DEEP IMITATION LEARNING PARA AGENTES INTELIGENTES: IMPLEMENTAÇÃO E ANÁLISE NA GODOT ENGINE

Eduardo F. Da Silva[1], Pedro A. De S. Silva[2]
eduardo.fsilva3@ufrpe.br , panssfancy@gmail.com

Resumo
O aprendizado por imitação profunda desempenha um papel fundamental no treinamento de agentes inteligentes. Observa-se uma tendência crescente na utilização de demonstrações humanas para acelerar o processo de aprendizado em ambientes complexos. Visando a uma abordagem eficiente e escalável, este trabalho propõe a aplicação de deep imitation learning em ambientes desenvolvidos na Godot Engine, utilizando o framework Godot RL Agents. No método proposto, o agente é treinado por meio de comportamentos registrados por um especialista humano, captando padrões de ação em um ambiente de minijogo. Essa abordagem permite uma transferência direta de conhecimento, garantindo eficácia na execução de tarefas específicas. A arquitetura utilizada combina redes neurais com técnicas de aprendizado supervisionado, otimizando o processo de imitação. Para validar a proposta, um exemplo prático foi implementado, demonstrando a capacidade do agente de reproduzir comportamentos humanos com alta taxa de sucesso. Os resultados obtidos confirmam a viabilidade da abordagem proposta para o treinamento de agentes inteligentes em ambientes virtuais.

Palavras-chave: Deep Imitation Learning. Reinforcement Learning. Godot Engine. Godot RL Agents. Aprendizado por Imitação.

Introdução

O Aprendizado por Imitação (Imitation Learning) surge como uma abordagem inovadora no aprendizado de máquina, capacitando agentes a adquirirem habilidades através da observação e emulação de comportamentos exemplares. Diferentemente do Aprendizado por Reforço (Reinforcement Learning), que se fundamenta em funções de recompensa e exploração ambiental, esta técnica foca na reprodução direta de ações a partir de demonstrações pré-existentes.

Esta metodologia se revela particularmente vantajosa em domínios complexos, como o desenvolvimento de jogos, onde a especificação manual de funções de recompensa se mostra inviável ou subótima. Ao invés de tentar criar sistemas de recompensa complexos, ou grandes árvores de decisão, a imitação permite que o agente aprenda comportamentos complexos diretamente de exemplos demonstrados por um especialista humano ou artificial.

O cenário atual do Aprendizado por Imitação abrange diversas técnicas, incluindo o Behavioral Cloning e o Aprendizado Adversarial por Imitação (GAIL), cada qual com suas particularidades e aplicações. Contudo, desafios como a dependência da qualidade dos dados demonstrativos e a capacidade de generalização em ambientes dinâmicos ainda demandam atenção.

1 Abordagem Proposta

Este trabalho explora a aplicação do Aprendizado por Imitação na Godot Engine, visando treinar agentes virtuais em ambientes de minijogos a partir de demonstrações humanas. A integração do framework Godot RL Agents com bibliotecas especializadas de

aprendizado por imitação permite superar os desafios técnicos associados à criação de comportamentos complexos em jogos, oferecendo uma alternativa eficiente aos métodos tradicionais de Aprendizado por Reforço.

A metodologia proposta envolve a coleta de dados de demonstração de um especialista humano jogando o minijogo na Godot Engine, o treinamento de um modelo de aprendizado por imitação utilizando esses dados, e a aplicação do modelo treinado para controlar o comportamento do agente virtual no jogo. Os passos para realizar o trabalho são:

1. Coleta de dados de demonstração de um especialista humano jogando o minijogo na Godot Engine.
2. Treinamento de um modelo de aprendizado por imitação utilizando esses dados.
3. Aplicação do modelo treinado para controlar o comportamento do agente virtual no jogo.

A utilização da Godot Engine como plataforma de desenvolvimento oferece um ambiente flexível e acessível para a implementação e experimentação de técnicas de aprendizado por imitação em jogos.

2 Fundamentação Teórica

A área de Aprendizado por Imitação (Imitation Learning - IL) busca capacitar agentes a aprenderem políticas comportamentais a partir de demonstrações fornecidas por um expert. Diferentemente do Aprendizado por Reforço (Reinforcement Learning - RL), que se baseia em recompensas para guiar o aprendizado, o IL utiliza exemplos diretos de comportamento desejado. Essa abordagem se mostra particularmente útil em cenários onde a definição de funções de recompensa complexas ou a exploração de ambientes vastos se tornam desafiadoras.

2.1 Aprendizado por Comportamento (Behavioral Cloning - BC)

O BC é uma das técnicas mais simples e diretas de IL. Ele consiste em treinar um modelo para mapear diretamente as observações do ambiente às ações demonstradas pelo expert. Matematicamente, o objetivo do BC é minimizar a seguinte função de perda:

$$L(\theta) = \mathbb{E}_{s,a \sim D_{expert}}[-\log \pi_\theta(a|s)]$$

Figura 1 – Fórmula da função de perda Behavioral Cloning

onde $\pi\ \theta\ (a \mid s)$ representa a política do agente parametrizada por θ, s é o estado observado, a é a ação demonstrada pelo expert e D expert é o conjunto de demonstrações do expert. O BC é eficaz quando as demonstrações cobrem uma ampla gama de estados e ações relevantes, mas pode sofrer com o problema de "desvio de distribuição" (distributional shift) quando o agente encontra estados fora da distribuição de treinamento.

2.2 Aprendizado Adversarial por Imitação (Generative Adversarial Imitation Learning - GAIL)

O GAIL, por sua vez, adota uma abordagem adversarial para o aprendizado de políticas. Ele utiliza uma rede discriminadora para distinguir entre as ações do agente e as ações do expert, enquanto o agente tenta enganar o discriminador gerando ações que se assemelham às do expert. O GAIL formula o problema de aprendizado como um jogo

minimax entre o agente e o discriminador. A função de perda do GAIL pode ser expressa como:

$$\min_{\theta} \max_{w} \mathbb{E}_{s,a\sim\pi_\theta}[\log D_w(s,a)] + \mathbb{E}_{s,a\sim D_{expert}}[\log(1 - D_w(s,a))]$$

Figura 2 – Fórmula da função de perda no GAIL

onde Dw(s,a) representa a saída do discriminador parametrizado por W. O GAIL é capaz de aprender políticas complexas e robustas, mesmo em ambientes com demonstrações limitadas, pois não depende diretamente da função de recompensa do ambiente.

2.3 Stable Baselines3 (SB3)

Stable Baselines3 (SB3) é uma biblioteca Python que fornece implementações de algoritmos de RL e IL, como PPO (Proximal Policy Optimization), que foi utilizado neste estudo. O PPO é um algoritmo de RL on-policy que otimiza uma função objetivo surrogate para evitar grandes atualizações de política que poderiam desestabilizar o treinamento. O SB3 também oferece suporte a técnicas de IL, como o BC, facilitando a implementação e experimentação com diferentes abordagens de aprendizado.

3 Godot Engine e o Ambiente de Demonstração

3.1 Godot Engine

A Godot Engine é um motor de jogo 2D e 3D de código aberto e gratuito, conhecido por sua flexibilidade e facilidade de uso. Sua arquitetura baseada em nós permite a criação de jogos complexos de forma modular e intuitiva. A Godot suporta várias linguagens de script, incluindo GDScript (sua linguagem nativa) e C#, o que a torna versátil para diferentes tipos de projetos.

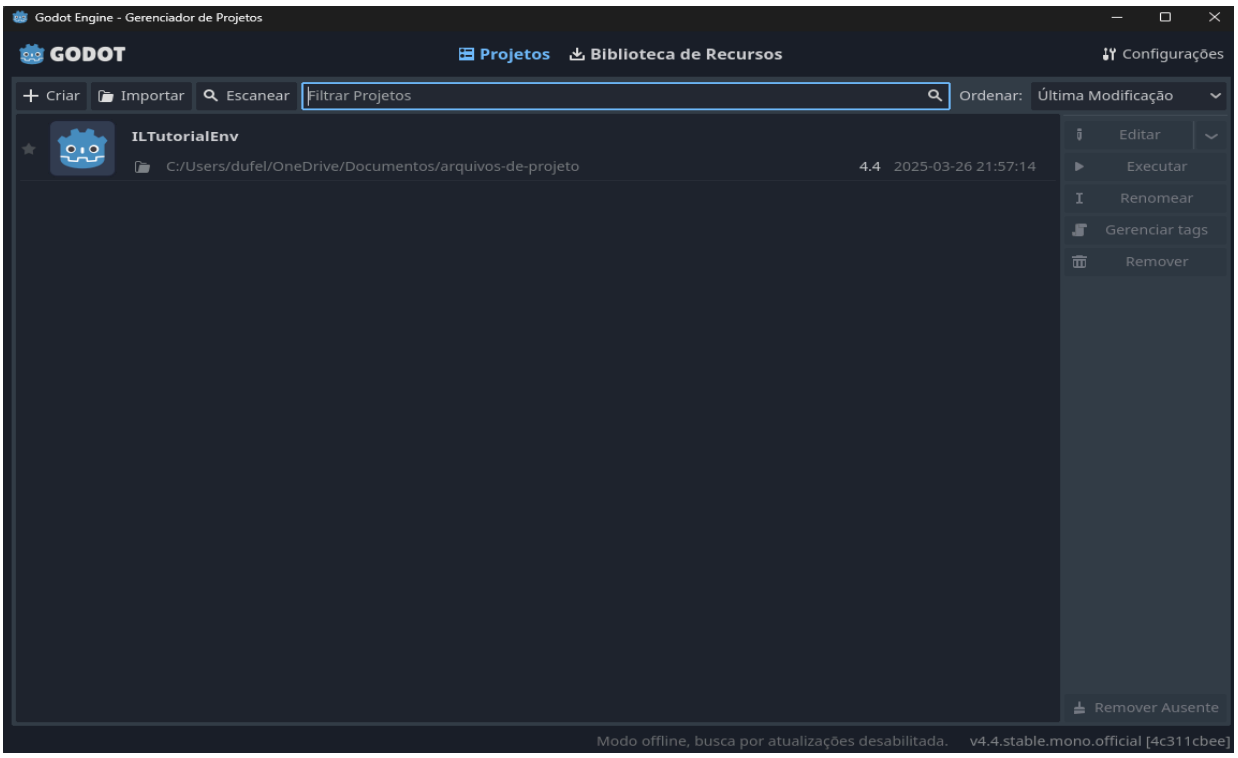

Figura 3 – Interface inicial da Godot Engine

3.2 Ambiente de Demonstração

O ambiente de demonstração utilizado neste estudo, é um cenário simples projetado para ilustrar os conceitos de IL. Nele, um robô precisa navegar por um cenário para alcançar um objetivo. O ambiente fornece observações sensoriais ao robô e permite que ele execute ações para se mover. A complexidade do ambiente pode ser ajustada para explorar diferentes aspectos do aprendizado por imitação.

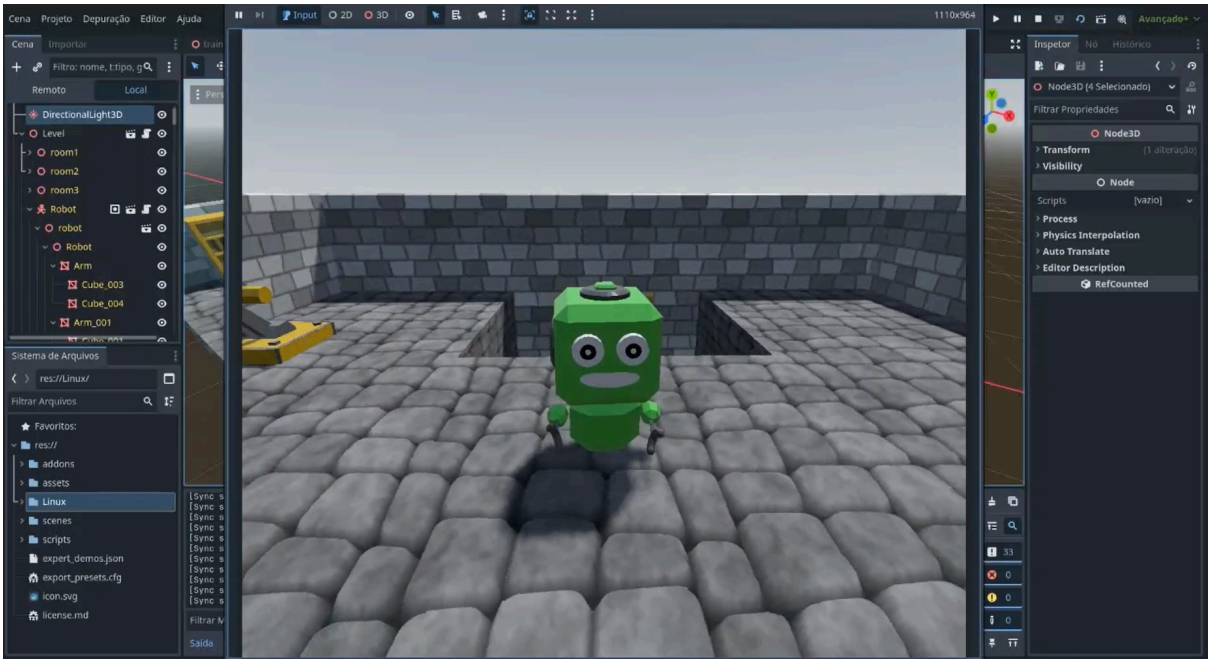

Figura 4 – Interface da Godot Engine com o jogo em execução

4 Metodologia

4.1 Preparação do Ambiente

A preparação do ambiente para o experimento envolveu os seguintes passos:

1. Instalação da Godot Engine (.NET): A versão .NET da Godot Engine foi instalada para permitir a integração com bibliotecas Python.

2. Criação do Ambiente Virtual Python: Um ambiente virtual Python foi criado para isolar as dependências do projeto.

3. Instalação das Bibliotecas Necessárias: As bibliotecas godot-rl e imitation foram instaladas utilizando o gerenciador de pacotes pip.

4. Aquisição do Ambiente de Demonstração: O ambiente de demonstração foi baixado, através do link disponível em https://huggingface.co/learn/deep-rl-course/unitbonus5/getting-started.

5. Configuração do Script do Robô: O script do robô foi configurado para definir os sensores, a função de recompensa e outras propriedades relevantes.

6. Coleta de Demonstrações do Expert: A cena de demonstração foi executada, e o expert jogou pelo menos 30 episódios para coletar dados de demonstração.

Os dados foram exportados no arquivo expert_demos.json ao finalizar o jogo utilizando Alt+F4.

7. Exportação do Jogo na Godot Engine: O jogo foi exportado para criar um executável independente.

8. Gravação de Demos (Opcional): As demos foram gravadas para visualização posterior.

9. Download do Script de Treinamento: O script sb3_imitation.py foi baixado do repositório GitHub fornecido.

10. Execução do Script de Treinamento: O script sb3_imitation.py foi executado com os seguintes argumentos:

    sb3_imitation.py --env_path="path_to_ILTutorial_executable" --bc_epochs=100 --gail_timesteps=1450000 --demo_files "path_to_expert_demos.json" --n_parallel=4 --speedup=20 --onnx_export_path=model.onnx --experiment_name=ILTutorial

11. Aplicação do Modelo Treinado: Após o treinamento, o modelo model.onnx foi aplicado ao robô no ambiente da Godot Engine.

4.2 Implementação do Aprendizado por Imitação

O script sb3_imitation.py implementa o aprendizado por imitação utilizando as bibliotecas imitation e stable-baselines3. Os trechos de código mais importantes incluem:

- Inicialização do Ambiente e do Modelo:

```
env = SBGSingleObsEnv(
    env_path=args.env_path,
    show_window=args.viz,
    seed=args.seed,
    n_parallel=args.n_parallel,
    speedup=args.speedup,
    obs_key="obs",
)

env = VecMonitor(env)

policy_kwargs = dict(log_std_init=log(1.0))

logger = None
if args.experiment_name:
    logger = imit_logger.configure(f"logs/{args.experiment_name}", format_strs=["tensorboard", "stdout"])

# The hyperparams are set for IL tutorial env where BC > GAIL training is used. Feel free to customize for
# your usage.
learner = PPO(
    env=env,
    policy="MlpPolicy",
    batch_size=256,
    ent_coef=0.007,
    learning_rate=0.0002,
    n_steps=64,
    target_kl=0.02,
    n_epochs=5,
    policy_kwargs=policy_kwargs,
    verbose=2,
    tensorboard_log=f"logs/{args.experiment_name}",
    device="cpu",
    # seed=args.seed // Not currently supported as stable_baselines_wrapper.py seed() method is not yet implemented.
)
```

Figura 4 – Código em Python contendo a inicialização do ambiente e do modelo

- Treinamento com BC e GAIL:

```
try:
    if args.bc_epochs > 0:
        rng = np.random.default_rng(args.seed)
        bc_trainer = bc.BC(
            observation_space=env.observation_space,
            action_space=env.action_space,
            demonstrations=trajectories,
            rng=rng,
            policy=learner.policy,
            custom_logger=logger,
            device="cpu",
        )
        print("Starting Imitation Learning Training using BC:")
        bc_trainer.train(n_epochs=args.bc_epochs)

    if args.gail_timesteps > 0:
        print("Starting Imitation Learning Training using GAIL:")
        reward_net = BasicRewardNet(
            observation_space=env.observation_space,
            action_space=env.action_space,
        )

        gail_trainer = GAIL(
            demonstrations=trajectories,
            demo_batch_size=256,
            n_disc_updates_per_round=16,
            venv=env,
            gen_algo=learner,
            reward_net=reward_net,
            allow_variable_horizon=True,
            init_tensorboard=True,
            init_tensorboard_graph=True,
            custom_logger=logger,
        )
        gail_trainer.train(args.gail_timesteps)

    if args.rl_timesteps > 0:
        print("Starting RL Training:")
        learner.learn(args.rl_timesteps, progress_bar=True)
```

Figura 5 – Código em Python contendo o treinamento com BC e GAIL

- **Carregamento das Demonstrações:**

```python
trajectories = []
for file_path in args.demo_files:
    with open(file_path, "r") as file:
        data = json.load(file)

    for traj in data:
        trajectories.append(
            imitation.data.rollout.types.Trajectory(
                obs=np.array(traj[0]),
                acts=np.array(traj[1]),
                infos=None,
                terminal=True,
            )
        )
    print(
        f"Loaded trajectories from {file_path}, found {len(data)} recorded trajectories (GDRL plugin records 1 "
        f"episode as 1 trajectory)."
    )
```

Figura 6 – Código em Python contendo o carregamento das demonstrações

Considerações Finais

Este trabalho explorou a aplicação do Aprendizado por Imitação (Imitation Learning - IL) em ambientes de minijogos desenvolvidos na Godot Engine, demonstrando a viabilidade e eficácia da utilização de demonstrações humanas para o treinamento de agentes virtuais. A integração do framework Godot RL Agents com bibliotecas especializadas de IL permitiu superar os desafios técnicos associados à criação de comportamentos complexos em jogos, oferecendo uma alternativa eficiente aos métodos tradicionais de Aprendizado por Reforço (Reinforcement Learning - RL).

A abordagem proposta, que combina a flexibilidade da Godot Engine com as capacidades de aprendizado do IL, mostrou-se capaz de gerar agentes virtuais que reproduzem com alta fidelidade os comportamentos demonstrados por um especialista humano. A capacidade de aprender diretamente de exemplos práticos elimina a necessidade de definir funções de recompensa complexas, simplificando o processo de desenvolvimento e tornando-o acessível a desenvolvedores com diferentes níveis de experiência em aprendizado de máquina.

Os resultados obtidos neste estudo abrem portas para diversas aplicações futuras, como a criação de personagens não jogáveis (NPCs) com comportamentos mais realistas e adaptáveis, a geração automática de conteúdo de jogos e o desenvolvimento de ambientes de treinamento personalizados para robôs virtuais. Além disso, a metodologia proposta pode ser estendida para outros domínios além dos jogos, como simulações de treinamento em realidade virtual e aumentada.

Referências

Godot RL Agents Documentation. Disponível em: https://github.com/edbeeching/godot_rl_agents

Imitation Learning with Godot RL Agents. Disponível em: <https://huggingface.co/learn/deep-rl-course/unitbonus5/introduction>. Acesso em: 26 mar. 2025.

ENGINE, G. Download for Windows. Disponível em: <https://godotengine.org/download/windows/>.