ANÁLISE DO USO DE ANIMAÇÃO COMPORTAMENTAL COM O MOTOR DE JOGOS UNITY

João Marcos Estevão, Dalton Solano dos Reis – Orientador

Curso de Bacharel em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

jmestevao@furb.br, dalton@furb.br

Resumo: *Este artigo descreve o processo de desenvolvimento de uma extensão de um simulador de ecossistemas através da adição de animais treinados utilizando aprendizado de máquina. Os animais foram treinados para se alimentar e saciar a sede utilizando recursos do cenário e interagindo uns com os outros. O trabalho foi desenvolvido utilizando o motor gráfico Unity em conjunto com a biblioteca Unity Machine Learning Agents. A definição dos animais e seus comportamentos foi feita em conjunto com uma professora do curso de Ciências Biológicas. Após realizados os testes e validações, a implementação e o treinamento dos animais se mostrou eficiente e a tecnologia utilizada se mostrou apropriada.*

Palavras-chave*: Simulador. Ecossistemas. Animais. Aprendizado de máquina. Unity.*

Com o rápido avanço da tecnologia na última década, o seu uso foi se tornando mais frequente em diversos ramos da sociedade. Um dos ambientes influenciados por esse período de informação digital é a educação e a expansão do uso da tecnologia trouxe muitas formas de melhorar e diversificar o aprendizado em diversas áreas, entre elas a de estudo do meio ambiente. Com o objetivo de facilitar o aprendizado de temas dos quais a observação de fatores no mundo real é complexa, foram desenvolvidos simuladores (GREIS; REATEGUI, 2010). Entre eles estão os simuladores de ecossistema, que tem como intuito demonstrar de forma facilitada o funcionamento de diversos aspectos do meio ambiente da maneira mais próxima possível a que existe na natureza.

Aldrich (2009) define simuladores como ambientes estruturados, abstraídos de alguma atividade da vida real, que permitem aos participantes praticar suas habilidades no mundo real, pois fornecem um feedback apropriado em um ambiente cujos resultados são controlados e previsíveis. Para Greis e Reategui (2010), as vantagens em se trabalhar com modelos simulados por computador no campo educacional são muitas. Entre elas tem-se a oportunidade de tornar possível a reprodução de processos lentos ou perigosos de ser reproduzir no ambiente natural, o controle das etapas necessárias para a observação dos fenômenos e até mesmo a redução dos custos envolvidos no projeto.

Segundo Dautenhahn e Nehaniv (2002) um dos aspectos que pode apresentar uma dificuldade no desenvolvimento de simuladores de ecossistemas, no entanto, é simular o comportamento existente dos seres vivos que compõem estes ecossistemas, visto que são criaturas providas de inteligência. Para este fim, podem ser utilizadas técnicas de inteligência artificial de forma a tentar reproduzir comportamentos compatíveis com o de criaturas encontradas na natureza.

> (...) imagine quão impressionante seria criar um robô que pudesse emular – isto é, observar a mudança realizada no ambiente e desenvolver sua própria maneira de reproduzir aquela mudança de estado. Nós presumimos que isso seria um desafio muito mais sério para os programadores do que desenvolver um robô que copie as ações de outros de forma não criativa. (DAUTENHAHN; NEHANIV, 2002, p. 226, tradução nossa) .

Decorrente destas considerações, o trabalho aqui apresentado tem como objetivo desenvolver uma inteligência artificial com aprendizado de máquina utilizando o Unity ML-Agents para ser aplicada em animais no ECOSAR, simulador de ecossistema desenvolvido por Pereira (2019), a fim de possibilitar que os agentes no cenário sejam capazes de interagir de forma autônoma, seguindo regras pré-determinadas para cada cenário. Os objetivos específicos são: adicionar animais no ECOSAR; desenvolver uma inteligência artificial utilizando aprendizado de máquina através do Unity ML-Agents para fazer com que os animais interajam com o ecossistema e uns com os outros de maneira inteligente.

1 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os aspectos da fundamentação teórica utilizados para o desenvolvimento deste trabalho. A seção 2.1 aborda o tema simuladores. Na seção 2.2 é apresentado o conceito de animação comportamental. A seção 2.3 explica sobre a biblioteca Unity Machine Learning Agents. Na seção 2.4 está a apresentação do aplicativo desenvolvido por Pereira (2019). Por fim, na seção 2.5 são apresentados os trabalhos correlatos comparando-os com o trabalho deste artigo.

1.1 SIMULADORES

Para Rosa (2008) os simuladores na área de computação podem ser explicados como a criação de um cenário virtual que cria um ambiente e o executa de forma mais próxima possível do mundo real. Segundo Greis e Reategui (2010), o modelo simulado oferece várias vantagens, como a possível reprodução de processos muito lentos ou perigosos para serem reproduzidos no ambiente real, a facilidade da observação de fenômenos e também a redução de custos. Os simuladores têm sido frequentemente utilizados em países como Alemanha, Reino Unido e Estados Unidos nas áreas de saúde, negócios e militar. Na área militar por exemplo o jogo “America's Army” é considerado uma das maiores fontes de recrutamento militar nos Estados Unidos (LOPES; OLIVEIRA, 2013).

Segundo Ulicsak e Wright (2010) os simuladores oferecem um mecanismo seguro e de baixo custo para treinamentos e atividades a serem realizadas em ambientes perigosos ou que sejam muito difíceis de recriar no mundo real. De acordo com Stone (2012) o alto nível de fidelidade, ou seja, a semelhança com os eventos reais permite esta transferência de ambientes.

Segundo Mendes e Fialho (2004) as técnicas de simulação, em particular aquelas que utilizam imagens interativas se destacam em relação a passividade proporcionada pela simples demonstração de vídeos ou animações sem interação, pois prolongam e transferem a capacidade de imaginação e de pensamento, apesar de não garantirem as inferências/raciocínios humanos. Além disso afirmam que a imersão proporcionada pela simulação ou pela realidade virtual, pode substituir algumas modalidades de experimentações práticas através de avaliações e estratégias específicas.

No entanto existe a necessidade de uma investigação científica criteriosa para cada área de conhecimento para comprovar a eficiência didática e, principalmente, a capacidade de aquisição de conhecimento proporcionada pela aplicação de softwares simuladores em experimentos práticos. Dessa forma é possível realizar um planejamento metodológico adequado do desenvolvimento das habilidades e competências esperadas com a devida dosagem de teoria, experimentação real e experimentação simulada (MENDES; FIALHO, 2004).

1.2 ANIMAÇÃO COMPORTAMENTAL

A animação comportamental é um conceito de controle de ações de objetos em ambientes computacionais de acordo com um comportamento dotado de inteligência artificial.

> A animação comportamental busca o realismo a nível de comportamento dos personagens em cena. Neste tipo de animação, os personagens são "atores sintéticos" dotados de personalidade e habilidades próprias. A atuação de um personagem não é mais oriunda exclusivamente de intervenções diretas do animador, mas sim fruto de sua personalidade, seu humor, suas metas e sua interação com os demais atores. (...) (SIMPÓSIO BRASILEIRO DE COMPUTAÇÃO GRÁFICA E PROCESSAMENTO DE IMAGENS, 1993, p. 1).

Os conceitos da animação comportamental são semelhantes aos dos agentes inteligentes. O primeiro conceito é o da percepção, que se resume a um processo de reconhecimento em que se capta informações por sensores externos e executa a transformação dos dados para uso do programa (INTELLIGENT AGENTS, 1995). O segundo conceito é o de raciocínio, que é a capacidade de tomar conclusões sobre um conjunto de hipóteses próprias e/ou de outros (INTELLIGENT AGENTS: AGENT THEORIES, ARCHITECTURES, AND LANGUAGES, 1997). Por fim o terceiro conceito é o de ação, que se trata de um comportamento de resposta a um estímulo externo (INTELLIGENT AGENTS, 1995).

A animação comportamental tem dado ímpeto a uma vasta gama de aplicações capazes de sintetizar, reunir e ensinar comportamentos de animais (EUROGRAPHICS/ACM SIGGRAPH SYMPOSIUM ON COMPUTER ANIMATION, 2005). Entre as áreas de aplicação dos conceitos de animação comportamental pode ser citado o desenvolvimento de jogos de videogame. Para os personagens principais do jogo, que aparecem em tela em períodos extensos, é necessária a utilização de técnicas que tornem o seu comportamento mais realista, visto que o jogador estará mais propenso a observar o padrão de ações destes personagens (INTERNATIONAL CONFERENCE, 2013).

1.3 UNITY MACHINE LEARNING AGENTS

Unity Machine Learning Agents, comumente abreviado para Unity ML-Agents ou ML-Agents, é uma biblioteca de código aberto para o motor de jogos Unity que permite que jogos e simulações sirvam como ambientes para treinar agentes inteligentes. Segundo Juliani et al. (2018) o ML-Agents permite aos desenvolvedores criar ambientes simulados utilizando o editor do Unity e interagir com eles via uma Application Programming Interface (API) em Python.

De acordo com Juliani et al. (2018), o Software Development Kit (SDK) do ML-Agents tem três entidades principais: o `Sensor`, o `Agent` e a `Academy`. O componente `Agent` é utilizado para indicar que um objeto de cena é um agente, o que significa que pode coletar informações por meio de observação, fazer ações e receber recompensas. As informações são coletadas pelos agentes através de uma variedade de sensores possíveis correspondendo a diferentes

formas de informação, como imagens renderizadas e resultados de *ray-cast*. Cada componente `Agent` possui uma política que é rotulada com um nome de comportamento. Uma política é a definição do comportamento do agente e pode ser utilizada por qualquer número de agentes em uma cena, os quais podem executar a mesma política e compartilhar dados de experiências durante o treinamento. Além disso, não há limite para o número de nomes de comportamentos para políticas dentro de uma cena, o que significa que é possível construir cenários com múltiplos agentes com grupos de agentes individuais executando vários comportamentos diferentes.

O sistema de recompensas é utilizado para enviar um sinal de aprendizado ao agente e por meio do sistema de *scripts* do Unity é possível defini-lo e modificá-lo a qualquer momento durante a simulação. Da mesma forma, a simulação pode ser definida como finalizada tanto no âmbito de um agente individual quanto em relação a todo o ambiente simulado (JULIANI et al., 2018). Isso pode ser definido tanto via chamadas de *scripts* do Unity quanto ao ser alcançado o número máximo de passos pré-definido.

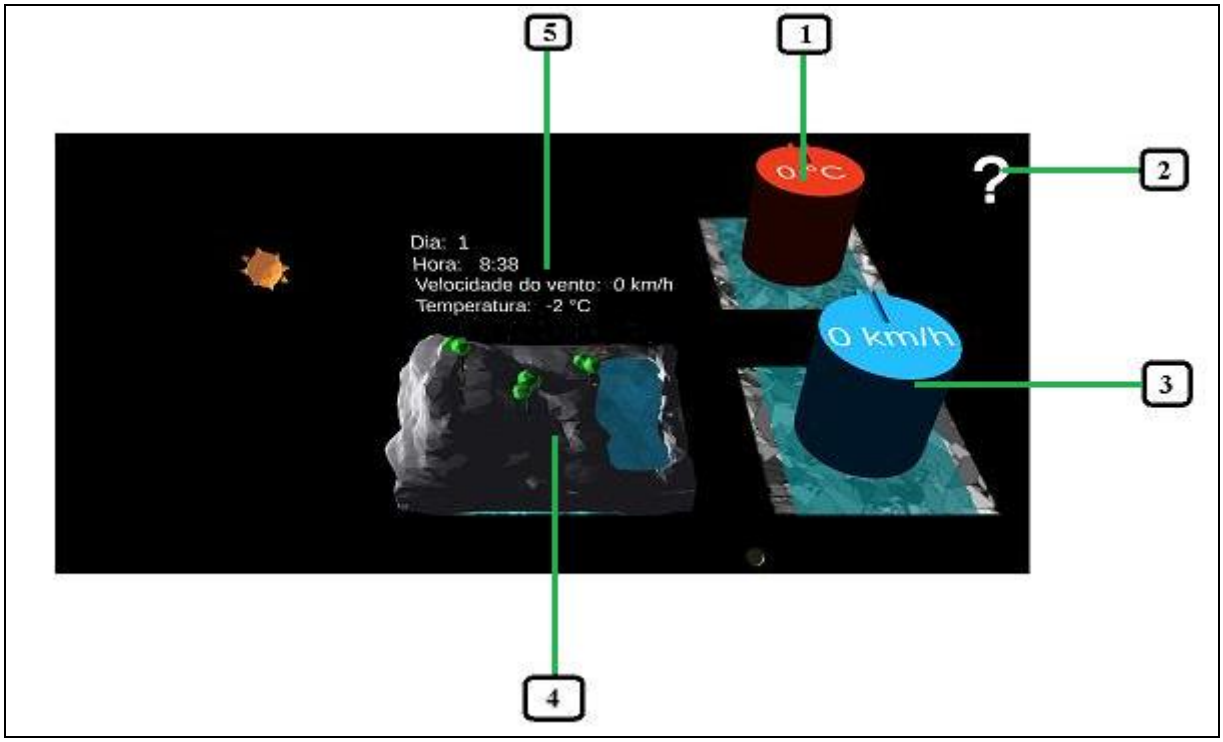
Juliani et al. (2018) explica que a `Academy` é um *singleton* e é utilizada para acompanhar o número de passos da simulação e gerenciar os agentes, além disso ela possui a capacidade de definir parâmetros de ambiente, que podem ser utilizados para mudar a configuração do ambiente em tempo de execução. Uma representação do funcionamento do ML-Agents pode ser visualizada no diagrama disponível no Apêndice C.

1.4 VERSÃO ANTERIOR DO APLICATIVO

O "ECOSAR – Simulador de Ecossistemas Utilizando Realidade Aumentada" foi desenvolvido por Pereira (2019) e sua proposta foi criar um ambiente virtual utilizando realidade aumentada para simular um ecossistema real. Além disso, o objetivo foi permitir ao usuário interagir com este ambiente através da alteração de elementos como temperatura, velocidade do vento e controle do ciclo dia/noite. Para o desenvolvimento do projeto foi utilizado o motor gráfico Unity em conjunto com a biblioteca Vuforia, além das ferramentas Photoshop CC 2019, AR Marker Generator para a geração dos marcadores e Blender para a modelagem dos objetos. Entre as propostas de extensão expostas por Pereira (2019) está a de inserção de animais na simulação. Este tópico será um dos objetivos deste artigo, visto que os animais serão os agentes que utilizarão o módulo de animação comportamental desenvolvido.

Para controlar estes elementos e mostrá-los na tela, o aplicativo faz uso da câmera do dispositivo móvel, utilizando-a em conjunto com marcadores (que funcionam em conjunto com a biblioteca Vuforia) para visualizar a aplicação (PEREIRA, 2019). A Figura 1 mostra a simulação ao ser iniciada. Ela começa com a temperatura em zero graus e com a velocidade do vento em zero quilômetros por hora. A partir deste ponto o usuário pode começar a interagir com a cena utilizando os marcadores.

Figura 1 – Início da simulação



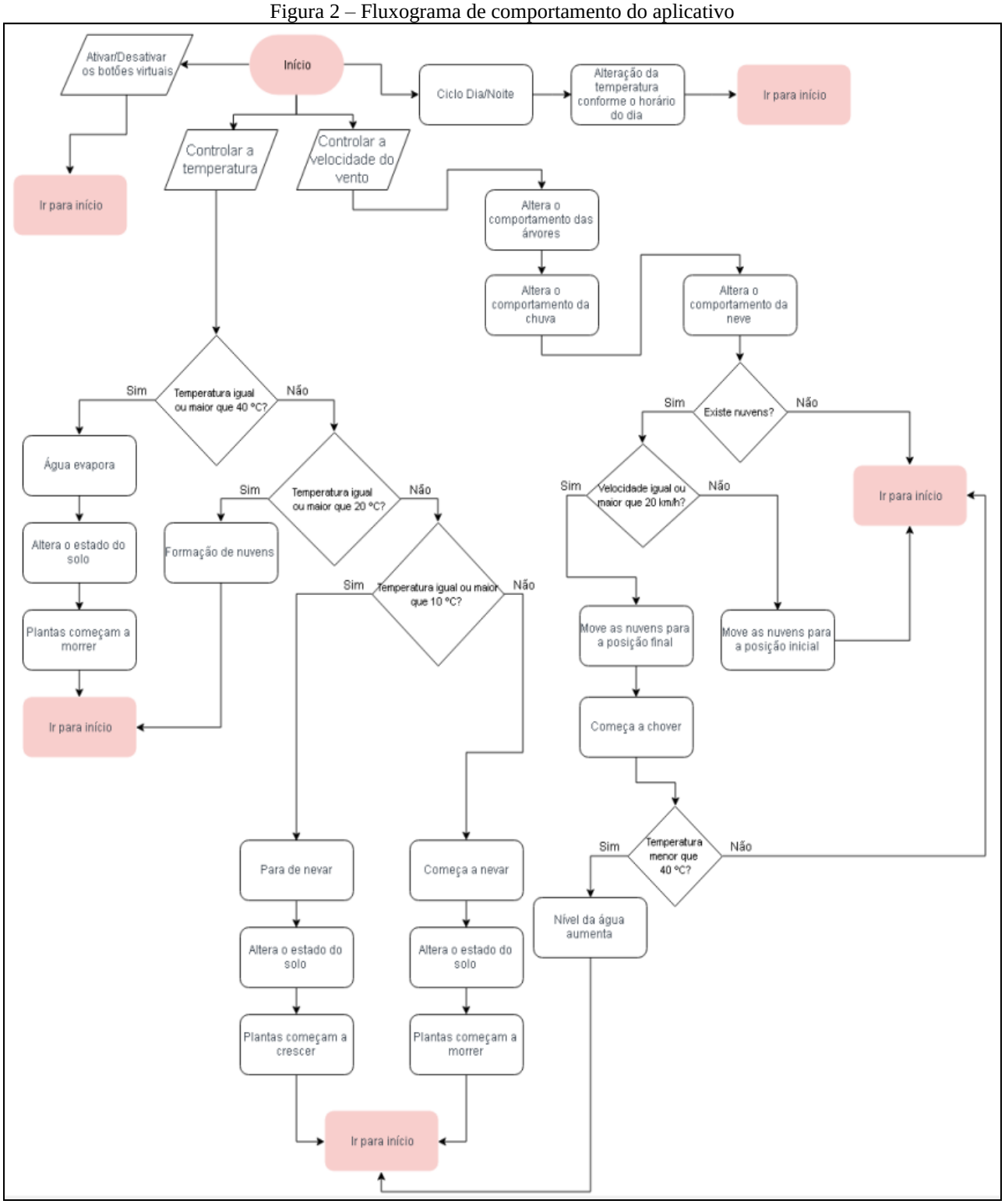
Fonte: Pereira (2019).

Na Figura 1, os itens 1 e 3 são os marcadores de controle do aplicativo, com os quais o usuário manipula a cena. Já o item 2 trata-se do botão de ajuda, que quando ativo mostra a *bounding box* dos marcadores e ativa os botões

virtuais. Estes botões (virtuais de ajuda) são utilizáveis através da Realidade Aumentada para mostrar textos de ajuda para cada marcador. E por fim, o item 4 é o marcador responsável pela visualização da simulação, enquanto o item 5 mostra um painel com as características atualizadas da simulação.

A Figura 2 apresenta um fluxograma que descreve os possíveis comportamentos que podem ser realizados dentro do aplicativo, sendo que os paralelogramos são os elementos que o usuário pode controlar, os losangos são as condições necessárias para cada processo e os retângulos correspondem aos resultados obtidos através dos controles realizados. Como pode ser visto, dependendo das alterações efetuadas pelo usuário, a simulação realiza diferentes comportamentos para diferentes alterações, sendo que mais de um comportamento pode ocorrer ao mesmo tempo, assim gerando um ambiente em que várias simulações ocorrem simultaneamente.

Figura 2 – Fluxograma de comportamento do aplicativo



Fonte: Pereira (2019).

1.5 TRABALHOS CORRELATOS

A seguir são apresentados trabalhos com características semelhantes aos principais objetivos do estudo proposto. O Quando 1 apresenta o primeiro, desenvolvido por Piske (2015), que é um programa que simula um ecossistema marinho dentro de um aquário utilizando animação comportamental. Já o Quadro 2 traz as informações do segundo, desenvolvido por Feltrin (2014), que consiste num módulo de animação comportamental para utilização em simuladores educacionais. O terceiro trabalho, apresentado no Quadro 3, foi desenvolvido por Fronza (2008) e trata-se de um simulador 3D de batalhas entre times de tanques de guerra.

Quadro 1 – VisEdu - Aquário Virtual: Simulador de ecossistema utilizando animação comportamental

Referência	Piske (2015).
Objetivos	Desenvolvimento de um simulador de ecossistema marinho em um aquário utilizando animação comportamental.
Principais funcionalidades	Ecossistema marinho com uma cadeia alimentar de três níveis onde o tubarão se alimenta da sardinha que por sua vez se alimenta dos plânctons.
Ferramentas de desenvolvimento	São: a) linguagem de programação Javascript; b) elemento canvas do HTML5; c) biblioteca gráfica ThreeJS; d) interpretador Jason em conjunto com a linguagem AgentSpeak para o desenvolvimento de agentes no modelo BDI.
Resultados e conclusões	Piske (2015, p.99) expõe nos resultados do trabalho que os objetivos de criar um aquário virtual que simulasse um ecossistema marinho foi contemplado. Da mesma forma os objetivos de estender o trabalho de Feltrin (2014) e o de permitir a inserção de agentes dotados de representações gráficas também foram contemplados.

Fonte: elaborado pelo autor.

O Aquário Virtual utiliza a animação comportamental para a criação de um cenário educativo de ecossistema. Na Figura 3 pode-se ver o cenário do ecossistema do aquário em execução e a edição dele utilizando a árvore de peças que serve para controlar os elementos que serão inseridos no cenário. Desta forma, cada peça representa um elemento gráfico (aquário, tubarão e sardinha), as quais encaixadas corretamente dentro da propriedade de mundo são exibidas na representação gráfica na tela.

Figura 3 – Aquário virtual no aplicativo VISEDU

Fonte: Piske (2015).

Quadro 2 – VisEdu-SIMULA 1.0: Visualizador de material educacional, módulo de animação comportamental

Referência	Feltrin (2014).
Objetivos	Extensão do motor de jogos 2D e do editor de jogos de Harbs (2013) com a criação de um simulador 2D para geração de animações comportamentais.
Principais funcionalidades	Simulador 2D para geração de animações comportamentais com controle de percepção, raciocínio e atuação dos personagens.
Ferramentas de	São:

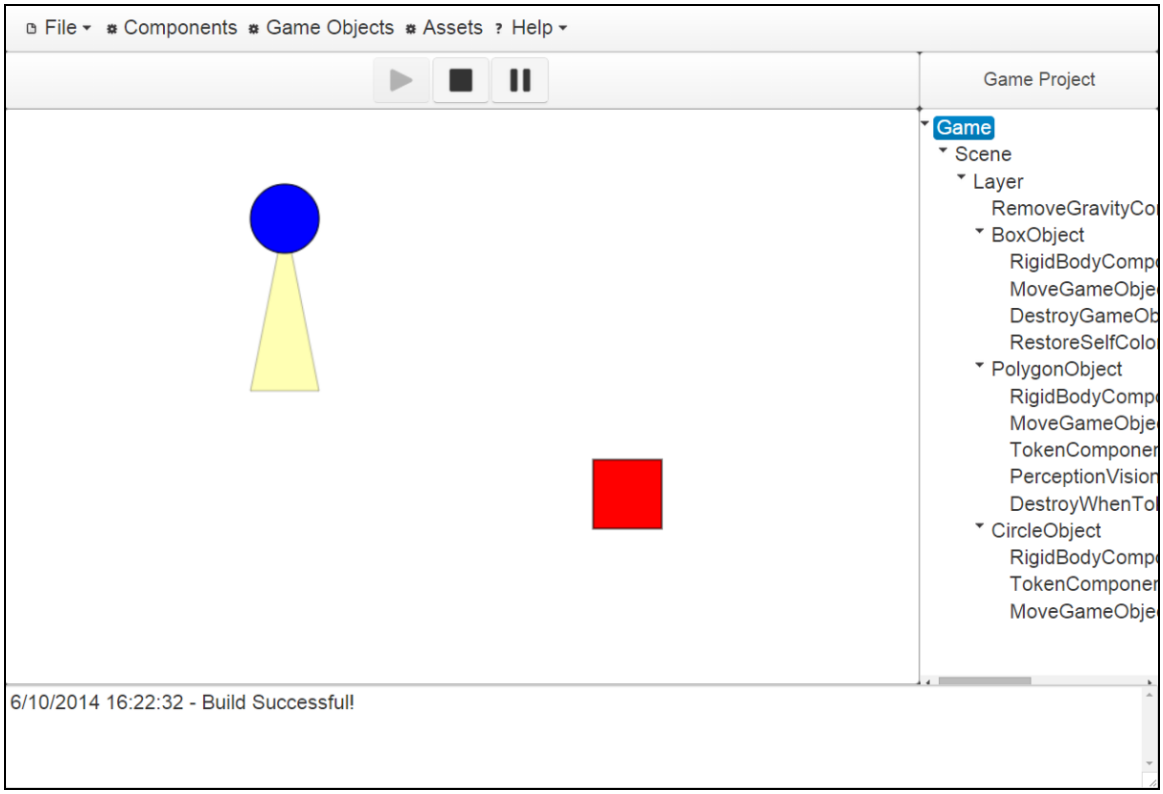
desenvolvimento	a) linguagem de programação Java, com os plugins JBoss Tools e Jasonide; b) interpretador Jason 1.4.1 para a criação do módulo de inteligência artificial; c) Apache Tomcat 7.0.55 para o servidor de aplicação; d) elemento canvas do HTML5.
Resultados e conclusões	Feltrin (2014, p.76) conclui que teve sucesso no objetivo de criar um simulador 2D para a geração de animações comportamentais, caracterizando a orientação a componentes como ponto forte do trabalho e o Editor de Jogos como o ponto fraco, explicando que sua utilização seria questionável visto que o Motor de Jogos poderia usar um editor de texto mais sofisticado.

Fonte: elaborado pelo autor.

O projeto faz uso de inteligência artificial para a criação de simuladores de objetivo educacional, utilizando o conceito de animação comportamental. Além da extensão e do desenvolvimento do módulo de inteligência artificial foi desenvolvida uma aplicação para testar a animação comportamental utilizando-se do modelo de presa e predador.

Na Figura 4 pode-se ver o protótipo da simulação de teste, no qual o quadrado vermelho representa o predador do ambiente, o círculo azul representa a presa e seu campo de visão é representado por um polígono triangular amarelo. Partindo do posicionamento inicial, a simulação pode alcançar três estados: a presa pode perceber o predador sem colidir com ele, a presa pode perceber o predador e estar colidindo com ele ou o predador pode colidir com a presa sem ser percebido pelo campo de visão dela.

Figura 4 – Execução do protótipo de simulação



Fonte: Feltrin (2014).

Quadro 3 – Simulador de um ambiente virtual distribuído multiusuário para batalhas de tanques 3D com inteligência baseada em agentes BDI

Referência	Fronza (2008).
Objetivos	Desenvolvimento de um simulador de batalhas entre tanques de guerra em 3D com suporte para partidas com múltiplos jogadores.
Principais funcionalidades	Simulador de batalhas de tanques de guerra com suporte para múltiplos jogadores e implementação de inteligência artificial para controle de tanques autônomos (não controlados pelos jogadores).
Ferramentas de desenvolvimento	São: a) linguagem de programação Java; b) biblioteca gráfica OpenGL; c) Blender 2.43 para a modelagem dos objetos; d) Apache Photoshop CS para a geração das texturas do cenário; e) Apache Ant para controle da parte do servidor.

Resultados e conclusões	Fronza (2008, p.130) expõe nas conclusões do trabalho que os objetivos foram contemplados. No entanto constatou que o simulador possuía problemas de performance e afirmou que o detalhamento do cenário poderia ser maior para que o nível de realidade fosse melhor.

Fonte: elaborado pelo autor.

Apesar de encontrar problemas com a performance do programa, Fronza (2008) constata que obteve sucesso em desenvolver os objetivos estabelecidos no seu artigo. Assim como é proposto neste trabalho, Fronza (2008) utilizou conceitos de inteligência artificial para desenvolver animação comportamental para os agentes do cenário do simulador. Para o controle dos tanques foi utilizado o modelo BDI, que utiliza os conceitos de crenças, desejos e intenções para desenvolver um comportamento. A Figura 5 mostra o simulador de batalhas de tanques de guerra de Fronza (2008).

Figura 5 – Execução do simulador de batalhas de tanques de guerra

Fonte: Fronza (2008).

2 DESCRIÇÃO DO APLICATIVO

Este capítulo pretende apresentar os detalhes da implementação do trabalho proposto. Para isso serão apresentadas duas seções. A primeira seção apresenta a visão geral da implementação, mostrando o funcionamento da simulação dos animais e explicando a metodologia utilizada para definição dos animais escolhidos e seus comportamentos. A segunda seção apresenta a implementação do trabalho (desenvolvido em C#), explicando os componentes utilizados e a implementação do aprendizado de máquina através da utilização da biblioteca do Unity ML-Agents.

2.1 VISÃO GERAL

O desenvolvimento deste projeto consiste na inclusão de animais de espécies distintas no simulador de ecossistemas desenvolvido por Pereira (2019) e a implementação de uma inteligência artificial para cada animal, permitindo aos animais interagir com o ambiente e uns com os outros através de um comportamento desenvolvido por meio de treinamento. Para definir o cenário e os animais que seriam utilizados foi realizada uma reunião com a professora Roberta Andressa Pereira (PEREIRA, 2020), professora do curso de Licenciatura em Ciências Biológicas da Universidade Regional de Blumenau. Além da explicação dos comportamentos possíveis de serem implementados também foi elaborado um documento com os *assets* de modelos de animais e suas animações disponíveis para o Unity. De acordo com esse documento em conversa com Pereira (2020) foram definidos os animais e comportamentos a serem utilizados.

Os animais selecionados para a inclusão no projeto foram o coelho, o veado e o lobo. A ideia do cenário é que tanto o coelho quanto o veado possam se alimentar da vegetação que cresce no cenário, de acordo com as condições de

temperatura, conforme implementação de Pereira (2019). O lobo por sua vez pode se alimentar tanto do veado quanto do coelho e todos os três animais saciam a sede com a água do lago existente no cenário.

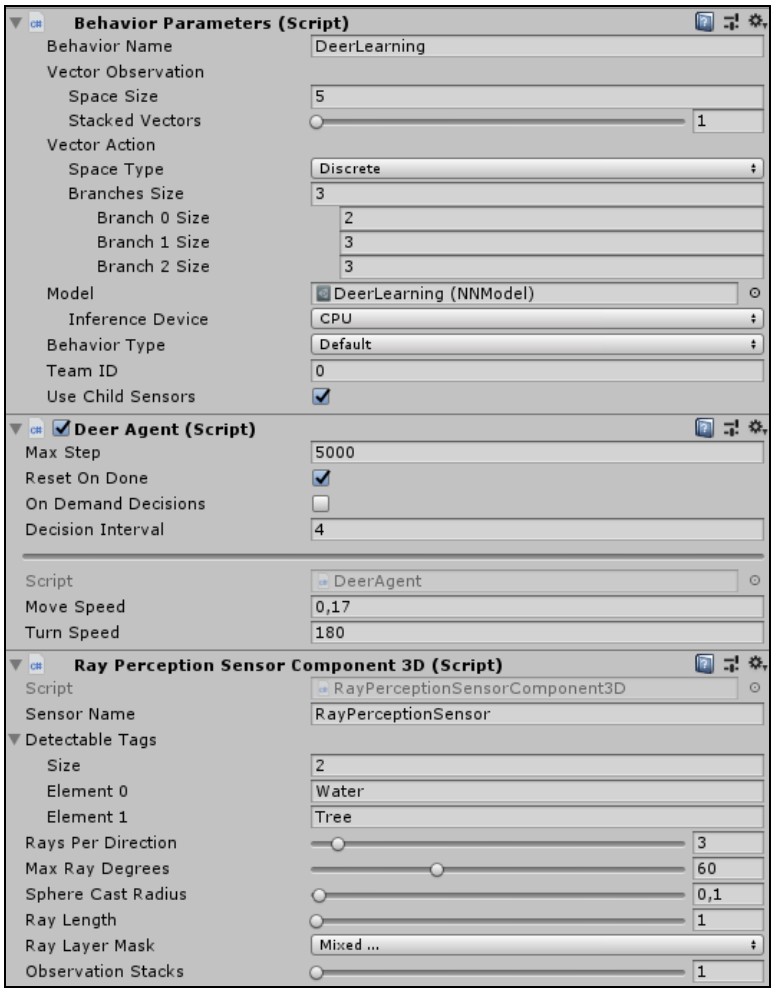
2.2 IMPLEMENTAÇÃO

Para o desenvolvimento do comportamento dos animais e adição dos modelos no ECOSAR foram utilizados o motor gráfico Unity em conjunto com a biblioteca Unity Machine Learning Agents. E o ambiente de desenvolvimento Visual Studio 2017 com o gerenciador de pacotes e ambientes Python Anaconda 2019.10, para interação com a API em Python do Unity ML-Agents responsável pelo treinamento dos agentes.

Dentro do componente `SceneTarget` do projeto estão presentes todos os objetos do marcador principal da aplicação, o qual contém o ambiente do ecossistema. Nele estão contidos os objetos de terreno, suas plantas; partículas de água, fogo e neve; as nuvens; e o controle de rotação do sol e da lua na cena. Os animais inseridos no projeto foram adicionados dentro do componente `Terrain`, que representa o terreno da cena e no qual também se encontram as árvores e flores. Além da adição dos animais com seus agentes também foi adicionado no âmbito principal da cena o componente `Academy`, utilizado para gerenciar o treinamento de todos os agentes da cena.

Entre os componentes utilizados em todos os três agentes da cena tem-se o `Animator`, o `RigidBody` e o `Collider`. O `Animator` serve para o gerenciamento das animações do personagem enquanto o `Rigidbody` adiciona ao objeto o controle de física do Unity, permitindo controle gravitacional, de massa, de rotação e posicionamento do personagem, além de outras funções. Como também para o controle de movimentação dos animais na cena. Por último, o `Collider` é um componente utilizado para identificação de colisão com outros objetos da cena por parte do controle de física do Unity. Com relação a parte de treinamento da inteligência artificial dos animais, foram adicionados três componentes, conforme pode ser visualizado na Figura 6.

Figura 6 – Propriedades do objeto de cena do veado



Fonte: elaborado pelo autor.

Os componentes `Behavior Parameters` e `Ray Perception Sensor Component 3D` são provenientes da biblioteca Unity ML Agents. O primeiro é responsável por especificar o número de decisões a serem feitas pelo agente assim como o número de ações disponíveis para cada decisão, além da configuração do tipo de comportamento

(controlado pelo usuário ou pela IA) e do modelo de comportamento, caso se deseje adicionar um modelo já treinado ao agente. O `Ray Perception Sensor Component 3D` é utilizado para configuração do componente de *RayCast*, especificando as *tags* detectáveis e estabelecendo as dimensões e ângulos dos raios. Por fim, o componente `DeerAgent` (Figura 6), é utilizado para definir a lógica de aprendizado dos animais. Dentro do *script* do agente (Quadro 4), está o método `AgentAction`, que é responsável pelo treinamento e controle do animal na cena.

Quadro 4 – Método `AgentAction` no script `DeerAgent`

```
public override void AgentAction(float[] vectorAction)
{
    if (currentAction == CurrentAnimalAction.dead)
        return;

    currentAction = CurrentAnimalAction.idle;

    // Converte a primeira ação em movimento para frente
    float forwardAmount = vectorAction[0];
    if (forwardAmount > 0)
        currentAction = CurrentAnimalAction.walking;

    // Converte a secunda ação em virar pra esquerda ou pra direita
    float turnAmount = 0f;
    if (vectorAction[1] == 1f)
    {
        turnAmount = -1f;
    }
    else if (vectorAction[1] == 2f)
    {
        turnAmount = 1f;
    }

    //Converte a terceira ação para a chamada do método correspondente (1 para beber, 2 para comer)
    if (vectorAction[2] == 1f)
    {
        Drink();
        ControlAnimation();
        return;
    }
    else if (vectorAction[2] == 2f)
    {
        Eat();
        ControlAnimation();
        return;
    }

    // Aplica o movimento
    rigidbody.MovePosition(transform.position + transform.forward * forwardAmount * moveSpeed * Time.fixedDeltaTime);
    transform.Rotate(transform.up * turnAmount * turnSpeed * Time.fixedDeltaTime);

    // Aplica um a pequena recompensa negativa para encorajar o personagem a fazer uma ação
    AddReward(-1f / agentParameters.maxStep);

    ControlAnimation();
}
```

Fonte: elaborado pelo autor.

Neste método (`AgentAction`) é feita a interpretação do vetor de ações do agente e o gerenciamento de recompensas. O vetor de ações tem o tamanho três, que é equivalente ao número de decisões que o agente deve tomar. Estas decisões são feitas a cada *frame* da execução da cena e são feitas pelo algoritmo de aprendizado de máquina do ML-Agents. No caso de cada animal da cena a primeira decisão a ser tomada é se o agente deve se movimentar para frente ou não, portanto são duas ações possíveis. Neste caso esta posição do vetor poderá vir com o valor zero, que significa não fazer nada ou um, que significa mover-se para frente. A segunda decisão a ser tomada é se o agente deve se virar para esquerda, para direita ou continuar na direção para a qual está virado. Nesta decisão portanto são três ações possíveis. A terceira decisão é a do que o agente fará na cena: beber, comer ou não fazer nada, sendo que no caso do lobo há uma quarta alternativa que é a de caçar.

A cada instante de execução do treinamento o *script* de treinamento irá enviar para este método um vetor com ações tomadas por ele, ou seja, um valor de zero a três, dependendo da ação, que irá representar a decisão tomada. O método irá então interpretar estas ações e convertê-las em movimentos do personagem na cena gráfica. Por fim, dependendo da ação tomada e do estado atual do agente em relação aos contadores de sede e fome, ele irá receber uma recompensa negativa ou positiva. Por exemplo, se o agente estiver com sede e decidir beber, irá ser recompensado positivamente para que este comportamento seja reforçado e se o agente estiver com fome irá receber uma recompensa negativa para que ele entenda que esta situação vai de encontro ao modo de agir desejado. De acordo com estas

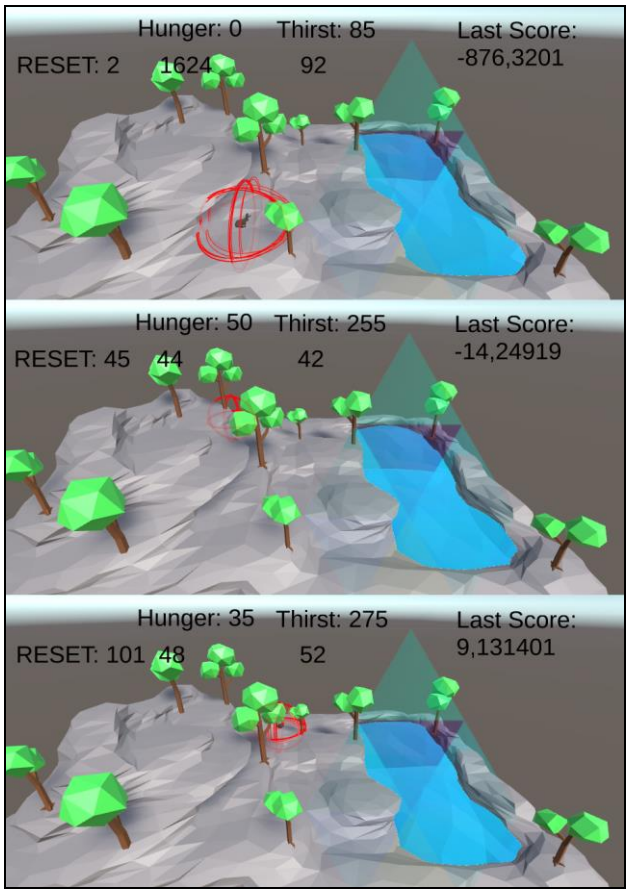
recompensas o script de treinamento irá moldar o comportamento do agente, buscando sempre alcançar a maior pontuação positiva possível em termos de recompensas para suas ações.

O método `AgentAction` irá então receber esse vetor de ações e irá executar essas ações. Por exemplo, se o lobo decidir caçar o método `Hunt` será chamado e da mesma forma as outras ações executam seus respectivos métodos. Além da execução há também a recompensa por determinadas ações. Se o animal estiver com sede e for beber ou se estiver com fome e comer será recompensado positivamente. Se o agente estiver com fome, sede ou se tentar realizar as ações de comer e/ou beber enquanto está cheio será recompensado negativamente. De acordo com as recompensas positivas e negativas o agente vai treinando seu comportamento e após cada sessão de treinamento é gerado um arquivo com um modelo de comportamento treinado utilizando os dados da sessão de treinamento, que pode então ser inserido no agente da cena. Com um modelo de comportamento adicionado a seu objeto de cena, o agente passa a tomar as decisões de acordo com a inteligência artificial treinada anteriormente. O diagrama de classes do projeto está disponível no Apêndice A.

Para poder realizar o treinamento dos agentes é necessário rodar o programa dentro do ambiente do Unity, porém foram encontradas dificuldades de rodar o ECOSAR devido à utilização do Vuforia. Devido a esta dificuldade e para montar um ambiente de treinamento funcional sem precisar alterar o projeto original, foi decidido criar um projeto para o treinamento dos animais. Neste projeto foram adicionados apenas os componentes essenciais para o treinamento, como o terreno, água e vegetação, além da inclusão de informações adicionais na execução do programa para facilitar a visualização do progresso do treinamento, tais como número de vezes que o animal comeu e bebeu, atual placar do treinamento, que exibe a pontuação acumulada de recompensas recebidas pelo agente em cada episódio do treinamento e número de vezes que o treinamento foi reiniciado. Além disso para agilizar e facilitar o processo de treinamento é possível multiplicar as instâncias do cenário na cena, fazendo com que vários cenários de ecossistema executem ao mesmo tempo e dessa forma todos participem do treinamento simultaneamente, acelerando o processo de aprendizagem do agente e consequentemente o alcance do modelo de comportamento desejado.

O treinamento de um agente é composto de várias etapas, visto que cada vez que o agente executa o número de passos limite estabelecido na configuração do treinamento, um episódio é finalizado e outro é iniciado. A cada episódio o agente realiza novamente o treinamento e vai utilizando a experiência das recompensas recebidas para realizar o aprendizado. A Figura 7 mostra três etapas diferentes do treinamento sendo executado no projeto de treinamento com o agente do coelho.

Figura 7 – Evolução do treinamento do agente do coelho

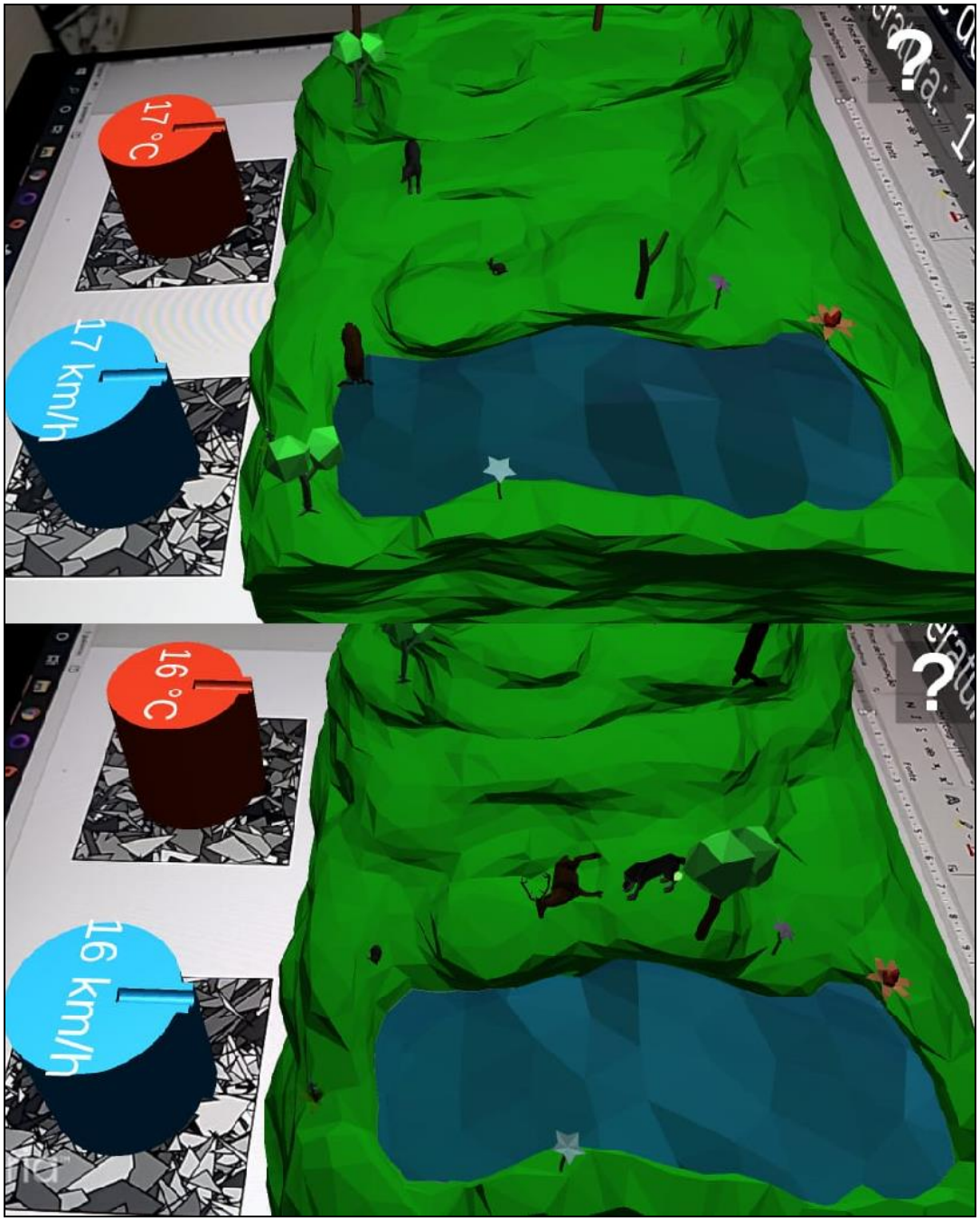


Fonte: elaborado pelo autor.

Na primeira parte da Figura 7 é mostrado o treinamento após dois episódios concluídos, onde o coelho ainda está com uma pontuação baixa. Na segunda e terceira partes o coelho já está em etapas mais avançadas, após quarenta e cinco e cento e um episódios concluídos, respectivamente. Nestas etapas é possível observar que o último placar aumentou, estando positivo na terceira parte, indicando que o coelho foi aprendendo a conseguir maiores pontuações de recompensas e desta forma foi tendo um comportamento mais próximo do desejado para o treinamento. Em `Hunger` e `Thirst` são exibidos os contadores de sede e fome do animal enquanto em `Reset` é possível visualizar quantos episódios do treinamento já foram realizados. Os números seguintes são contadores das quantidades de vezes que o animal fez a ação de comer e beber no último episódio. Por último o `Last Score` mostra a pontuação de recompensas atingida pelo agente no último episódio de treinamento realizado. A configuração para permitir a execução do treinamento está disponível no Apêndice C.

Na Figura 8 pode ser visualizada a execução do ECOSAR com os animais adicionados. A primeira imagem mostra o veado saciando a sede no lago enquanto na segunda imagem podemos ver o veado morto após ser atacado pelo lobo que irá se alimentar dele.

Figura 8 – Execução do ECOSAR com os animais



Fonte: elaborado pelo autor.

Para os modelos dos animais e suas animações foi utilizado o pacote de *assets* Low Poly Animated Animals desenvolvido por Polyperfect e disponível na loja de *assets* da Unity.

3 RESULTADOS

Ao fim do treinamento realizado foram inseridos os modelos de comportamento treinados aos animais na cena. Após a execução do simulador com estes modelos constatou-se que houve sucesso em obter o comportamento desejado dos animais. Ao se observar o treinamento dos agentes na cena pode-se constatar que a cada etapa do treinamento o placar de recompensas recebidas pelo agente durante o episódio aumentava, chegando após um certo tempo e número de episódios concluídos ao comportamento desejado para cumprir o objetivo do trabalho. Todas as três espécies passaram a se mover e realizar ações de maneira coerente com a proposta, se alimentando e saciando a sede conforme necessário e de maneira eficiente.

No início do desenvolvimento foi encontrado um empecilho para a utilização do Unity ML-Agents no projeto do ECOSAR, visto que era impossível rodá-lo dentro do ambiente do Unity devido a problemas com o Vuforia, o que impossibilitaria a realização do treinamento em conjunto com a API em Python. Devido a esta restrição foi decidido criar um projeto separado baseado no ECOSAR, porém sem os componentes de realidade aumentada do Vuforia. A longo prazo viu-se que não só essa solução era efetiva para realizar o treinamento como também propiciou um ambiente mais fácil de ser manipulado e que facilitou o processo de treinamento, visto que foram removidos a maioria dos componentes não necessários da cena, como efeitos de partículas e transição do sol/lua. Além disso o projeto separado deu maior liberdade para implementação de placares para visualização do andamento do treinamento e da multiplicação de cenários, que acelera o processo de aprendizagem dos agentes.

Um outro problema encontrado em relação a extensão proposta do projeto do ECOSAR foi que o cenário restringia a utilização de um número maior de animais, visto que era pequeno e aumentá-lo envolveria um problema maior já que está vinculado ao marcador utilizado para a cena. Além disso o formato do terreno dificultava a movimentação dos personagens, resultando numa movimentação menos fluida. Estes problemas acabaram limitando a possibilidade de implementação de outros recursos como reprodução dos animais ou da criação de uma forma para adicioná-los e/ou removê-los do cenário, visto que com três animais o cenário já se mostrava cheio.

O trabalho se mostrou relevante pois como pode ser observado em relação aos trabalhos correlatos selecionados, os projetos anteriores procuravam desenvolver um modelo de inteligência artificial apenas aplicado aos seus respectivos trabalhos, sem a utilização de um editor de jogos consolidado. Com a pesquisa realizada e o desenvolvimento deste trabalho, pode-se aprender sobre a biblioteca Unity ML-Agents e compreender como utilizá-la em um projeto, facilitando trabalhos futuros e explicando esta tecnologia que é implementada utilizando um motor de jogos amplamente utilizado por alunos e desenvolvedores de jogos e simuladores.

4 CONCLUSÕES

Com a análise da execução dos modelos de comportamento treinados provou-se que os animais foram capazes de se comportar da maneira desejada e, portanto, tornaram o simulador de ecossistemas mais robusto, adicionando uma nova camada de estudo e aprendizado do meio ambiente. Além disso as tecnologias utilizadas para o desenvolvimento do trabalho mostraram-se apropriadas, sendo o Unity ML Agents uma ferramenta completa e eficiente, permitindo desenvolvimento de modelos de comportamentos complexos e desenvolvimento de múltiplos agentes com sucesso em uma cena do Unity.

Cumpriu-se o objetivo de pesquisar sobre tecnologias de inteligência artificial disponíveis para o Unity e posteriormente estudar e desenvolver uma solução utilizando a biblioteca Unity ML-Agents. Essa pesquisa torna disponível o conteúdo sobre essa biblioteca que está fortemente integrada ao motor de jogos Unity, amplamente utilizado em outros projetos acadêmicos e recebe atualizações constantemente, recebendo foco da equipe de desenvolvimento do Unity para se tornar uma solução cada vez mais robusta.

As possíveis extensões propostas para continuar a linha de pesquisa desse projeto são: ampliação do cenário do marcador principal do ECOSAR; reprodução dos animais; interação dos animais com os ciclos de dia/noite do cenário; possibilitar adição/remoção de animais no cenário; adição de animais aquáticos e/ou voadores.

REFERÊNCIAS

ALDRICH, Clark. **Learning online with games, simulations and virtual worlds**. San Francisco, CA: Jossey-Bass, 2009.

DAUTENHAHN, Kerstin; NEHANIV, Chrystopher L. **Imitation in Animals and Artifacts**. Cambridge, MA; The MIT Press, 2002.

EUROGRAPHICS/ACM SIGGRAPH SYMPOSIUM ON COMPUTER ANIMATION, x., 2005, Los Angeles. **Proceedings...** New York: Association for Computing Machinery, 2005. 274 p.

FELTRIN, Gustavo R. **VISEDU-SIMULA 1.0: Visualizador de material educacional, módulo de animação comportamental**. 2014. 91f. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

FRONZA, Germano**. Simulador de um ambiente virtual distribuído multiusuário para batalhas de tanques 3D com inteligência baseada em agentes BDI**. 2008. 141f. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

GREIS, Luciano K; REATEGUI, Eliseo. Um Simulador Educacional para Disciplina de Física em Mundos Virtuais. **Renote: Revista Novas Tecnologias na Educação**, Porto Alegre, v. 8, n. 2, p.1-10, jul. 2010.

HARBS, Marcos. **Motor para jogos 2D utilizando HTML5**. 2013. 77f. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

INTELLIGENT AGENTS, 1., 1994, Amsterdã. **Proceedings...** New York: Springer, 1995. 412 p.

INTELLIGENT AGENTS: AGENT THEORIES, ARCHITECTURES, AND LANGUAGES, 3., 1996, Budapeste. **Proceedings...** Berlim: Springer, 1997. 401 p.

INTERNATIONAL CONFERENCE, 13., 2013, Edimburgo. **Anais...** Berlim: Springer, 2013. 493 p.

JULIANI, Arthur et al. **Unity: A General Platform for Intelligent Agents**. San Francisco: Unity Technologies, 2018. 28 p. Disponível em: https://arxiv.org/pdf/1809.02627.pdf. Acesso em: 6 jul. 2020.

LOPES, Nuno; OLIVEIRA, Isolina. Videojogos, Serious Games e Simuladores na Educação: usar, criar e modificar. **Educação, Formação & Tecnologias** - ISSN 1646-933X, Lisboa, v. 6, n. 1, p.1-20, jul. 2013.

MENDES, Mauricio A.; FIALHO, Francisco A. P. **Avaliação de simuladores aplicados na educação tecnológica a distância.** 2004. Disponível em: <http://www.abed.org.br/congresso2004/por/htm/036-TC-B1.htm> . Acesso em: 12 de abril de 2020.

PEREIRA, Roberta Andressa. **Definição dos animais e seus comportamentos para o aplicativo ECOSAR**. 2020. Entrevistador: João Marcos Estevão. Blumenau. 2020. Entrevista feita através de conversação – não publicada.

PEREIRA, Rodrigo W. **ECOSAR – Simulador de ecossistemas utilizando realidade aumentada**. 2019. 21f. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

PISKE, Kevin E. **VISEDU – Aquário virtual: Simulador de ecossistema utilizando animação comportamental**. 2015. 113f. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

ROSA, Thomas da. **Simulador de animais vivos: Meios alternativos**. 2008. 59f. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SIMPÓSIO BRASILEIRO DE COMPUTAÇÃO GRÁFICA E PROCESSAMENTO DE IMAGENS, 6., 1993, Recife. **Anais...** Recife: SBC/UFPE, 1993. 122 p.

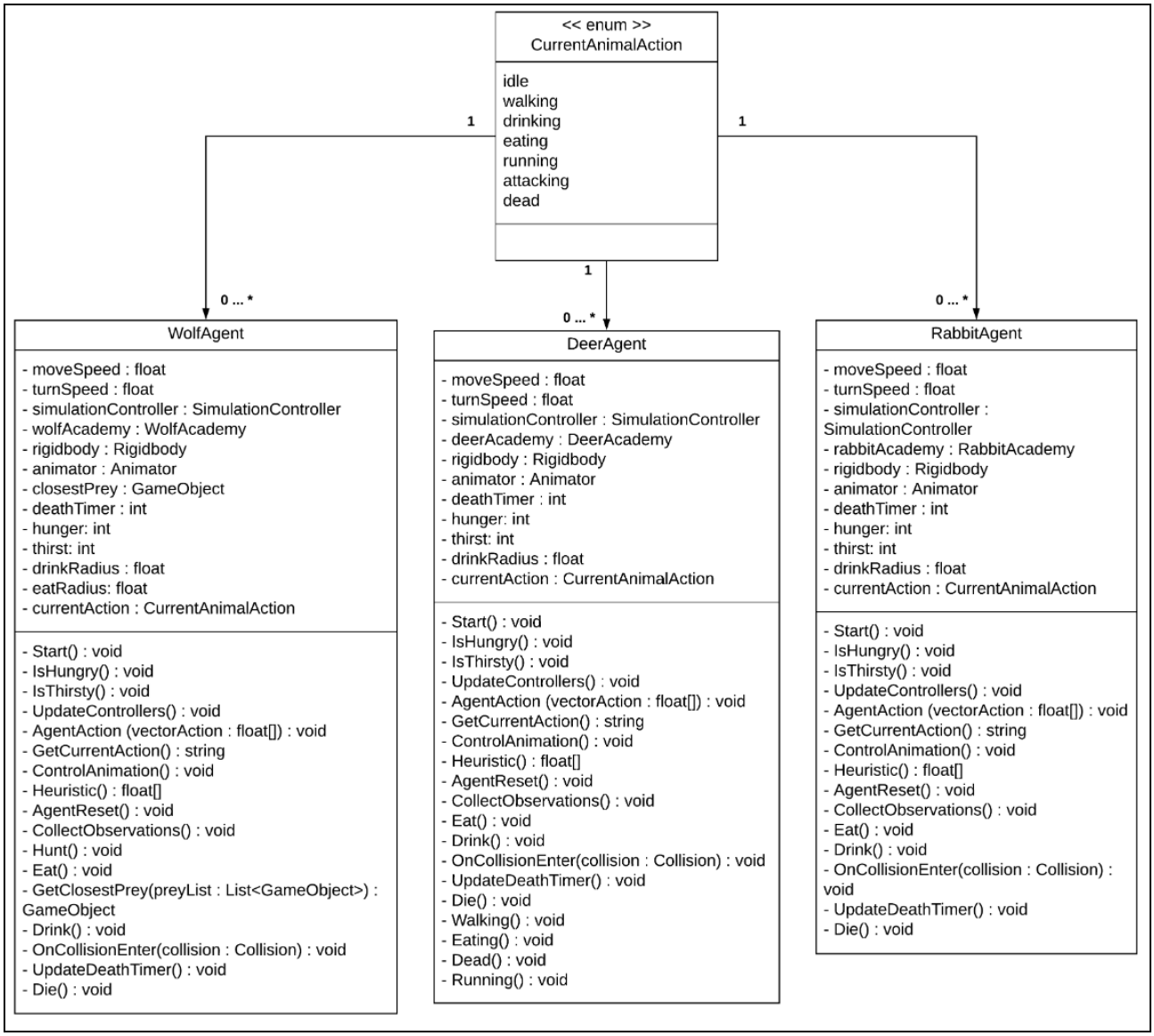
STONE, Robert. **Human Factors Guidelines for interactive 3D and Games-based training Systems**. 2012. Disponível em: <https://www.birmingham.ac.uk/Documents/college-eps/eece/research/bob-stone/human-factors-guidance.pdf>. Acesso em: 27 de outubro de 2019.

ULICSAK, Mary; WRIGHT, Martha. **Games in Education: Serious Games**. 2010. Disponível em: <https://www.nfer.ac.uk/media/1823/futl60.pdf>. Acesso em: 27 de outubro de 2019.

APÊNDICE A – DIAGRAMAS DE CLASSES

Este apêndice apresente os diagramas de classe do trabalho. Na Figura 9 são apresentados os diagramas das classes dos agentes do veado, do coelho e do lobo. Neles como pode ser visualizado estão os contadores de sede e fome, representados pelas variáveis `hunger` e `thirst`, assim como o componente `Animator` para gerenciamento das animações, o `Rigidbody` para controle de física e uma variável que conterá a instância da `Academy` de cada animal. Além disso cada classe tem algumas variáveis de controle como por exemplo para controlar a velocidade do agente, a distância que ele pode beber a água do lago e a ação atual do personagem. Entre os métodos pode-se destacar o `AgentAction`, explicado no trabalho, que faz o gerenciamento de ações e controle do personagem, o `CollectObservations` que irá gerenciar quais informações do cenário o usuário poderá coletar a partir da observação e utilizar para treinar seu comportamento e o `Heuristic` que serve para permitir o controle do personagem pelo usuário, para facilitar o teste de animações, movimentação e outros aspectos mais facilmente. Além disso os métodos `Eat` e `Drink` realizam as ações de comer e beber do agente e estipulam a recompensa a ser repassada ao agente por estas ações de acordo com as condições em que elas foram executadas. Outros métodos também fazem o gerenciamento das animações, o controle e atualização dos contadores de sede e fome e a atualização do objeto de cena ao morrer, entre outras funções.

Figura 9 – Diagrama de classes dos agentes

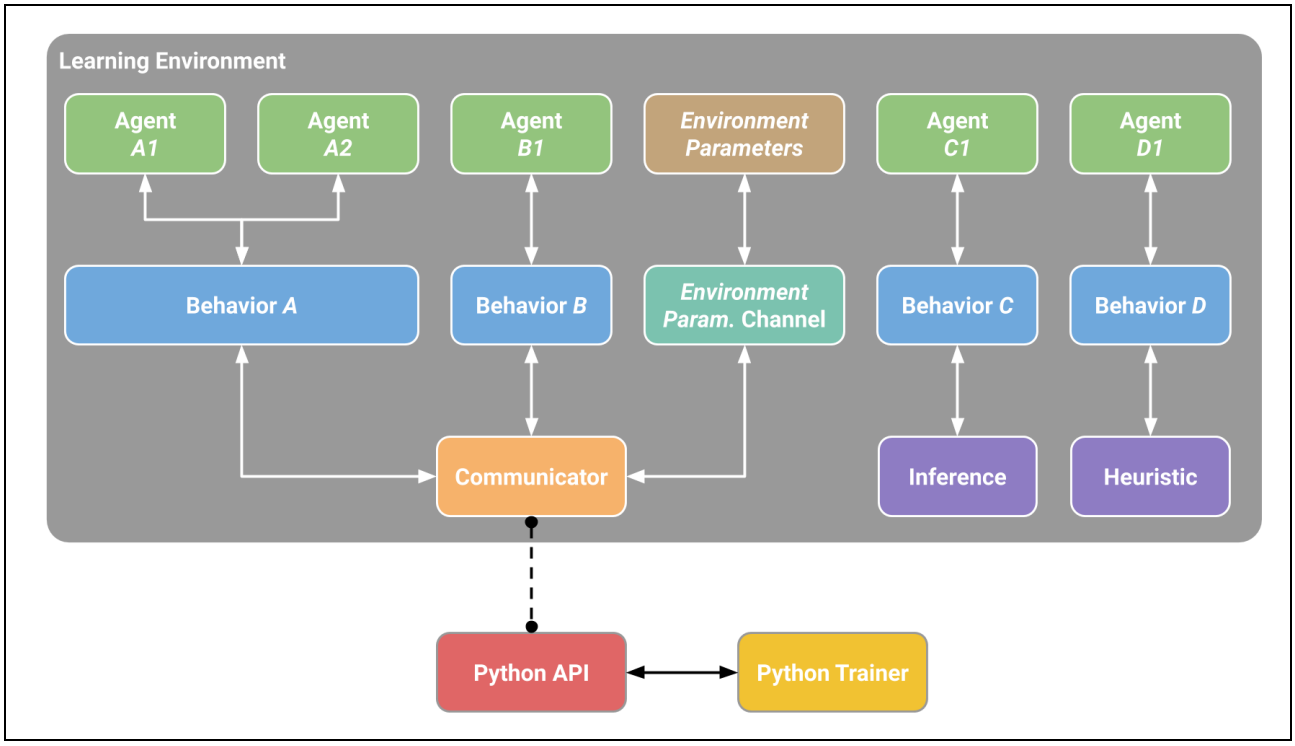


Fonte: elaborado pelo autor.

APÊNDICE B – CONFIGURAÇÃO DE AMBIENTE PARA UTILIZAÇÃO DO UNITY ML-AGENTS

No diagrama da Figura 10 está representado o comportamento do ML-Agents em uma cena do Unity. Os componentes verdes representam os agentes, que possuem uma política de comportamento, representada em azul. Esta política, conforme pode ser visualizado, pode ser compartilhada entre mais de um agente e podem ser treinadas simultaneamente em uma cena, conforme se pode visualizar através da conexão com o componente `Communicator` em laranja, que representa a comunicação realizada no treinamento entre o editor de jogos do Unity e o *script* de treinamento do ML-Agents, através da API em Python. O agente pode ter outros dois tipos de controle além do *script* de treinamento, representados pelos componentes roxos. `Inference` se refere ao controle do personagem por um modelo de comportamento já treinado, resultado da execução de uma rotina de treinamento do ML-Agents, que gera um arquivo que pode ser inserido no objeto de cena para definir sua inteligência artificial. Já o `Heuristic` se refere ao controle do agente pelo próprio usuário, através do teclado ou algum outro tipo de comando. Por último o componente `Environment Parameters` se refere aos parâmetros que podem ser adicionados ao `Academy` em uma cena e serão utilizados para interferir no treinamento de acordo com o objetivo desejado pelo desenvolvedor.

Figura 10 – Diagrama de representação do funcionamento do ML-Agents



Fonte: Juliani et al. (2018).

APÊNDICE C – CONFIGURAÇÃO DE AMBIENTE PARA UTILIZAÇÃO DO UNITY ML-AGENTS

Este apêndice irá explicar como funciona a instalação e configuração do ambiente com o Anaconda para permitir a realização do treinamento dos agentes de cena utilizando o Unity. Foram utilizados o Unity ML-Agents na versão 0.13.1 e o Anaconda na versão 2019.10, com a versão 3.7 do Python. Ambos podem ser baixados através dos sites oficiais dos distribuidores destes componentes.

Após a instalação do Anaconda você deve executar o programa Anaconda Prompt, onde irão ser executados os comandos para configuração do ambiente. Primeiramente é preciso criar um ambiente Python para a execução dos *scripts* de treinamento do ML-Agents. Para isso deve ser executada uma linha de comando que irá criar um ambiente Python na versão 3.7 chamado ml-agents. Após a criação do ambiente é necessária a execução de uma outra linha de comando para ativar o ambiente. Ambas as linhas de comando podem ser visualizadas na Figura 10.

Figura 10 – Comandos para criação do ambiente

```
conda create -n ml-agents python=3.7
```

```
conda activate ml-agents
```

Fonte: elaborado pelo autor.

Após a ativação do ambiente outras linhas de comando devem ser executadas para instalação dos componentes do ML-Agents. Para isso você deve navegar através do Anaconda Prompt para a pasta de instalação do ml-agents na sua máquina. A partir dessa pasta serão rodados comandos para instalação de todos os componentes contidos dentro de duas subpastas deste diretório. A sequência de comandos necessária para instalação é demonstrada na Figura 11. Após a execução dos comandos mencionados acima a instalação estará concluída. Caso o Anaconda Prompt seja fechado e aberto novamente em outro momento é necessário apenas rodar a linha de comando para ativação do ambiente, demonstrada anteriormente na Figura 10.

Figura 11 – Comandos para instalação do Unity ML-Agents

```
cd ml-agents-envs
```

```
pip install -e ./
```

```
cd ..
```

```
cd ml-agents
```

```
pip install -e ./
```

Fonte: elaborado pelo autor.

Para o treinamento dos agentes primeiramente é necessário navegar até a pasta config, localizada dentro do diretório de instalação do ml-agents, e alterar o arquivo trainer_config.yaml. Lá deve ser adicionado um novo bloco de parâmetros de configuração para o treinamento do seu agente. Um exemplo dos parâmetros utilizados para treinamento do coelho pode ser visualizado na Figura 12.

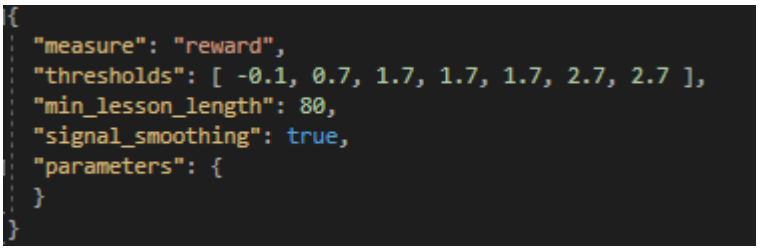
Figura 12 – Bloco de configuração geral do treinamento do coelho

```
RabbitLearning:
    summary_freq: 5000
    time_horizon: 128
    batch_size: 128
    buffer_size: 2048
    hidden_units: 256
    beta: 1.0e-2
    max_steps: 1.0e6
```

Fonte: elaborado pelo autor.

Após a alteração do arquivo de treinamento deve ser criada uma pasta, nomeando-a da forma desejada, dentro do diretório curricula, localizado dentro da pasta config, acessada no passo anterior. Dentro desta pasta deve ser criado um arquivo no formato JavaScript Object Notation (JSON) com as configurações para o treinamento de cada agente que se deseje treinar na cena. Como exemplo, neste trabalho foi criado o arquivo RabbitLearning.json para o treinamento do agente do coelho. É importante ressaltar que o nome deste arquivo deve ser o mesmo nome atribuído no atributo `Behavior Name`, localizado dentro do componente `Behavior Parameters`, que irá ser adicionado no seu agente de cena dentro do editor do Unity. Se o nome estiver diferente o treinamento não será efetuado. A Figura 13 mostra o conteúdo do arquivo RabbitLearning.json, utilizado neste trabalho.

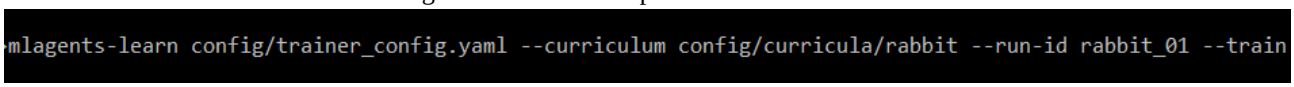
Figura 13 – Arquivo json com parâmetros para o treino do coelho



Fonte: elaborado pelo autor.

Após a criação das pastas e arquivos citados anteriormente o treinamento pode ser executado. Para isso devem ser configurados os componentes e *scripts* no editor do Unity conforme o desejado e então deve ser feita a execução de uma linha de comando para iniciar o treinamento. Esta linha de comando irá ativar a API de treinamento do ML-Agents de acordo com os parâmetros configurados nos arquivos citados nos passos anteriores. Após a execução desta linha de comando será exibida no *prompt* uma mensagem informando que o treinamento pode ser iniciado ao clicar na tecla Play no editor do Unity. Ao fazer isso os agentes iniciarão seu treinamento, que pode ser visualizado no Unity. No momento que a execução for interrompida no ambiente do Unity, o treinamento será gerado um arquivo de comportamento para o agente, que poderá ser adicionado ao atributo `Model`, localizado no componente `Behavior Parameters` no objeto de cena do agente no Unity, para que ele passe a agir de acordo com o treinamento efetuado. A Figura 14 mostra a execução da linha de comando para início de treinamento do agente do coelho utilizado no trabalho.

Figura 14 – Comando para iniciar treinamento



Fonte: elaborado pelo autor.