AI4U Connect Godot Edition

Guia do Desenvolvedor – Versão Beta 2 (16 de dezembro de 2022)

Gilzamir Gomes (Programador/Analista)

Eduardo Nogueira (Game Designer)

Sumário

Introdução

AI4U Godot Edition (AI4UGE) possibilita a conexão transparente de Python com Godot para a criação de experimentos de inteligência artificial com personagens virtuais. Com isso, você pode usar o potencial de frameworks de aprendizado de máquina disponíveis em Python ou em C# e criar modelos de controladores de NPCs (*Non-Player Characters*) ou de personagens virtuais baseados em redes neurais. Além disso, pode-se conectar qualquer script em Python baseado em AI4UGod para controlar itens de jogos.

AI4UGE foi refinada para o desenvolvimento de experimentos de desenvolvimento de NPCs usando inteligência artificial. Para isso, fornece uma forma lúdica e transparente de modelagem de NPCs por meio de uma abstração de agentes inteligentes. Você constrói o NPC adicionando atuadores e sensores, como se estivesse construindo um robô, contudo, com um corpo virtual. Isso nos provê, além de um escopo de experimentação, um apelo educacional poderoso.

Portanto, as principais características de AI4UGE são:

- desenvolvimento de controles de jogos baseados no paradigma de agentes em Inteligência Artificial;
- *multi-engine*, suporta jogos modelados na *Unity* ou na *Godot*, sem a necessidade de reprogramação dos scripts Python; e
- Modelagem de ambiente baseado no framework *Gym*.

A Modelagem do Ambiente

A modelagem do ambiente, apesar do apelo visual, deve enfatizar os aspectos físicos, como obstáculos, pisos, construções, passagens e colisões. A AI4UGE possui sensores de colisão com as formas padrões de corpo rígido disponíveis na *Godot*. Portanto, essa modelagem é muito dependente do motor de jogos que você usa.

Godot possui uma interface agradável para modelagem do ambiente. Na Figura 1, pode-se visualizar uma cena muito simples, mas com os elementos necessários em um ambiente: dois objetos (uma capsula com seta que representa o corpo do agente e um cubo que representa um item que o agente tem que capturar), uma câmera e um piso (um terreno plano simples). O que não se pode ver é o motor de física que adiciona gravidade ao ambiente, fixa o piso como um chão impedindo os objetos de caírem para sempre e o mecanismo que permite a colisão entre objetos. Todos estes elementos estão presentes no projeto AI4UGTesting (disponível em *exemplos/serverside/Godot/AI4UGTesting*). Iremos nos basear neste projeto para explicar os diferentes componentes da AI4UGE.

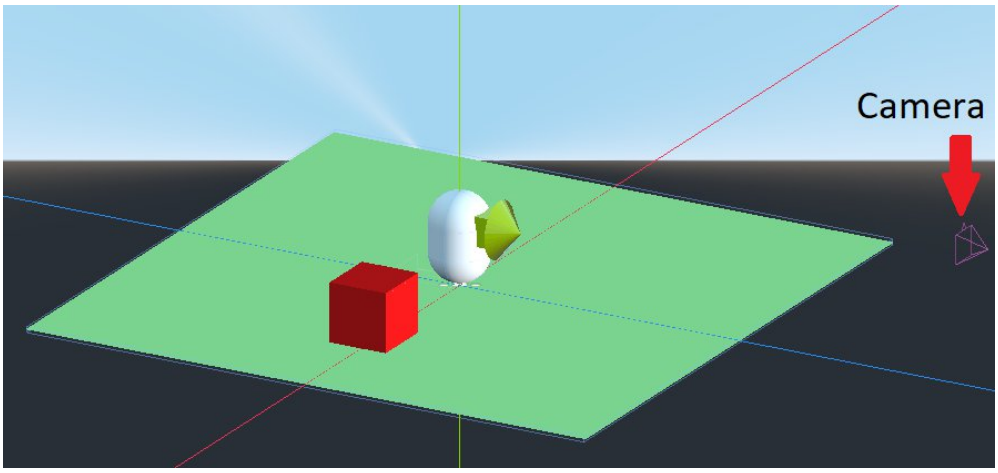

Figura 1. Uma cena muito simples modelada usando Godot.

O ambiente pode ser modelado como em qualquer jogo, exceto que os aspectos de interação do agente com o ambiente devem ser especificados. Portanto, o próximo passo para se compreender a modelagem do ambiente é compreender a modelagem do agente.

Modelagem do Agente

O primeiro passo para a criação de um agente com um corpo físico é criar uma hierarquia com um nó do tipo *RigidBody* como raiz dessa hierarquia, como mostrado na Figura 2.

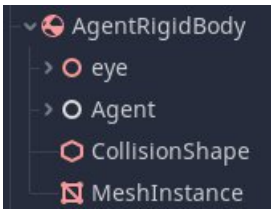

Figura 2. Configuração de um Agente na Godot.

O objeto *AgentRigidBody* (Figura 2) é um objeto do tipo *RigidBody* e tem quatro nós filhos: *eye*, *Agent*, *MeshInstance* e *CollisionShape*. O penúltimo nó serve para dar uma forma física ao objeto e o último serve para se criar uma representação visual e geométrica dessa forma física. Resumindo, criar nós *MeshInstance* e *CollisionShape* como filhos do RigidBody constitui a forma de se modelar um corpo rígido em Godot. Além disso, para que um corpo rígido interaja com o ambiente e produza o comportamento esperado pela AI4UGE, é necessário configurar adequadamente os parâmetros da classe *RigidBody*. Exibe-se na Figura 3 um exemplo de configuração adequada de um corpo rígido que pode ser comportar como um NPC controlado pelo atuador *RBMoveActuator*.

Figura 3. Exemplo de configurações adequadas de um corpo rígido.

Já o nó *Agent* representa o “cérebro” do agente. Para criarmos um agente, precisamos criar um objeto do tipo *BasicAgent*. Para isso, primeiro adicionamos um nó do tipo *Node* como filho do objeto *AgentRigidBody*, então o associamos ao *script* da classe *BasicAgent*. Este *script* vai se encarregar de conectar os sensores e atuadores ao corpo rígido. Um objeto do tipo *BasicAgent* depende de 4 tipos de componentes para funcionar:

- Actuator;
- Sensor;
- ControlRequestor; e
- Brain.

Estes componentes são criados da mesma forma que criamos o objeto *Agent*, mas os associamos a *scripts* diferentes, por exemplo, um nó atuador pode ser do tipo *RBMoveActuator* e um exemplo de sensor é a classe *RaycastingSensor*. Para serem reconhecidos como parte do agente, estes componentes devem ser adicionados como filhos do nó *BasicAgent*. A seguir, estes componentes serão explicados separadamente.

A classe *ControlRequestor*

Um agente possui sensores e atuadores e, além disso, uma forma de contagem de tempo própria, sincronizada com o tempo físico do motor de física do jogo. *Godot* provê o método *_PhysicsProcess* de atualização da física do jogo. Ou seja, *_PhysicProcess* executa o laço de atualização física do jogo. Dentro deste laço de atualização física do jogo, rodamos o laço de tomada de decisão, representado pela classe *ControlRequestor*. Em analogia a um cérebro biológico, *ControlRequestor* funciona como o hipotálamo e o nó Agent do tipo *BasicAgent* como o próprio cérebro.

Durante o laço físico do *ControlRequestor*, ocorre um ciclo de decisão. Em um ciclo de decisão, o agente envia uma mensagem para um objeto do tipo *Brain* requisitando alguma informação de controle (sincronização de dados do ambiente ou de ações) e recebe de volta informações do ambiente. Estas informações são de controle e o próprio estado do ambiente atualizado. Portanto, um ciclo de decisão pode ocorrer ao longo de várias iterações do laço de atualização física do jogo. A forma exata como este ciclo de decisão opera depende de sete atributos da classe *ControlRequestor*:

Atributo	Descrição
Skip Frame	O número de iterações do laço de física (execuções do método *_PhysicProcess*) que deve ser ignorado entre ciclos de decisão.
Repeat Action	Se a ação escolhida no início de um ciclo de decisão deve ser repetida nas iterações ignoradas (definidas pelo atributo *skip frame*) do laço de física.
Default Time Scale	O valor do atributo *Engine.TimeScale*. Isso tende a diminuir o intervalo de tempo entre iterações físicas. Veja mais sobre isso na documentação oficial da *Godot*.
Brain Mode Path	O modo de execução do laço de decisão: remoto (carrega o *script RemoteBrain*) ou local (carrega o *script LocalBrain*). Se o *script RemoteBrain* for carregado, um protocolo de comunicação baseado em UDP é aberto com um *script* remoto, geralmente codificado em *Python*.

Na Figura 4, mostra-se as propriedades do *ControlRequestor* usada no projeto de exemplo *AI4UGTesting*.

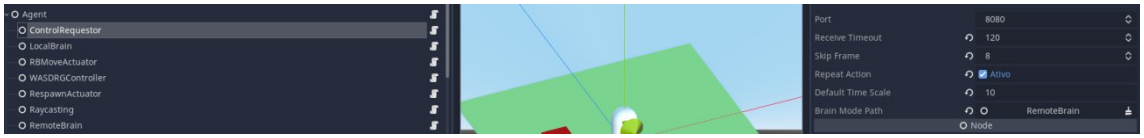

Figura 4. Exemplo de configuração do ControlRequestor.

O par *BasicAgent/ControlRequestor* estabelece uma ordem de chamada dos métodos padrões da AI4UGE:

1. O método *OnSetup* de *BasicAgent* é executado uma vez na criação do cenário e realiza a execução do método *OnSetup* dos componentes do agente, nesta ordem: funções de recompensa (objetos do tipo *RewardFunc*), sensores (objetos do tipo *Sensor*) e atuadores (objetos do tipo *Actuator*). Apenas os componentes ativos filhos do nó *BasicAgent* são adicionados efetivamente ao agente.
2. Sempre que um agente é reinicializado (o método *Reset*) do agente é chamado, todos os componentes reinicializáveis (geralmente sensores, atuadores e funções de recompensa) são reinicializados (a função *OnReset* destes componentes é executada).
3. No início de um ciclo de decisão, o agente aciona os sensores chamando o método *Get<Type>Value()*, onde *<Type>* pode ser um dos tipos de sensores suportados: *Float*, *FloatArray*, *Bool*, *Int*, *IntArray*, *String* e *ByteArray*. Isso faz com que os sensores retornem valores que são enviados ao controlador local ou remoto (dependendo do tipo de Brain utilizado).
4. Depois que todos os dados dos sensores serem capturados, estes dados são enviados para um controlador definido por um objeto do tipo *Brain* (*LocalBrain* ou *RemoteBrain*). O controlador envia de volta uma ação. Esta ação possui um nome. Um atuador que combina com o nome da ação é acionado. Isso significa que o método *Act* do atuador é acionado a cada ciclo físico durante o ciclo de decisão do agente (se a propriedade *Repeat Action* estiver habilitada) ou apenas uma vez no início do ciclo (se a propriedade *Repeat Action* não estiver habilitada). O atuador acionado pode receber os dados enviados pelo controlador chamando métodos adequados do agente. Por exemplo, o método *GetStateAsFloatArray()* de instâncias da classe *Agent* retorna os dados enviados pelo controlador como um arranjo de números reais.
5. Depois que todas as ações do ciclo de decisão forem executadas (geralmente uma), o método *OnUpdate* é executado para cada função de recompensa (objetos do tipo *RewardFunc*) do agente. Isso é necessário para que a recompensa do agente seja calculada como consequência das ações executadas, mantendo a consistência de algoritmos de aprendizado por reforço que se baseiam no ciclo de decisão do agente.

Um programador também pode adicionar controladores de evento disponíveis em *BasicAgent* que possibilitam adicionar comportamento de sensores e atuadores e de quaisquer outros objetos da *Godot* entre as etapas apresentadas:

Evento	Descrição
beforeTheResetEvent	Evento antes de qualquer componente do agente ser reinicializado. Pode ser usado para configurar propriedades necessárias para a inicialização do agente.
endOfEpisodeEvent	Evento que ocorre quando um episódio termina (propriedade *Done* do agente muda para *true*).
beginOfEpisodeEvent	Evento que ocorre na inicialização (e reinicialização) de um agente.
endOfStepEvent	Evento que ocorre no final de um ciclo de decisão do agente.
beginOfStepEvent	Evento que ocorre no início de um ciclo de decisão do agente.
beginOfUpdateStateEvent	Evento que ocorre em todo ciclo antes dos sensores produzirem qualquer valor.
endOfUpdateStateEvent	Evento que ocorre em todos os ciclos depois

	que todos os sensores produziram seus valores de ciclo.
beginOfApplyActionEvent	Evento que ocorre antes de qualquer ação ser executada.
endOfApplyActionEvent	Evento que ocorre depois de todas as ações terem sido executadas.

Atuadores

Um agente (objeto do tipo *BasicAgent*) deve ter um ou mais atuadores (objetos do tipo *Actuator*) para poder funcionar adequadamente. Por padrão, a *AI4UGE* contém uma classe *RBMoveActuator* (que herda de *Actuator*) e que permite criar instâncias que movimentam objetos do tipo *RigidBody*. O desenvolvedor pode criar seus próprios atuadores criando uma classe que herda de *Actuator*.

A Classe *Actuator*

A classe *Actuator* provê uma abstração para o significado de uma ação que afeta o ambiente. Além disso, esta classe provê as informações gerais da ação: o nome, o tipo e a forma.

O nome do atuador é único (dois atuadores não podem ter o mesmo nome), pois é por meio do nome que se determina qual a ação o agente deve executar em determinado momento. O nome do atuador é representado pelo atributo *actionName* e pode ser definido tanto no painel de propriedades do editor da Godot em uma classe concreta que herda da classe *Actuator* quanto no método *OnSetup* da classe filha.

O tipo define o tipo de dado que o atuador recebe para executar a ação que representa e é representado pelo atributo *isContinuous*. Este atributo é protegido e pode apenas ser definido por herança nas classes concretas que herdam de *Actuator*. Recomenda-se definir este valor no construtor da classe filha ou no método *OnSetup*. O tipo sempre será numérico e pode ser contínuo ou não contínuo. Por exemplo, um atuador pode operar recebendo sinais escalares que indicam graus de rotação ou intensidades de força que devem ser aplicadas em um corpo rígido. Outro caso é o atuador representa uma ação categórica, como escolher uma opção dentre muitas opções disponíveis.

A forma do atuador define como os dados sobre os quais o atuador opera estão organizados e é representada pelo atributo protegido *shape*. Este atributo é do tipo *int[]* e deve ser instanciado junto com o tipo do atuador no construtor ou no método *OnSetup* classe filha. Por exemplo, um atuador que produz movimentos de corpos rígidos pode receber dados na forma (4,) que representa um *array* de quatro elementos. No caso específico do RBMoveActuator, esse array é tem os elementos $[f, t, j, jf]$, onde f representa a intensidade do movimento para frente/para trás, t representa o ângulo de rotação do corpo rígido em torno do próprio eixo, j representa um salto vertical e jf, um salto para frente. Essas informações podem ser usadas por um controlador para produzir os dados de entrada do atuador adequadamente.

A AI4UGE possui um atuador *builtin* chamado de *RBMoveActuator*. Este atuador é capaz de controlador objetos do tipo *RigidBody*.

A classe *RBMoveActuator*

Um objeto do tipo *RBMoveActuator* controla objetos do tipo *RigidBody*, eis porque possui a sigla *RB* do início de seu nome. Além disso, como em Godot um objeto *RigidBody* possui quatro modos diferentes, é importante destacar que o atuador *RBMoveActuator* foi projetado

especificamente para controlar objetos do tipo *RigidBody* no modo *RigidBody*. (Veja o campo *mode* na Figura 3. Portanto, não é recomendado usar esse tipo de atuador em um *RigidBody* com outro tipo de modo.

Para adicionar um *RBMoveActuator* ao agente, abaixo do nó *Agent* (veja Figura 2), cria-se um nó do tipo *Node* e o associa ao *script* da classe *RBMoveActuator*.

Figura 5. Exemplo em que se adicionou um objeto do tipo RBMoveActuator ao agente.

Um objeto do tipo *RBMoveActuator* possui seis campos:

Campo	Descrição
Action Name	Deve ser um nome único para cada atuador, pois é por meio deste nome que o atuador será reconhecido pelo agente.
Move Amount (ma)	A quantidade de movimento para frente.
Turn Amount (ta)	A intensidade do giro. Pode ser um valor que varia continuamente entre -1 (giro à esquerda) e 1 (giro à direita).
Jumper Power (jp)	A intensidade máxima do salto (apenas vertical).
Jumper Forward Power (jpf)	A intensidade do salto para frente.

Um agente com um atuador deste tipo pode receber ações da forma *[f, t, j, jf]*, onde $f > 0$ representa um movimento para frente (direção positiva do eixo *z* do objeto) e $f < 0$ representa o movimento oposto; $t > 0$ representa giro para a direita e $t < 0$ representa giro para a esquerda; *j* representa a intensidade do salto e deve ser maior ou igual a zero; e *jf* representa um salto para frente, também deve ser maior ou igual a zero. A intensidade final dos movimentos depende dos valores dos campos do objeto *RBMoveActuator* e será igual a *[f * ma, t * ta, j * jp), jf * jpf]*, em que * representa a multiplicação escalar.

Finalmente, é recomendado que o nome da ação (*Action Name*) seja configurado como “move”, pois este é o nome esperado pelos exemplos de controladores disponíveis na AI4U. Exibe-se na Figura 6 um exemplo completo de configuração deste tipo.

Figura 6. Configuração de um atuador do tipo RBMoveActuator.

A Classe Sensor

Um agente deve ter um modo de perceber o mundo. Para isso, provemos a classe *Sensor*, que permite o agente perceber as propriedades do ambiente no qual atua. A classe *Sensor* provê atributos gerais presentes em todos os sensores. Estes atributos podem ser definidos no próprio editor da *Godot*, mas também podem ser definidos em código, por meio de herança. Na verdade, a classe *Sensor* não representa um sensor específico, mas sim aglomera as propriedades gerais de sensores e os métodos que retornam todos os tipos de dados suportados pela AI4U. Os atributos gerais são cinco:

Atributo	Descrição
perceptionKey	Identificador único do sensor. Esta informação será usada pelo agente para determinar determinado tipo de informação esperada pelo agente.
stackedObservations	A quantidade de informação que o agente recebe no ciclo de decisão *t*. Essa quantidade deve ser maior ou igual a 1. Se for 1, o agente recebe no ciclo de decisão *t* apenas a informação capturada no início do ciclo. Se for $k > 1$, no ciclo de decisão *t*, o agente recebe também das informações capturadas nos ciclos de decisão *t*, *t-1*, *t-2*, ..., *t-k+1*.
isActive	Ativa ou desativa o sensor. Se o sensor estiver desativado antes do início do jogo, o agente ignora este sensor durante a execução do jogo. Isso pode exigir alterações no *script* de controle remoto do agente, pois do ponto de vista é externo, este sensor não fica mais visível.
normalized	Indica ao programador do sensor se os dados devem ser normalizados ou não.
resetable	Indica se o método *OnReset* do sensor deve ser executado quando o agente for reinicializado.

Além dos atributos públicos e externos (que podem ser modificados na interface do editor da Godot), um sensor tem outros quatro atributos que devem ser definidos no construtor ou no método *OnSetup* do sensor:

1. *Type*: o tipo do sensor, que é do tipo de enumeração *SensorType*. *SensorType* pode ser:
    a. *SensorType.sint*,
    b. *SensorType.sfloat*,
    c. *SensorType.sbool*,
    d. *SensorType.sstring*,
    e. *SensorType.sfloatarray*,
    f. *SensorType.sbytearray*,
    g. *Sensortype.sintarray*.
2. *IsState*: indica se a informação do sensor deve ser incorporada ao estado do ambiente ou indica apenas uma configuração inicial do ambiente (algo que não muda ao longo do tempo).
3. *Shape*: as dimensões e seus respectivos tamanhos. Se o sensor produz apenas um valor escalar, a forma é vazia, representada por um instância vazia de um arranjo inteiro: *new int[0]*.
4. *agente*: uma referência para o agente dono do sensor. Apenas um agente pode receber dados de um sensor por vez.

Além destas propriedades, ao criar um sensor, deve-se implementar o método que retorna os dados correspondentes ao tipo do sensor. Por exemplo, se o sensor é do tipo *SentorType.sfloatarray*, deve-se implementar o método *GetFloatArrayValue*() que retorna um arranjo de números reais. Os métodos possíveis para cada tipo de dados são:

Tipo	**Método**
SensorType.sfloat	GetFloatValue()
SensorType.sstring	GetStringValue()
SensorType.sbool	GetBoolValue()
SensorType.sbytearray	GetByteArrayValue()
SensorType.sint	GetIntValue()
SensorType.sintarray	GetIntArrayValue()
SensorType.sfloatarray	GetFloatArrayValue()

A AI4UGE provê alguns sensores embutidos: *ActionSensor*, *DoneSensor*, *FloatArrayCompositeSensor*, *IDSensor*, *OrientationSensor*, *PositionSensor*, *RayCastingSensor*, *RewardSensor* e *StepSensor*.

ActionSensor: este sensor retorna a(s) última(s) ação(ões) realizada(s) pelo agente. Em alguns problemas de aprendizagem por reforço, o histórico de ações facilita a aprendizagem do problema. Este método é do tipo *SensorType.sfloatarray* e a quantidade de ações pode ser definida por meio da propriedade *actionSize* (*Action Size* no editor da Godot).

DoneSensor: este sensor retorna um booleano indicando se o episódio terminou ou não. É do tipo *SensorType.sbool* e não suporta empilhamento de observações, pois serve apenas para indicar aos controladores que um episódio terminou ou não. O valor padrão da propriedade *perceptrionKey* deste sensor é *done*. Todo agente do tipo *BasicAgent* possui este sensor.

FloatArrayCompositeSensor: este objeto se comporta como um sensor, mas é um agregador de sensores de diversos tipos. Este sensor serve para criar uma composição de sensores. Para isso, você deve adicionar ao nó do tipo *FloatArrayCompositeSensor* um ou mais nós filhos de um tipo que herda de *Sensor*. A informação destes sensores é convertida em números reais e colocadas em um arranjo de números reais contendo os valores de todos os sensores agregados. Portanto, este objeto deve ter um ou mais filhos do tipo *Sensor*. Os filhos podem ser de qualquer tipo numérico. O tipo deste sensor é *SensorType.sfloatarray*. Os *scripts* controladores em Python disponível no diretório de exemplos pressupõem que a propriedade *perceptionKey* deste sensor tem o valor *array*.

IDSensor: retorna o identificador do agente no ambiente. Este sensor retorna um código de identificação único do agente no ambiente e está disponível para todos os agentes do tipo *BasicAgent*. O tipo deste sensor é *SensorType.sstring* e o valor padrão da propriedade *perceptionKey* é *id*. Este sensor não pode empilhar informações ao longo de vários ciclos de decisão.

OrientationSensor: retorna dois números reais que indicam a orientação relativa e a distância do agente para um alvo definido pela propriedade *target*. A orientação é calculada como o coseno do ângulo entre os vetores **u** e **v**, em que **u** representa a direção entre o agente e o alvo e **v** representa a direção do agente (eixo que indica a direção do eixo de

visão do agente). A distância é a distância euclidiana entre o agente e o alvo. Este sensor se mostrou eficiente para a geração de comportamentos de navegação espacial. O tipo do sensor é *SensorType.sfloatarray*, que retorna um arranjo $[o, d]$, onde o é a orientação e d é a distância do agente em relação ao alvo. O valor da propriedade *perceptionKey* geralmente é definido pelo usuário na interface da *Godot*, como mostrado na Figura 7.

PositionSensor: retorna a posição do agente em coordenadas locais ou em coordenadas globais. O sensor é do tipo *SensorType.sfloatarray* e retorna um arranjo de números reais na forma [x, y, z].

RayCastingSensor: retorna uma matriz de dimensão *mxn* contendo códigos de objetos detectados no campo de visão do agente. Raios são lançados em perspectiva de uma posição definida na propriedade *eye* do sensor. Para cada objeto intersectado pelo raio, ou o código do objeto ou a distância do agente para o objeto é registrado na matriz na posição correspondente à intersecção do raio no plano entre o observador e o objeto na cena. Este sensor é do tipo *SensorType.sfloatarray* e produz um arranjo com *m x n* elementos. No controlador, geralmente esse arranjo é colocado novamente no formato matricial.

RewardSensor: este sensor retorna a última recompensa produzida pelo usuário. O tipo deste sensor é *SensorType.sfloat* e não suporta a propriedade *stackedObservations*.

StepSensor: retorna o ciclo de decisão atual do agente. O tipo deste sensor é *SensorType.sint*. Este sensor não suporta a propriedade *stackedObservations*.

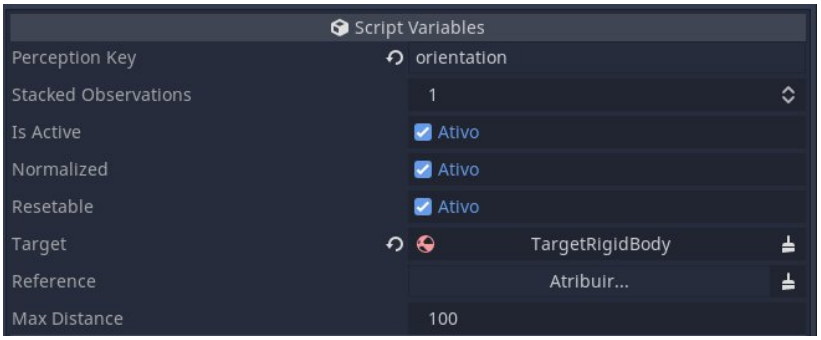

Figura 7. Exemplo de configuração de um sensor do tipo OrientationSensor.

As Classes *Brain*, *RemoteBrain* e *LocalBrain*

O componente *ControlRequestor* depende de um objeto do tipo *Brain*, que pode ser *RemoteBrain* ou *LocalBrain*. O tipo de *Brain* definido em *ControlRequestor* determina o tipo de controlador do agente.

Um *Brain* do Tipo *RemoteBrain*

Um *Brain* do tipo *RemoteBrain* permite o agente ter um controlador remoto, implementado em qualquer linguagem que implemente o protocolo privado da AI4U. Por enquanto, mantemos este protocolo implementado apenas na linguagem Python e testado especificamente na versão *Python 3.7*.

Um *Brain* do Tipo *LocalBrain*

Já um *Brain* do tipo *LocalBrain*, permite o agente ser controlado por *scripts* implementados na própria *engine* de jogos, usando C# ou outra linguagem suportada pela *engine de jogos*. Um objeto *LocalBrain* depende de um objeto de um tipo que herda de *ai4u.Controller*. Por enquanto, o único controlador concreto embutido na AI4UGE é o *ai4u.WASDRBMoveController*, que permite controlar o agente usando as teclas *WASD* do teclado de um PC ou de um notebook.

Contudo, o usuário poderia implementar seu próprio controlador para, por exemplo, executar uma rede neural implementada diretamente em C#. Isso permitiria treinar um controlador usando Python e usar a rede neural depois de treinada sem a necessidade de manter a conexão com a linguagem Python. Essa é uma funcionalidade futura que pretendemos adicionar à AI4UGE.

Criando seu Próprio *Controller*

Um controlador precisa sobrescrever basicamente dois métodos: *GetAction* e *NewStateEvent*. O método *GetAction* é executado quando o *ControlRequestor* envia uma solicitação de controle, então *GetAction* deve retornar uma ação codificada em uma *string*. O programador pode utilizar métodos acessórios, como as várias versões do método *ai4u.Utils.ParseAction*, que transforma um comando em sua codificação como *string*.

No método *NewStateEvent*, o programador pode utilizar diversos métodos acessórios (como *GetStateSize*, *GetStateName* e *GetStateAsFloat*) para verificar qual o estado atual do agente.

Enquanto não se produz uma documentação mais ampla, recomendamos basear o código do controlador no código da classe *ai4u.WASDRBMoveController*.

Funções de Recompensa

Um elemento centralizador de algoritmos de aprendizado de máquina é o tipo de *feedback* que se usa para treinamento dos controladores. AI4U é provê os elementos básicos para o aprendizado por reforço.

A classe *BasicAgent* provê métodos que permitem ao programador adicionar recompensas ao agente em qualquer ponto de um ciclo de decisão do agente. Contudo, para garantir a consistência do laço de decisão e a propriedade *markoviana* de um processo de decisão, deve-se adicionar a recompensa correspondente à ação do agente no ciclo de decisão *t* no final do ciclo de decisão *t*. É importante cada recompensa ficar restrita ao ciclo de decisão na qual foi adicionada. Há duas formas de se fazer isso com AI4U:

- Implementar uma classe que herda de *ai4u.RewardFunc*; criar um nó do tipo dessa classe e filho do nó *BasicAgent*; neste caso, o deve-se sobrescrever o método *OnUpdate*, que é onde se deve adicionar recompensa do ciclo de decisão do agente; ou
- Adicionar um controlador do evento ao evento *endOfStepEvent* de *BasicAgent* e adicionar a recompensa neste controlador.

O interessante da segunda possibilidade é que é possível adicionar recompensa de qualquer componente do agente, seja dentro de um sensor ou de um atuador criados pelo usuário. Na Figura 8, observa-se uma função que adiciona uma recompensa quando o agente toca um cubo. Observe que a variável *acmReward* é zerada (linha 32 do respectivo código-fonte) depois que a

recompensa é adiciona ao agente. Isso evita a propagação da recompensa nos ciclos de decisão subsequentes.

```
using Godot;
using System;
using ai4u;
namespace ai4u {
    public class TouchRewardFunc : RewardFunc {
        [Export]
        public float reward = 0.0f;
        [Export]
        public NodePath targetPath;
        private Node target;
        private float acmReward = 0.0f;
        private BasicAgent agent;

        public override void OnSetup(Agent agent) {
        }

        public void body_shape_entered(RID body_rid, Node body, ]
                                       int body_shape_index,
                                       int local_shape_index ) {
            if (body == target) {
                acmReward += this.reward;
            }
        }

        public override void OnUpdate() {
            this.agent.AddReward(acmReward, this);
            acmReward = 0.0f;
        }

        public override void OnReset(Agent agent) {
            acmReward = 0.0f;
        }
    }
}
```

Figura 8. Exemplo de função de recompensa.

O uso de um controlador local do Agente

Uma vez que o agente e todos os seus componentes foram configurados, pode-se configurar o *ControlRequestor* para manipulação do corpo do agente por meio de um controlador local. Para isso, deve-se criar um nó do tipo *Node*, associá-lo ao *script LocalBrain* e então configurar a propriedade *Brain Mode Path* de *ControlRequestor* para apontar para este nó do tipo *LocalBrain*.

O nó do tipo *LocalBrain* deve apontar para um controlador. Um controlador define a forma como o agente se comportará. Para isso, deve-se criar um nó do tipo *Node* e associá-lo ao *script* que implementa o controlador. Vamos usar como exemplo o controlador embutido *WASDRBMoveController*, que permite controlador o agente por meio das teclas *WASD* do telcado alfa-numérico. Na Figura 9, mostra-se como este controlador foi configurado no projeto *AI4UGTesting*.

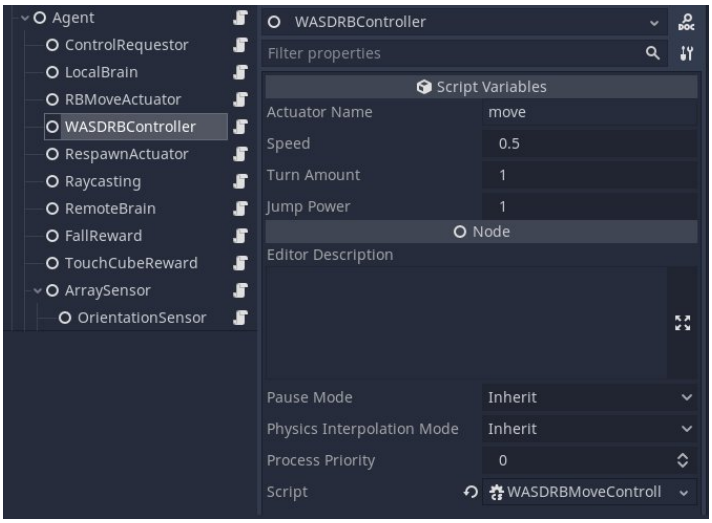

Figura 9. Exemplo de configuração de um controlador WASDRBMoveController. Speed é o multiplicador de velocidade do agente quando a tecla de movimento para frente/traz é pressionada. TurnAmount é a quantidade de graus que o agente riga quando A (giro para a esquerda) ou D (giro para a direita) são pressionados. E Jump Power é força do salto do agente, quanto maior o valor deste parâmetro, mas alto é o salto do agente.

Depois de o controlador ser configurado, deve-se configurar a propriedade *Controller Path* de *LocalBrain* para apontar para este controlador, como mostrado na Figura 10.

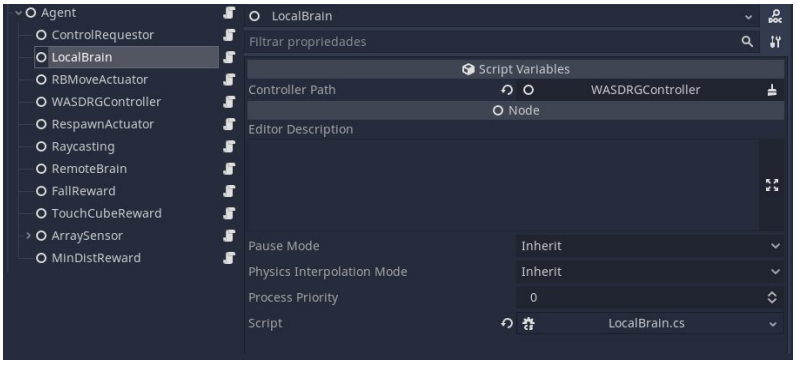

Figura 10. Configuração de um nó LocalBrain.

O uso de um controlador remoto de Agente.

Pode-se definir um controlador remoto de agente por meio de um objeto do tipo *RemoteBrain*, que deve substituir o objeto *LocalBrain* na configuração do *Brain Mode Path* do componente *ControlRequestor* do agente. Para isso, primeiro, deve-se criar e configurar um nó do tipo *RemoteBrain*. Na Figura 11, observa-se a configuração de um nó do tipo *RemoteBrain*. Cria-se um nó desse tipo do mesmo modo que se cria um nó do tipo *LocalBrain*, exceto que o *script* usado é *RemoteBrain*.

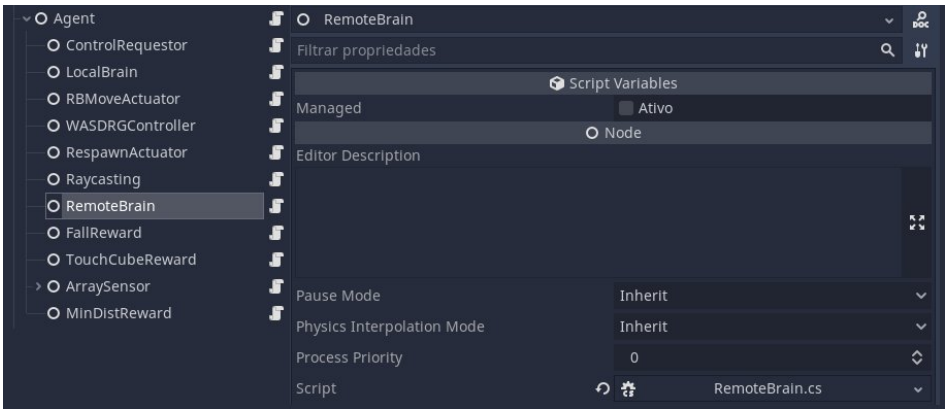

Figura 11. Configuração de um componente do tipo RemoteBrain.

Um objeto do tipo *RemoteBrain* se conecta com um *script* na rede local e recebe comandos de um controlador externo, passando estes comandos ao agente por meio do *ControlRequestor*. As configurações de conexão remota são feitas no próprio *ControlRequestor*, como mostrado na Figura 4. A Linguagem suportada pelo AI4U para controle externo do agente é Python. Esta linguagem pode ser usada para treinar uma rede neural que controla o agente. A linguagem Python foi usada devido à sua facilidade e à disponibilidade de *frameworks* e ferramentas de aprendizado de máquina que a suportam.

Treinamento e manipulação externa do Agente

O treinamento e manipulação externa (por meio de ferramentas fora da *engine de jogos*) pode ser feita por meio de *scripts* em Python. A forma de implementar este controle é a mesma tanto para a *Unity* quanto para a *Godot*. Por isso, toda a documentação referente a controle externo está disponível em arquivos com extensão *.md disponíveis no diretório *doc* da AI4U.

Contudo, é importante considerar as configurações adequadas da Godot para que o treinamento do agente seja bem-sucedido. Durante o treinamento, configure a propriedade *Physics FPS* do projeto (*Project → Project Settings→ Physics*) para um valor adequado às configurações da máquina em uso. Para um computador com GPU dedicada *Nvidia 1650* de 4G de VRAM, processador Intel Core i5 da geração 11 e memória RAM de 4GB, utilizou-se o valor 320 (veja a Figura 12) no projeto *AI4UGTesting*.

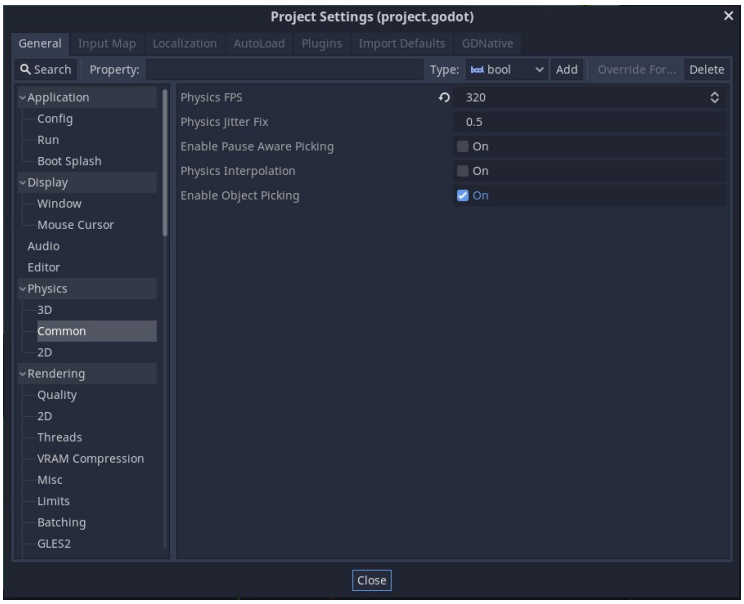

Figura 12. Configurações de física recomendadas para treinamento do agente.

Depois de configurar adequadamente o ambiente e o agente, defina o valor da propriedade *Brain Mode Path* do *ControlRequestor* selecionando a opção *RemoteBrain*. Então, entre no diretório *examples/ai4upe/scene_samplescene* (localizado no repositório da AI4U) e execute o comando:

>> *python appgym_sb3train.py*

Este comando aciona um *script* que usa o algoritmo SAC disponível no *framework stable-baselines3* para treinar um modelo controlador do agente. Este modelo consegue alcançar o cubo sem cair da plataforma. Para testar o controlador, execute o comando:

>> *python appgym_sb3test.py*

Ao executar este comando, você escolherá se quer executar o modelo treinado para controlar um agente da Unity, da Godot ou o último modelo treinado (como mostrado na Figura 13).

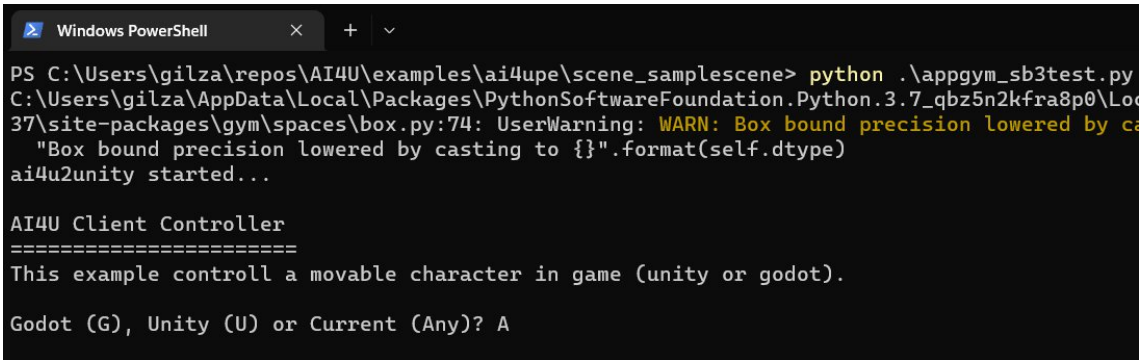

Figura 13. Inicializando um script que carrega uma rede neural para controlar um agente na Unity/Godot.

Limitações

Até o momento, a AI4UGE suporta agentes que controlam corpos físicos do tipo *RigidBody* em ambientes tridimensionais e possui um conjunto limitado de sensores.

Créditos

Gilzamir Gomes: *design* e desenvolvimento.

Eduardo Nogueira (não informado): *game design*.