

How Pluto Plays

Notes on the Brood War bot that won CoG 2026, worked out from its public release

1 October 2026

In short

- Pluto is one neural network that plays the whole game as all three races: build, macro, scouting and micro. Nobody wrote its strategy or its micro by hand. It learned them by playing against itself.
- It does not follow build orders. Before the game it is told the name of an opening, in the form of a few on/off switches, and it decides every Drone and every building itself. It keeps to the broad idea, such as pool first or hatch first, and plays the details loosely.
- It sees about what a player with perfect attention would see: every visible unit on the map at once, nothing in the fog, and no game clock.
- It makes one decision about every quarter second, and each decision is one order. One order can go to dozens of units, which is why its APM in a replay looks far higher than four a second.
- In its own practice games, Terran was its best race and Zerg its weakest.

WORDS USED IN THIS DOCUMENT

- **The network:** the part that decides. It is a very large calculation. It takes in a description of the game and gives out an order.
- **Weights:** the 315 million numbers inside that calculation. They decide what comes out. Nobody set them by hand; training set them.
- **Training:** playing a very large number of games and adjusting the weights a little after each batch, so that the network plays better.
- **The bot file:** a small, ordinary program that runs inside StarCraft. It describes the game to the network and carries out the orders the network gives.

Much of this document is reconstruction, and some of it is guesswork. Each point is marked with how it is known:

- *(read from the program)*: seen directly in the code or in its data;
- *(computed from the weights)*: calculated from the network's weights;
- *(educated guess)*: fits the evidence but is not proven by it.

What Pluto is

Pluto was written by tscmoo (Vegard Mella), who has made StarCraft bots for many years and also wrote OpenBW, a reimplementation of the Brood War engine. In 2026 Pluto won the CoG StarCraft AI Competition, beating PurpleWave 4–1 in the final.

The release is three files and a short README: the bot file, which loads into StarCraft through BWAPI; a second program that runs the network; and the weights. Unlike most bots, the bot file reads the game's memory directly and sends orders as raw game commands, and uses BWAPI only for a few things such as chat and leaving the game. There is no source code, and the README describes the training in one line, as self-play. Everything here was pieced together from those files and from independent reverse-engineering notes (see the last section).

The bot file and the network work together in a loop. Every 6 game frames, about a quarter of a second on Fastest, the bot file describes the current game to the network, the network answers with one order, and the bot file gives that order in the game. Then the loop starts again. The network never touches the game directly. *(read from the program)*

The quarter-second rhythm is the one Pluto was trained at, so the released version keeps to it: if the computer is too slow, the game waits for Pluto rather than letting it fall behind, unless an answer is more than a second late. No graphics card is needed. The README asks for a fairly recent processor with at least 6 cores, and gives about 20 milliseconds per decision on a current desktop. *(read from the program and the README)*

According to the release notes, the network and the program that runs it are the same ones that played the competition. The bot file is a new build. The main difference is timing: in the competition, an answer that came too late could miss its moment, while the released version waits for it. This document describes the released version.

IN PLAIN TERMS

The bot file is a go-between. Four times a second it describes the game to the network, gets one order back, and gives that order in StarCraft. All the deciding happens in the network.

The rest of the document follows a game in order: first what happens before the game starts, then what the network knows during the game, then how it turns that into an order, and finally what it does in a fight.

Before the game: the opening

Choosing an opening

Pluto has a fixed list of named openings, 21 to 23 per race, counting variants. That count includes a “free” option, in which it is given no opening at all.

Race	Openings in the list
Terran	BBS, 1 Rax FE, 2 Rax Bio FE, Rax Fact, 1 Fact CC, 2 Fact Vulture, Siege Expand, 2 Port Wraith, Goliath, CC First, 14 CC, Vessel
Zerg	4 Pool, 9 Pool, 9 Pool Speed, 12 Pool, 12 Hatch FE, Sunken, 2 Hatch Hydra, 3 Hatch Muta, Lurker, Lurker Defiler, Hydra Mass, Defiler, Ultra Ling

Race	Openings in the list
Protoss	1 Gate Core, 2 Gate, 4 Gate, 1 Gate Robo, 12 Nexus, 14 Nexus, FFE, Speedlot, Corsair, DT Rush, Storm, Carrier, Arbiter

Before each game, the bot file looks up Pluto’s record against that opponent. It favours openings that have won before, and still tries less-tested ones now and then, so that it keeps learning which openings work. It starts from how well each opening did in Pluto’s own practice games, counted as if it had already played six games with each opening, and adds its real results against this opponent on top. The record is a plain text file that can be edited by hand. *(read from the program)*

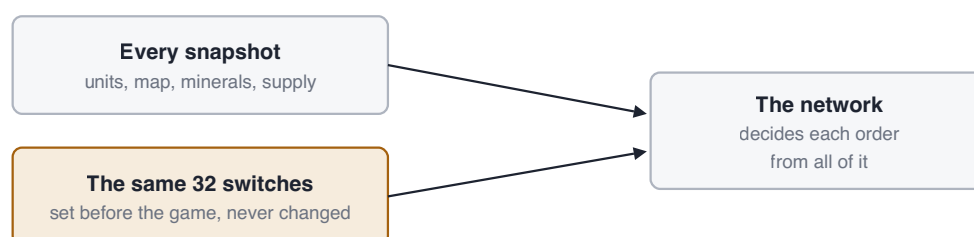
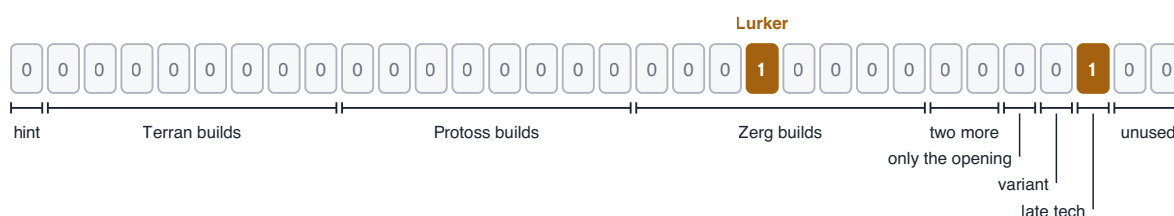
What the network is told

The program contains no build orders at all: no supply counts and no “Pool at 9”. Instead, an opening is a name plus a row of 32 switches. Each switch is either on or off, and each opening turns on one or two of them (“free” turns on none). For Lurker Defiler, the “Lurker” switch and a “late tech” switch are on, and the other 30 are off. *(read from the program)*

These 32 on/off are all the network ever learns about the opening. They are set before the game and never change. The network sees them at every decision for the whole game, next to everything else it sees. Nothing checks whether the build is being followed. *(read from the program)*

Lurker Defiler, as the network receives it

32 on/off switches. Two are on: the Lurker build and the late-tech extension.



No list of build steps is sent at any point, and the program contains none.

Lurker Defiler as the network receives it: the Lurker switch plus a late-tech switch. *(layout read from the program; what the modifier switches mean is an educated guess from the names)*

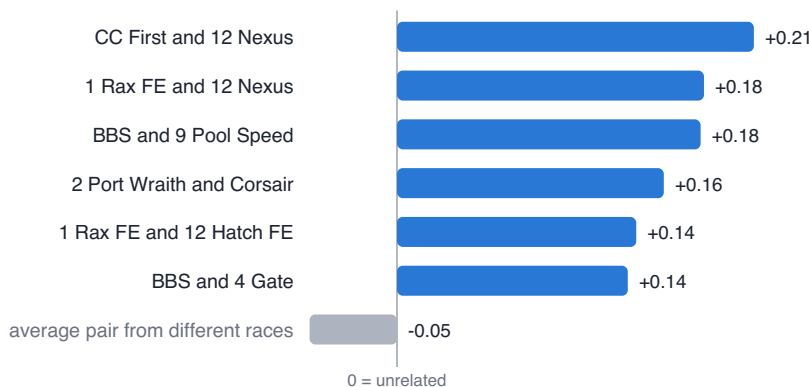
IN PLAIN TERMS

The switches are like a coach saying “play Lurker Defiler” before the game. They are not a written build order. The player still decides every step, and nobody checks during the game whether the plan is being followed.

Do the switches change anything?

Yes. In practice games as Terran against Protoss, the same network won 34% of games with the Goliath switch on and 68% with the 2 Fact Vulture switch on. *(read from the program's opening table; that these are practice results is an educated guess, explained under "How strong it is")* Since the network was the same in both cases, the difference has to come from the switch.

The network also reacts to some switches from different races in a similar way, for example CC First (Terran) and 12 Nexus (Protoss). These two are never on in the same game, so the network was not told they are related; it found that out in training. This suggests that a switch stands for an idea, such as "expand early" or "rush", rather than for a list of buildings. *(computed from the weights)*



How alike the network's reaction is to pairs of switches from different races. 0 means no relation; the grey bar shows a typical pair for comparison. *(computed from the weights)*

How loosely it plays them

For Zerg, the switches mostly settle one question: does the first building come before the first Overlord, or after? *(computed from the weights)*

- **Pool switches, and "free":** nearly always a building before the first Overlord. With 9 Pool, the Pool usually starts at 8 or 9 supply.
- **Hatch switches:** nearly never. Instead it makes the Overlord at 9, then more Drones, and often a Hatchery at 12 to 15.
- **The first minute is the same whatever the switch.** The switches only start to make a difference at 9 supply.
- **The details vary.** Timings move around, "4 Pool" does not produce a 4 Pool, and an Evolution Chamber or Creep Colony sometimes appears when the network wants to build something before it can afford a Pool.

SO FAR

Before the game, Pluto picks an opening from its record. The network is told the opening only as a few switches. It follows the broad idea, and decides every step itself.

During the game: what it knows

Every quarter second, the bot file gives the network a fresh snapshot of the game. The network works through it in four steps, and this section takes them in order:

1. what is in the snapshot;
2. reading the map;
3. reading the units;
4. adding what it remembers.

The result of these four steps is what the decision is made from.

Step 1: what is in the snapshot

Pluto has no screen and no camera. The snapshot has three parts:

- **A list of units:** one row for every unit it can see anywhere on the map, its own, the enemy's and neutral ones. Each row holds about what a player would learn by clicking the unit: type, owner, position, HP, shields, weapon cooldown, what it is doing, burrowed, cloaked, detected, spells on it and visible upgrades. For its own units it also holds energy, production, and whether the unit is selected or in a control group.
- **The map:** for every tile, the terrain, whether it has been explored, whether it is in sight now, and creep and buildings where it has vision.
- **A few general facts:** its minerals, gas and supply, the opponent's race, the opening switches, and which kinds of order are possible right now.

It gets nothing from the fog, nothing about the enemy's money, nothing about what a visible enemy building is producing (only whether it is busy), and nothing about undetected cloaked or burrowed units beyond what a player would see. It is never told the game time. (*read from the program*)

IN PLAIN TERMS

Pluto sees what a player could see if they could look at the whole map at once and never missed anything. It has no maphack: it has to scout like anyone else.

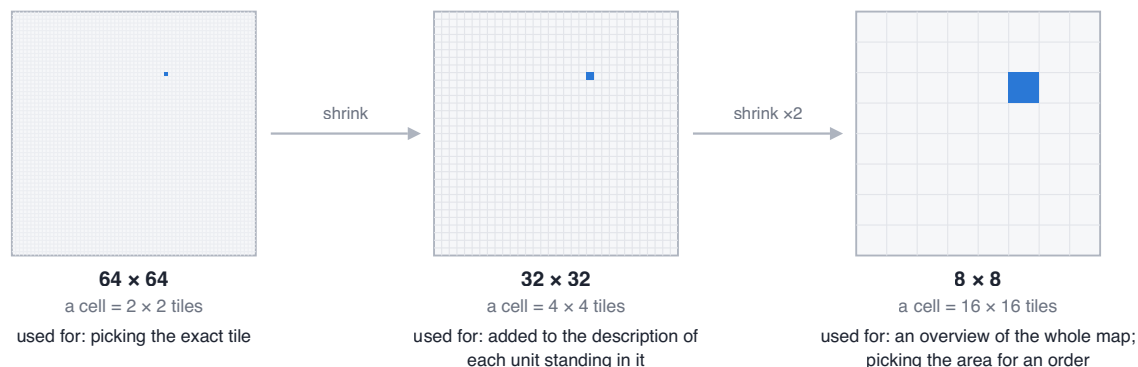
Step 2: reading the map

The map is not a picture. It is a table with one entry for each of the 128×128 tiles, and each entry holds a few facts about that tile. The units from the unit list are written onto the tiles they stand on, so the map and the list describe the same units. A tile in the fog keeps what is known about its terrain but has no units on it. (*read from the program*)

A single tile says very little. To get a view of whole areas, the network shrinks the table in steps. Each step combines neighbouring cells so that the grid is half as wide, and each cell describes a larger area, for example "an enemy army on high ground, on creep". How to combine them is something it learned in training. It keeps three sizes, because each is used for a different job later on. (*read from the program*)



The map is then shrunk in steps. Each step combines 2×2 neighbouring cells into one, so a cell describes a bigger area.



Top: what goes into the map. Bottom: the smaller grids made from it, and what each is used for. The blue cell is the same area of the map at each size.

Step 3: reading the units

A bare row in the unit list describes only that one unit: “a Marine, 40 HP, at this tile”. It does not say that the Marine is standing next to a burrowed Lurker, or that Hydralisks are behind the Lurker. A player sees that at a glance. The network has to work it out.

It does that in two ways. First, each unit gets the cell of the 32×32 map grid that it stands in added to its row, so the row now also says something about the ground and the units around it. Second, the whole list goes through a part called the *unit encoder*. In the unit encoder, every row looks at every other row and takes in what is relevant to it. This is repeated six times. (*read from the program*) The unit encoder is built from the same kind of layer as current language models, which do the same thing with the words in a sentence.

IN PLAIN TERMS

After step 3, each row no longer describes just a unit. It describes a unit in its situation, something like “a Lurker, burrowed but detected, next to 12 Marines, covered by Hydralisks”. The network does not use words. Each row is a list of 768 numbers, but this is the kind of thing those numbers hold.

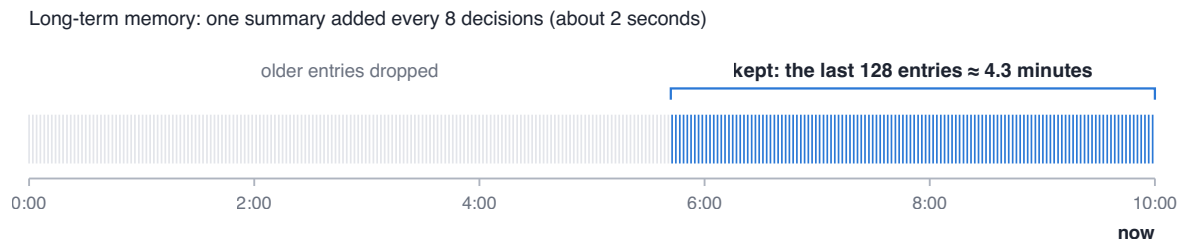
Step 4: adding what it remembers

The snapshot shows only the present. A player remembers what they scouted a few minutes ago, so Pluto keeps two memories. (*read from the program*)

- **A short memory**, carried from one decision to the next and updated every quarter second.

- **A long memory**, which stores a summary about every two seconds and keeps the last 128 of them, a little over four minutes of game time. At every decision the network looks back through all of them.

This is how it can still act on a Spire it saw three minutes ago, even though the Spire is now in the fog.



Separately, the short-term memory is updated at every decision, four times per second.

The long memory: one summary every two seconds, the newest 128 kept.

The result: a summary of the moment

After these four steps, everything the network knows is combined into one long list of 4,096 numbers. In this document it is called the **summary of the moment**. Nobody can read it directly. It is the starting point of every decision. When a decision needs a unit or a place, the network also goes back to the unit rows and the map grids from steps 2 and 3, as explained later. (*read from the program*)

SO FAR

Every quarter second Pluto gets a snapshot: a list of units, a map and a few facts. It reads the map at three sizes, works out each unit's situation, and adds what it remembers. All of that becomes one summary of the moment. The next section is about how an order comes out of that summary.

How a decision is made

What an order looks like

In Brood War, an order a player gives has up to three parts:

- “Build a Factory here”: a kind of order (build), which one (Factory) and a place (this tile).
- “Right-click this Lurker”: a kind of order (right-click a unit) and a unit (the Lurker). There is no “which one”.
- “Stim”: a kind of order (use an ability) and which one (Stim Packs). There is no place or unit.

Pluto's orders have the same parts. There are 20 kinds of order: do nothing; four kinds of select; move (right-click a place); attack-move; right-click a unit; stop; hold position; two kinds of train; build; research, upgrade or add-on; three kinds of ability (on its own, at a place, on a unit); and three for control groups. (*read from the program*) The network does not choose a whole order in one go. It chooses one part at a time.

How one part is chosen

Each part is chosen by its own small piece of the network. This document calls each part a **choice**: the kind of order is one choice, which one is another, and the unit or place is a third. The piece of the network that makes a choice is usually called a *head* in machine learning.

Every choice is made in the same three steps:

1. **Score.** The network takes the summary of the moment and calculates a score for every option. A higher score means the option looks better in this moment. How to calculate the scores was learned in training.
2. **Odds.** The scores are turned into percentages that add up to 100%. A higher score gets a larger share, and a much higher score gets most of it.
3. **Draw.** One option is drawn at random with those odds, like rolling a weighted die.

The bot file also tells the network which options are impossible right now, such as a unit Pluto cannot afford or a spell without enough energy. Those options are removed before the draw, so the network never picks them. (*read from the program*)



The first choice: what kind of order. On the left is the summary of the moment. Each kind of order gets a score, the scores become odds, and one is drawn. The numbers are made up for illustration.

In this made-up example, attack-move has the highest score and gets 55%. So in this exact moment Pluto would attack-move a little more than half the time, move about one time in five, and so on.

IN PLAIN TERMS

A choice works like a points table. Every option gets points, and more points means a better chance of being drawn. The option with the most points is the most likely, but not certain. How many points each option gets in each situation is what training set.

Because the option is drawn, the same situation does not always lead to the same order. Usually one option gets most of the odds, so Pluto mostly does the obvious thing, but sometimes it does something less likely.

Choices in a row

After the kind of order has been drawn, the next choice is made, and it is told what the first choice was. This decides which options it has:

- If the first choice was “build”, the second choice is between buildings.
- If it was “train”, the second choice is between units.
- If it was “use an ability”, the second choice is between abilities.

The third choice, when it is needed, is a unit or a place, and it is told both earlier choices. So after “build” and “Factory”, the choice of place knows it is placing a Factory, not a Pylon or a Sunken. *(read from the program)*

Not every order needs all three choices. “Stim” stops after the second. “Right-click a unit” skips the second and goes straight to the unit. A choice that is not needed is simply skipped.

The figure below shows four complete orders. Read each row from left to right. “Not needed” means that choice is skipped for that kind of order. The last column shows what the bot file then does with the finished order inside StarCraft.



Four orders, choice by choice.

SO FAR

An order has up to three parts. Each part is its own choice, made one after another, and each choice knows the ones before it. Every choice scores its options, turns the scores into odds and draws one.

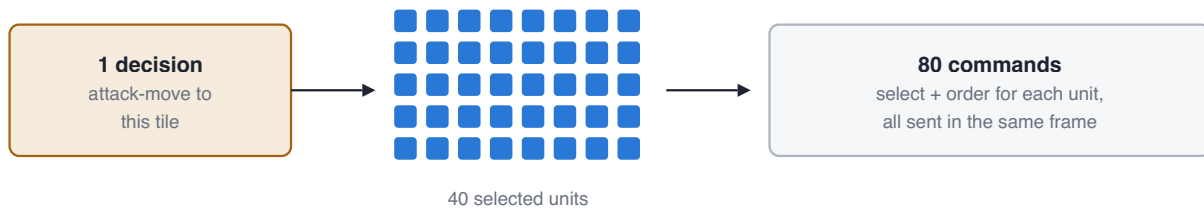
Selecting, and where the APM comes from

In Brood War, an order goes to whatever units are selected. Pluto works the same way. Selecting is one of the 20 kinds of order, and it takes one decision. *(read from the program)*

To select, the network picks one unit (in the same way it picks a target, explained below) and how many to take: the nearest 1, 2, 4, 8, 16 or all units within about four tiles of it. To get a larger group, it adds more units with another decision, or recalls a control group, which has no size limit. *(read from the program)* So “attack with this army” usually takes at least two decisions: first select the army, then give it the order.

Each selected unit gets its own copy of the order, and the replay counts each copy as an action. So one order to 40 selected units shows up as 80 actions in the replay: a select and an order for each unit. That is where the very high APM comes from. The number of real decisions is about 240 a minute, four per second. (*read from the program*)

The bot file also does a few small things a player would click through: one kind of train order goes to the least busy building and the other trains in every selected building, a gas building snaps onto the geyser, and add-ons are placed automatically. (*read from the program*)



One decision, 40 units selected, 80 actions in the replay.

IN PLAIN TERMS

Pluto's APM is not a sign of very fast thinking. It makes four decisions a second. The large number comes from counting each unit in a group separately.

Choosing a unit

Some orders need a unit: right-click this unit, cast on this unit, or select starting from this unit. For these, the options are not a fixed list like “Factory, Barracks, Starport”. They are the units Pluto can see right now, and those change every quarter second. So this choice works a little differently from the others.

It starts from the unit list, the same list as in step 1:

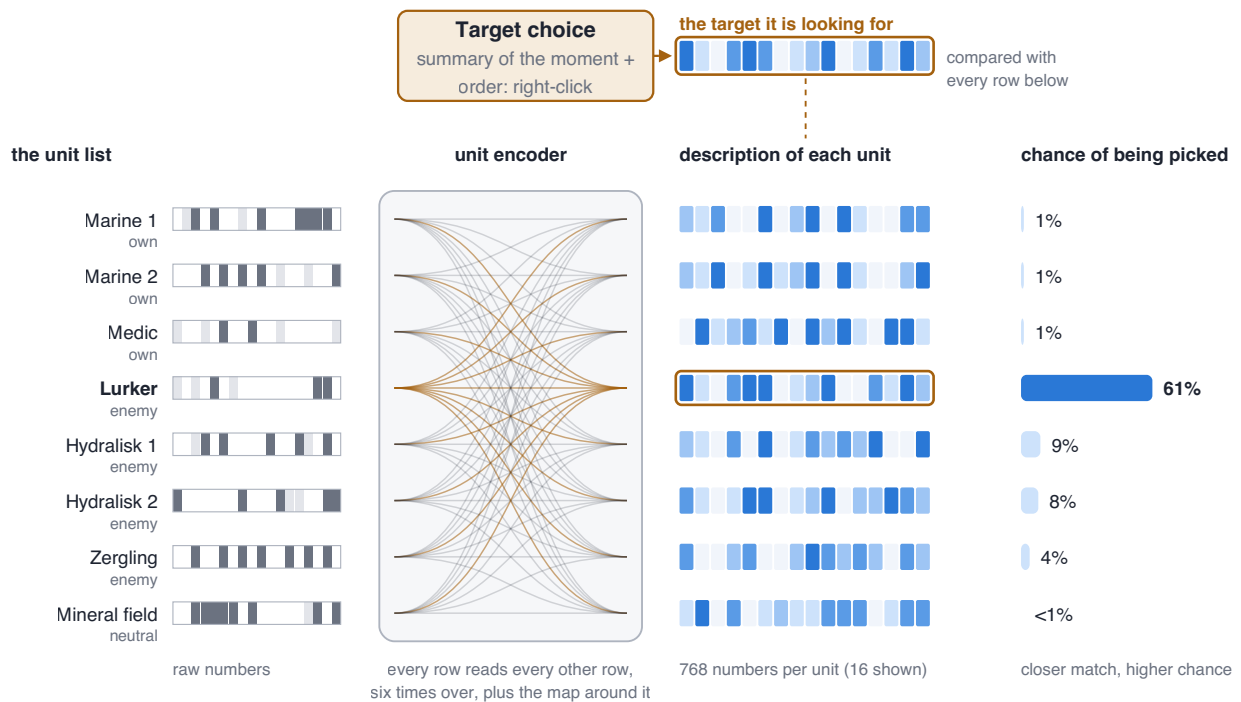
#	unit	side	tile	HP	weapon	doing now	status	selected	last ordered
1	Marine	own	41, 63	40/40	ready	attack-move	stimmed	yes	0.5 s ago
2	Marine	own	42, 63	28/40	reloading	attacking	stimmed	yes	0.5 s ago
Marines 3–12									
13	Medic	own	39, 64	60/60	—	healing	energy 112	no	4 s ago
15	Lurker	enemy	52, 60	125/125	reloading	attacking	burrowed, detected	—	—
16	Hydralisk	enemy	55, 57	80/80	ready	attacking	range upgrade	—	—
19	Zergling	enemy	51, 66	35/35	ready	moving	speed upgrade	—	—
all other visible units, anywhere on the map									
88	Mineral field	neutral	12, 8	—	—	—	1,500 left	—	—

Each row really holds 6 labels and 109 numbers. The ten columns above are a sample; the rest cover spells and timers, upgrades, cargo, direction, production, and which control groups the unit is in. Up to 768 rows, rebuilt at every decision.

Part of the unit list during a fight between Marines and Medics and a Lurker with Hydralisks and Zerglings. The fields are real; the values are made up for illustration.

By now every row has been through the unit encoder (step 3), so each row describes a unit in its situation. The choice is then made like this: *(read from the program)*

1. From the summary of the moment and the earlier choices, it builds a description of the unit it is looking for, in the same form as the rows.
2. It compares that description with every row. The more alike they are, the higher that unit's score.
3. From there it is the same as any other choice: the scores become odds, and one unit is drawn.



How a right-click target is picked. The colour strips stand for the descriptions; the description the network is looking for is closest to the Lurker's, so the Lurker gets the best odds. Strips and numbers are illustrative; the steps are real.

IN PLAIN TERMS

The network does not ask "which unit is the Lurker?". It asks "which unit looks most like what I want to hit right now?". If the answer is usually the Lurker, that is because training taught it to want units like the Lurker in moments like this one.

So there is no written rule like "shoot the Lurker first". Units that are visible but not detected, such as a shimmering Dark Templar, are on the list but are removed as targets before the draw, as they would be for a player. *(read from the program)*

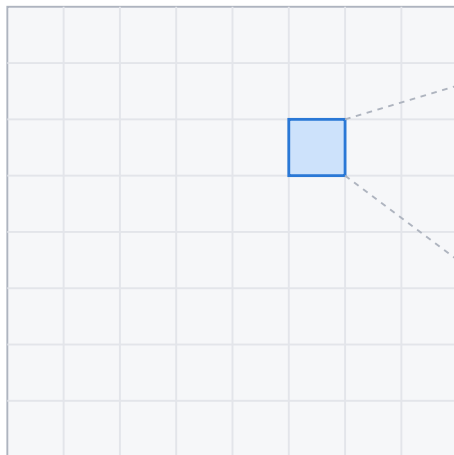
Choosing a place

Some orders need a place: move here, attack-move here, build here, Storm here. The map has 16,384 tiles, too many to score one by one, so the choice of place is split into two steps, using two of the map sizes from step 2: *(read from the program)*

1. It scores the 64 areas of the 8×8 grid and draws one area.
2. It scores the 256 tiles inside that area, using the detailed grid, and draws one tile.

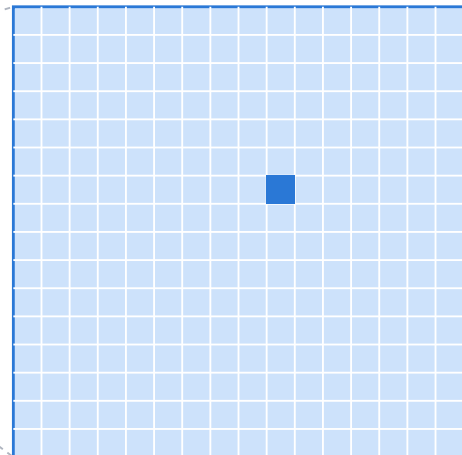
Both steps use the same score, odds and draw as every other choice.

Step 1: one of 64 areas



the 128 × 128-tile map

Step 2: one of 256 tiles in it



16 × 16 build tiles (32 px each)

First an area, then a tile inside it. This picks one exact tile anywhere on the map, without moving a screen.

Move and attack-move can aim anywhere on the map. Buildings and most spells can only be placed near the selected units. So to take an expansion across the map, Pluto first has to walk a worker there, as a player would. *(read from the program)*

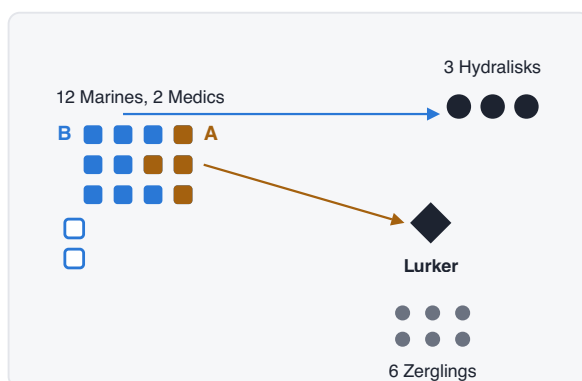
SO FAR

To give an order, Pluto first selects units, then makes up to three choices in a row. A unit is chosen by comparing a description of what it wants with every unit it can see. A place is chosen as an area first and then a tile inside it.

In a fight

Splitting fire

Because each order goes to whatever is selected, splitting fire takes several decisions in a row: select one group, give it a target, select the next group, give it another target. *(read from the program)*



- 1 select the 4 nearest a front Marine group A (orange)
- 2 right-click unit → the Lurker
A shoots the Lurker
- 3 select the 8 nearest a back Marine group B (blue)
- 4 attack-move → a tile by the Hydralisks
B fights whatever it meets there

Four decisions, about one second of game time.

Splitting fire takes four decisions, about one second of game time. The situation is made up for illustration; the kinds of order and the group sizes are real.

Marines against a Lurker

Take 12 Marines selected on open ground, 2 Medics beside them, and a burrowed Lurker with 3 Hydralisks and 6 Zerglings to one side. What the network tells the Marines depends on three things: how far away the Lurker is, whether it is detected, and where on the map this happens. (*computed from the weights*)

In the figure, the left side shows the order given in each case: each square is one distance, from 1 to 6 tiles, with the Lurker detected (top row) or not (bottom row). The right side shows the map, with an arrow from the Marines to where the move order sends them.

What 12 selected Marines are told to do

tiles between Marines and Lurker

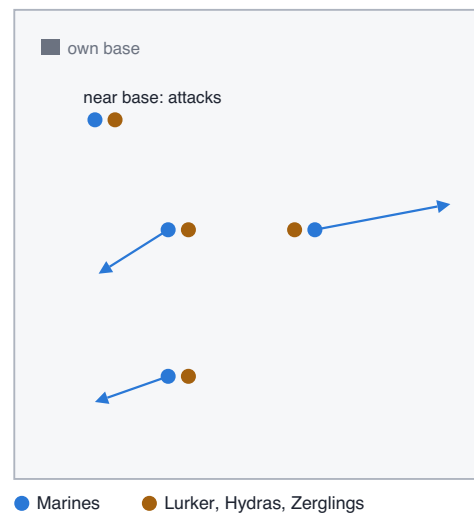
	1	2	3	4	5	6
detected	attack	attack	move	move	move	move
not detected	move	move	move	move	move	move

attack = right-click, 95% on the Lurker

move = a move order, about 95%

Each cell is a separate situation.

Where the move order points (4 tiles, detected)



Left: the order the network gives in each case. Right: where the move order points. The group moves about 20–40 tiles away from the enemy, whichever side the enemy is on.

- If the Lurker is detected and within 2 tiles, the Marines are told to attack it, and nearly always the Lurker rather than the Hydralisks or Zerglings.
- From 3 tiles out, or if the Lurker is not detected, the Marines are moved away from the enemy, in whichever direction that is.
- In one case placed near Pluto's own base, the Marines attack from 4 tiles instead of pulling back. A little further from the base, they pull back as usual.
- In all of these cases the network expects to lose, and it does not use Stim.

IN PLAIN TERMS

If the Lurker is close and can be shot, the Marines kill it. If it is further away, or cannot be shot, they back off. In one case near home, they stood and fought.

In a real game, terrain, a Science Vessel and what happened earlier also matter. So this shows how the network reads the situation, not exactly what it would do in every game.

How strong it is

Practice results

The stored record that Pluto starts from, for each opening, is its own practice results. Those numbers have a pattern. In practice, Terran won 83% of games against Zerg, and Zerg won 20% against Terran. Those two add up to about 100%. The same holds for every pair of opposite matchups. That is what you would expect if every one of these games was Pluto playing against Pluto: when one side wins, the other side loses. (*educated guess, but a strong one*)

Pluto as	vs Zerg	vs Terran	vs Protoss
Zerg	47%	20%	60%
Terran	83%	56%	59%
Protoss	42%	45%	52%

Practice win rates with the “free” option. Read across: the row is Pluto’s race, the column the opponent’s race.

The chart below shows the same practice results for every opening. Each row is one matchup, and each dot is one opening’s win rate in that matchup.



Each dot is one opening’s practice win rate in that matchup; the black tick is the “free” option.

The race it played mattered more than the opening it chose: in most matchups the dots sit fairly close together, while the matchups are far apart. Terran against Protoss is the exception, with a wide spread. Because these are results against itself, they say more about Pluto than about the races. Its Terran is clearly stronger than its Zerg.

When it gives up

During a game, the network also keeps an estimate of who is winning, and this can be shown as a graph on screen. If, after the first four minutes, the estimate stays at a near-certain loss for ten seconds, Pluto types “gg” and leaves. (*read from the program*)

How it learned

The training code was not released, so this is the least certain part.

The author describes the training as self-play. The network plays a very large number of games against copies of itself. After each batch of games, its weights are adjusted a little, so that the decisions made in games that went well get higher scores next time, and the decisions made in games that went badly get lower scores.

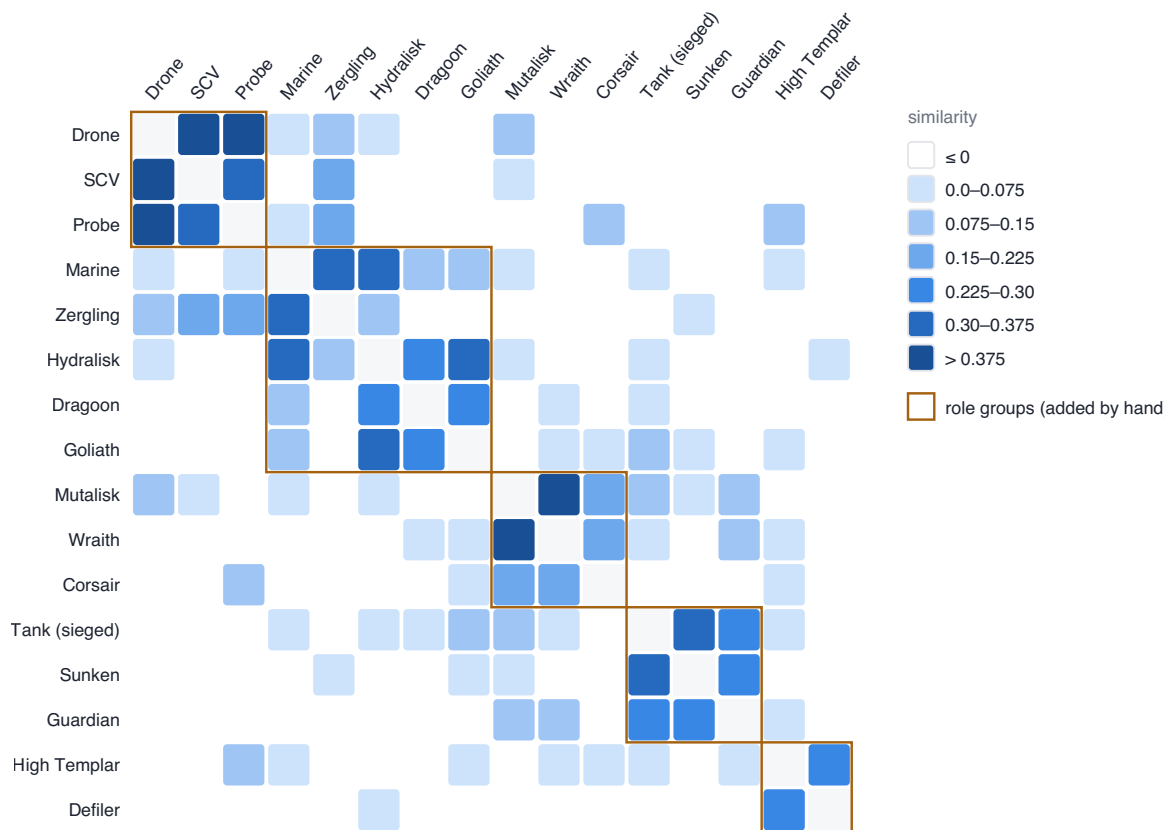
That leaves a problem: a game has thousands of decisions and only one result, so which decisions deserve the credit? Parts removed from the weights file before the release show that a *critic* was used to help with this. A critic is an extra part that estimates, at every moment, how likely a win is. If the estimate goes up after a decision, that decision probably helped. *(educated guess from the names in the file)*

A removed part also suggests that the network practised guessing hidden information, such as what the opponent has. Training settings left in the file suggest that the critic could see both players' units, which a player never can, and the file's original path suggests that it also played against a pool of earlier versions of itself. *(educated guess from the names and settings in the file)*

IN PLAIN TERMS

Pluto learned like a player who has played millions of games against themselves, with a coach who could see both screens and said after every moment whether things were going better or worse.

One result of this training can be seen in the weights. Each unit type has its own entry inside the network, and comparing these entries shows which unit types the network treats as alike. It put Drone, SCV and Probe together, Mutalisk near Wraith, and Marine near Hydralisk, though nobody told it which units are alike. *(computed from the weights)*



How similar the network's entries for 16 unit types are. Darker means more alike; the boxes mark role groups added by hand for comparison.

What this cannot tell you

- How exactly it was trained, for how long and on what hardware, and whether it started from human replays.
- How it behaves in real games. This document describes what the programs and the network are built to do.

How this was worked out

The analysis was done with Claude Code, Anthropic's AI coding assistant, running the Claude Opus 5.5 model. A person asked the questions and decided what to look into; the assistant did the reading, the tooling and the calculations. It read the text and settings inside the files, decompiled both programs (turned the machine code back into rough, readable code), and calculated things from the network's weights. Existing reverse-engineering notes by someone else were used as a starting point and checked against the code. Pluto's own programs were not run, and no real games were watched.

Sources

- Pluto repository, README and release notes: github.com/tscmoo/pluto, release `cog2026-2578600`

- CoG 2026 StarCraft AI Competition results: davechurhill.ca/starcraft/cog/results/2026/
- Independent reverse-engineering notes: github.com/hwkim3330/pluto-re
- BWAPI 4.4.0 headers: github.com/bwapi/bwapi
- Ghidra 12.1.4 (decompiler): github.com/NationalSecurityAgency/ghidra