**General Documentation for Unity ML Agents**

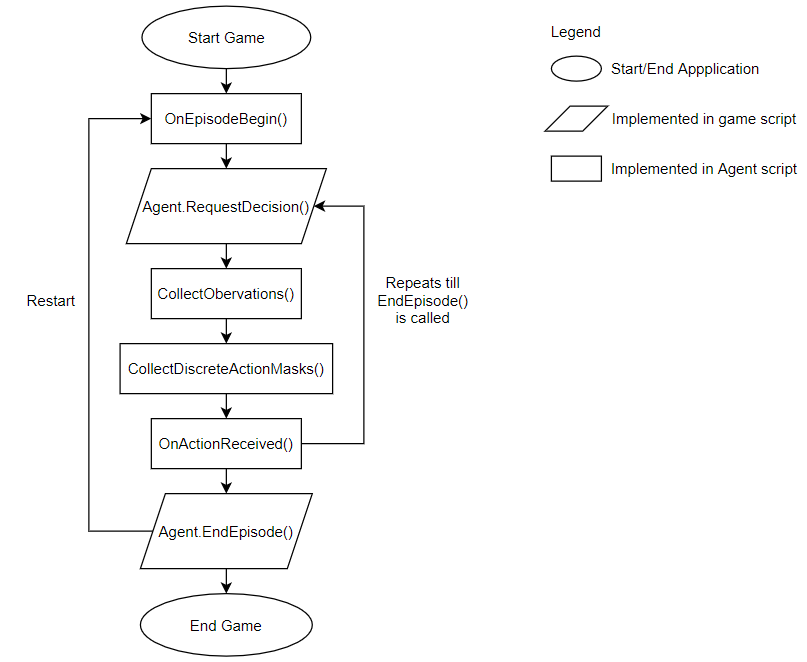
|  |  |
| --- | --- |
| Siew Jun Leong | 18-Oct-20 |

**What is the ML Agents Toolkit?**

It is a project by Unity to enable any game or simulations created through Unity to serve as training environment for the agents. It contains:

* 15 ML Agents examples.
* 2 deep reinforcement learning algorithms, Proximal Policy Optimisation (PPO) and Soft-Actor Critic (SAC).
* Self-play mechanism for agents in adversarial scenarios.
* Utilises Unity Inference Engine for the usage of pre-trained neural network models in the created Unity game.
* Many other learning approaches which are irrelevant at the current stage of the project.

**Codes before Training**



Flowchart for Sequence of Agent Functions

The agent script will inherit from the Agent class provided by Unity, which allows interfacing the game with the AI script.

The important portions of the code are the ones in the loop as shown in the figure above: CollectObservations() and OnActionReceived(). In this case, where we do not want the agent to override any of the slots that were already occupied in the case of the TicTacToe example, CollectDiscreteActionMasks() is important as well.

CollectObservations() code will be called automatically whenever Agent.RequestDecision() is called in the program script. Any information the Agent is supposed to access can be added here. This information will be fed into the neural network, and an output will be given based on the input.

OnActionReceived() will be called when an action is received from the neural network. Actions that the agent can take will be placed in here and rewards are usually assigned here based on the outcome of the action taken (eg. Agent got closer/further away from target, time based penalty). In the case of the TicTacToe example, rewards are going given based on the outcome of the game, such as win, lose or draw.

The listw below shows more information with regards to the functions.

To allow the agent to interact with the program a script inheriting from the Agent class is required, and implementation of the following code are also required:

* Agent.OnEpisodeBegin()
  + This function is called at the beginning of the Agent episode, which includes the start of the simulation.
* Agent.CollectObservations(VectorSensor sensor)
  + This function is called every time the Agent.RequestDecision() is called.
  + The [Observable] attribute can also be added to fields and properties on the Agent.
  + Must include VectorSensor.AddObservation(input) where input types can be:
    - Integers
    - Boolean
    - Vecter2

Unity Specific data types

* + - Vector3
    - Quaternion
  + Under Behaviour Parameters in Unity Editor, number of inputs taken in using CollectObservation() must be defined in Vector Observations > Space Size
    - Branch size is the number of observations and value in the sub-branch will be 1, unless an array is used, which it will then take the size of the array.
* Agent.OnActionReceived()
  + Called every time the Agent receives an action to take. Receives the action chosen by the Agent. It is also common to assign a reward in this method.
* Agent.CollectDiscreteActionMasks(DiscreteActionMasker actionMasker) (Not Mandatory)
  + This function is to prevent the agent from taking actions that are deemed impossible (eg. Pressing the direction to bang to the wall).
  + Only possible when the AI is controlled by a neural network and the action space is discrete (eg. Directional keys).

**Training Process**

1. AI starts with random actions that make no sense to us humans.
2. Due to reward functions in place, it will slowly realise what it is doing is not right, as AI aims to get the highest amount of rewards possible.
   1. For example the AI in TicTacToe, the reward function is as follows:
      1. If AI win, and
         1. AI started first, +0.75f reward
         2. AI started second, +1f reward
      2. If AI lose, and
         1. AI started first, -1f reward
         2. AI started second, -0.25f

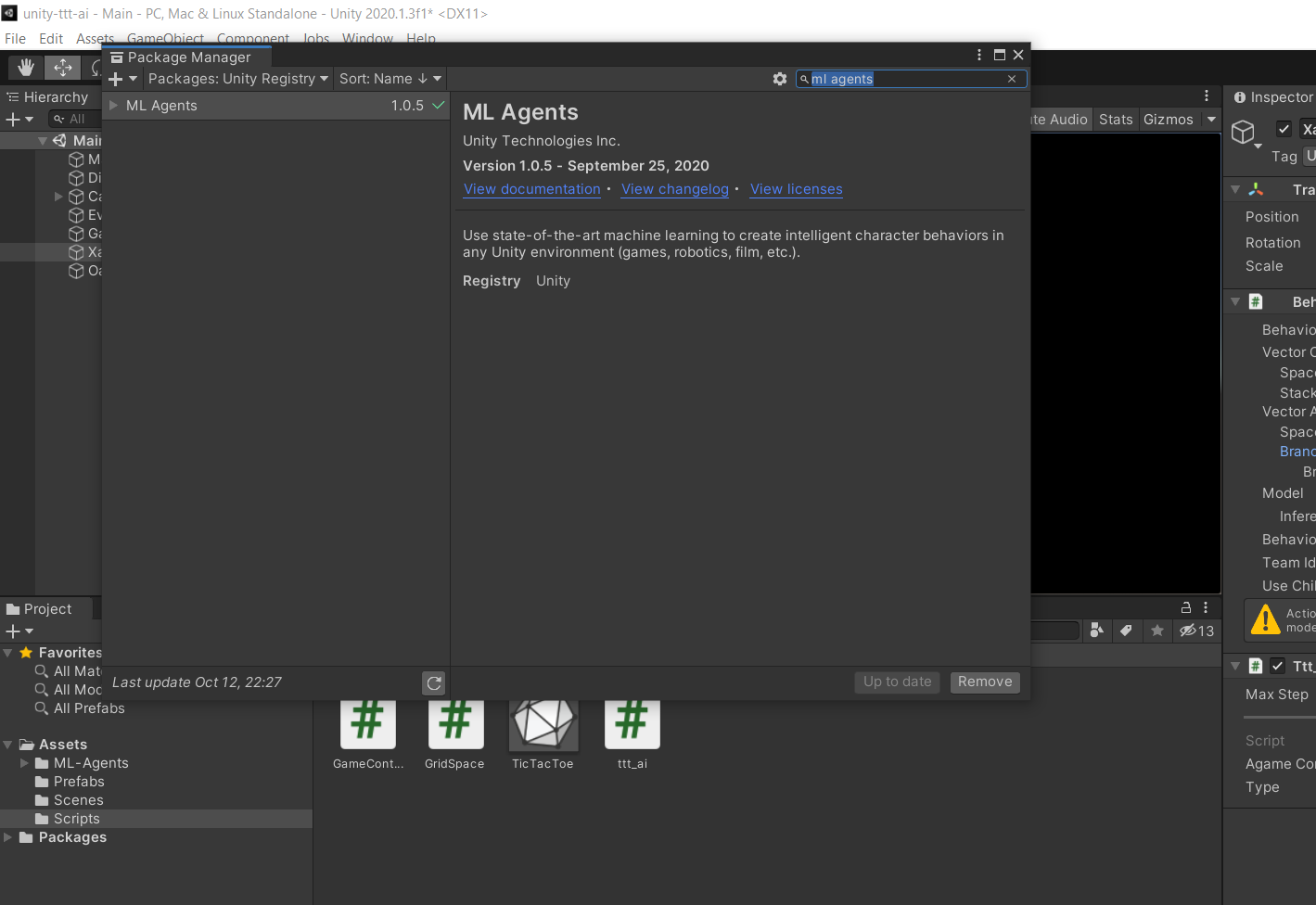
This is to condition the AI to aim for draw more often instead of just rewarding +1f reward to winner and -1f vice versa. This implementation calls the reward function when the game has ended

* 1. Another example is in a snake game. Small penalties can be implemented for every action step taken by the AI, thus allowing the AI to learn to use the shortest path possible. Ending the game by banging onto the wall or itself to incur large penalties, and positive rewards for every apple eaten.

1. If the graph shown with regards to the training is not ideal, reward function is required to be tweaked to optimise AI training.

**Steps used for Training**

1. Install ML agents package in Unity for the project (First Time Setup for Project).



Window > Package Manager

1. Prepare a training configuration file (eg. TicTacToe.yaml), where behaviour name in the file must be the same as the behaviour name in Unity

|  |  |
| --- | --- |
|  |  |

1. Use virtual environment so that management of the dependencies of the project is easier. Run the following code
   1. Go to the project directory from cmd
   2. Make a new directory for the virtual environement, md python-envs (First Time Setup).
   3. Create environment using python -m venv python-envs\tttAIenv (First Time Setup).
   4. Activate the environment through python-envs\tttAIenv\Scripts\activate
   5. Change *activate* to *deactivate* in the code above to deactivate the virtual environment
2. Run training by mlagents-learn trainingConfig/TicTacToe.yaml --run-id=tttTraining
   1. Run id is unique if training a new Neural Network
      1. If resuming, append *--resume* at the end of the training command
      2. If training new Neural Network with the parameters of an existing model, append this command at the end of the training command --initialize-from=tttTraining where tttTraining is the run id of the older training
   2. Press the play button in Unity to start
3. To observe training
   1. Enter tensorboard --logdir results into another cmd window
   2. Head to *localhost:6006* in the browser to view the progress

Tidy documentation on mlagents unity project (understand and describe mlagents)

**References**

[1] Juliani, A., Berges, V., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., Lange, D. (2020). Unity: A General Platform for Intelligent Agents. arXiv preprint arXiv:1809.02627. https://github.com/Unity-Technologies/ml-agents.