will be as follows:

While episodes not finished:

1.For each episode:

1. The *captain* collects all the observations.
2. Based on their values chooses the appropriate sight range for each agent.
3. The agents get their observations based on their new sight range.
4. Agents choose an action based on their action-value function.

2. The *captain* and the agents receive their reward and update their values.

In our method the captain uses QMIX for its training. So, it considers all the agents while choosing an action for each agent.

**Hyperparameters:**

* **Training time:** 5e5 timesteps
* **Epsilon:** decays from 1 to 0.01 in 1/5 of total timesteps
* **Evaluation:** the game is run for 100 times in test mode for a total of 25 times.
* **Agent's network:** QMIX with double Q learning with 64 neurons for MLP and RNN layers**.** Also uses hard update
* **Captain's Network:** Same as above, but doesn't use RNN.
* **:** 100

# Results so far

To test our method, we first ran the environment without captain for 6 and 12 sight range (two extremes). And them, added captain to see where its win rate lies between the two.

These are the results for 10-marines map:

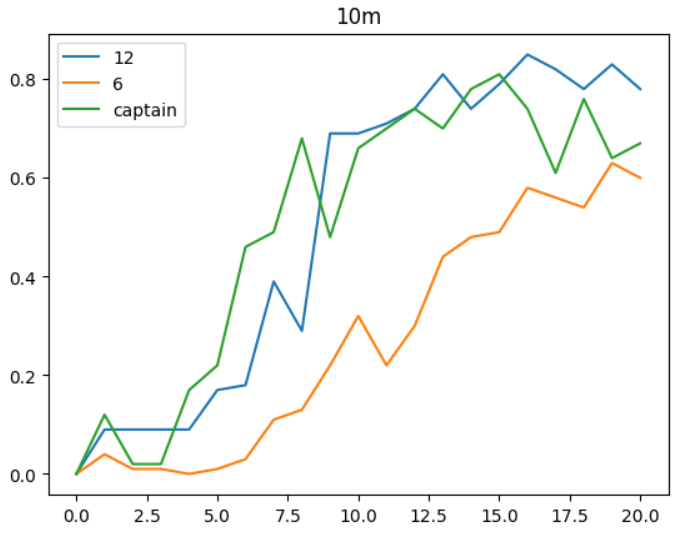


Figure Win rates for 10-Marines map

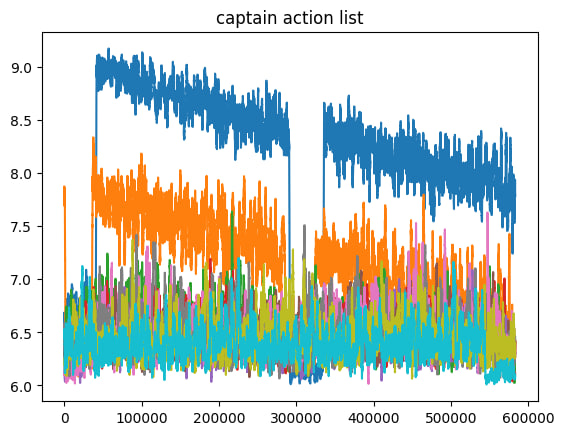


Figure Captain action list for 10 Marines map

And, these are the results for 5 marines map:

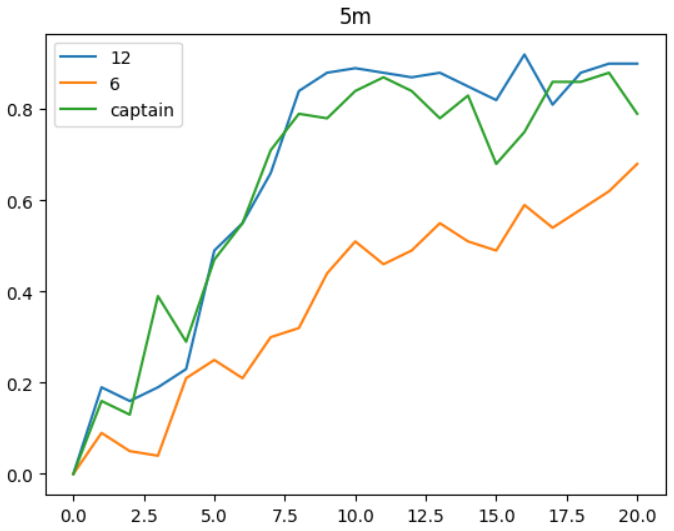


Figure Win rates for 5 Marines map

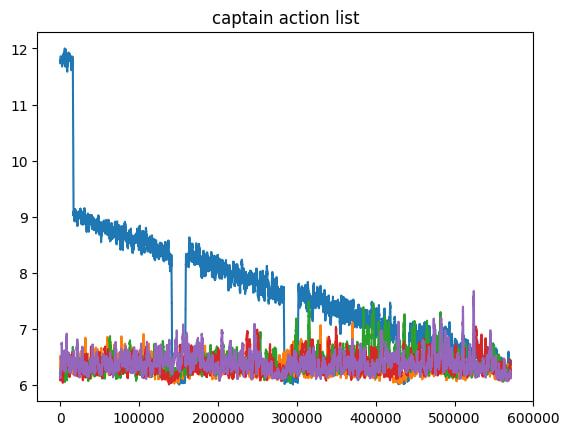


Figure captian action list for 5 marains map

The results show that, captain successfully rose win rate close to upper extreme while keeping the average sight range just above 6. However, by carefully looking at captain action list we see that agent 1(in all scenarios) is the only agent that has a falling average sight range and all other agents have an average of 6.2 with negligible changes.

Although exact analysis of captain performance must be carries out within one episode (not around 50000 independent episodes stacked above together), it is almost impossible to do this as game is in speed-run and has no playback.

So, one way to deeper analyze captain performance is to change default shoot range to 9. This gives enemy first, see first shoot advantage when ally's sight range is reduced to 6.

These results will be ready soon.