

CSCI599: Life Simulator

Team: ZLX

Zhihan Zhang

Chuanzhe Li

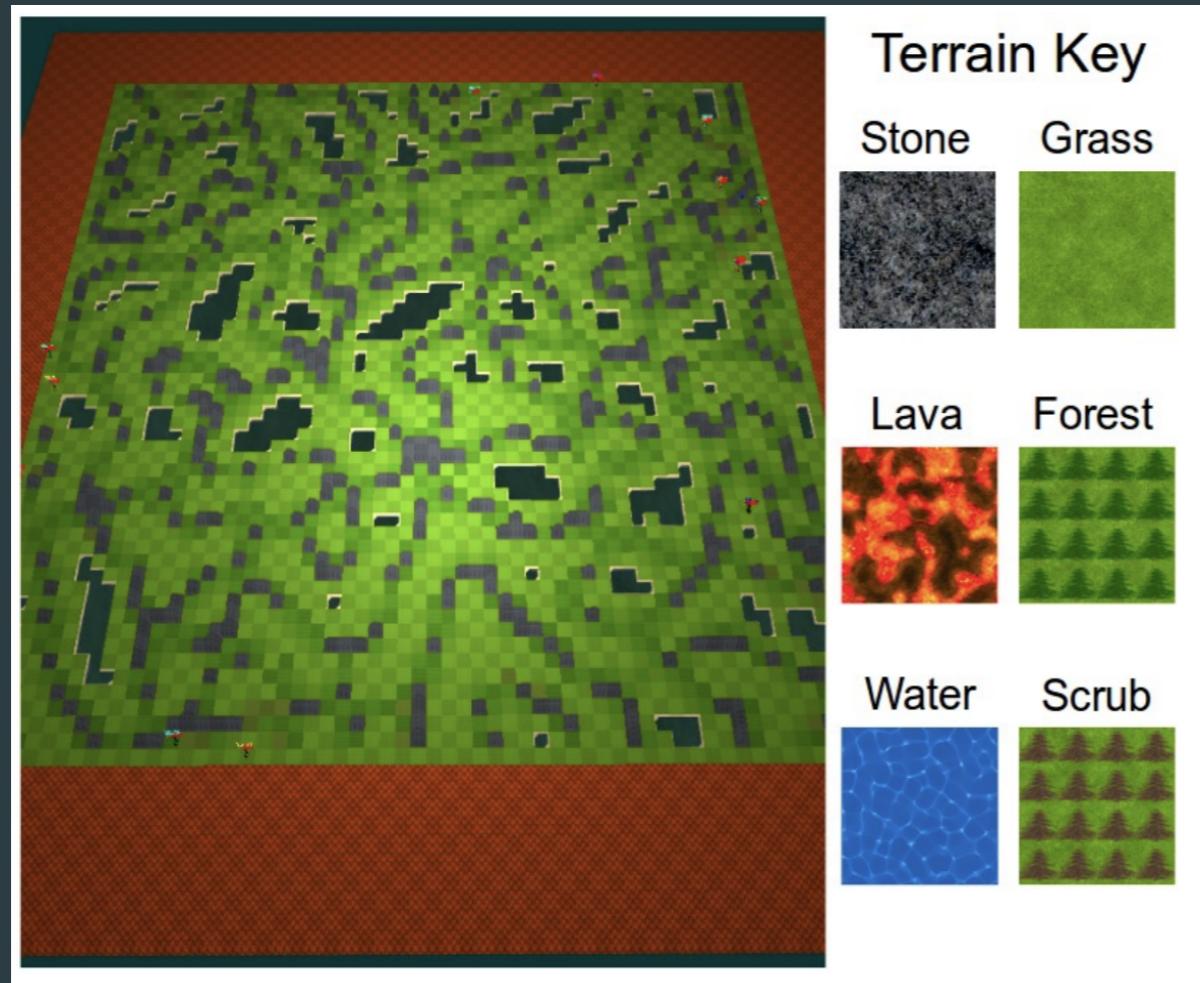
Jia Xu

Introduction

Genre: Neural Massively Multiplayer Online Role-Playing Games (MMO)

Character: AI agents

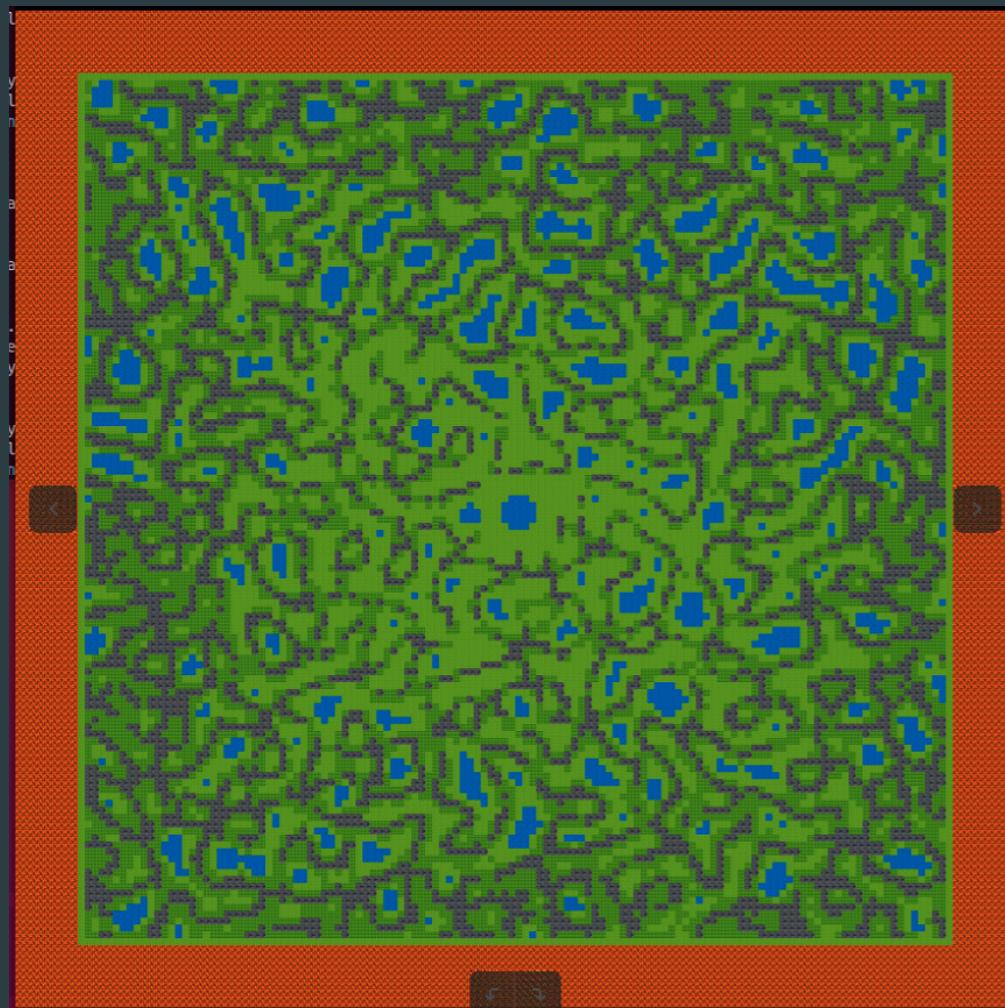
Purpose: simulate the evolution of a large number of players competing to maximize survival time in a resource-limited environment



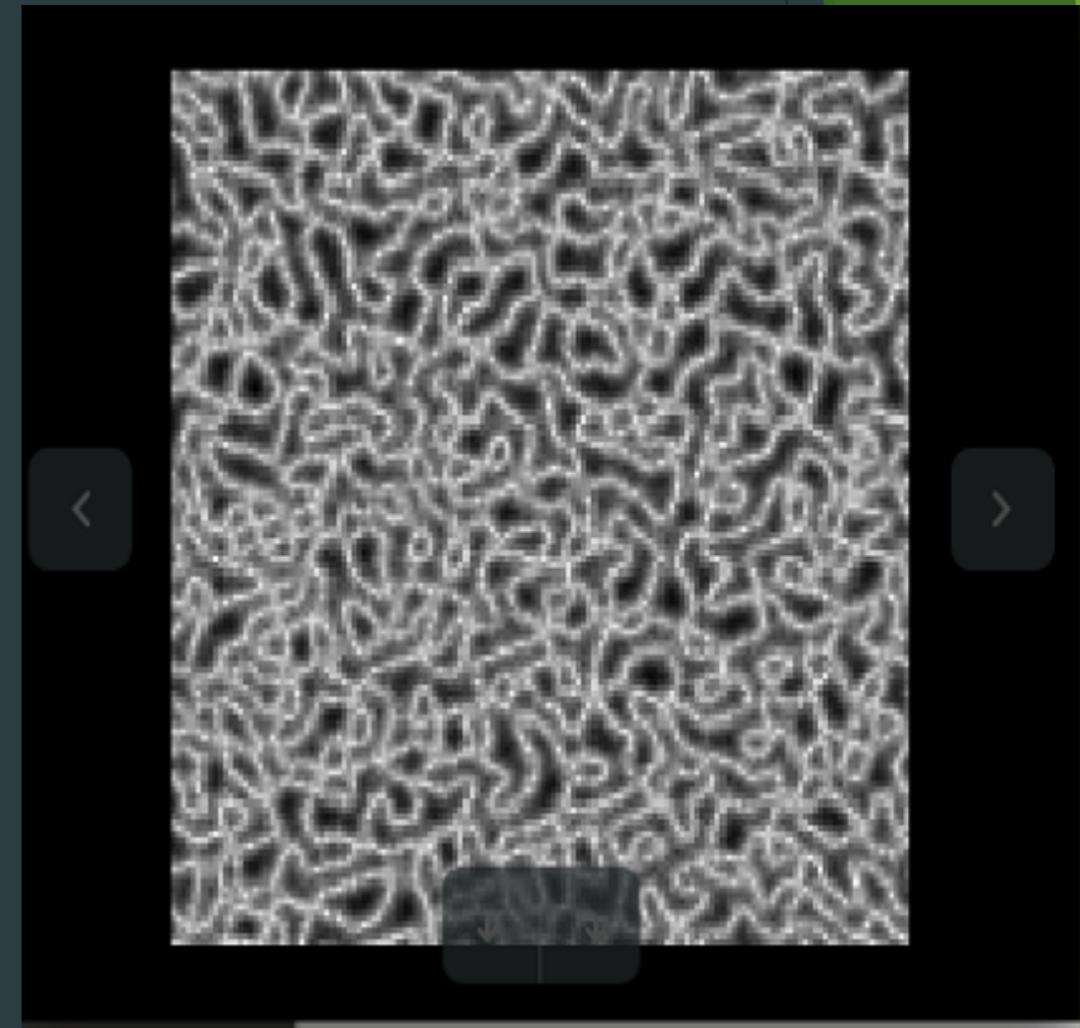
Map terrain:

- **Stone:** inaccessible
- **Grass & Scrub:** traversable
- **Lava:** die once enter
- **Forest:** traversable, get 5 points food, resource has 2.5% to regenerate
- **Water:** inaccessible, can get water near it

Introduction



Tile-based terrain map



Texture map

Introduction

Agent's attributes:

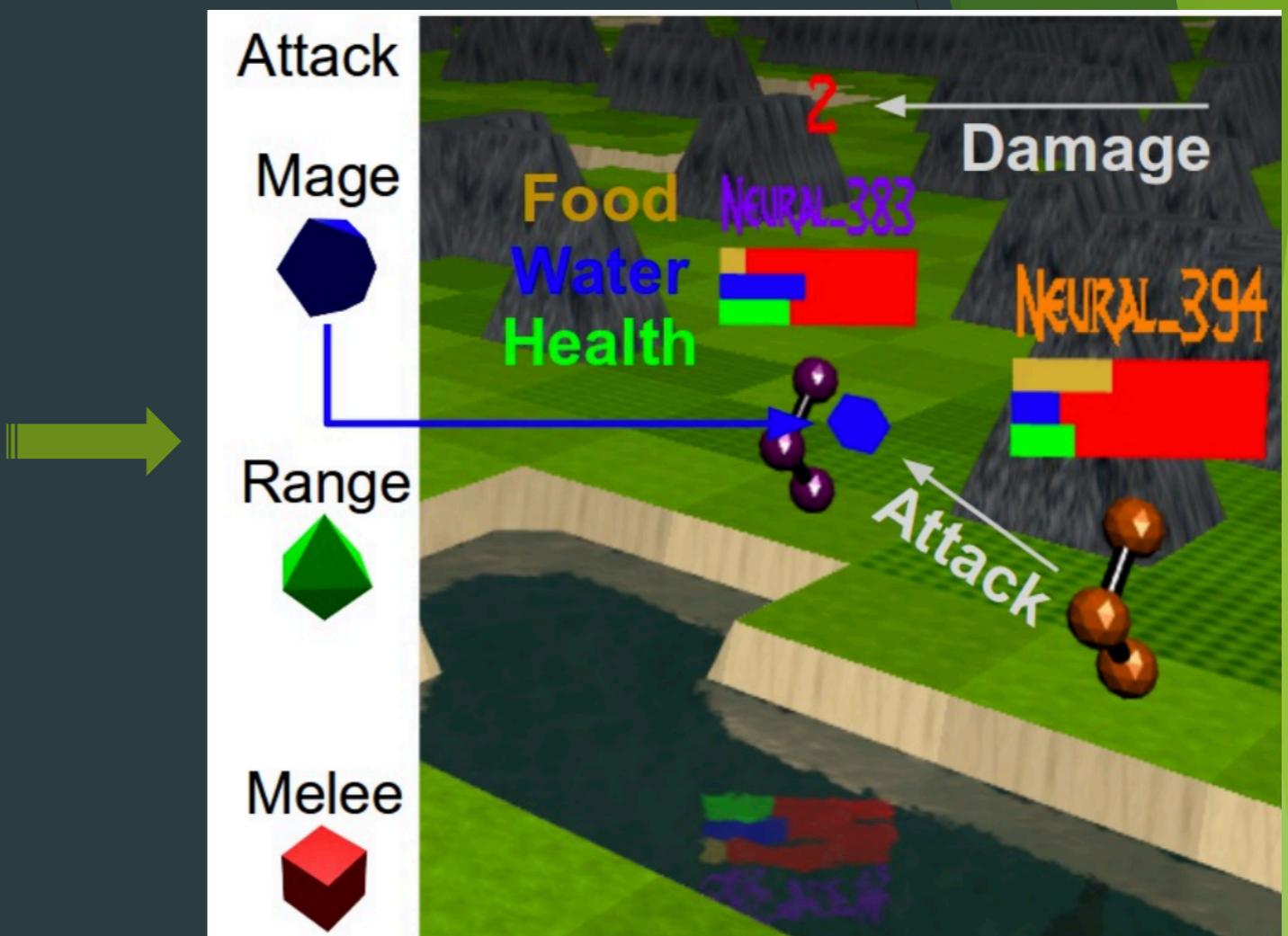
- **Food:** Agents die at 0 health (**Yellow**)
- **Water:** Agents begin taking damage at 0 food or water (**Blue**)
- **Health:** Agents begin taking damage at 0 food or water (**Green**)



Introduction

Combat system:

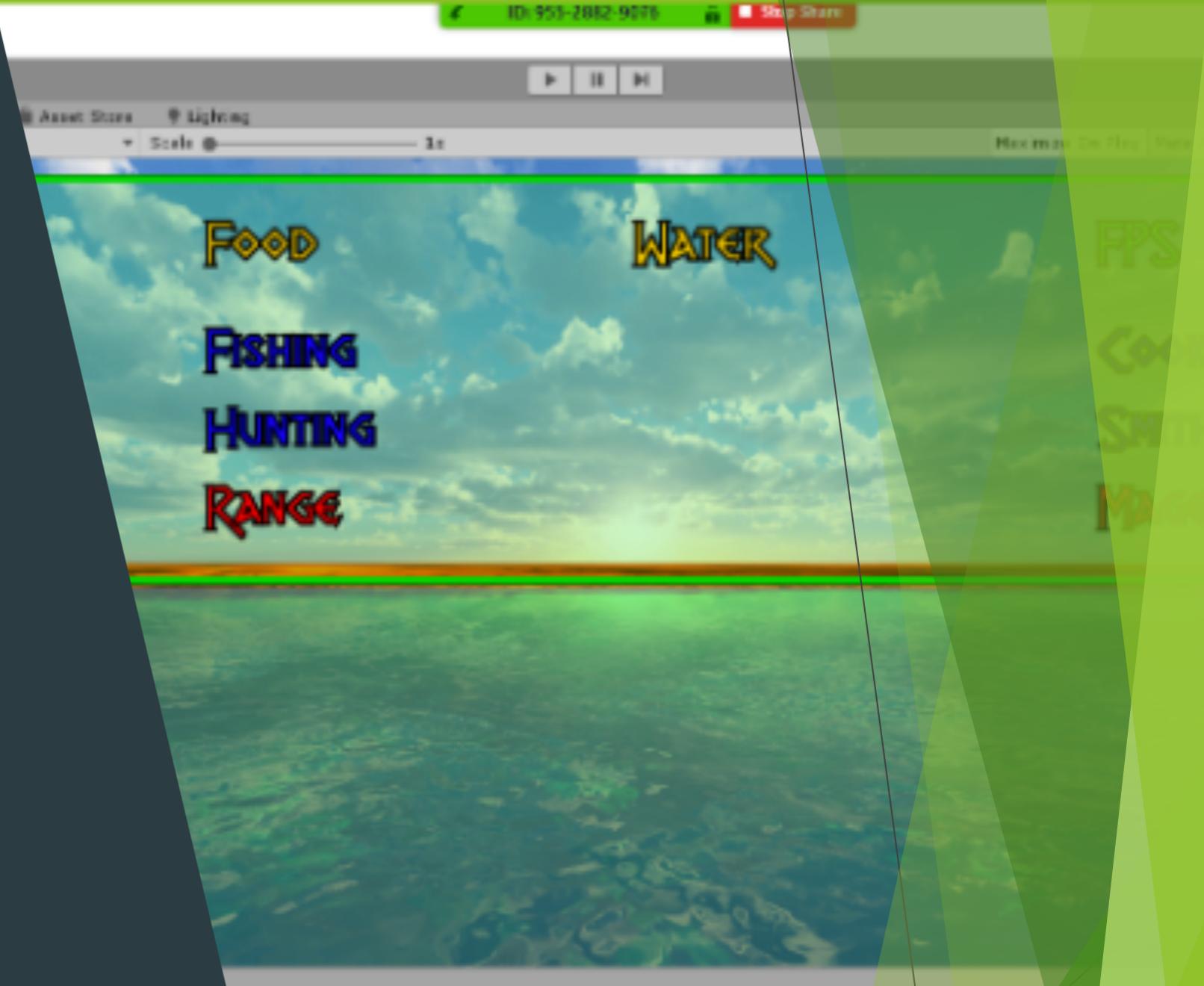
- **Mage:** Inflicts 10 damage at 1 range
- **Range:** Inflicts 2 damage at 1-2 range
- **Melee:** Inflicts 1 damage at 1-3 range and freezes the target in place, preventing movement for two ticks



How we build it

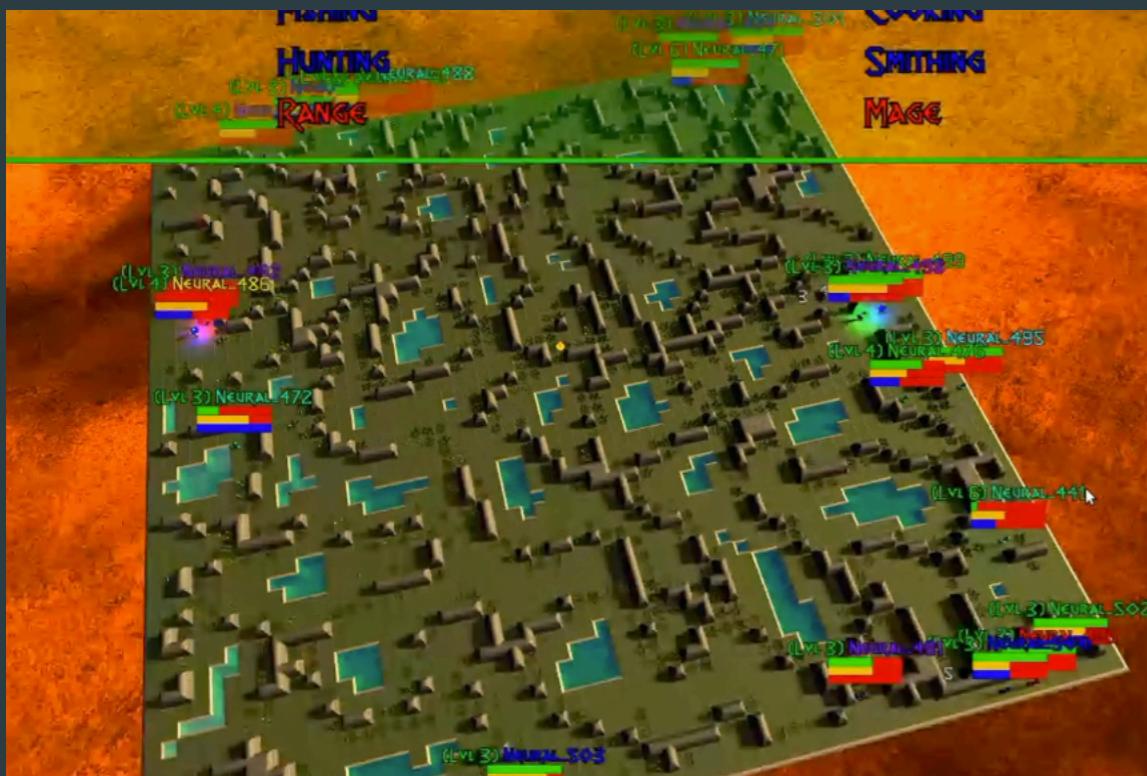
UI

- ▶ Unity client UI is built with UI assets implemented by Unity hub
- ▶ Optimized the network connection setting

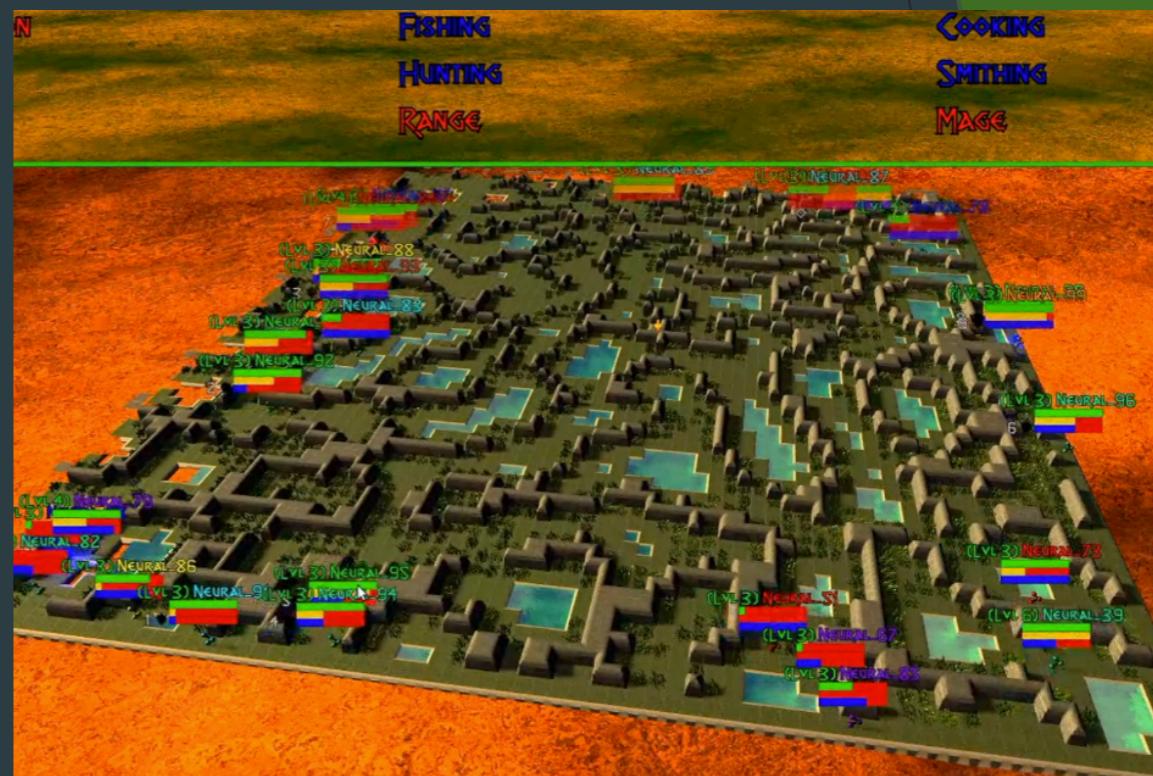


UI - changeable map size

- ▶ Changed UI map drastically to support different size of maps with support to much more clients live in the same map



80 * 80



144 * 144

Architecture and Training - Define terms in our model

- s_t : the current state of given agents given time tick t , state contains position information, current food, water, health etc.
- o_t : the current observation based on one agent.
- a_t : the action made by given agent which have five types, North, West, East, South, Pass
- $\Pi_\theta(a_t|s_t)$: Policy(fully observed)
- τ : the trajectory of given agents which is a sequence of observation, action and reward. $\tau = (o_t, a_t, r_t, \dots, o_T, a_T, r_T)$
- $R(\tau)$: reward given trajectory which is the sum of each survival one in every time tick with discount which by default is 0.99 $R(\tau) = \sum_t^T \gamma^t r_t$

Architecture and Training - Training Steps

The purpose is to maximize the reward in given trajectory:

$$\underbrace{p_{\theta}(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T, \mathbf{a}_T)}_{p_{\theta}(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^T \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) \quad (1)$$

$$\theta^* = \arg \max_{\theta} E_{\tau \sim p_{\theta}(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right] \quad (2)$$

Here we want to optimize the learned parameters maximize the expected survival rewards by adopting policy gradient method:

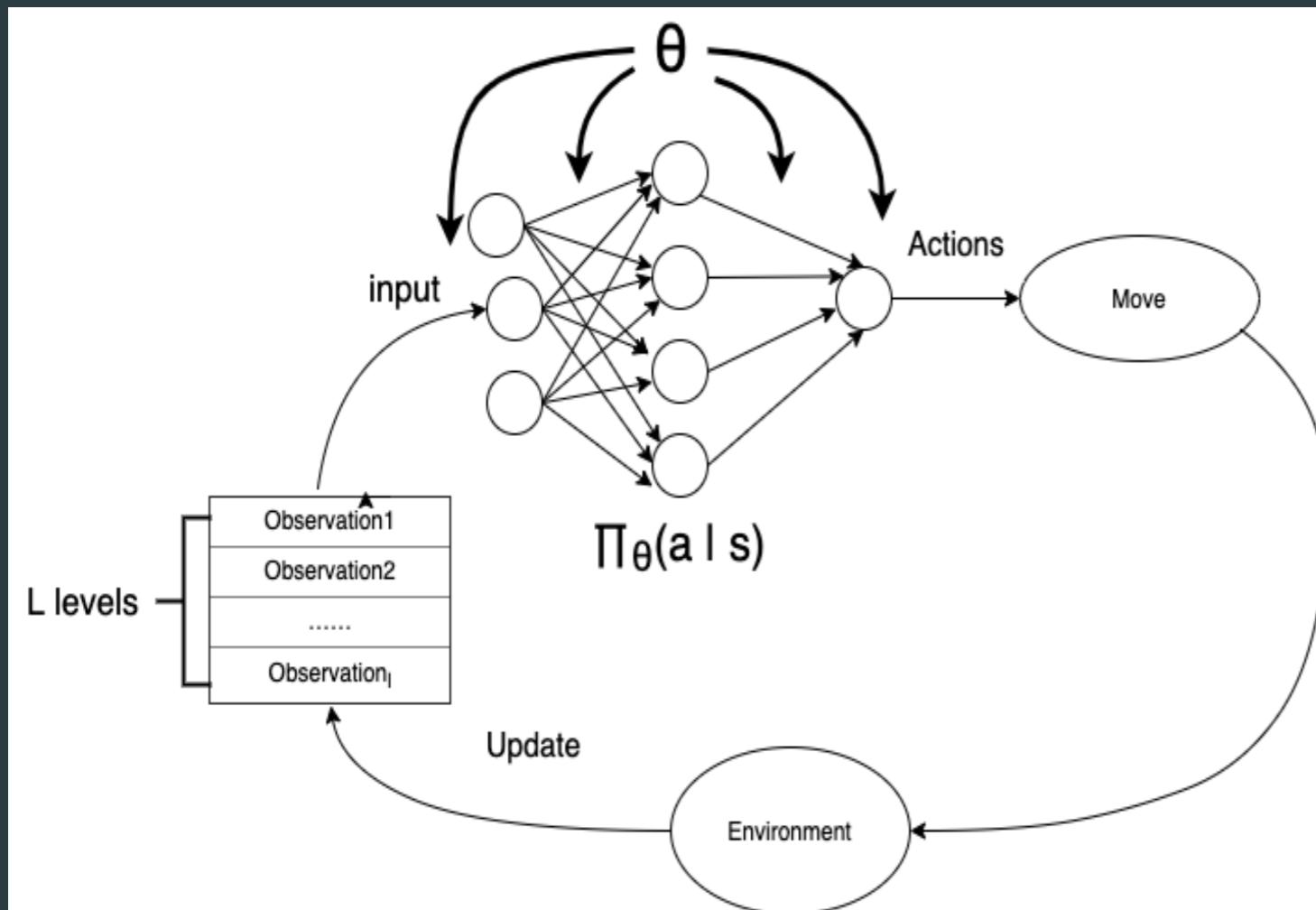
$$J(\theta) = E_{\tau \sim p_{\theta}(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right] \approx \frac{1}{N} \sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \quad (3)$$

$$\nabla_{\theta} J(\theta) = E_{\tau \sim \pi_{\theta}(\tau)} \left[\left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) \right) \left(\sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right) \right] \quad (4)$$

We will update based on epoch:

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta) \quad (5)$$

Architecture and Training -- Training model



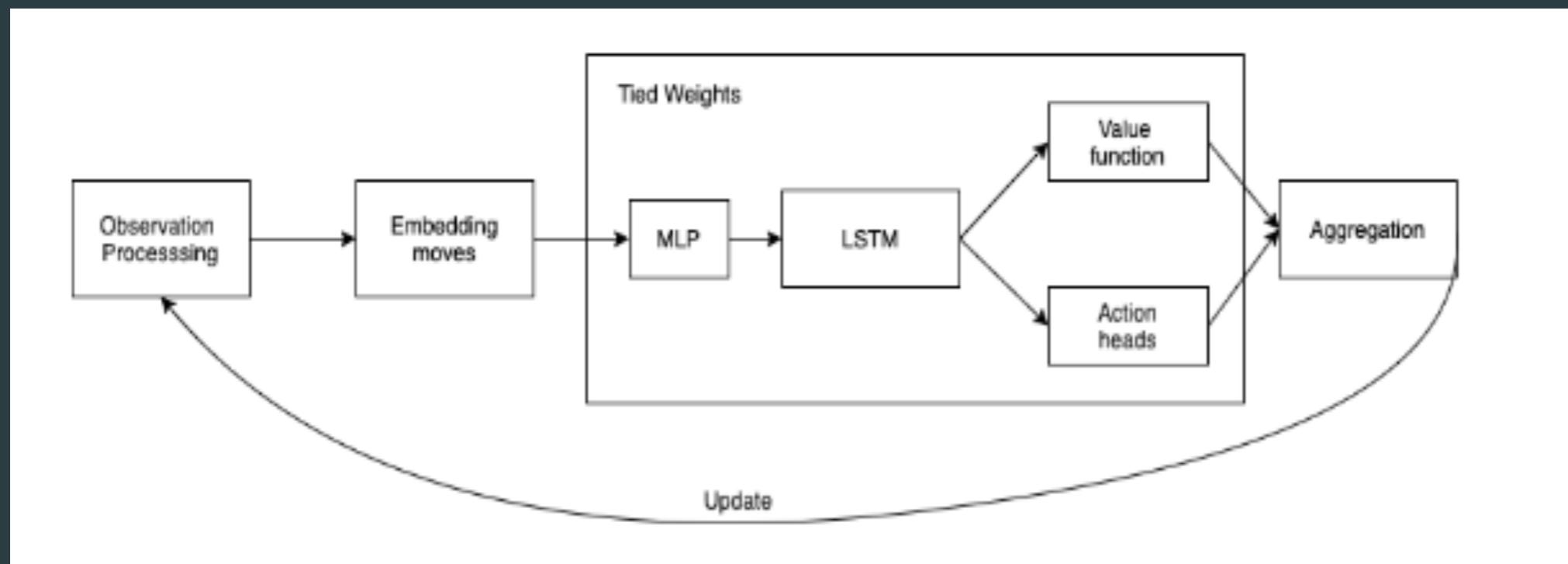
- ▶ Linear model
- ▶ Use embedding and flatten agents' reward to a one-dimensional encoded vector and fit it into a linear layer of with trainable parameters.

Architecture and Training -- Training model improvement tried

- ▶ Tried many different network layer architectures to optimize the training speed and the performance of agents after training:
 - ▶ LSTM layers
 - ▶ Attention layers
 - ▶ Transformer

Architecture and Training -- Training model improvement tried

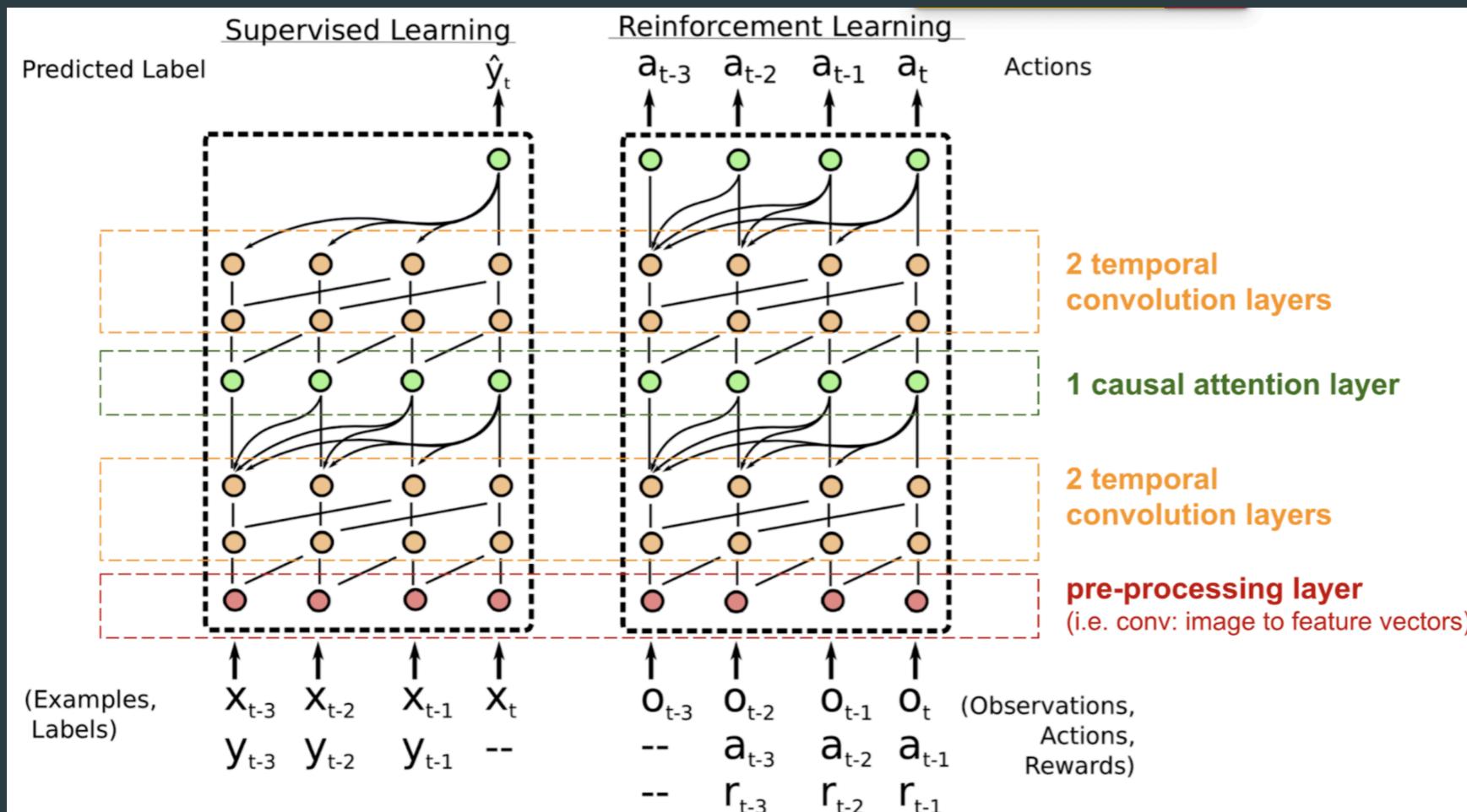
Model with LSTM:



LSTM is widely used in Time series data prediction, thus we want to using it to explore the action in a time-serial based trajectory

Architecture and Training -- Training model improvement tried

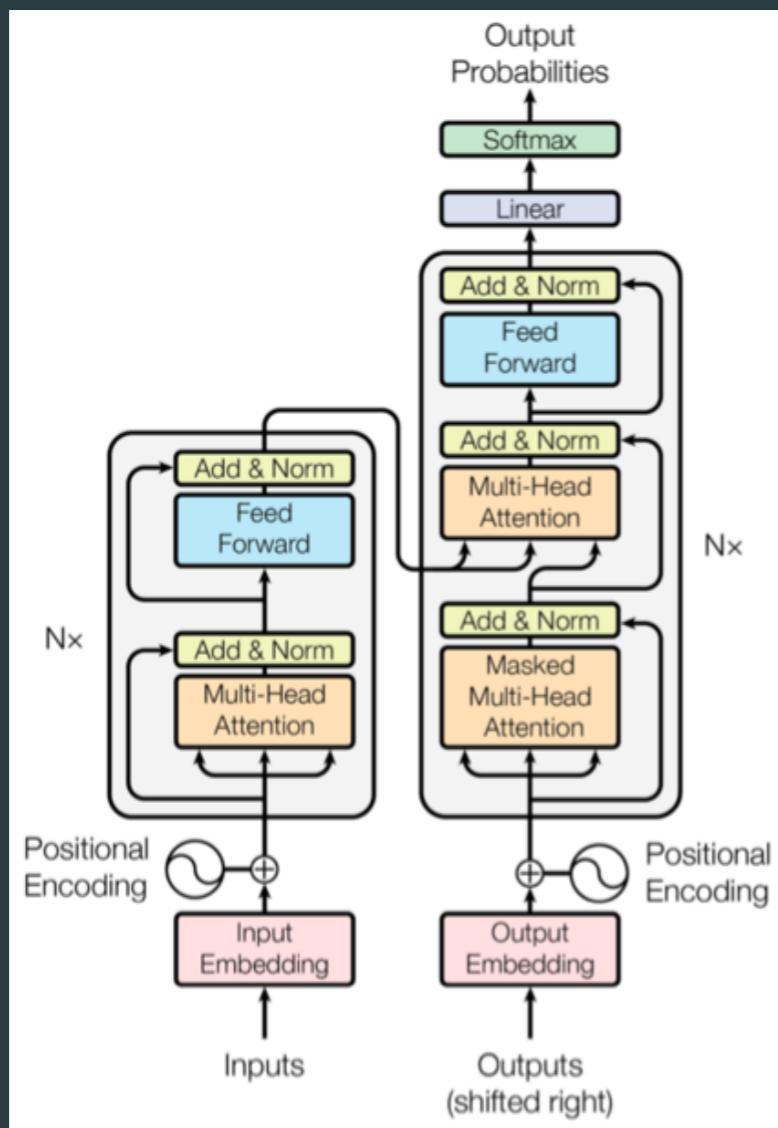
Attention:



Achieve better decision making by adopting global level understanding to the whole trajectory.

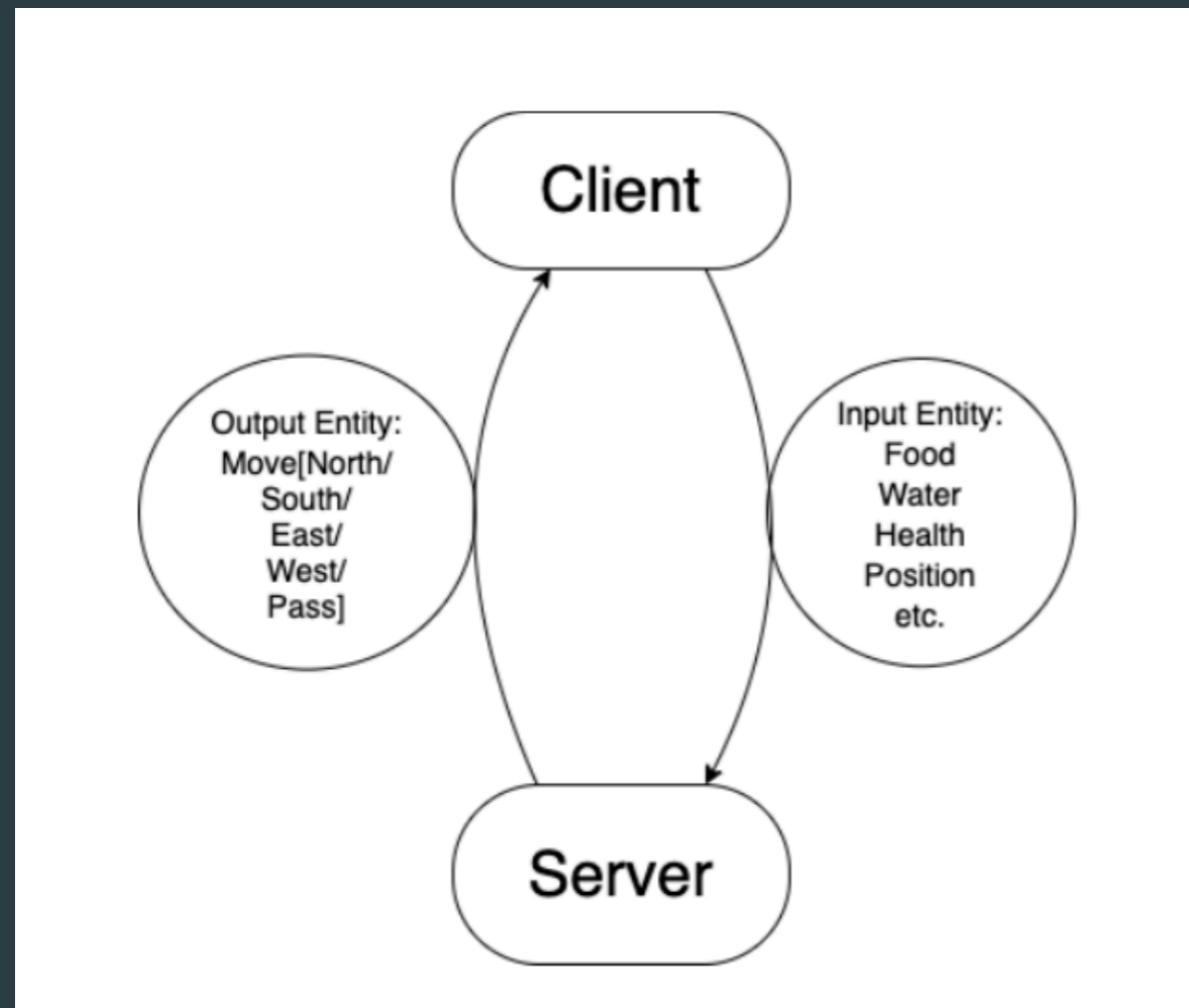
Architecture and Training -- Training model improvement tried

Transformer:



Neural machine translation may be applicable to our neural network because trajectory information may be viewed as a foreign language sentence.

Communication network -- Basic network architecture

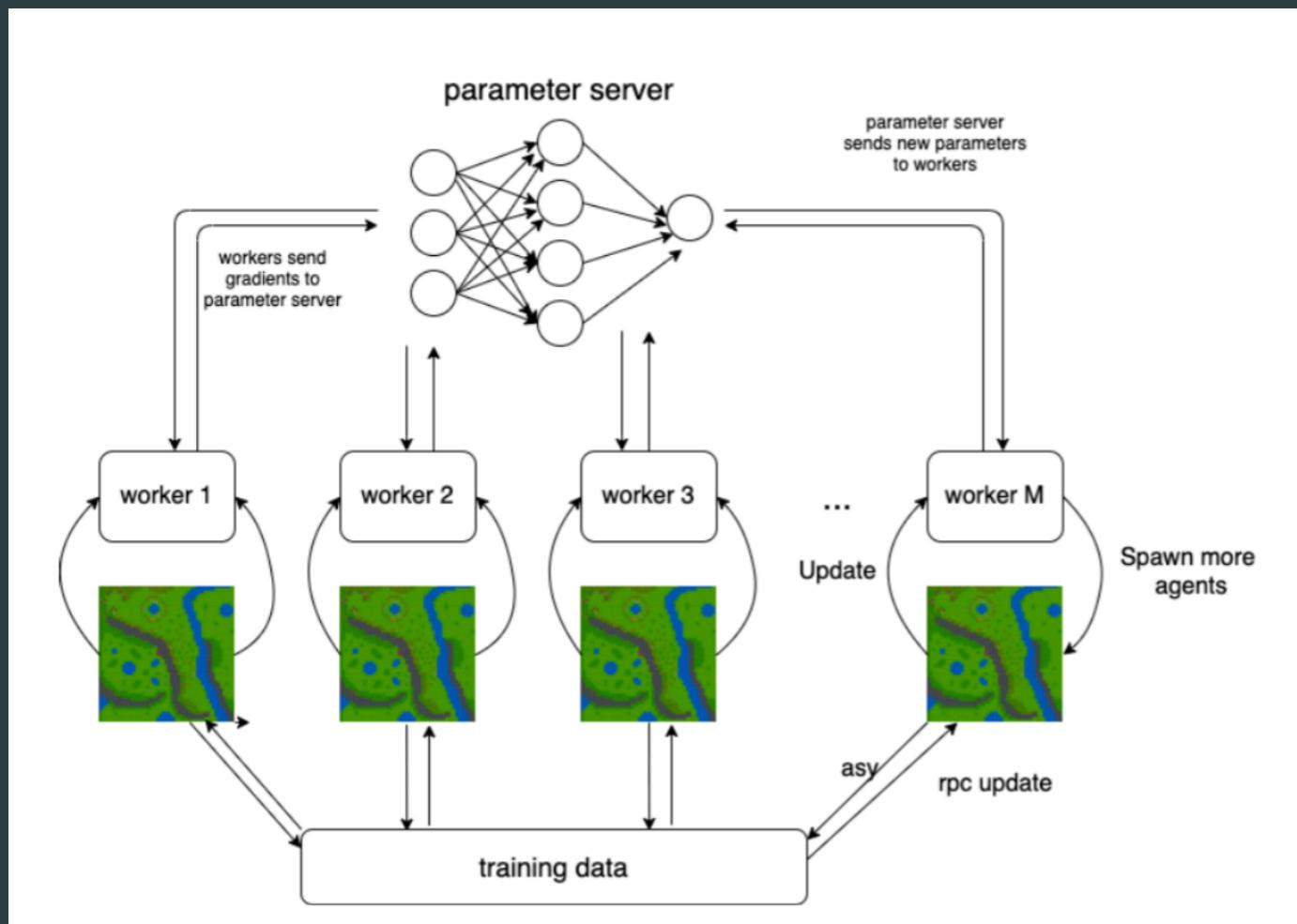


- ▶ Use Twisted Network Library to communicate packet information between client and server

Communication network -- scaling network with parameter server

- ▶ The initial training model with a single-layer neural network took several days to get results.
- ▶ Consider methods to optimize the algorithm by distributing training to different processes and utilizing the computing resources completely.
- ▶ The key principle of parallel computing is to overlap computation and communication to avoid high latency.
- ▶ Using a parameter server helps us to solve this problem

Communication network -- scaling network with parameter server



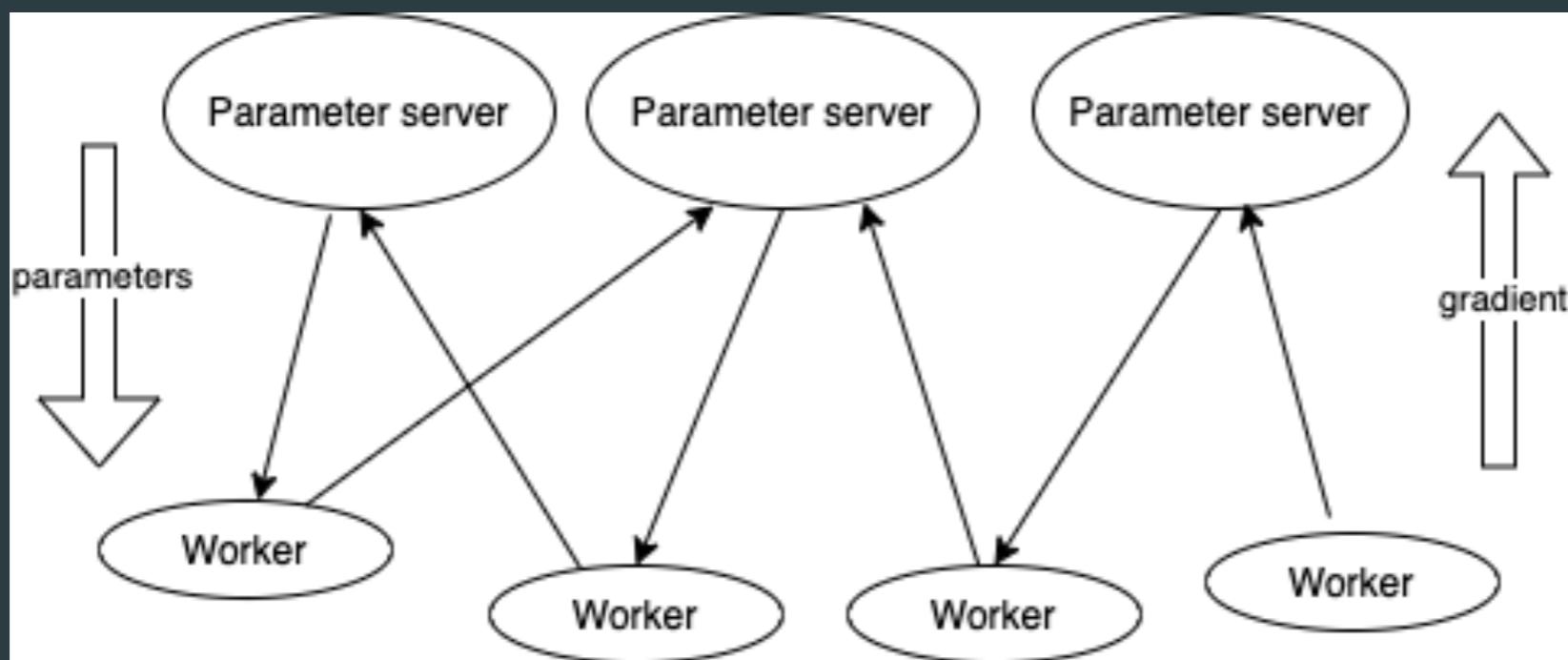
- ▶ Step 1: The param server get the parameters computed and pushed by workers
- ▶ Step 2 : Param server Broadcasts its updated parameters to all the other worker machines
- ▶ Step 3: Workers use the updated parameters to compute gradients.

Communication network -- Implement parameter server with Ray

- ▶ What's Ray? -- A high-performance distributed framework for large-scale machine learning and reinforcement learning applications.
- ▶ Why Ray? -- To simplify the development of high-performance distributed application that runs efficiently on a cluster.

Communication network -- Implement parameter server with Ray

- With the Ray framework, we implement the parameter server pythonically in only a few lines of code, also, it allows us to scale our algorithm that runs on a laptop into a high-performance distributed application that runs efficiently on a cluster with lightweight APIs



Communication network -- Implement parameter server with Ray

```
@ray.remote
class ParameterServer(object):
    def __init__(self, keys, values):
        # These values will be mutated, so we must create a local copy.
        values = [value.copy() for value in values]
        self.parameters = dict(zip(keys, values))

    def get(self, keys):
        return [self.parameters[key] for key in keys]

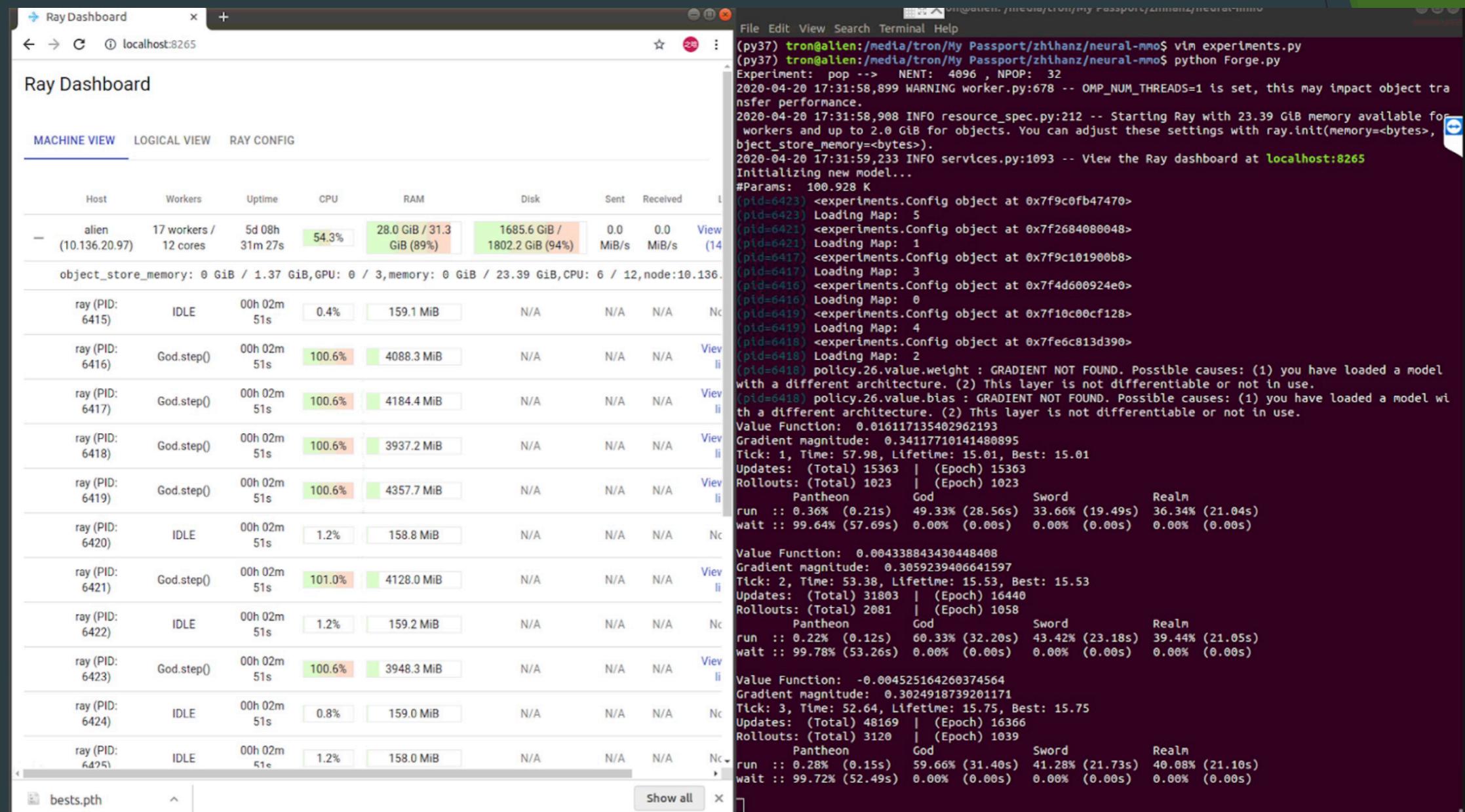
    def update(self, keys, values):
        # This update function adds to the existing values, but the update
        # function can be defined arbitrarily.
        for key, value in zip(keys, values):
            self.parameters[key] += value

@ray.remote
def worker_task(parameter_server):
    while True:
        keys = ['key1', 'key2', 'key3']
        # Get the latest parameters.
        values = ray.get(parameter_server.get.remote(keys))
        # Compute some parameter updates.
        updates = time.sleep(1)

        # Update the parameters.
        parameter_server.update.remote(keys, updates)

    # Start 4 long-running tasks.
    for _ in range(4):
        worker_task.remote(parameter_server)
```

Communication network -- Implement parameter server with Ray



The image shows a terminal window with two tabs open. The left tab is titled 'Ray Dashboard' and shows the Ray Dashboard interface. The right tab is titled 'experiments.py' and displays the output of a Python script running on a Ray cluster.

Ray Dashboard (Left Tab):

- MACHINE VIEW:** Shows a table of workers on the 'alien' host (10.136.20.97). There are 17 workers (12 cores) with 5d 08h uptime. CPU usage is 54.3%, RAM usage is 28.0 GiB / 31.3 GiB (89%), and Disk usage is 1685.6 GiB / 1802.2 GiB (94%). Sent and Received rates are 0.0 MiB/s.
- LOGICAL VIEW:** Shows 14 Ray objects. The first object is 'bests.pth' (type: File, status: IDLE, 0.4% CPU, 159.1 MiB RAM). The second object is 'Forge.py' (type: File, status: God.step(), 100.6% CPU, 4088.3 MiB RAM).
- RAY CONFIG:** Shows the configuration for the experiment, including the experiment name 'pop', NENT (4096), NPOP (32), and the command to run 'Forge.py'.

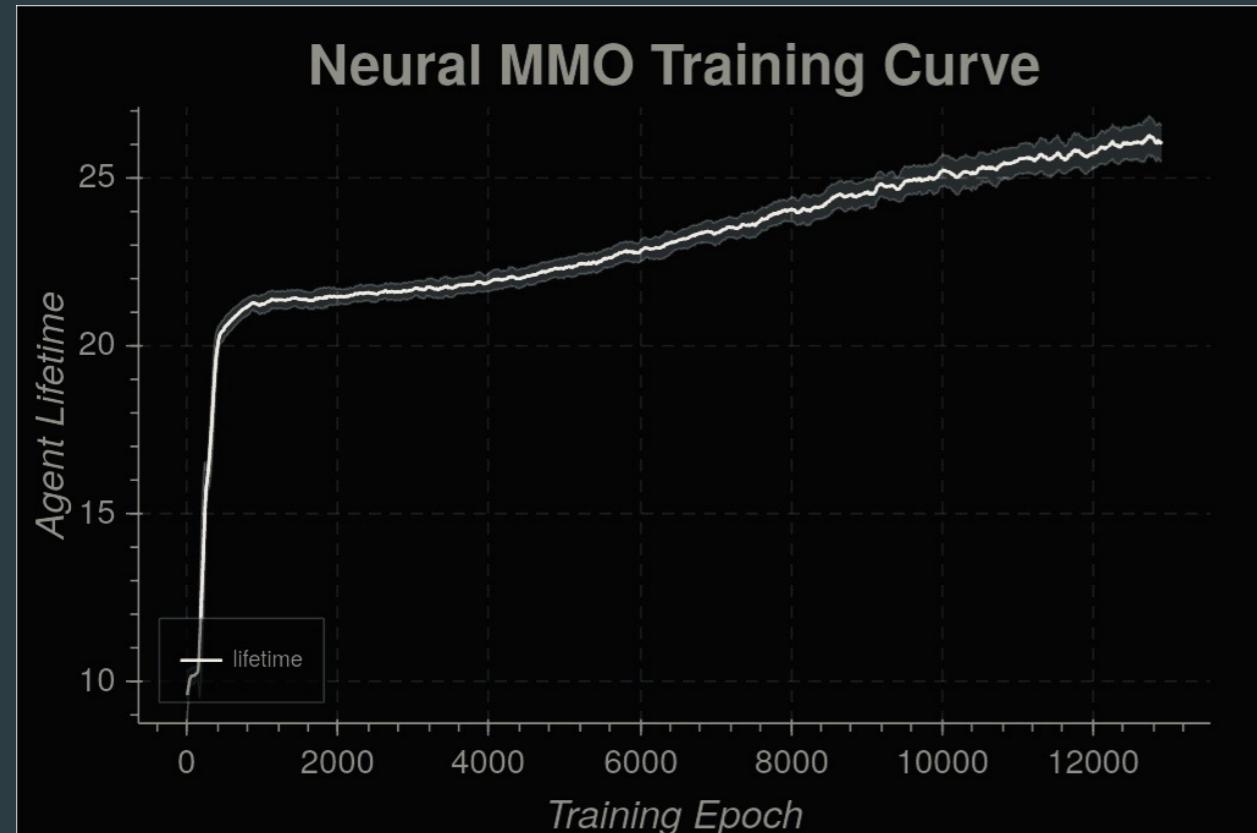
experiments.py (Right Tab):

```
File Edit View Search Terminal Help
(py37) tron@alien:/media/tron/My Passport/zhihanz/neural-mmo$ vim experiments.py
(py37) tron@alien:/media/tron/My Passport/zhihanz/neural-mmo$ python Forge.py
Experiment: pop --> NENT: 4096, NPOP: 32
2020-04-20 17:31:58,899 WARNING worker.py:678 -- OMP_NUM_THREADS=1 is set, this may impact object transfer performance.
2020-04-20 17:31:58,908 INFO resource_spec.py:212 -- Starting Ray with 23.39 GiB memory available for workers and up to 2.0 GiB for objects. You can adjust these settings with ray.init(memory=<bytes>, object_store_memory=<bytes>).
2020-04-20 17:31:59,233 INFO services.py:1093 -- View the Ray dashboard at localhost:8265
Initializing new model...
#Params: 100.928 K
(pid=6423) <experiments.Config object at 0x7f9c0fb47470>
(pid=6423) Loading Map: 5
(pid=6421) <experiments.Config object at 0x7f2684080048>
(pid=6421) Loading Map: 1
(pid=6417) <experiments.Config object at 0x7f9c101900b8>
(pid=6417) Loading Map: 3
(pid=6416) <experiments.Config object at 0x7f4d600924e0>
(pid=6416) Loading Map: 0
(pid=6419) <experiments.Config object at 0x7f10c00cf128>
(pid=6419) Loading Map: 4
(pid=6418) <experiments.Config object at 0x7fe6c813d390>
(pid=6418) Loading Map: 2
(pid=6418) policy.26.value.weight : GRADIENT NOT FOUND. Possible causes: (1) you have loaded a model with a different architecture. (2) This layer is not differentiable or not in use.
(pid=6418) policy.26.value.bias : GRADIENT NOT FOUND. Possible causes: (1) you have loaded a model with a different architecture. (2) This layer is not differentiable or not in use.
Value Function: 0.016117135402962193
Gradient magnitude: 0.34117710141480895
Tick: 1, Time: 57.98, Lifetime: 15.01, Best: 15.01
Updates: (Total) 15363 | (Epoch) 15363
Rollouts: (Total) 1023 | (Epoch) 1023
Pantheon God Sword Realm
run :: 0.36% (0.21s) 49.33% (28.56s) 33.66% (19.49s) 36.34% (21.04s)
wait :: 99.64% (57.69s) 0.00% (0.00s) 0.00% (0.00s) 0.00% (0.00s)

Value Function: 0.004338843430448408
Gradient magnitude: 0.3059239406641597
Tick: 2, Time: 53.38, Lifetime: 15.53, Best: 15.53
Updates: (Total) 31803 | (Epoch) 16440
Rollouts: (Total) 2081 | (Epoch) 1058
Pantheon God Sword Realm
run :: 0.22% (0.12s) 60.33% (32.20s) 43.42% (23.18s) 39.44% (21.05s)
wait :: 99.78% (53.26s) 0.00% (0.00s) 0.00% (0.00s) 0.00% (0.00s)

Value Function: -0.004525164260374564
Gradient magnitude: 0.3024918739201171
Tick: 3, Time: 52.64, Lifetime: 15.75, Best: 15.75
Updates: (Total) 48169 | (Epoch) 16366
Rollouts: (Total) 3120 | (Epoch) 1039
Pantheon God Sword Realm
run :: 0.28% (0.15s) 59.66% (31.40s) 41.28% (21.73s) 40.08% (21.10s)
wait :: 99.72% (52.49s) 0.00% (0.00s) 0.00% (0.00s) 0.00% (0.00s)
```

Training result and analysis



- ▶ The baseline model gets to on average > 28 lifetime after training for several days on 12 cores.
- ▶ Agents do learn something after training, like not to run into lava first, as indicated by the steep initial learning curve slope.

Training result

Evaluation Results



Limitation

- ▶ Map size rendering optimization (currently the model FPS drop drastically if size become 500*500) (show later in the demo)



Future Work

- ▶ Map size rendering optimization
- ▶ Add more terrain setups and explore of environment complexity on population behavior (like will people can choose to live near river and generate collaboration instead of competition)
- ▶ Deploy AI training to cloud and realize cloud native AI training, deployment and visualization pipeline

Demo

- ▶ <https://www.youtube.com/watch?v=5qFh01Kd6vg&feature=youtu.be>

Thank you !