Project Darwin Final Report

ECE324

December 6th, 2020

Aidan Lawford-Wickham, Rui (Eric) Liang, Young Seok Seo

Word count (excluding figures): 1989

Permissions

- I, **Aidan Lawford-Wickham**, give permissions to the following:
    a) Presentation video being posted
    b) This final report being posted
    c) The source code of the project being posted

- I, **Rui (Eric) Liang**, give permissions to the following:
    a) Presentation video being posted
    b) This final report being posted
    c) The source code of the project being posted

- I, **Young Seok Seo**, give permissions to the following:
    a) Presentation video being posted
    b) This final report being posted
    c) The source code of the project being posted

Introduction

The prospect of having an intelligent computer that can balance rational thought with human-like creativity could potentially be a groundbreaking innovation for applications like personal assistants, healthcare, military, self-driving cars, and more. When one thinks about what it means for an intelligent being to act in human ways, one may recall the Turing Test, which was proposed in 1950 to test artificial beings on a number of topics deemed 'important' to true intelligence.

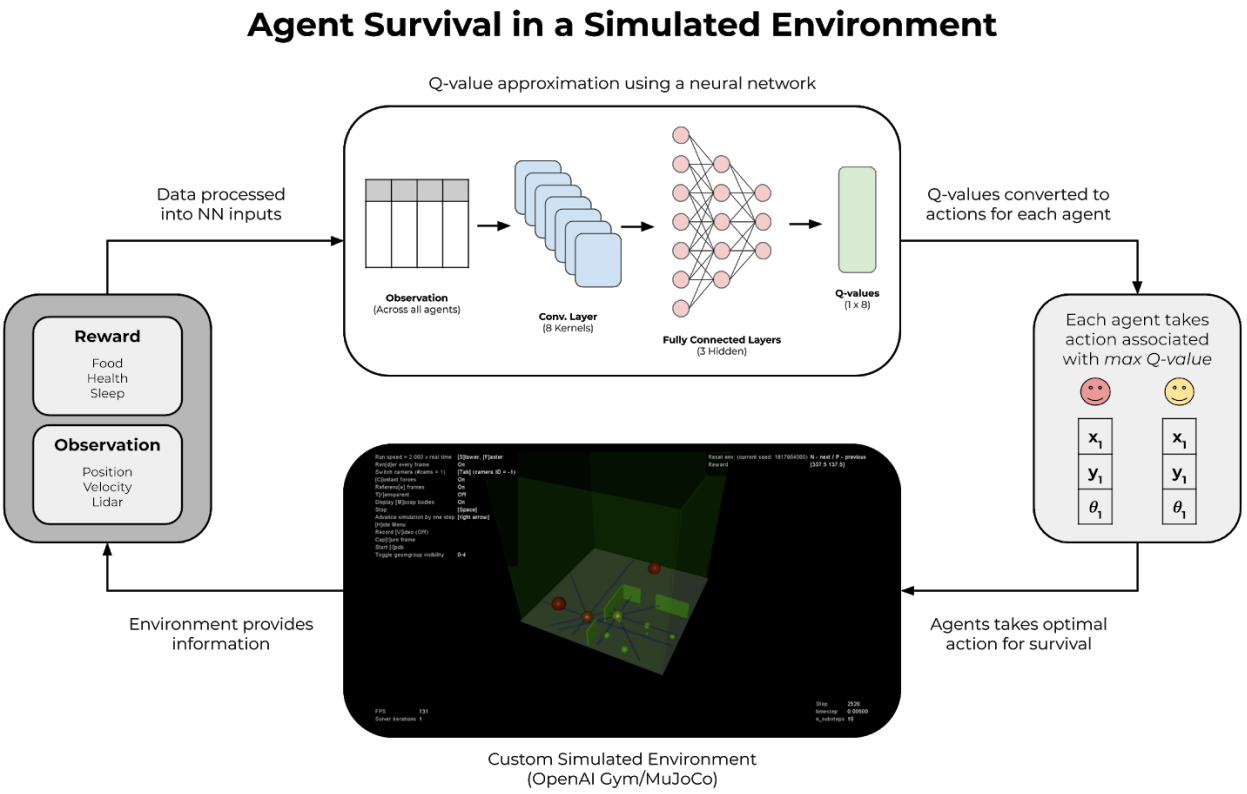
Another way to think about whether an artificial being is intelligent is asking whether it could survive if placed an a 'realistic' world simulation alongside humans and/or other artificial beings, one where it needed to fulfill human-like survival behaviours like eating, sleeping, building shelter, and strategizing with or against others also living in the simulation to better the chance of survival. This is the challenge that our project seeks to investigate, that is, **whether we can build an intelligent entity capable of surviving when placed under pseudo-realistic constraints for survival in a simulated world**. Albeit less rigorous for proving intelligence than measures like the Turing Test, and despite only being pseudo-realistic, our line of thought is that if simulations of this sort were made increasingly complex over time, eventually they may reach a realistic level and the artificial beings capable of surviving within them could reach human-level survival intelligence.

Background & Related Work

When thinking about placing an intelligent entity inside a simulation and deducing an algorithm to generate actions, one can quickly see a parallel to reinforcement learning (RL). Past RL algorithms have proved highly successful at making an 'agent' play through the game-like environment at a superhuman level (e.g. OpenAI Five, which defeated the world champion Dota2 team, or Deepmind's AlphaGo [1][2]). Thus, modelling the survival simulation as a game-like RL problem can be a highly beneficial approach to this problem. The agent can be modeled as a neural network or other optimizer which can determine the best action to take at any step (**Appendix A**).

A successful example is OpenAI's Multi-Agent Hide and Seek [3]. Using RL methods to influence the actions of the agents, they were able to observe agents coming up with creative, human-like strategies to succeed over their competitors. By constructing a similar simulation environment, then adding our own pseudo-realistic survival game rules, we see a promising RL approach to creating intelligence capable of strategically competing for survival. Another motivation for approaching this problem with RL is its unsupervised nature; the agents will only be restricted by the environment constraints that we set. Instead, if they learn to survive it will be only through their own exploration and learning.

Illustration / Figure



Data & Data Processing

In the Darwin project, the "data"—consisting of each agent's state and the reward they receive from the environment—is continuously collected and processed (**Appendix B**). As such, it is essential to discuss the multiple Darwin Environments that we constructed.

Since the goal of the project is to test agents' intelligence in a reality-like survival environment, it is essential to construct the following objects:

1. Two or more learning agents
2. Multiple food-sites, as agents need to consume food in order to survive
3. Walls/obstacles in the environment to increase complexity
4. Sensors on the agents allowing them to detect surroundings.

Our 4 different environment designs:

	Baseline 1	Baseline 2	Survival
Wall placement	Quadrant		
# of Agents	2	4	2
Agent placement	Outside quadrant		
# of Lidars per agent	8		
# of Food-sites	10	10	5
Maximum food health (per food site)	50	50	15
Food-sites placement	Within quadrant		
Reward function	Discussed in section below		
Addition features	n/a	n/a	Sleep sites

***Table 1**. Summary Table of 4 constructed environments*

Simulation Environments

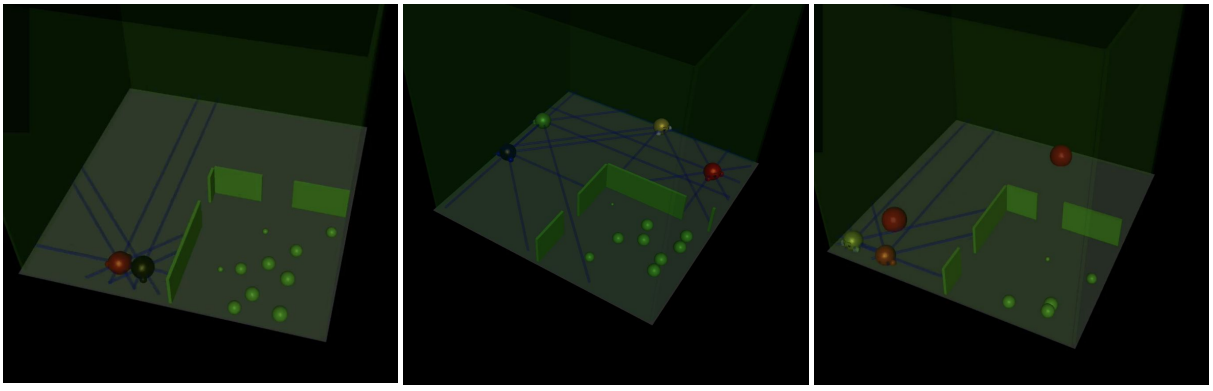


***Figure 1.** From left: Baseline 1, Baseline 2, Survival*

Components

- Agents
  - Action represented by an array of velocities (x, y, θ) with possible values +/-1.
  - 8 lidars representing lines of vision.

- Food

- Agent gains reward inside a predefined eating threshold.

- Reward
  - We want the agents to locate and move towards the food in the shortest number of steps.
  - *Number of steps*
    - "Punish" agents based on the number of steps taken to find food, by assigning a reward of -α at each step.
  - *Food*
    - Assign rewards based on distance between agent and food to aid in "learning". To guide the agents toward food, the "eating threshold" covers the entire environment with a decaying reward function:

$$food\ reward\ =\ 50\ \times sech(\lambda d)$$

      - λ: deceleration factor used to flatten the *sech* curve (between 0 and 1 if $d > 0$).
      - *d*: distance between agent and food
  - Through experiments, α = 1 and λ = 5 work the best; Thus:

$$Reward\ =\ -1\ +\ 50\ sech(5d)$$

  - The survival environment includes sleep sites, where agents in the proximity gain "sleep health" — an additional reward. The final proposed reward function for the survival environment is:

$$Reward = \begin{cases} +50, & health_{food} + health_{sleep} \leq 50 \\ -50, & health_{food} + health_{sleep} > 50 \end{cases}$$

The environment returns a dictionary of the agents' states and rewards, visualized below:

	Agent 1	Agent 2
Agent positions		
Agent velocities		
Lidar		

Each agent has three arrays that contain 8 coordinates for each agent's joints (default from MuJoCo), 8 rotational quaternion and velocities for joints, and the vision data from 8 lidar (to match other observations) [4]. Thus, for an environment with two agents, the dictionary can be split into two 8x3 matrices per agent, which can be concatenated together to form the final

8x6 matrix. The reward array is used to track the agents' performance during training (see Results).

The same process is repeated for the other 3 environments with slightly different input and output sizes.

Baseline Model

The simplest model used was Q-learning. Through the Bellman equation, optimized with dynamic programming, the Q ("quality") values can be found for each state and action without machine learning [5]. The agent explores a simple environment over a fixed number of episodes, updating the maximum Q value (optimal action for each state) using a lookup table [6].

For context, the Q-learning baseline supported that the agent requires a neural network in order to survive in our simulated environment. Due to the environment's complexity and number of states, exploring and storing every possible state is too costly, so the agent failed to converge to an optimal policy with this approach.

Architecture

The network architecture for this problem focuses on using an agent's observation of the environment (position, velocity, and lidar data) to make an informed decision. Each of the agents' observations are stacked into a matrix of (*n_agents+1*) columns, which can be processed using a 2D convolutional layer. This convolution ensures that the neural network accurately captures the relationship between each agent's current observation (input columns).

The data then enters a fully connected MLP which learns the relationship between the input matrix and the output Q-values. Batch normalization is applied to prevent exploding gradients. The network outputs 8 Q-values associated with each possible action. The maximum Q-value is the next "step" for the agent, which maximizes their reward and thus chances of survival (**Appendix A**).

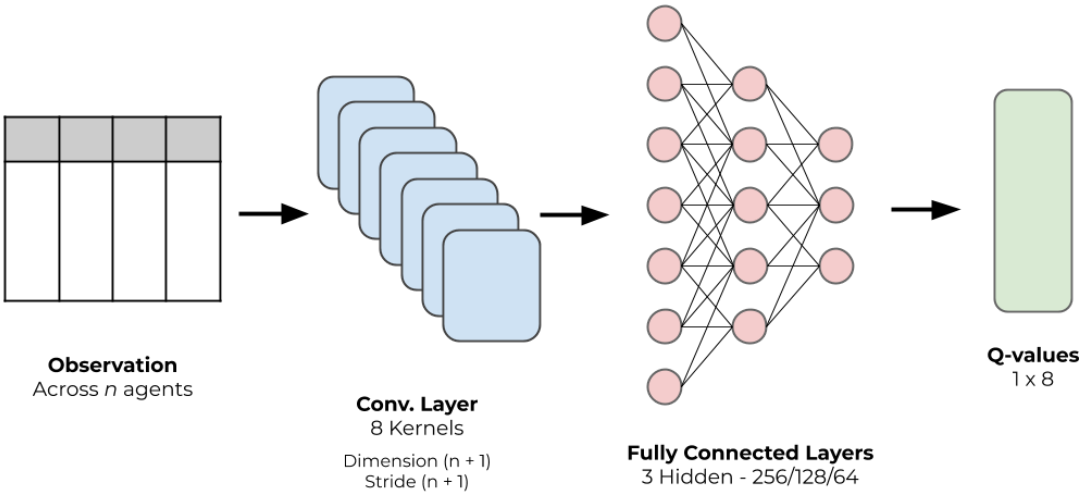


Figure 2. *Final CNN Model Architecture*

Since 'labels' or optimal actions are not known ahead of time, the training procedure of the *main network* (above), uses the *target network* to predict labels. Then, a MSE loss can be computed and used for SGD. Using the best action predicted by the main network, the training loop simulates another step, producing another observation that is fed into the target network generating the agent's *next* response. The output of the target network is then applied to the Bellman equation to generate labels [5]. The target network is not trained; instead, its parameters are copied from the main network at a set interval of steps [7]. This interval is an important hyperparameter that allows the main network to converge. The model also collects every step into a "replay buffer", which acts as a constantly-changing training dataset for random sampling [8].

Qualitative & Quantitative Results

The two main metrics of success used to judge whether the agent was successfully surviving in our environment were loss (MSE loss from model) and average reward collected over training (**Appendix C**).

In our baseline environment, the combined loss and reward plots show that the agents collectively were able to converge on an optimal policy during training. However, we noticed competitive behaviour between two agents; as seen in the individual reward plot, *agent0* took bolder actions earlier on, which increased its loss at first but eventually allowed it to collect more rewards than *agent1*. Following a series of "wins" for *agent0* through steps *5000~10000*, the reward plot becomes more balanced and even shows *agent1* winning most of the steps between *10000* and *15000*. We believe that longer training would display more strategies emerging between the two competing agents.

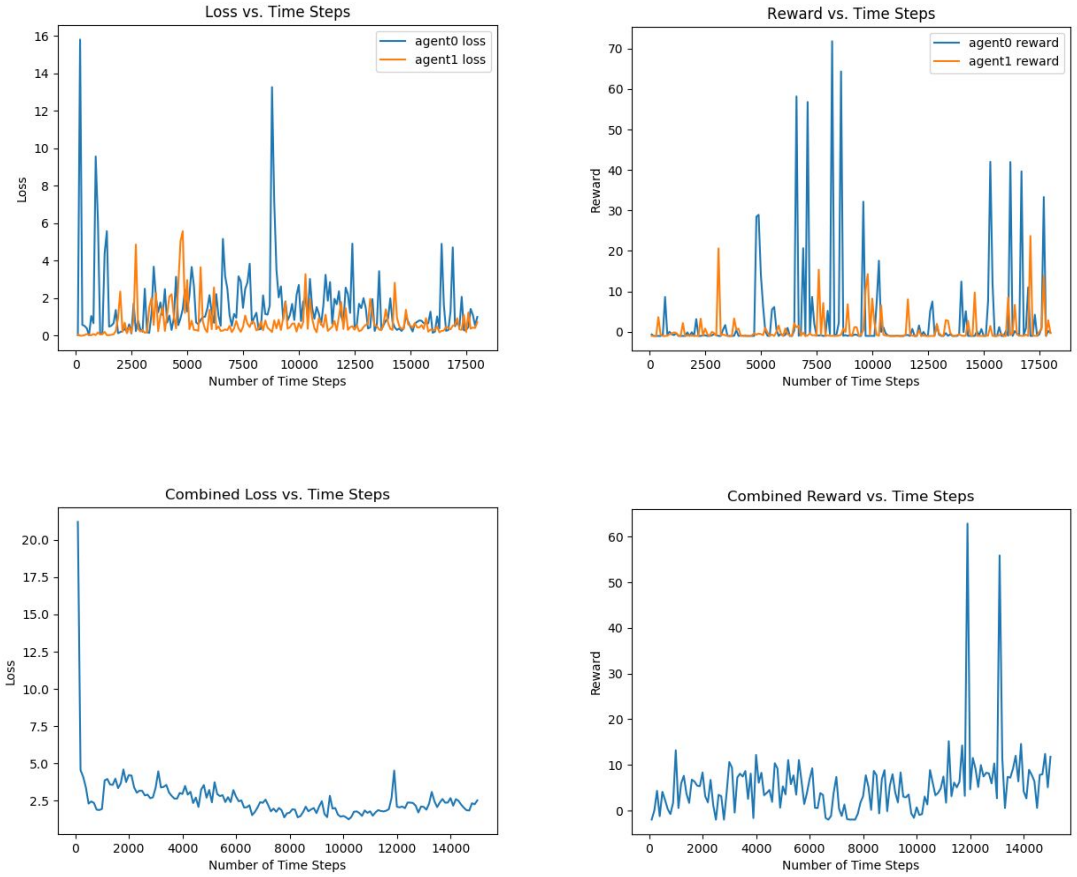


Figure 3. *Quantitative results for the Baseline environment.*

Baseline with *four* agents showed a similar behaviour. The loss trend was decreasing, signifying that all four agents were converging. Following an initial stage of random exploration, *agent0 (blue)* begins to succeed in collecting the majority of the rewards. *Agent3 (red)* adapts to this emerging threat and builds a strategy to maximize its reward while competing with *agent0*, whose reward begins to decrease. Near the end of training, *agent1* adjusts its strategy to account for the other two dominant agents, gaining higher reward.

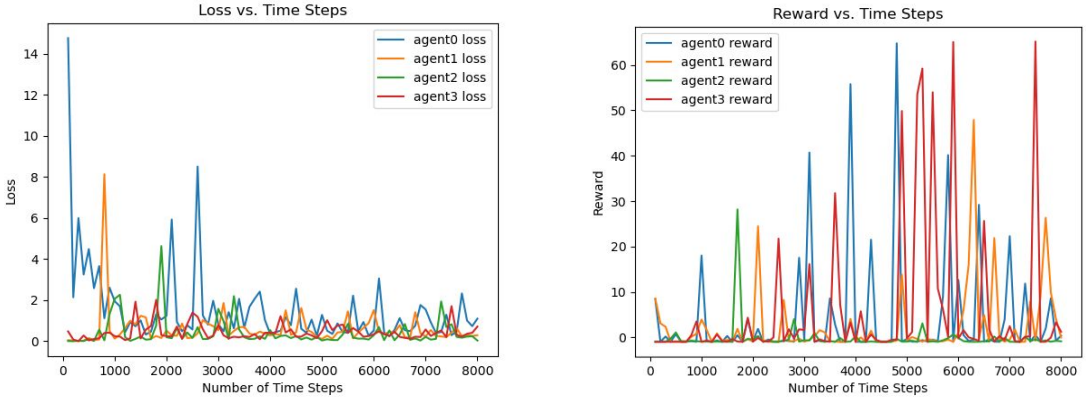


Figure 4. *Quantitative results for the Baseline environment with four agents.*

The results for the Survival environment are slightly less definitive due to its increased complexity. The loss plot shows a decreasing trend, but the magnitude of the loss is still much larger than other environments; a longer training loop would allow the loss to stabilize. The oscillating behaviour in the reward plot also suggests a need for an extended training period.

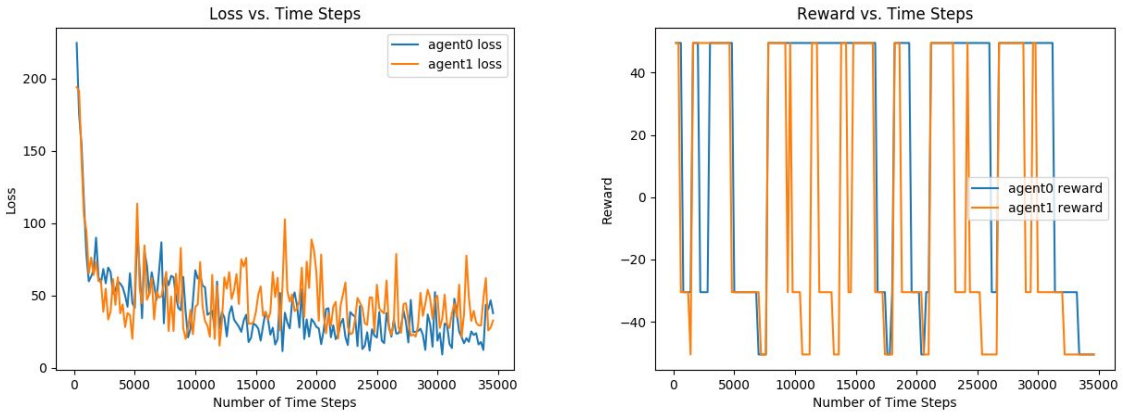


Figure 5. *Quantitative results for the Survival environment.*

The training results were also recorded in video formats for evaluation purposes, and the qualitative assessment of the agents' ability to survive was based upon whether the agents could visibly navigate to the food/sleep sites. For further research purposes, the models trained on the baseline environment were applied to navigation and survival environments to observe the effects of transfer learning. See these videos on our website [10].

Discussion and Learnings

Although the models in each environment converged so that one of the agents ‘won’, this does not signify that models of the ‘losing’ agents did not converge; rather, they are temporarily being defeated by another model that converged to a more optimal policy during training. For example in the Baseline 2 environment, three agents rush towards the nearest door to collect the food but they all get stuck trying to enter at the same time. The fourth agent recognizes this and enters through the far door, travelling extra distance but reaching the food first [11]. This is an exhibition of autocurricula, as observed in other RL problems [9]: one agent inevitably finds a better solution to the problem, making the environment more difficult to solve for other agents. In the future, training for longer or introducing more complexity to the environment might allow us to see the losing agents emerge as the eventual winner.

We also found that although a model may find the set of actions leading to the highest reward, it may not be converging to a set of actions that meet our actual intention, which is for agents to actually survive. Whether these two things align is dependent on the reward function, and early on our reward function was not representative of our desired outcome. As

a result, we found simplifying the environment and reward could act like creating an ‘overfit dataset’, which would allow us to first concentrate on debugging our neural network.

Ethical Framework

The Darwin project can be considered as a research effort in the field of RL. Stakeholders are listed below:

1. RL researchers
    a. The environments, reward, and model that we designed can be applied elsewhere. The environments can be used to train and test other RL algorithms, and mathematical concepts reward functions and model architecture can be evaluated in the design stage of other projects.
    b. High beneficence/nonmaleficence; low autonomy as our project can hinder the work of other researchers who hope to pursue similar research.

2. Team Darwin
    a. As the project involves RL, we were able to explore a different branch of machine learning compared to supervised learning that was taught in class, allowing us to explore more opportunities involving RL in the future.
    b. High beneficence/non-maleficence/autonomy.

3. Robotics companies and research institutes
    a. An important application of the Darwin project is transfer learning [12]. Robotics companies could apply our models/simulation to train complex robots that otherwise require significant resources to develop and test. However, with the help of concepts in this project like simulation, their process could be optimized.
    b. High beneficence/non-maleficence/autonomy.

References

[1] https://docs.google.com/document/d/1str7oBVTUKIl2vMH90PFwD3jLDett1shij9nA5DrgjI/edit

[2] https://deepmind.com/research/case-studies/alphago-the-story-so-far

[3] https://arxiv.org/abs/1909.07528

[4] http://www.chrobotics.com/library/understanding-quaternions

[5] https://towardsdatascience.com/reinforcement-learning-markov-decision-process-part-2-96837c936ec3

[6] https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56

[7] https://arxiv.org/abs/1509.06461

[8] https://towardsdatascience.com/reinforcement-learning-with-hindsight-experience-replay-1fee5704f2f8

[9] https://arxiv.org/pdf/1903.00742.pdf

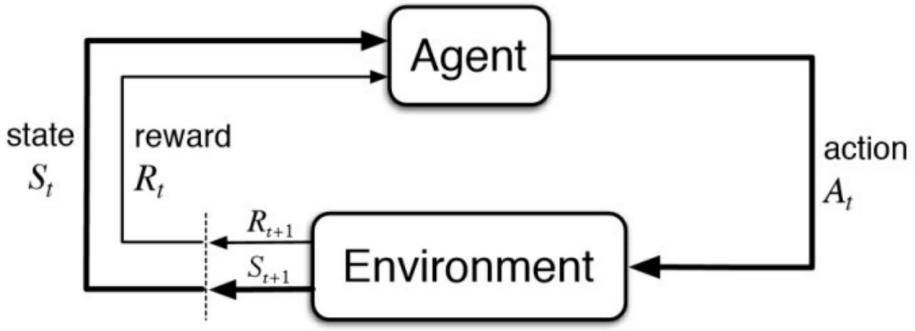
[10] https://darwin-app.web.app/showcase/baseline1

[11] https://darwin-app.web.app/showcase/baseline2

[12] https://tspace.library.utoronto.ca/handle/1807/70527

Appendix

Appendix A: Formulation of a typical RL problem



A typical RL problem is framed as the following: given a physical environment and a number of agents, design an algorithm (i.e, NN) that optimizes the agents' actions with respect to their states and rewards throughout the simulation. Hence, the 'data' that is being continuously collected and processed as the input to the model is: 1) Agents' current states; 2) The rewards (represented as continuous numbers) associated with the agents' current states (note: every agent, in 1 state, has only 1 reward that is generated by the environment). **By calling the environment to take a single "step", the environment object updates all the objects in the environment itself and returns a list of variables that contains states and rewards**.

Appendix B: Key terms in RL

1) *Environment*: Physical world in which the agent operates.
2) *State*: Current situation of the agent. Often used interchangeably with "**observations**". Every agent, after taking one action, has a set of observations that are collectively referred to as the agent's current state.
3) *Reward*: Feedback from the environment.
4) *Policy*: Method to map agent's states to actions.
5) *Value*: Future reward that an agent would receive by taking an action in a particular state.

Appendix C: Metrics of Success in RL

The training and evaluation process for agent survival using RL was quite different compared to supervised learning; unlike a classification or regression problem where the performance of the model itself can be represented numerically (e.g. accuracy, confusion matrix), RL agents can be evaluated on the reward it collects over the steps. In addition, if the agent utilizes a neural network to optimize its policy, an added metric of loss can be used to determine whether the model is converging.