Agents in reinforcement learning

GROUP 5

Ander Salaberria Eneritz Domínguez

Abstract

This document introduces the investigation that DeepMind has done based on the StarCraft II game. This is a multi-agent problem with multiple players interacting; there is imperfect information due to a partially observed map; it has a large action space involving the selection and control of hundreds of units; it has a large state space that must be observed solely from raw input feature planes; and it has delayed credit assignment requiring long-term strategies over thousands of steps. They have provide an open source Python-based interface for communicating with the game engine. In addition to the main game maps, they have provide a suite of mini-games focusing on different elements of StarCraft II gameplay. Finally, we present the reinforcement learning, which is the method they have used to do the investigation.

1. Introduction

First of all, is important to know what is StarCraft II. It is a real-time strategy game developed by Blizzard in which players need to build armies and vie in order to control the battlefield. The armies can be as small as we want, and the player have to command their armies in real time. Strategic thinking is key to success: we need to gather information about our opponents in order to anticipate their moves and formulate a winning strategy.

2. About their work

What DeepMind wants to do is to push the boundaries of artificial intelligence (AI), developing programs that can learn to solve any complex problem without needing to be told how. Computer games are perfect to do this. They provide a compelling solution to the issue of evaluating and comparing different learning and planning approaches on standardised tasks, and is an important source of challenges for research in AI. Besides, they are externally defined to be difficult and interesting for a human to play. This ensures that the challenge itself is not tuned by the researcher to make the problem easier for the algorithms being developed; and in some cases a pool of avid human players exists, making it possible to benchmark against highly skilled individuals. In summary, games are perfect to do this because they allow us to develop and test smarter, and provide us instant feedback on how we're doing through scores.

In 2015, DeepMind created a Go program called Alpha-Go which beat the world champion in a 3 game match two years later of its creation. Now, they are disbanding the team that worked on the game while continuing AI research in other areas, such as StarCraft II.

3. Processes of the investigation

StarCraft II is a very complex game. Compared to chess or Atari games, in StarCraft II we don't have all the information we need to make our best possible movement, it's an imperfect information game. So, in order to make a better investigation, they divided the game in several mini-games.

These are focused scenarios on small maps that have been constructed with the purpose of testing a subset of actions or game mechanics with a clear reward structure. They have initially made seven mini-games:

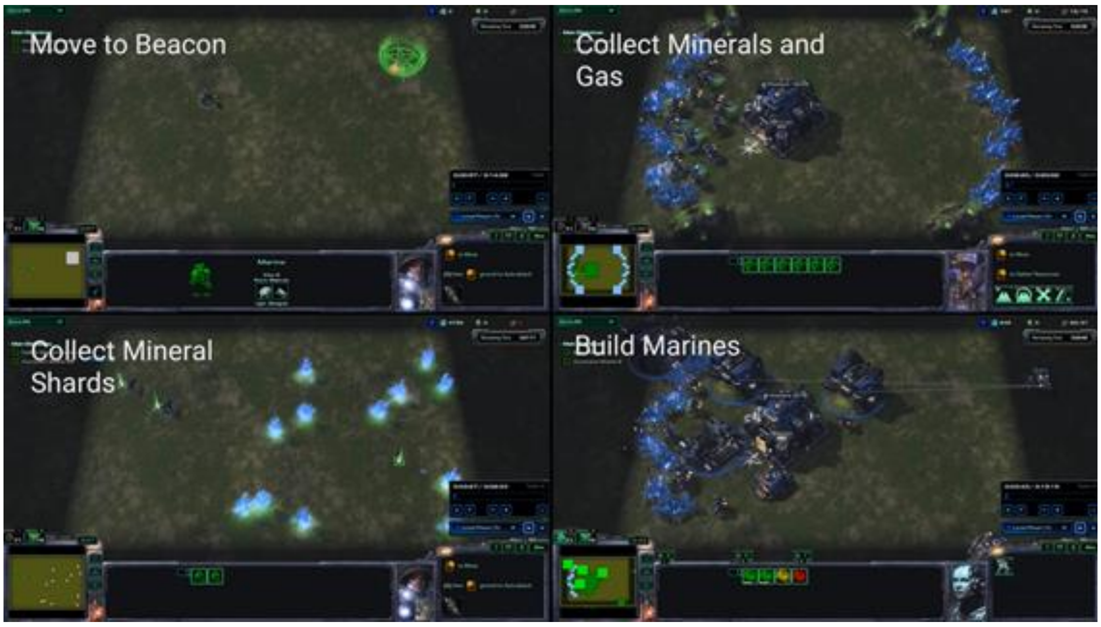

Figure 1. Here we've got some examples of the mini-games they have created. An example of these games are shown in the following link. In total, there are seven initial mini games designed by DeepMind, but programmers have resources to make new ones to see their agent's behavior. https://www.youtube.com/watch?v=6L448yg0Sm0&feature=youtu.be

- MoveToBeacon: The agent has a single marine that gets +1 in its score each time it reaches a beacon.
- CollectMineralShards: The agent starts with two marines and must select and move them to pick up mineral shards, which are around the map.
- FindAndDefeatZerglings: The agent starts with 3 marines and must explore a map to find and defeat individual Zerglings.
- DefeatRoaches: The agent starts with 9 marines and must defeat 4 roaches. Every time it defeats all of the roaches it gets 5 more marines as reinforcements and 4 new roaches spawn. The reward is +10 per roach killed and -1 per marine killed.

- DefeatZerglingsAndBanelings: The same as DefeatRoaches, except the opponent has Zerglings and Banelings, which give +5 reward each when killed. This requires a different strategy because the enemy units have different abilities.
- CollectMineralsAndGas: The agent starts with a limited base and is rewarded for the total resources collected in a limited time.
- BuildMarines: The agent starts with a limited base and is rewarded for building marines. It must build workers, collect resources, build Supply Depots, build Barracks, and then train marines.

To do all of this, they are using an environment, SC2LE, which is composed by three sub-components: a Linux StarCraft II binary, the StarCraft II API, and PySC2. They use the API to start a game, get observations, take actions, and review replays. Using this API they have built PySC2, which is a Python environment that helps the StarCraft II API to facilitate the interaction between Python reinforcement learning agents and StarCraft II.

4. Observating the environment

StarCraft II uses a game engine which renders graphics in 3D, but, instead of using those 3D rendered graphics, the StarCraft II API generates a set of “feature-layers”, which abstract away from the RGB images seen during human play and, still, maintain the core spatial and graphical concepts of StarCraft II (see Figure 2).

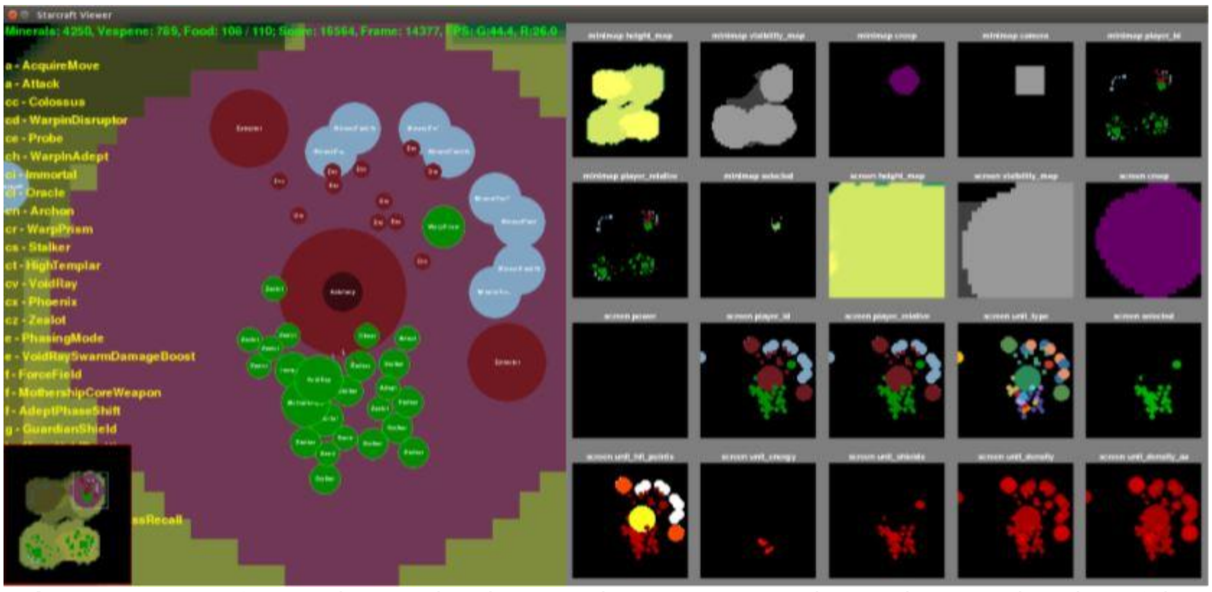

Figure 2. Here we’ve got the API they have made. We can see a human interpretable view of the game on the left, and coloured versions of the feature layers on the right. More examples in https://www.youtube.com/watch?v=-fKUyT14G-8&feature=youtu.be

Each of these layers represents something specific in the game, which can represent the minimap or the screen. Some of these (e.g., hit points, height map) are scalars, while others (e.g., visibility, unit type, owner) are categorical. Apart from those, there are also various non-spatial observations that the human interface provides and are relevant for the agent, such as minerals and gas quantity.

To simplify measures, the screen is rendered with a top down ortographic projection, which is scaled into a N x N pixel map (being N = 64, in this case). So, as the reader may think, there are an uncountable amount of different states in the game, and it is infeasible to precompute all the best possible actions for each different state.

5. Action pool

StarCraft II is not a simple game, that is true. In order to represent the full action space we define approximately 300 action-function identifiers with 13 possible types of arguments. In StarCraft II, not all the actions are available in every game state. For example, the move command is only available if a unit is selected.

As you can see in Figure 2, there is a list of available actions to be taken in that specific state (those actions can be read on the left side of the image, in yellow).

We could formally represent all actions as a composition of a function identifier a^0 and a sequence of L arguments which that function identifier requires: $a^1, a^2, ..., a^L$. You can see an example here (Figure 3):

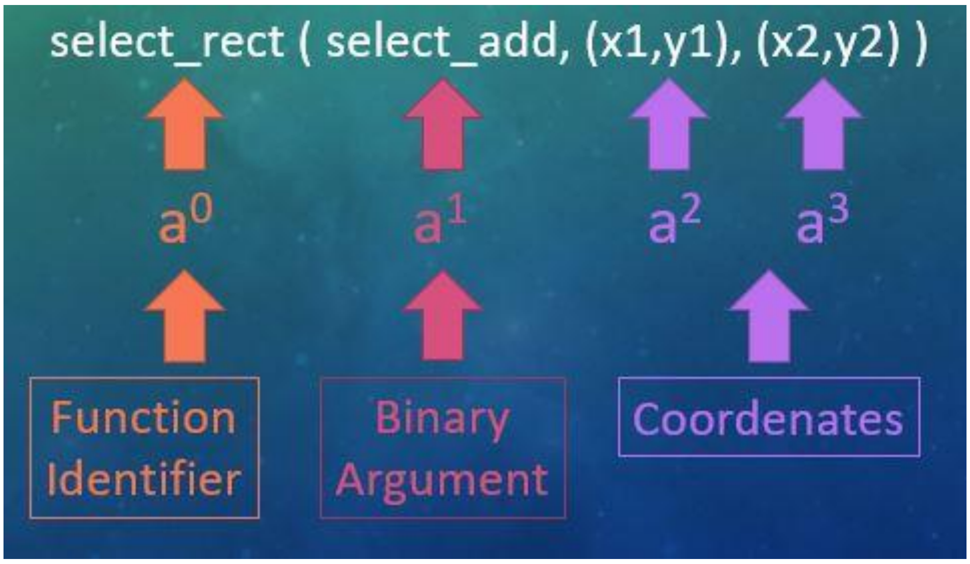

Figure 3. This is an example of the function (a^0) that selects units drawing a rectangle with the mouse. The first argument (a^1) is a binary argument which decides if the new selected units are going to be added with the previously selected units. The last two parameters (a^2 and a^3) are pixel coordinates that defines the rectangle on the screen.

Humans typically make between 30 and 300 actions per minute (APM), roughly increasing with player skill, with professional players often spiking above 500 APM. This means that the agent must decide which action to choose as fast as possible (and as good as possible too). In Figure 4, the reader will be able to see the difference between how actions are made by a human and an agent.

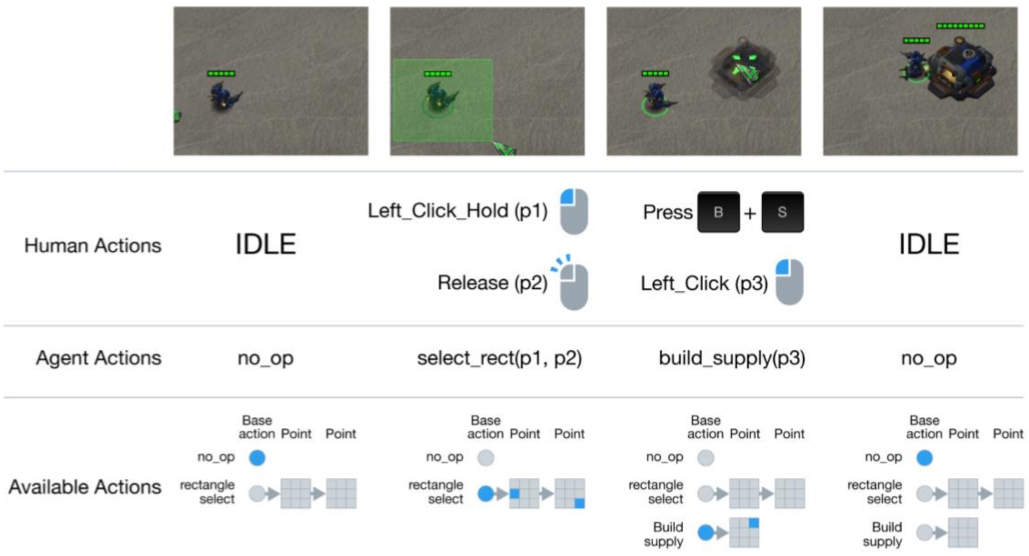

Figure 4. Human and agent actions can be compared in this figure. The action space were designed to be as close as possible to human actions. In the last row, we can see a simplified version of the list of available actions. Note that the first two columns 'build supply' action isn't in the list, because a worker has to be selected first.

6. Reinforcement Learning: Baseline Agents

Now that we know how the agent can observe the environment and how would be able to make some actions depending of its surroundings, the question is, how do they learn what to do in each specific scenario? Reinforcement learning may be the answer.

At the core of DeepMind's team's reinforcement learning agents sits a deep neural network with parameters θ, which defines a policy π_θ. At time step t, the agent receives observations s_t, selects an action at with probability π_θ $(a_t|s_t)$, and then receives a reward r_t from the environment. The goal of the agent is to maximise the return G_t, where γ is a discount factor (whose values are between 0 and 1).

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

In this case, DeepMind uses Asynchronous Advantage Actor Critic (A3C) in order to learn the parameters θ, which has shown to produce state-of-the-art results on some mini-games.

As it is said before, the API exposes actions as a nested list a which contains a function identifier a^0 and a set of arguments. Since all arguments including pixel coordinates on screen and minimap are discrete, a naive parametrization of a policy π(a|s) would require millions of values to specify the joint distribution over a, even for a low spatial resolution. Using a chain rule such as the next one can be a solution.

$$\pi(a|s) = \prod_{l=0}^{L} \pi(a^l|a^{<l}, s)$$

This representation, if implemented efficiently, is arguably simpler as it transforms the problem of choosing a full action a to a sequence of decisions for each argument a^l.

Decisions about where to click on a screen naturally depend on the purpose of the click. So, the idea is to first decide for the function identifier, then all categorical arguments and, finally, if relevant, pixel coordinates.

7. Mini Game results

As the full game results were really bad and the agents aren't able to keep logical strategies as a human would make, one can avoid the complexity of the full game by defining a set of minigames which focus on certain aspects of the game.

Although many agent architectures are used, we aren't going to make any distinctions about their own results and we are going to analyze all of them as equal agent (as we only want to see what are they capable of).

Agent	Metric	MoveToBeacon	CollectMineralShards	FindAndDefeatZerglings	DefeatRoaches	DefeatZerglingsAndBanelings	CollectMineralsAndGas	BuildMarines
Random policy	mean	1	17	4	1	23	12	< 1
	max	6	35	19	46	118	750	5
Random search	mean	25	32	21	51	55	2318	8
	max	29	57	33	241	159	3940	46
DeepMind human player	mean	26	133	46	41	729	6880	138
	max	28	142	49	81	757	6952	142
StarCraft GrandMaster	mean	28	177	61	215	727	7566	133
	max	28	179	61	363	848	7566	133
Atari-net	best mean	25	96	49	101	81	3356	< 1
	max	33	131	59	351	352	3505	20
FullyConv	best mean	26	103	45	100	62	3978	3
	max	45	134	56	355	251	4130	42
FullyConv LSTM	best mean	26	104	44	98	96	3351	6
	max	35	137	57	373	444	3995	62

Table 1. This is a table in which the average and max score of both human players and artificial agents are compared. We can see some different agent architectures, but, overall, an agent's performances is decreased as the strategic difficulty of the mini game increases.

In table1 we can see the results of two real people in those mini games (one being an amateur and the other one a grand master) as well as the agent's scores. Strategically speaking, the easiest one is "Move to Beacon", which mainly precise of good mechanics and reaction. This is the unique mini game in which the agents beat the humans.

Mini games that require to collect resources are a little bit more difficult for the agents and they normally get a worse score than both the amateur and professional human beings.

Mini games that are focused on combat strategy are better managed by these agents, and in some cases, they reach the level of an amateur player. But, when a mini game is more strategically demanding like "Build Marines", the agents aren't able to apply some logic as a human would be able to and they get really bad scores. This demonstrates that even relatively simple mini-games present interesting challenges, which we hope will inspire new research.

8. Final conclusion

In StarCraft II, mini games are primarily unit-tests which the agent must beat in order to succeed on the full game. As we can see in the results, this goal seems to be still far away, and a lot of research needs to be done if we want to see a decent agent.

These new technologies that are being investigated (such as deep learning and reinforcement learning), seems to be the key to a world full of intelligent agents that will ease our lives, and it is essential to invest a lot of efforts in this area if we want to see that becoming true.

As far as we are concerned, the Real Time Strategy game called StarCraft looks like a very good way to test our agent's capacity to learn and reconsider new thoughts and strategies.

9. Bibliography

- Article - "Release of the SC2LE" by DeepMind

https://deepmind.com/blog/deepmind-and-blizzard-release-starcraft-ii-ai-research-environment/

- Paper - "StarCraft II: A New Challenge for Reinforcement Learning" by DeepMind & Blizzard

https://deepmind.com/documents/110/sc2le.pdf

- Video - "A guide to DeepMind's StarCraft AI Environment" by Siraj Raval

https://www.youtube.com/watch?v=URWXG5jRB-A