# Artificial Intelligence for You - Versão TorchSharp

Guia do Usuário

# AI4USharp (*Artificial Intelligence for You – TorchSharp Version*) é um framework que te permite projetar agentes autônomos para atuar como personagens da Godot. Este framework procura ser amigável, entre ser didático e obter a melhor otimização, optarei pela didática. Com a abstração de agentes da AI4USharp, ao mesmo tempo que você tem uma interface amigável, obtém flexibilidade e poder. Você pode fazer muita coisa sem código, mas se quiser, pode colocar a mão-na-massa do código.

# Instalando a AI4USharp

Faça uma cópia do repositório <https://github.com/gilzamir18/AI4USharp> e comece pelo projeto de exemplo disponível no diretório godotproject. Ou inicie seu próprio projeto, crie um diretório e o nomeie como *addons*, cole nele o diretório ai4u disponível em godotproject/addons/ai4u. Fazendo isso, as classes e objetos da *AI4USharp* ficarão disponíveis para o teu *game*. Se começou um projeto novo, adicione os pré-requisitos da AI4USharp com o comando:

*dotnet add package TochSharp-cpu*

*.*

# O Projeto Test

Se você baixar a AI4USharp e abrir na Godot o padrão em *godotproject*, você poderá explorar um exemplo muito básico de uso da *AI4USharp*. Este exemplo tem o único propósito de teste da convergência do algoritmo PPO. Para começar, inicie a cena *main.tscn*, que será exibida aproximadamente assim

Tela de computador

Descrição gerada automaticamente. Neste exemplo, um bastão (um retângulo preto com pontas vermelha e branca) tem que aprender a tocar na caixa com círculo vermelho e bordas pretas sem colidir com as paredes (retângulos brancos). Você pode modificar este exemplo para fazer algo mais complexo. Ou pode simplesmente executar o treinamento, que inicia com o bastão agindo aleatoriamente e termina com o bastão aprendendo a tocar o alvo.

Para habilitar o treinamento, deixe habilitada a propriedade *Enabled* do componente *TrainerController* e desligando a propriedade *Enabled* do componente *ModelController*. Acompanhe a evolução do treinamento na *aba inferior* ***Output***. Depois de 2000 atualizações, simplesmente feche a janela do jogo.

Observe que o treinamento ocorre em alta velocidade (se você tiver um bom computador). Para executar o modelo (a rede neural) treinada que aprendeu a controlar o bastão, depois do treinamento, altere a propriedade *Default Time Scale* do objeto *LoopController* de 10 para 1. Isso fará com que o agente (a rede neural treinada) execute em uma velocidade normal. Agora desligue a propriedade *Enabled* do *TrainerController* e habilite a propriedade *Enabled* do *ModelController*. Isso fará com que a rede neural seja executada sozinha (já treinada, portanto, sem a necessidade do algoritmo de treinamento em execução). O componente TrainerController executa um treinador (no caso, o MLPPPOTrainer, que é o único disponível, por enquanto). Um treinador é uma implementação de um algoritmo de aprendizado de máquina baseado em aprendizado por reforço. Este algoritmo irá adaptar os pesos da rede para que ela faça o que é dito por meio de funções de recompensa. Neste exemplo, todos estes componentes já foram configurados e a execução da rede neural treinada mostrará o comportamento adequado do agente.

# Funcionalidades e Limitações

Entre ter a implementação de vários algoritmos de aprendizado por reforço, optamos pela implementação de um único algoritmo. Pela experiência do projetista da AI4USharp, o melhor algoritmo para cenários de jogos é o algoritmo PPO (*Proximal Policy Optimization*), pois é um algoritmo fácil de implementar, não exigente em termos de *hardware* comparado a outros algoritmos igualmente eficientes e de fácil parametrização.

A Tabela 1 resume as funcionalidades atuais da AI4USharp. A meta deste projeto é ter todos os casos apresentados nesta tabela implementados. A implementação atual traz o algoritmo PPO assíncrono (inspirado no algoritmo A3C) e na forma *standalone*. Mas ainda não há a implementação em que a entrada é um dicionário e nem suporte a redes recorrentes. Multi-Layer Perceptron (MLP) é a única arquitetura suportada no momento. A funcionalidade MultiInput significa que a especificação de k sensores com k chaves (*Perception Key*) resultará em uma rede neural com *k* entradas nomeadas. Ainda não suportamos esse tipo de rede neural e não há previsão se iremos suportar isso no futuro. Por enquanto, você pode concatenar vários sensores por meio de um sensor do tipo *FloatArrayCompositeSensor*. O algoritmo pode ser assíncrono ou *standlone*. Assíncrono significa a execução simultânea de vários ambientes, cada ambiente computando seus próprios gradientes, que são somados em uma rede neural compartilhada. Essa é uma estratégia semelhante à usada no algoritmo A3C (*Asyncronous Advantage Actor-Critic*). A implementação atual não suporta múltiplas saídas nomeadas (*MultiOutput*). A apenas uma saída, que pode ser discreta ou contínua.

Tabela 1. Algoritmos e suas características.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Algoritmos | MLP | MultiInput | RecurrentNet | Assíncrono | MultiOutput | Contínuo |
| PPO | Sim | Não | Não | Sim | Não | Sim |

Apesar da limitação de que apenas sensores de *arrays* de números de ponto-flutuante podem ser usados na entrada da rede neural, pode-se usar um sensor do tipo *FloatArrayCompositeSensor*, que agrega vários sensores colocamos como filho deste nó. Assim, cria-se uma estrutura em que um *FloatArrayCompositeSensor* é colocado como pai e com vários sensores do tipo *array* de *float* como filhos. Com esta configuração, a AI4USharp apenas enxerga o sensor *FloatArrayCompositeSensor*, mas este sensor agrega em um único *array* os dados retornados pelos seus filhos que seja do tipo *array* de *float* ou de um tipo *primitivo* como *float*, *bool* ou *int* (*bool será interpretado como zero* ou *um*).