A-life using Reinforcement learning and Neural Network.

Shammi Seth (2005262)

Coursework for MAT501-Applied Mathematics and Artificial Intelligence Part 2, Term 1 20/21

School of Design and Informatics (SDI)
University of Abertay Dundee

Contents

Introduction

As humans we live in a world as perceived by us, we have the ability to dream which is what is called a dream world which is as sophisticated as the real world and the experience and joys and sorrow in the deal world appear to be as real that it is difficult to differentiate between the true experience or the real experience. Computer games introduce another set of experiences called Virtual world or Game world. There is also an intermediary between the virtual and real world called an augmented world; this experience overlaps the experience of the virtual and real.

The experience in the virtual world needs to appear and behave as believable as in real life to have this experience registered as real. Realism in the virtual/ game world can be achieved using several techniques 1. Graphical techniques for example using photo realistic virtual environments, animation using motion capture etc., 2. Human interaction techniques by making use of hardware and peripherals to improve the interaction of humans with machines example use of VR, and 3. AI techniques by use of expert systems, state machines, algorithms for example graduated difficulty levels, distinct movement patterns, intelligent game characters NPCs, procedurally generated worlds, creating player opponents etc. With the advent of advanced computer graphics technology and human interaction in a virtual world the video games also need to advance in terms of Artificial Intelligence (AI).

Automation exists in real life where we have robots to do repeated tasks, in decision support systems for medical diagnosis, in internet intelligence where we have semantic web to do search and recommender systems when shopping online. To implement automation in Games, AI engines need to have the blocks of an artificial life like sensing, thinking, and acting. Intelligent agents have cognitive ability, personality and moods, reactive or impulsive actions. An AI engine can be based on complexity of execution and complexity of sophistication. However, an AI engine does not need to be as sophisticated in the virtual\game world, there is not enough resource and real time computation power at the disposal it can smoke and mirror.

Types of Intelligent Agents in AI

intelligent agent (IA) refers to an autonomous entity which acts, learns or uses knowledge directing its activity towards achieving their goals. AI behaviours can be simplified as follows:

1. Reactive AI - CODE/ INFER
   This is implemented by explicitly creating knowledge or triggers.
   The examples of Reactive AI are using Finite state machines, Rules based systems, Behaviour trees, Utility AI etc.
2. Deliberative AI - SOLVE
   This makes use of Algorithms to optimize search and predict based on the knowledge gathered from environment and goals.
   The examples of Deliberative AI are A * algorithm, Navigation mesh, Goal Oriented Action Planning GOAP, AI Planner, STRIPS, Case based Reasoning etc.
3. Machine Learning AI -LEARN
   This makes use of algorithms to learn from large amounts of datasets and ability to generalize a given problem (train) and then apply the learning (predict).
   The examples of AI using Machine learning are Regression Classifiers, Clustering Algorithms, Genetic Algorithm, Reinforcement learning, Imitation learning.

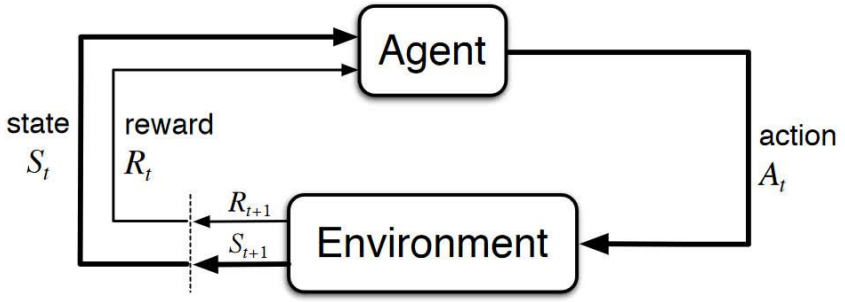
Knowledge representation, goal-directed behaviour and knowledge reusability are all directly relevant to improving intelligent agents in computer games.

Approach using Reinforcement learning

Reinforcement learning is the ability to define a system where the AI is able to acquire knowledge and learn from it. The AI is able to use a combination of cognition and knowledge in the system which interfaces with the environment. The system should be placed in the environment so that the acquisition of knowledge can be active. This is similar to how an adolescent learns from the environment by using curiosity to explore and and exploitation of knowledge from the previous experiences to achieve its goal.

The autonomous agent is able to perceive the environment which can be in different states. The agent then processes the data based on inputs from the environment and the possible actions for the states. The agent then selects an action which results into a transition of the environment to a new state. Agent is the given reward based on the consequence of the previous action. Learning occurs as a series of discrete time steps.

Markov decision processes MDP has formalized sequential decision making as a basis for structuring problems that are solved with reinforcement learning. The components of MDP are Agent, Environment, set of State (S), Set of actions (A), Reward (R).



R and S are finite steps Rt and St have well defined probability distributions. These distributions depend on the preceding state and action that occurred in the previous time step T is the final time step. Expected return G = R_{t+1} +R_{t+2} + ..R_T
It is the agent's goal to maximize the expected return of rewards. Agent environment interaction can be broken down into a series of steps called as episodes. episode and is at a terminal state of time T, which is followed by resetting the environment to some standard starting state or to a random sample from a distribution of possible starting states.

In case of Reinforcement learning using a neutral network. It is the agent's goal to maximize the expected discounted return of rewards. The discount rate will be the rate for which we discount future rewards and will determine the present value of future rewards. Learning Rate γ a number between 0 and 1

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$
$$= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}.$$

(Deeplizard.com)

Agent will care more about the immediate reward over future rewards since future rewards will be more heavily discounted.

$$= R_{t+1} + \gamma\big(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots\big) \quad G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}.$$
$$= R_{t+1} + \gamma G_{t+1}$$

(Deeplizard.com)

How a State is a Good State

How likely is for the agent to select a specific action for a given state. The strategy that the agent used to determine its net action. Policy (π) a function that maps a given state to probabilities of selecting each possible action from that state. An agent follows policy π at time t, then $\pi(A_t|S_t)$.

State-Value Function are functions of states, or of state-action pairs that estimates how good it is for the agent to perform a given action in a given state is is given in terms of expected return. The state-value function for policy π, denoted as Value of state Vπ. The value of state S_t under policy π is the expected return from starting from state S at time t and following policy π thereafter

$$v_\pi(s) = E_\pi[G_t \mid S_t = s]$$
$$= E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right].$$

(Deeplizard.com)

Action-Value Function is a function of actions that estimates how good the action is under policy. Value of action A_t in state St under policy π is the expected return from starting from state St at time t, taking action At, and following policy π thereafter.

$$q_\pi(s, a) = E_\pi[G_t \mid S_t = s, A_t = a]$$
$$= E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right].$$

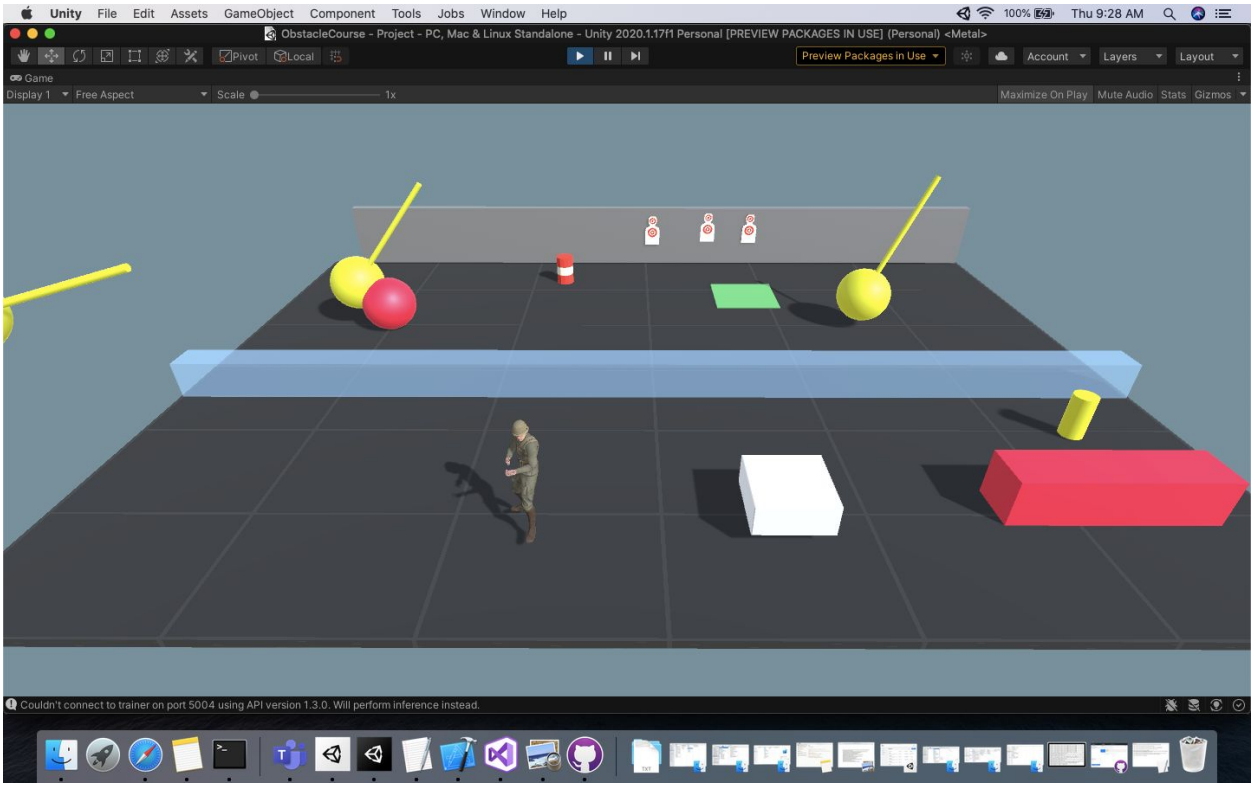
(Deeplizard.com)

The action-value function q π is referred to as the Q-function, and the output from the function for any given state-action pair is called a Q-value.

Exploration Vs. Exploitation

Exploration is the act of exploring the environment to find out information about it. Exploitation is the act of exploiting the information that is already known about the environment in order to maximise the return.

A-life Army Soldier in a training environment.



This game application is a simulation training for an Army soldier Agent. Just like the Army soldiers train in a boot camp. There are a number of obstacles, and the Agent needs to perform several tasks as a series of steps, the animation component has been added for the automated agent to transition to for example an animation for walking idle, walk, run, jump, crawl, push objects etc. The other possible action by the agent would be to hit dummy targets and climb a wall. The AI needs to prevent collisions and go out of bounds. The game is reset after every end of steps by the agents or if the agents fall off the bounds. The Agent needs to maximize the rewards for the training. The agent once changed using the Machine learning process can be switched with another humanoid character. If there are multiple agents they may appear to complete to grab the block to climb up or they may cooperate, this gives rise to emergent behaviour which was not coded.

The asserts used in the application have been downloaded from unity assets store and are free to use. This include Obstacle Course Pack (Obstacle Course Pack, AISUKAZE STUDIO) and Rifle 8-Way Locomotion pack downloaded from Maximo

Unity ML Agents and Python Tensorflow.

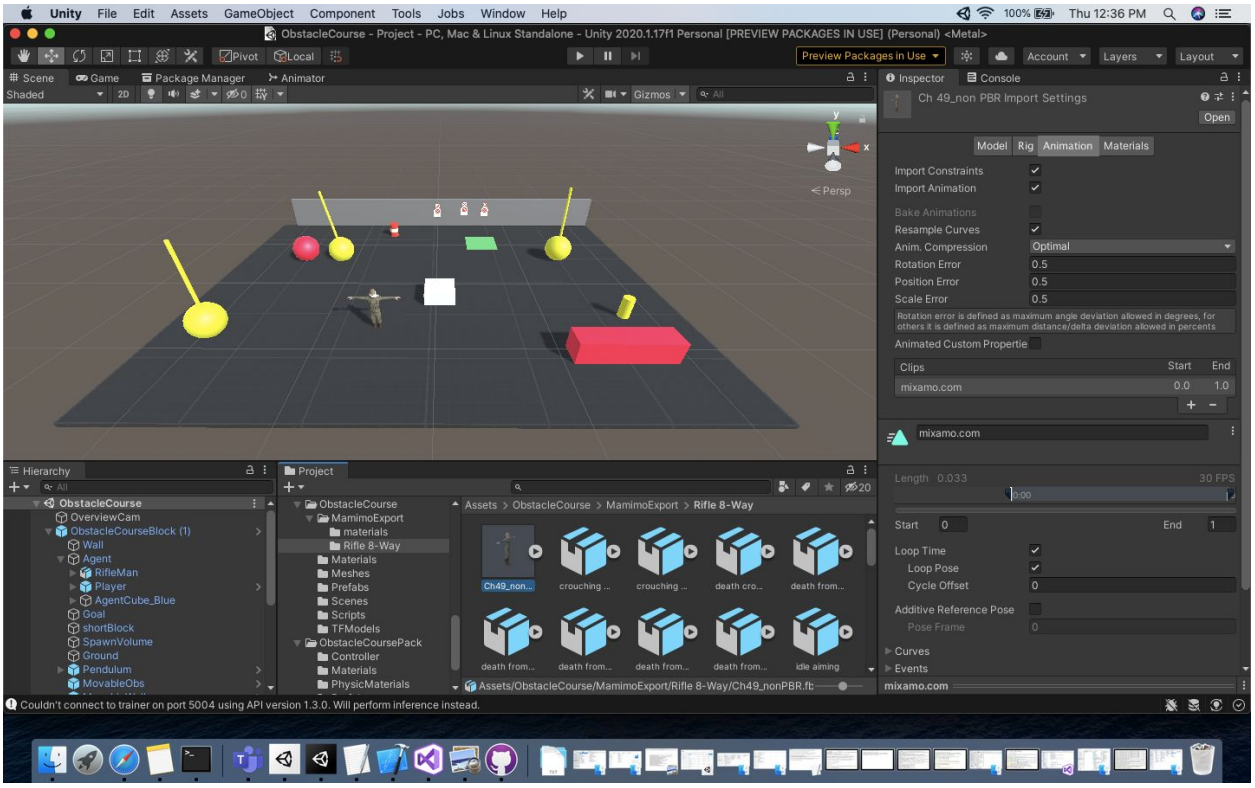
(Unity Machine Learning Agents) With Unity Machine Learning Agents (ML-Agents), you are no longer “coding” emergent behaviors, but rather teaching intelligent agents to “learn” through a combination of deep reinforcement learning and imitation learning. Using ML-Agents allows developers to create more compelling gameplay and an enhanced game experience.

ML-Agents can be downloaded via github my cloning the repository. The application uses the latest files from revision 12 updated on 23 December 2020. The initial setup on Mac and windows can be courorsome, there is help available on stackoverflow.

The current application is using an existing template for ML Agents. This is based on the knowledge gained after going through a couple of tutorials on ML agents on Udemy.com and youtube.com. the references have been included for the same.

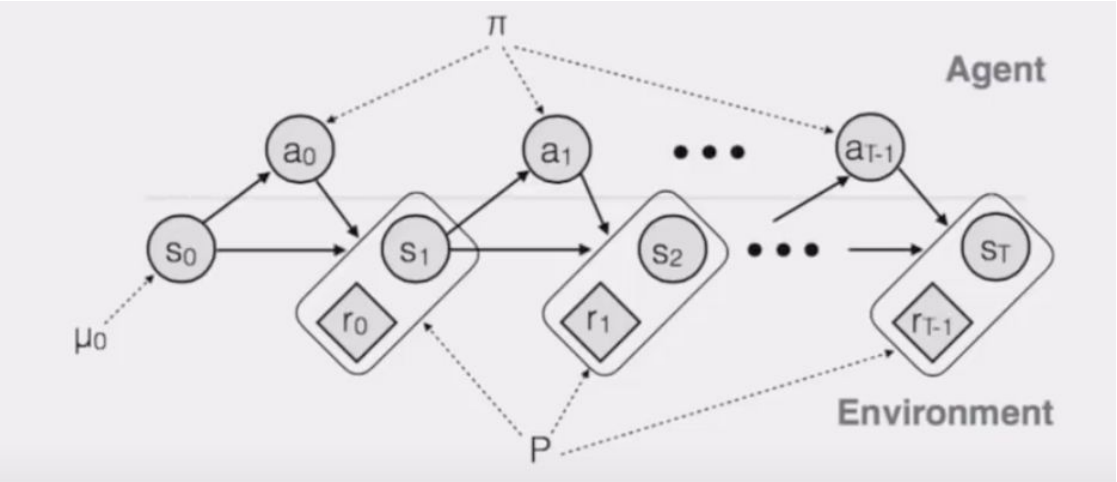
The application does not modify the python code written in Pytorch framework, as this is for advance purpose.

Unity Application setup unity game engine version 2020.17f is being used and the assets are converted to prefabs so they can be added to a scene as one object. This also increases the speed of training the agent using deep neural network.



As this application is making use of neural network we are interested in Value function and the learning rate parameters. The Agent used discrete action and not continuous as parameterized action for MDP. With a **discrete action space**, the agent decides which distinct action to perform from a finite action set.

PPO = Proximal policy optimization (Policy gradient method.) maximize η(π), where $\eta(\pi) = \Sigma [r_0 + r_1 + \dots r_{T-1} \mid \pi(At|St)]$



(Lex Fridman, Deep Reinforcement learning)

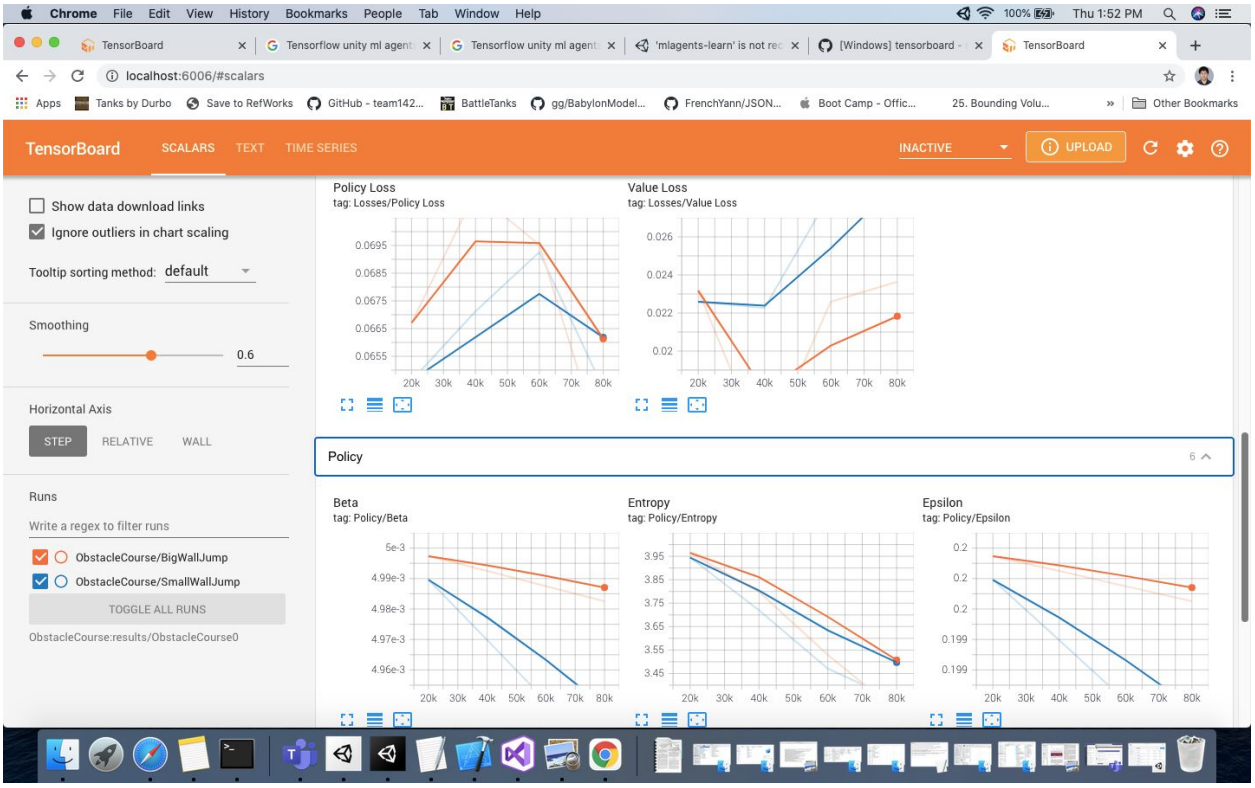
Epison = between 0 and 1. value for exploration < 0.5 or exploitation > 0.5.

Once the model has been trained it is able to generalize it;s learning and able to perform based on the best strategy / policy that gets it maximum rewards.

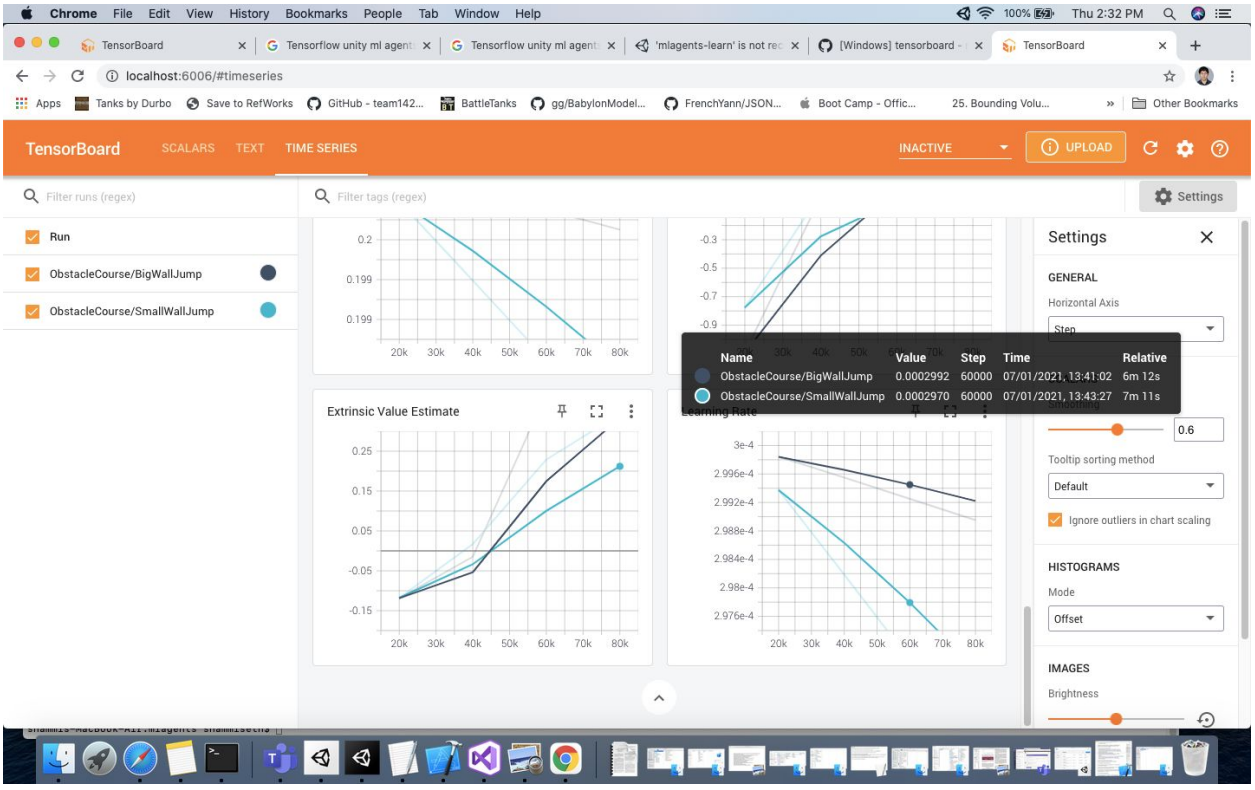
One the training is complete the Learned parameters is saved a a json file and this contains all the weights for the q-value, other learnable parameters more importantly if the agent has learnt enough it is able to use the same strategy for similar scenarios thereafter.

Tensorflow board.

This is part of the Python libraries tensorflow 1.17 has been installed ML Artificial Neural Network the gaphich api provides real time data visulation to look athe learning outcomes form the training. can be started by tensorboard --logdir=summaries



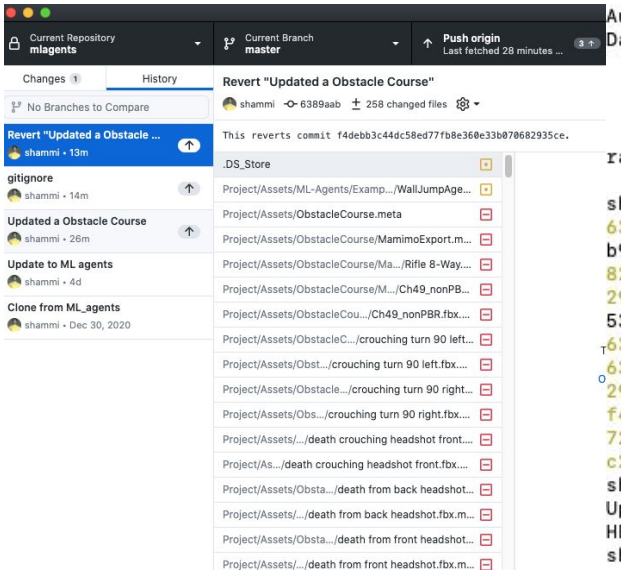
Above image shows the data shown by tensorflow grams during the training of ObstacleCourse It seems that the Agent is not able to learn enough about how to jump a big wall. This is using the time series tab.



Issues.

During the training it was observed that the model was stuck on the barrel for a long time and this needed to reduce the number of max steps in an epoch.

As the project was using github on two occasions there seems to be some issue with github desktop it would accept a commit and not be able to push updates as the maximum size of file was 100 mb. the locomotion characters files size exceed the limit. by mistake a reverted the history and most of the changes were lost. When the user restores this from the local repo then it creates a new branch, however the second time was not pushed to the remote. And the branch was lost, this requires a special command git reflog to restore the branch only from the local. This lead to a lot of problems, its better to use git bash then to use the application with limited features.




```
Author: shammi <ssshammi@hotmail.com>
Date:   Wed Dec 30 15:54:27 2020 +0000

    Clone from ML_agents

    Clone from https://github.com/Unity-Technologies/ml-agents/tree/release_12_b
ranch Branch 12 release.

shammis-MacBook-Air:mlagents shammiseth$ git reflog
6389aab (HEAD -> master) HEAD@{0}: checkout: moving from 825b61d776131d1082e57c8
b9394429a6b8fe0a6 to master
825b61d HEAD@{1}: commit: updated again All done
29b08eb (tag: was-working) HEAD@{2}: checkout: moving from master to 29b08eb00cb
5370288cd44d139fb13aa3651b2ff
6389aab (HEAD -> master) HEAD@{3}: reset: moving to HEAD
6389aab (HEAD -> master) HEAD@{4}: revert: Revert "Updated a Obstacle Course"
29b08eb (tag: was-working) HEAD@{5}: commit: gitignore
f4debb3 HEAD@{6}: commit: Updated a Obstacle Course
72a06ac (origin/master) HEAD@{7}: commit: Update to ML agents
c2102bc HEAD@{8}: commit (initial): Clone from ML_agents
shammis-MacBook-Air:mlagents shammiseth$ git reset --hard 825b61d
Updating files: 100% (639/639), done.
HEAD is now at 825b61d updated again All done
shammis-MacBook-Air:mlagents shammiseth$
```

Results

Below are the test results from the machine learning

Cumulative reward with Discount

Hyper parameters in Deep Neural Network

The hyperparameters for neural network and other inputs are located needs to be in the rage as below. The initial configuration is listed in the table below.

Learning Rate Range: 0.003 to 5e-6

Discount Factor Gamma Range 0.8 to 0.9997

Minibatch, timesteps_per_batch 4 to 4096

Epoch 3 to 30

```
 Version information:
  ml-agents: 0.23.0,
  ml-agents-envs: 0.23.0,
  Communicator API: 1.3.0,
  PyTorch: 1.7.1
2021-01-07 13:31:49 INFO [learn.py:275] run_seed set to 5972
2021-01-07 13:31:49 INFO [environment.py:205] Listening on port 5004. Start training by pressing the Play button in the Unity Editor.
2021-01-07 13:31:56 INFO [environment.py:111] Connected to Unity environment with package version 1.7.2-preview and communication version 1.3
2021-01-07 13:31:56 INFO [environment.py:271] Connected new brain:
SmallWallJump?team=0
2021-01-07 13:31:56 INFO [environment.py:271] Connected new brain:
BigWallJump?team=0
2021-01-07 13:31:56 WARNING [stats.py:190] events.out.tfevents.1610026288.shammis-MacBook-Air.local.7637.0 was left over from a previous run.
2021-01-07 13:31:56 INFO [stats.py:147] Hyperparameters for behavior name BigWallJump:
```

```
2021-01-07 13:31:56 WARNING [stats.py:190]
events.out.tfevents.1610026288.shammis-MacBook-Air.l
ocal.7637.1 was left over from a previous run. Deleting.
2021-01-07 13:31:56 INFO [stats.py:147]
Hyperparameters for behavior name SmallWallJump:
        trainer_type:       ppo
        hyperparameters:
          batch_size:       128
          buffer_size:      2048
          learning_rate:    0.0003
          beta:     0.005
          epsilon: 0.2
          lambd:  0.95
          num_epoch:        3
          learning_rate_schedule:     linear
```

```
network_settings:
            normalize:      False
            hidden_units:   256
            num_layers:     2
            vis_encode_type: simple
            memory:         None
reward_signals:
            extrinsic:
              gamma:        0.99
              strength:     1.0
          init_path: None
          keep_checkpoints: 5
          checkpoint_interval:          500000
          max_steps:        5000000
          time_horizon:     128
          summary_freq:     20000
          threaded:True
          self_play:None
          behavioral_cloning:           None
          framework:        pytorch
```

Mean Square and Loss function

```
2021-01-07 13:34:49 INFO [stats.py:139] BigWallJump. Step: 20000. Time Elapsed: 180.724 s. Mean Reward:
-1.094. Std of Reward: 0.493. Training.
2021-01-07 13:36:16 INFO [stats.py:139] SmallWallJump. Step: 20000. Time Elapsed: 267.000 s. Mean Reward:
-0.776. Std of Reward: 0.704. Training.
2021-01-07 13:37:15 INFO [stats.py:139] BigWallJump. Step: 40000. Time Elapsed: 326.753 s. Mean Reward:
-0.001. Std of Reward: 0.801. Training.
2021-01-07 13:40:18 INFO [stats.py:139] SmallWallJump. Step: 40000. Time Elapsed: 509.278 s. Mean Reward:
0.023. Std of Reward: 0.852. Training.
2021-01-07 13:41:02 INFO [stats.py:139] BigWallJump. Step: 60000. Time Elapsed: 553.112 s. Mean Reward:
0.498. Std of Reward: 0.636. Training.
2021-01-07 13:43:27 INFO [stats.py:139] SmallWallJump. Step: 60000. Time Elapsed: 698.089 s. Mean Reward:
0.210. Std of Reward: 0.831. Training.
```

```
2021-01-07 13:44:31 INFO [stats.py:139] BigWallJump. Step: 80000. Time Elapsed: 762.181 s. Mean Reward:
0.697. Std of Reward: 0.436. Training.
2021-01-07 13:46:53 INFO [stats.py:139] SmallWallJump. Step: 80000. Time Elapsed: 904.306 s. Mean Reward:
0.356. Std of Reward: 0.774. Training.
2021-01-07 13:47:30 INFO [subprocess_env_manager.py:186] UnityEnvironment worker 0: environment stopping.
2021-01-07 13:47:31 INFO [trainer_controller.py:189] Learning was interrupted. Please wait while the graph is
generated.
2021-01-07 13:47:31 INFO [model_serialization.py:104] Converting to
results/ObstacleCourse0/BigWallJump/BigWallJump-95727.onnx
2021-01-07 13:47:31 INFO [model_serialization.py:116] Exported
results/ObstacleCourse0/BigWallJump/BigWallJump-95727.onnx
2021-01-07 13:47:31 INFO [torch_model_saver.py:116] Copied
results/ObstacleCourse0/BigWallJump/BigWallJump-95727.onnx to results/ObstacleCourse0/BigWallJump.onnx.
2021-01-07 13:47:31 INFO [model_serialization.py:104] Converting to
results/ObstacleCourse0/SmallWallJump/SmallWallJump-83813.onnx
2021-01-07 13:47:31 INFO [model_serialization.py:116] Exported
results/ObstacleCourse0/SmallWallJump/SmallWallJump-83813.onnx
2021-01-07 13:47:31 INFO [torch_model_saver.py:116] Copied
results/ObstacleCourse0/SmallWallJump/SmallWallJump-83813.onnx to
results/ObstacleCourse0/SmallWallJump.onnx.
2021-01-07 13:47:31 INFO [trainer_controller.py:85] Saved Model
```

Conclusions

Autonomous agents need to depict the impression of intelligent behaviour like understanding decisions, collision avoidance, emotions, predictable actions etc. In case multiple agents automation behaviours need to depict swarms, flocking, cohershin, formations etc. Autonomous character determines its own actions, at least to a certain extent. This gives the character some ability to improvise, and frees the animator from the need to specify each detail of every character's motion. These improvisational characters are puppets that pull their own strings.

Further evaluation functions are required for states like seeking an ambush position, retreating, infiltrating, and others. Multi objective optimization in filtering out those possibilities which are obviously inferior to others.

Implementation of AI needs to be in terms of resource utilization and computational efficiency. Game AI needs to be result oriented, allowing inexact and calculated concessions and having enjoyable gameplay experience.

Running the Build

This will be packaged as an executable will only infer from the learning data added. Thus there is no user interaction in this build. To modify the code and debug follow below steps

1. Go to mlagents folder
2. pip3 install agents_envs
3. Open unity Project
4. mlagents-learn config/trainer_config.yaml --run-id=uniqueName --resume or —force
5. tensorboard --logdir_spec ObstacleCourse:results/ObstacleCourse

References

Millington, I 2019, AI for Games, Third Edition, Taylor & Francis Group, Milton. Available from: ProQuest Ebook Central. [27 October 2020].

Artificial Intelligence in Games, Food for Thought Series. Available at: http://www.stat.columbia.edu/~jakulin/FT/

Lex Fridman, Deep Reinforcement Learning (John Schulman, OpenAI). Available at:. https://www.youtube.com/watch?v=PtAIh9KSnjo [Accessed:. 28 Dec 2019]

Reinforcement Learning - Goal Oriented Intelligence, Deep Learning Course. Available at:.https://deeplizard.com/learn/video/nyjbcRQ-uQ8 [Accessed:. 30 Dec 2019]

A Beginner's Guide To Machine Learning with Unity Available at: https://www.udemy.com/course/machine-learning-with-unity

Unity Machine Learning Agents. Available at:. https://unity.com/products/machine-learning-agents

Animate 3D characters for games, Maximo (Available At: https://www.mixamo.com/#/?page=1&query=Soldier&type=Motion%2CMotionPack