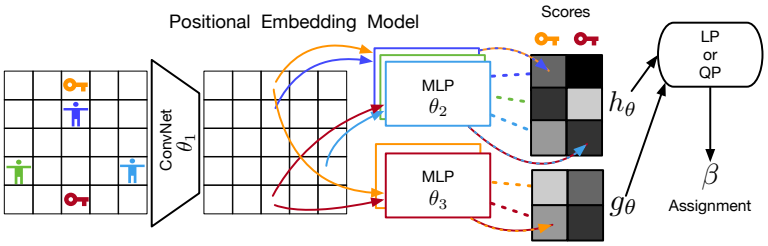


Figure 2: The PEM model for 3 agents and 2 tasks, arrows of interactions are only drawn for 2 agents each with one task, and one interaction of the two tasks. Agents are in cold (blue/green) colors, tasks are in warm (red/orange) colors.

A The Positional Embedding Model

A more complete illustration of the PEM model is reported in Fig. 2.

Algorithm 1 Structured Prediction RL algorithm - Worker thread

// p is the number of correlated steps, σ the exploration standard deviation
// We denote by Queue_p a queue of fixed size p. We assume we can sum all the elements in it.
$T \leftarrow 0$
repeat
  Start new a episode, $t \leftarrow 0$
  for all agent i and task j **do**
    Init noise history for the linear part to 0: $noise\_hist\_lin(i,j) = \text{Queue}_p(0)$
  end for
  for all pair of task j, k **do**
    Init noise history for the quadratic part to 0: $noise\_hist\_quad(j,k) = \text{Queue}_p(0)$
  end for
  repeat
    Observe state s_t
    for all agent i and task j **do**
      Compute $h_\theta(i,j,s_t)$ using the network
      Sample the actual action $H_{i,j}(s_t) \sim \mathcal{N}\left(h_\theta(i,j,s_t) + \sum noise\_hist\_lin(i,j), \frac{\sigma}{p}\right)$
      Store the current noise $noise\_hist\_lin(i,j)$.append$(H_{i,j}(s_t) - h_\theta(i,j,s_t))$
      Compute $\mu_{i,j}(s_t)$ the contribution of the agent to the task
      Compute $u_j(s_t)$ the capacity of the task
    end for
    for all pair of task j, k **do**
      Compute $g_\theta(j,k,s_t)$ using the network
      Sample the actual action $G_{j,k}(s_t) \sim \mathcal{N}\left(g_\theta(j,k,s_t) + \sum noise\_hist\_quad(j,k), \frac{\sigma}{p}\right)$
      Store the current noise $noise\_hist\_quad(j,k)$.append$(G_{j,k}(s_t) - g_\theta(j,k,s_t))$
    end for
    Compute the assignment using the constrained optimizer $\beta \leftarrow \text{solve}(H, G, \mu, u)$
    Execute the assignment in the environment, observe reward r_t
    The policy is $\pi(s_t) = [h_\theta(s_t), g_\theta(s_t)]$, and the action $a(s_t) = [H, G]$
    Send to the learner thread $(\pi(s_t), a(s_t), s_t, r_t)$
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until Episode ends
until $T > T_{max}$

Algorithm 2 Structured Prediction RL algorithm - Learner thread

// p is the number of correlated steps, N is the return length we use, σ the exploration standard deviation
γ is the discount factor, λ is the policy weight
repeat
Reset gradients $d\theta \leftarrow 0$
Receive N consecutive samples from a worker thread $[(\pi_1, a_1, s_1, r_1), \ldots, (\pi_N, a_N, s_N, r_N)]$
$R = \begin{cases} 0 & \text{if } s_N \text{ is terminal} \\ V(s_N, \theta) & \text{otherwise} \end{cases}$
for $t = N - 1$ **to** 1 **do**
$R = \begin{cases} r_t & \text{if } s_t \text{ is terminal} \\ r_t + \gamma R & \text{otherwise} \end{cases}$
Accumulate value loss $d\theta \leftarrow d\theta + \nabla_\theta \left(|R - V(s_t, \theta)|\right)$
Compute advantage $A_t \leftarrow R - V(s_t)$
Compute new policy $\pi_{\text{new},t}(s_t, \theta)$ using the network
Compute likelihood of action according to old policy (assuming $a_t \sim \mathcal{N}(\pi_t, \sigma)$) $l_{\text{old}} \leftarrow$ NormalPDF(a_t, π_t, σ)
Compute likelihood according to new policy (assuming $a_t \sim \mathcal{N}(\pi_{\text{new},t}, \sigma)$) $l_{\text{new}} \leftarrow$ NormalPDF$(a_t, \pi_{\text{new},t}, \sigma)$
Compute importance ratio $ir \leftarrow \frac{l_{\text{new}}}{l_{\text{old}}}$
Accumulate policy loss $d\theta \leftarrow d\theta + \lambda \nabla_\theta (ir * A * \log(l_{\text{new}}))$
end for
Take an optimization step according to $d\theta$.
until training ends

B Formal description of the training algorithm

In the experiments of this paper, we chose to build upon A2C with continuous actions, but in principle, any continuous policy-gradient algorithm could be used. The pseudo-code of the learning algorithm is presented in Algorithm 1 for the worker threads and Algorithm 2 for the learning threads.

C Search and rescue Experimental Setup

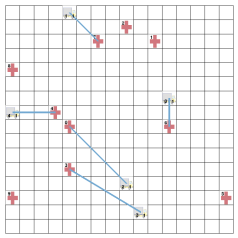


Figure 3: Search and rescue environment, with 5 ambulances and 10 victims.

C.1 Details of the Problem

We consider a search and rescue problem on a 16x16 grid. Each ambulance can move in any of the 8 adjacent cell at each step, and is assumed to be able to pick-up an infinite number of victims. At the beginning of each episode, m victims and n ambulances are spawned uniformly at random on the grid. An ambulance picks up a victim as soon as it reaches its cell, regardless of whether this ambulance was effectively assigned to that victim or if it visited the cell contingently. When it happens, the state of the victim changes to reflect the fact that that task is solved, yet nothing prevents the agent to keep assigning ambulances to it.

C.2 Training

We optimize the hyper-parameters of the learning procedure using a random search. We sample 128 sets of hyper-parameters for each combination of model/scenario, and we train the models during 8 hours on a Nvidia Volta. We report the performance of the best performing model after training according to the average reward on training problems. The parameters that were tuned are the following: learning rate of the value network (we sample $c \in [0, 5]$, uniformly at random, and use 10^{-c}), the learning rate of the policy network (same sampling), the variance of the random policy (uniformly in $[0.1, 2]$), the number of correlated steps in the exploration (uniformly in $[1, 10]$), the number of steps of the n-step return (uniformly in $[2, 5]$), and the optimization algorithm (SGD or Adam [16]).

C.3 Optimal algorithm

For the small instances of the problem, it is possible to use an exact algorithm (topline) that solves the Multi-Vehicle Routing Problem exactly. To do it efficiently, we first pre-compute for each subset of victims (there are 2^m possible subsets), the shortest path to visit them all, starting from each of the victims of the subset (that is, if a subset contains 5 victims, we compute the 5 shortest paths that go through all of them starting from each of the victims). This pre-computation phase is done using a dynamic programming algorithm that has a complexity of $O(2^m m^3)$. This allows us to compute, for a given ambulance, what is the optimal way of visiting a given subset of the victims: we simply need to choose which victim to visit first and then execute the optimal path that we have cached between the remaining ones. What remains to be done is to partition the victims in n sets, and assign those sets to the n ambulances, so that the maximal time required by the ambulances to visit their subset optimally is minimized. To solve this partition/assignment problem efficiently, we model it as an Integer Linear Program, and solve it exactly using an efficient branch-and-cut solver (SCIP [10])

C.4 Model

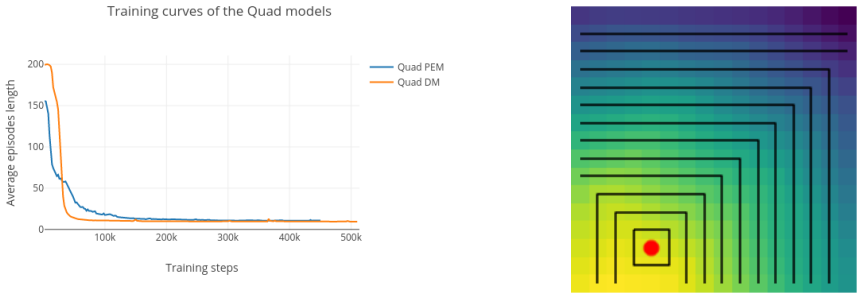
Features. The input of the networks consists in 4 feature planes: the first one contains a 1 if the corresponding cell contains a victim (0 otherwise), the second contains a 1 if the corresponding cell contains an ambulance, and the last two planes contain respectively the x/y coordinates of the entity (victim or ambulance) contained in that cell, if there is any.

Network. The value network is a residual network ([13]) made of 3 residual blocks of 2 convolutional layers, with kernel size 3, stride 1, and padding such that the output of each layer has the same spatial dimension as the input. Each convolutional layer outputs 32 feature planes, and we use ReLUs as non-linearities, as well as BatchNorm layers for regularization. The output of the last block is connected to a fully connected layer, which outputs one single value, the value function. For the direct model, we use a simple fully connected network, with 3 linear layers separated by ReLUs, with 32 hidden units each. The architecture of the PEM network is the same as the value network.

We train the model using a synchronous Advantage-Actor-Critic algorithm (A2C) (see [20] for the asynchronous version), using the ELF platform [30]. We batch 128 observations together, and run 256 agents in parallel.

C.5 Detailed results analysis

For the AMAX strategy, the experiments show that the model that has access to the the full view of the map (PEM) performs better on the domain where it was trained, w.r.t. a model that only has access to the features of the objects (DM). This means that the model learns something that is more useful (as a parameterization of the LP /QUAD) than simply the distance between objects, and probably takes into account some local patterns of the other objects on the map. However, this better performance in-domain comes at a cost of poor generalization: on the 8×15 scenario, the AMAX PEM model and the QUAD PEM model trained on the 2×4 gets stuck in a loop in a fraction of the evaluation episodes, for which they are not able to complete the overall problem (in the table we report the performance averaged over non-failing episodes). This is probably because the receptive fields of the network are significantly more saturated (their average activations have a higher value) in the 8×15 scenario than the models learned in the 2×4 training scenario. By contrast, the models trained on 5×10 is not affected by this. The phenomenon also occurs in the opposite direction: the QUAD PEM



(a) (b)

Figure 4: (left) Learning curves of the Quad models; (right) The distance computed by model (depicted as the density), is skewed towards the corner, compared to the ground-truth distance (depicted as the contours)

trained on 5×10 experiences failures on the smaller environment 2×4, indicating some form of overfitting.

Another interesting observation is that the AMAX PEM agent performs on average 7% better than the baseline. This is unexpected since the only features it has access to are the positions of the ambulance and the victim, thus it would be expected to learn to send each ambulance to the closest victim. We credit this development to the ability of the model to learn to break ties between equally close victims, which it does by choosing the victims that are on the furthest peripheral positions of the map. Since these positions tend to be the outliers, they are on average more difficult to access than the ones in the middle, so it is beneficial to favor them. The baseline breaks ties randomly and thus does not account for that. This can be observed in Fig 4b in which we plotted the similarity function learned by the model, by computing the score the model would give when the agent is located at the red dot and the task is located anywhere else. While the function it generates resembles the ground-truth distance, the distribution is slightly skewed towards the bottom-left corner.

Overall, the constrained strategies (LP and QUAD) perform better than the AMAX strategy, and the the QUAD strategy is better than the LP strategy, irrespective of the feature model used. This is an evidence that the structure introduced into the inference procedure is indeed able to improve the performance of the agent. These strategies also leverage the structure of the problem through the constraints on the assignment, which means that there is less room for learning something that cannot be computed directly from the pairwise positions using the additional information provided by the access to the full view. Since the extra information does not directly help, it slows down the learning procedure, and hinders the final performance. This can be seen in Fig 4a, where the PEM agent is slower to converge, and converges to a worse performance than the DM agent, when both use the QUAD strategy.

D StarCraft experimental setup

All the experiments are played on a fully empty map, where the units are centered in the middle, outside their respective fire range. Because of the kind of movement policy we use, the units never go near the edge of the map, hence there is no collision involved but the collisions between units themselves. The bot takes its actions every 3 frames, but we reevaluate the target assignment only every 6 frames. If the target of a unit dies in the meantime, then it does nothing until the next re-assignment. The reward at time t is defined as

$$r_t = (\text{ourHealth}_t - \text{ourHealth}_{t-1} + \text{theirHealth}_{t-1} - \text{theirHealth}_t)/\text{ourHealth}_0.$$

At the beginning of every battle, the units are spawned according to a seeded normal distribution, which is skewed towards the Y dimension, in order to form more coherent initial spread of the units. Even though the starting positions are deterministic with respect to the seed, the outcome of a battle is not, even if the policy of the players are totally deterministic. This is due to the internal randomness of the game engine, which affects things like the random miss probability of each attack (any attack has a $\frac{1}{256}$ probability of failing) and the initial orientation of the units.

The opponent army is controlled by the built-in AI. In practice, it means that we give an "attack-move" order to the opponent, with the target position being the centroid of our units. This order is repeated every 60 frames with an updated centroid. The attack-move order causes the built-in AI to take over the unit, globally moving it towards the target position and attacking any visible unit along the way.

We wish to emphasize the fact that all the details of the experimental protocol have a significant impact on the outcome of the battles. In particular, the frequency at which the opponent's attack-move command is re-issued matters: if the frequency is too high, then the units tend to attack less, because spamming orders have some weird effects on the game engine. Conversely, if the frequency is too low, then the attack point might be significantly off the centroid of our army, leading to sub-optimal attacking behaviour. Similarly, the initial positions of the units plays an important role. As a result, the exact win-rate are not directly comparable with previous works, where neither precise details of the setup nor source code are available. The differences can be observed even in the win-rate of the baselines heuristics. For this work, we refer to the source code in the supplementary material for the exact details used.

D.1 Scenarios

We consider three different kinds of scenarios:

Wraith These are ranged flying units, which means that they don't collide with any other unit. We denote these scenarios as wNvM where N is the size of our army, and M is the size of the enemy army. Following the previous works, we train on the imbalanced scenario w15v17, where the two additional units given to the opponent are required to make the scenario challenging.

Marines These are ranged ground units, which do have collisions. We denote these scenarios as mNvM where N is the size of our army, and M is the size of the enemy army. Since the built-in AI has a better control policy for ground units, we train on the balanced scenario m10v10, which is challenging enough.

Zergling-Hydralisk Zerglings are fast melee units (they can only attack if they are in contact of their target), while Hydralisks are slower ranged units. In this scenario, we investigate the opportunity to learn distinct behaviours depending on the type of the agent or its task. To further amplify their relative capabilities, we give the Zerglings a speed boost ("Metabolic Boost" upgrade), and the Hydralisks a range boost ("Grooved Spines" upgrade). We denote these scenarios as "zh$N$vM" which correspond to N Zerglings and N Hydralisks versus M Zerglings and M Hydralisks. We train on zh10v10.

D.2 Baseline Heuristics

Here is a description of the heuristics used as a comparison:

Closest (c) Each unit independently picks the units that is the closest, as measured by the distance function used by the game engine. Ties are broken using the unit internal ID, which is randomly assigned at the beginning of the game but consistent for the duration of the episode.

Weakest Closest (wc) All units are collectively assigned to the weakest enemy unit. The distance is used as a tie breaker.

Weakest closest No-Overkill (wcnok) . We select the weakest-closest enemy unit, then assign greedily in an arbitrary order as many units as possible as long as the total sum of damage to this unit is lesser than its health. When this enemy is saturated, if some of our units don't have a target yet, then we select the second weakest-closest, and so on until all enemy units have been exhausted or all our units have a target.

Weakest Closest No-Overkill No Change(wcnoknc) Same as wcnok, but once a unit starts attacking a target, it keeps doing so until the target dies. When the target dies, a new target is computed as in wcnok Keeping the same target can reduce some instability in the assignment found by wcnok, but can lead to over-killing.

Experiment	Model	Learn. rate	Policy-loss weight λ	Explor. std-dev σ	Returns length	# correlated steps
Marine	QUAD	4.272e-05	1.585e+01	2.910e+00	5	10
	LP	2.280e-05	2.157e-03	7.017e-01	6	5
	AMAX	4.330e-05	5.396e-01	2.418e+00	2	2
Wraith	QUAD	2.826e-05	2.514e+02	1.899e+00	6	7
	LP	5.600e-05	5.534e-01	4.244e-01	4	6
	AMAX	2.525e-05	6.847e+01	1.994e+00	8	10
Zergling-Hydra	QUAD	5.325e-05	6.277e-01	7.185e-01	3	3
	LP	6.534e-05	4.455e-02	2.902e+00	8	1
	AMAX	1.885e-05	2.119e-02	1.881e+00	3	10

Table 3: Best hyper-parameters found through random search on the different models

Weakest Closest No-Overkill Smart (wcnoks) Same as wcnoknc, except that the target is kept only as long as it doesn't risk causing over-killing. When it does, a new target is computed as in wcnok.

Random No change (rand-nc) Each unit pick a target at random at the beginning of the episode, and keep attacking it until it is dead. When that happens, a new target is picked randomly.

D.3 Model

The scoring models h_θ and g_θ are fully-connected networks consisting of 3 linear layers with ReLU. To compute $h_\theta(i, j)$, we give as input to the network the concatenation of the features of ally unit i and the features of enemy unit j, along with 2 additional features: a boolean flag that indicates whether i was attacking j in the previous step (this is meant to facilitate temporal consistency of the actions), and the distance between both units, as computed by the internal game engine. The input of $g_\theta(j, k)$ only contains the features of enemy units j and k, with no additional features. In this experiment, we use 64 agents in parallel during training, and batch 32 observations together. The reward is joint, and consist in the normalized instantaneous delta of health of our units and the enemy units.

D.4 Hyper-parameters

All models use the same network architecture, using 3 layers of fully-connected with 32 units and ReLUs in-between. The value function uses a residual network made of 5 blocks of 2 convolutional layers, with 16 feature layers and kernel size 3, operating on a 100x100 view of the state. The discount factor γ is set to 0.999.

For all the models, we sample randomly the remaining hyper-parameters, as follows: For the learning rate we sample c u.a.r in $[-6, -3]$, and use 10^c, for the policy-loss weight λ, we sample d u.a.r in $[-3, 3]$ and use 10^d, the exploration standard deviation σ is sampled u.a.r in $[0.1, 3]$, the return-length used in A2C is an integer sampled u.a.r in $[2, 10]$, and the number of correlated exploration steps is an integer sampled u.a.r. in $[1, 10]$. We report the best parameters found for all model in all experiments in Table 3

D.5 Detailed Results

In Table 4, we report the performance of all the heuristics, as well as the confidence intervals. In the Marine and Zergling-Hydralisk scenarios, the "closest" heuristic tends to dominate, suggesting that there is not much value in learning a focus-firing policy. However, in the Wraith scenarios, the best performing heuristic are the more complicated ones that focus fire on the weakest while preventing over-killing. This is the type of policies that only the QUAD model is able to represent, explaining its dominance in this scenario. AMAX and LP manage to do better than the closest heuristic by using enemy positions as a way to reach a consensus: they tend to favor picking targets which have a high y coordinate, for example, so that they all end-up picking the same in the beginning, creating ad-hock collaboration. However, as the battle goes on, this coordination scheme is not as efficient, since the battle tend to be messy.

		Heuristics						RL		
Train	Test	c	wc	wcnok	wcnoknc	wcnoks	rand-nc	LP DM	QUAD DM	AMAX
m10v10	m5v5	0.77 ± 0.03	*0.88 ± 0.02*	0.56 ± 0.03	0.86 ± 0.02	0.83 ± 0.02	0.15 ± 0.02	**0.90 ± 0.02**	0.83 ± 0.02	0.84 ± 0.02
	m10v10	*0.77 ± 0.03*	0.44 ± 0.03	0.00 ± 0.00	0.45 ± 0.03	0.56 ± 0.03	0.01 ± 0.01	**0.94 ± 0.02**	0.83 ± 0.02	0.82 ± 0.02
	m10v11	*0.25 ± 0.03*	0.07 ± 0.02	0.00 ± 0.00	0.05 ± 0.01	0.11 ± 0.02	0.00 ± 0.00	**0.52 ± 0.03**	0.28 ± 0.03	0.29 ± 0.03
	m15v15	*0.75 ± 0.03*	0.03 ± 0.01	0.00 ± 0.00	0.14 ± 0.02	0.18 ± 0.02	0.00 ± 0.00	**0.92 ± 0.02**	0.69 ± 0.03	0.77 ± 0.03
	m15v16	*0.40 ± 0.03*	0.00 ± 0.00	0.00 ± 0.00	0.03 ± 0.01	0.02 ± 0.01	0.00 ± 0.00	**0.68 ± 0.03**	0.32 ± 0.03	0.43 ± 0.03
	m30v30	*0.69 ± 0.03*	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	**0.74 ± 0.03**	0.06 ± 0.02	0.36 ± 0.03
w15v17	w15v17	0.07 ± 0.02	0.00 ± 0.00	0.58 ± 0.03	0.33 ± 0.03	*0.81 ± 0.02*	0.01 ± 0.01	0.53 ± 0.03	**0.89 ± 0.02**	0.30 ± 0.03
	w30v34	0.01 ± 0.01	0.00 ± 0.00	0.36 ± 0.03	0.31 ± 0.03	*0.90 ± 0.02*	0.01 ± 0.01	0.76 ± 0.03	**0.99 ± 0.01**	0.37 ± 0.03
	w30v35	0.00 ± 0.00	0.00 ± 0.00	0.10 ± 0.02	0.08 ± 0.02	*0.60 ± 0.03*	0.01 ± 0.00	0.56 ± 0.03	**0.94 ± 0.02**	0.24 ± 0.03
	w60v67	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	*0.07 ± 0.02*	0.00 ± 0.00	0.33 ± 0.03	**0.72 ± 0.03**	0.13 ± 0.02
	w60v68	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	*0.01 ± 0.01*	0.00 ± 0.00	0.21 ± 0.03	**0.52 ± 0.03**	0.07 ± 0.02
	w80v82	0.00 ± 0.00	0.00 ± 0.00	0.09 ± 0.02	0.08 ± 0.02	*0.32 ± 0.03*	0.00 ± 0.00	0.11 ± 0.02	**0.36 ± 0.03**	0.03 ± 0.01
zh10v10	zh10v10	*0.86 ± 0.02*	0.26 ± 0.03	0.00 ± 0.00	0.54 ± 0.03	0.64 ± 0.03	0.00 ± 0.00	**0.90 ± 0.02**	0.83 ± 0.02	0.84 ± 0.02
	zh10v11	*0.30 ± 0.03*	0.01 ± 0.01	0.00 ± 0.00	0.04 ± 0.01	0.09 ± 0.02	0.00 ± 0.00	**0.46 ± 0.03**	0.24 ± 0.03	0.40 ± 0.03
	zh10v12	*0.03 ± 0.01*	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	**0.06 ± 0.02**	0.01 ± 0.01	**0.06 ± 0.01**
	zh11v11	***0.87 ± 0.02***	0.15 ± 0.02	0.00 ± 0.00	0.38 ± 0.03	0.56 ± 0.03	0.00 ± 0.00	**0.87 ± 0.02**	0.75 ± 0.03	0.80 ± 0.02
	zh12v12	***0.85 ± 0.02***	0.05 ± 0.01	0.00 ± 0.00	0.23 ± 0.03	0.42 ± 0.03	0.00 ± 0.00	0.82 ± 0.02	0.64 ± 0.03	0.75 ± 0.03

Table 4: Results on StarCraft: average win-rate of the different methods and all the heuristics. Bests results are in bold, the best heuristic on each scenario is in italics. We report 95% confidence intervals (using the Normal approximation interval).

In the Marine scenario, all three models usually learn to wait in the beginning of the battle, and start picking a target only when the enemies are about to get in range. This strategy allows them to keep a rather good formation, which they would otherwise last, had they picked a wrong target. This gives them a little edge in the battle, and then they proceed by following a strategy that resemble attacking the closest unit.

In the Zergling-Hydralisk scenarios, the models learn to wait in the beginning as well. This is an exploit on the rushing behaviour of the opponent: the faster zerglings of the enemy will engage first, while its hydralisks are lagging behind. This allows the model to first clear up the zergling wave with the combined forces of their zerglings and hydras, before turning to the enemy hydralisks.

We provide some replay video as a supplementary file.