UNIVERSIDADE ESTADUAL DE CAMPINAS

Faculdade de Engenharia Elétrica e de Computação

Wandemberg Santana Pharaoh Gibaut

Uma Arquitetura Cognitiva para o Aprendizado Instrumental em Agentes Inteligentes

Campinas

2018

Wandemberg Santana Pharaoh Gibaut

Uma Arquitetura Cognitiva para o Aprendizado Instrumental em Agentes Inteligentes

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica, na Área de Engenharia de Computação.

Orientador: Prof. Dr. Ricardo Ribeiro Gudwin

Este exemplar corresponde à versão final da dissertação defendida pelo aluno Wandemberg Santana Pharaoh Gibaut, e orientada pelo Prof. Dr. Ricardo Ribeiro Gudwin.

Campinas

2018

Agência(s) de fomento e nº(s) de processo(s): CAPES, 1548944
ORCID: https://orcid.org/0000-0001-7322-5399

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Luciana Pietrosanto Milla - CRB 8/8129

G35a

Gibaut, Wandemberg Santana Pharaoh, 1992-
Uma arquitetura cognitiva para o aprendizado instrumental em agentes inteligentes / Wandemberg Santana Pharaoh Gibaut. – Campinas, SP : [s.n.], 2018.

Orientador: Ricardo Ribeiro Gudwin.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Agentes inteligentes. 2. Memória episódica. I. Gudwin, Ricardo Ribeiro, 1967-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: A cognitive architecture for instrumental learning in intelligent agents
Palavras-chave em inglês:
Intelligent agents
Episodic memory
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora:
Ricardo Ribeiro Gudwin [Orientador]
Romis Ribeiro de Faissol Attux
Angelo Conrado Loula
Data de defesa: 31-01-2018
Programa de Pós-Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - DISSERTAÇÃO DE MESTRADO

Candidato: Wandemberg Santana Pharaoh Gibaut **RA:** 106992
Data da defesa: 31 de janeiro de 2018

Titulo da Dissertação: Uma Arquitetura Cognitiva para o Aprendizado Instrumental em Agentes Inteligentes

Prof. Dr. Ricardo Ribeiro Gudwin (Presidente, FEEC/UNICAMP)
Prof. Dr. Angelo Conrado Loula (UEFS)
Prof. Dr. Romis Ribeiro de Faissol Attux (FEEC/UNICAMP)

A ata de defesa, com as respectivas assinaturas dos membros da Comissão Julgadora, encontra-se no SIGA (Sistema de Fluxo de Dissertação/Tese) e na Secretaria de Pós-Graduação da Faculdade de Engenharia Elétrica e de Computação.

Dedico esta dissertação a Maria Aparecida Santana pelo apoio e amor incondicional por toda uma vida e, in memoriam, a Robson Pharaoh Gibaut pelo seu amor e zelo, e que tinha o sonho de ver o filho doutor.

Agradecimentos

Agradeço primeiramente a minha mãe, Maria Aparecida, pelo apoio incondicional afetivo, psicológico e financeiro durante toda minha vida, especialmente durante a minha trajetória acadêmica, desde 2010 até aqui. Agradeço também a Eugênio Rodrigues e Gustavo Menezes pelo apoio como amigos, incentivadores e as vezes conselheiros, a André Paraense pela sempre disponível ajuda quando precisei e a Suelen Mapa pelos conselhos e desabafos. À Helena Amélia, um agradecimento especial pelas noites viradas no laboratório, companhia nas madrugadas silenciosas da FEEC. Agradeço ainda à CAPES pelo apoio financeiro dado durante este período.

"If a machine is expected to be infallible, it cannot also be intelligent."
(Alan Turing)

Resumo

Este trabalho apresenta uma proposta de Arquitetura Cognitiva para controle em sistemas inteligentes capaz de aprender autonomamente baseada na observação do ambiente e no resultado de suas ações. Para isso, a arquitetura utiliza de técnicas diversas, tais como o Aprendizado por Reforço, Redes Neurais e Memória Episódica, que permitem ao agente, dentre outras coisas, gerar expectativas para suas ações e relembrar experiências passadas. Para validar a arquitetura, realizamos um teste de desempenho da mesma no ambiente virtual do jogo *Minecraft*.

Palavras-chaves:Sistemas Cognitivas; Agentes Inteligentes; Memória Episódica; Aprendizado por Reforço.

Abstract

This work proposes a Cognitive Architecture for control in intelligent systems, able to learn autonomously, based on the results of its own actions observed from the environment. To accomplish this, the Architecture counts on different techniques, such as Reinforcement Learning, Neural Networks and Episodic Memory, which allows the system to generate expectations for its actions and to retrieve past experiences. To validate the architecture, we developed a performance test, using for that the virtual environment of the game *Minecraft*.

Keywords: Cognitive Systems; Intelligent Agents; Episodic Memory; Reinforcement Learning.

Lista de ilustrações

Sumário

1 Introdução

Carros autônomos, casas inteligentes, *bots* de internet ... cada vez mais a presença de sistemas inteligentes, sejam eles virtuais ou ciber-físicos, vem ganhando espaço e tornando-se mais marcante em tudo que cerca a vida humana. Esses sistemas são muitas vezes responsáveis por tarefas muito sensíveis a erros e/ou por casos onde é impossível prever todas as situações a serem encontradas, a priori. Tendo em mente isso, faz-se necessário um controle inteligente adequado para os sistemas atuais e vindouros, com especial interesse naqueles capazes de apresentar modelos generalizados o suficiente para suprir demandas de diversas naturezas, com destaque para a tomada de decisões sob fortes incertezas e informações parciais, isto é, quando não se possuem todas as informações acerca da tarefa, mas ainda assim precisa-se atuar. Este destaque se dá devido a estas situações geralmente representarem aplicações de maior interesse no que tange a sistemas com espaços de estados de alta dimensionalidade e parcialmente percebíveis, como por exemplo o mundo real.

1.1 Motivações e Objetivos

A principal motivação desse trabalho foi estudar como o uso de uma arquitetura cognitiva poderia ser empregado para controlar uma criatura artificial em um jogo de computador de alta complexidade (no caso, o Minecraft, um sofisticado ambiente 3D) e avaliar mecanismos de aprendizagem que permitissem que a criatura pudesse aprender sozinha, por meio de tentativa e erro, a operar em seu ambiente. Para tanto, foram utilizadas técnicas de aprendizado não-supervisionado (por exemplo, técnicas de aprendizado por reforço) e redes neurais, bem como modelos de memória episódica, explorando-se como tais técnicas podem ser utilizadas em arquiteturas cognitivas.

Em virtude dessa motivação, o objetivo do presente trabalho foi propor uma versão alternativa da arquitetura cognitiva MECA (GUDWIN *et al.*, 2017), sendo desenvolvida em nosso grupo de pesquisa, que pudesse incluir a questão do aprendizado instrumental, estando alinhada a teorias bem estabelecidas e apresentando uma visão ampliada do funcionamento de uma mente. Para isso, alguns módulos originais do MECA foram suprimidos e outros foram incorporados. A arquitetura foi testada tendo como estudo de caso seu uso no controle de uma criatura artificial em um jogo de computador com alto grau de liberdade de ações, no caso o *Minecraft*, um ambiente onde o elevado tamanho do espaço de estados torna formas mais convencionais de controle pouco eficientes e consideravelmente limitadas. De modo particular, pretendeu-se analisar como modelos de Memória Episódica, um sistema de memória independente no qual as informações são

contextualizadas no tempo, como um filme autobiográfico, podem auxiliar no processo de aprendizagem e tomada de decisão de um agente cognitivo de forma que o agente possa explorar e aprender sobre seu ambiente, gerando comportamentos que façam bom uso dos recursos disponíveis de modo a maximizar seus indicadores de desempenho. Para tanto, foram utilizadas técnicas de aprendizado não-supervisionado (por exemplo, técnicas de aprendizado por reforço, apresentadas no próximo capítulo) e redes neurais, bem como modelos de como as mesmas podem ser utilizadas em arquiteturas cognitivas.

Alguns dos maiores desafios do trabalho foi contornar de forma eficiente o problema do grande número de possibilidades de estados do ambiente, mantendo uma boa relação custo-benefício entre explorar novas possibilidades e fazer bom uso do conhecimento já adquirido e contornar problemas relacionados a convergência de aproximadores em técnicas de Aprendizado por Reforço. No experimentos desenvolvidos, utilizou-se o *Project Malmo* (JOHNSON *et al.*, 2016), uma camada de abstração para o jogo *Minecraft*, desenvolvido e disponibilizado publicamente pela empresa Microsoft para a comunidade acadêmica. Para implementar a arquitetura cognitiva controlando o agente, foi utilizado o CST (RAIZER *et al.*, 2012; PARAENSE *et al.*, 2016) um projeto em desenvolvimento no DCA-FEEC-UNICAMP pelo grupo de pesquisa orientado pelo Prof. Gudwin.

1.2 Sobre este Trabalho

Este trabalho está divido em cinco capítulos. No presente capítulo foram mostradas as motivações e objetivos deste projeto. No capítulo 2, faz-se uma pequena excursão pelos principais conceitos envolvidos na proposta, como Aprendizado Instrumental e Memória Episódica. No capítulo 3, explicamos o funcionamento da principal ferramente utilizada, o CST, e o modelo de representação de conhecimento trabalhado. No capítulo 4, são explicados em detalhes a plataforma utilizada, a proposta de arquitetura propriamente dita e um controlador dedicado a uma aplicação, que serve como prova de conceito. O quinto capítulo é dedicado à descrição, aos resultados e à análise dos experimentos realizados. Na Conclusão, apresentamos um sumário sobre as principais contribuições do trabalho e especulamos sobre possíveis trabalhos futuros.

2 Inteligência, Cognição e Aprendizado

O termo *inteligência*, derivado do substantivo em latim *intelligentia* que por sua vez remete ao verbo *intelligere* (algo como 'compreender', 'perceber'), já possuiu diversas definições ao longo do tempo e mesmo hoje não existe um consenso de qual delas melhor expressa o conceito. Uma definição formal é apresentada em Gottfredson (1997) que, em uma tradução livre, diz:

> Uma capacidade mental bem geral que, entre outras coisas, envolve a habilidade de raciocínio, planejamento, resolução de problemas, pensamento abstrato, compreender ideias complexas, aprender rapidamente e aprender a partir de experiências. Ela (a inteligência) não é meramente relacionada ao aprendizado em livros, uma habilidade acadêmica específica ou conhecimento a partir de testes. Ao contrário, ela reflete a capacidade mais ampla e profunda de compreensão dos nossos arredores - "compreendendo", "dando sentido" a coisas ou "entendendo" o que fazer.

Dentre outras definições, inteligência pode ser entendida como a capacidade ou inclinação a perceber ou deduzir informações e retê-las como conhecimento a ser aplicado em comportamentos adaptativos dentro de um ambiente ou contexto pertinente. Isto é, este conhecimento pode vir do ambiente, por meio de novas informações sensoriais, ou inferidas pelo agente a partir de dados já previamente captados.

Fortemente relacionado a este conceito mais abstrato, o conceito de cognição apresenta-se como a ação mental ou processo de aquisição de conhecimento e entendimento via pensamento, experiência e sentidos (BISTRICKY, 2013). Esse conceito engloba processos como conhecimento, atenção, memória e memória de trabalho, julgamento e avaliação, raciocínio, resolução de problemas e tomadas de decisão e compreensão e produção de linguagem. A cognição pode apresentar-se de forma consciente ou inconsciente, concreta ou abstrata e utilizar conhecimento existente pra gerar conhecimento novo.

Aprendizado, por sua vez, é entendido como o processo de adquirir ou modificar conhecimento, habilidades, valores, comportamentos ou competências e pode ser observado em variações de comportamento de vários níveis de complexidade, dos mais simples aos mais sofisticados (GROSS, 2010).

A seguir, faz-se um pequeno resumo de uma série de conceitos relevantes para a obtenção dos objetivos deste trabalho, de acordo com o que se pode encontrar na literatura.

2.1 Agentes Inteligentes e Agentes Cognitivos

De acordo com Franklin e Graesser (1997), um agente autônomo pode ser definido como um sistema que, sendo parte de um ambiente, encontra-se situado dentro deste, percebe e age sobre tal ambiente ao longo do tempo buscando seus próprios propósitos, de modo a efetivar o que eles determinam para o futuro. Dependendo da maneira como esse agente determina sua atuação sobre o ambiente, podemos ter diversas classes (RUSSELL; NORVIG, 2003), desde agentes puramente reflexivos, ou reativos, que meramente executam uma função sobre os dados sensoriais, até agentes que buscam atingir um estado futuro pré-determinado, chamados de *agentes inteligentes*. Um humano é um exemplo de agente inteligente biológico com órgãos dos sentidos como olhos, ouvidos etc, como sensores e outras partes como mãos, pés e boca como atuadores. Agentes autônomos podem variar em complexidade de capacidade de interferência com o ambiente, de forma que um humano, um carro autônomo e um termostato podem, os três, enquadrar-se no conceito. Existem ainda diversas outras definições de agentes autônomos que podem ser encontradas em Franklin e Graesser (1997) mas, para o presente trabalho, esta foi considerada a mais adequada.

Imaginemos um agente que, por meio de sua atuação sobre o ambiente, consiga chegar a um determinado estado desejado. De acordo com nossa definição, esse agente seria inteligente. Entretanto, pode haver diversas maneiras de se chegar a esse estado, sendo que uma maneira pode ser mais custosa ou mais demorada do que outra. Dessa forma, podemos imaginar que um outro agente, que chegue ao mesmo estado, porém com uma otimização de certas variáveis como custo, tempo etc., poderia ser classificado como sendo mais eficiente do que o primeiro agente, que simplesmente atingiu o estado desejado sem levar esses parâmetros em consideração. Podemos ainda imaginar que um agente utilize sempre um mesmo conjunto de princípios para sua operação, inseridos em sua codificação desde que o mesmo começou a operar, e operando sempre da mesma maneira. Esse agente, por sua vez, pode ser visto como menos eficiente, no tocante a lidar com novas situações, do que outro que consiga modificar seu comportamento, adaptando-se ao ambiente e aprendendo novas habilidades à medida que interage com ele.

Em seres humanos, é possível identificar diversas capacidades mentais, que fazem diferir um indivíduo de outro. Essas capacidades, tais como a percepção, a imaginação, a abstração, o reconhecimento e a associação de padrões, a emoção, o raciocínio, a tomada de decisões, a memória, a aprendizagem, a linguagem, o planejamento e a execução de comportamentos sequenciais motivados, a consciência etc. são chamadas de capacidades cognitivas, e envolvem processos (ainda em parte desconhecidos) por meio do qual o ser humano é capaz de conhecer seu ambiente e modelá-lo de tal forma a prever seu comportamento e atuar sobre ele de forma a modificá-lo de acordo com seus propósitos, da maneira mais eficiente possível. Em ciência cognitiva, essas capacidades são modeladas

de diversas maneiras, resultando em diferentes modelos cognitivos que tentam explicar como a mente humana funciona (JOHNSON-LAIRD, 1980). Quando nos servimos destes modelos cognitivos para a criação de mentes artificiais para agentes inteligentes, nos referimos a esses agentes como sendo agentes cognitivos.

2.2 Redes Neurais

De acordo com Haykin (1998), uma rede neural artificial (RNA) é um processador paralelo e distribuído constituído por unidades de processamento simples que tem uma propensão natural para armazenar conhecimento e permitir um processamento do mesmo. Uma RNA assemelha-se a um cérebro em dois aspectos:

1. O conhecimento é adquirido pela rede, a partir de seu ambiente, através de um processo de aprendizado.
2. As intensidades das conexões entre neurônios, os pesos sinápticos, são utilizados para armazenar o conhecimento adquirido.

O poder computacional de uma Rede Neural vem de sua estrutura distribuída e paralela e de sua habilidade de generalização, isto é, de sua capacidade de apresentar respostas plausíveis para entradas que não estavam presentes durante a fase de treinamento. Dessas duas capacidades derivam algumas propriedades interessantes (HAYKIN, 1998), sendo que as propriedades a seguir são de especial relevância para o presente trabalho:

- Mapeamento entrada-saída: Através de métodos de aprendizado supervisionado, uma rede neural aprende a partir dos exemplos a ela apresentados, construindo um mapeamento entre entrada e saída para um dado problema.
- Adaptatividade: Redes Neurais têm uma capacidade inerente de adaptar seus pesos sinápticos às mudanças em seu ambiente, podendo passar por retreinamentos para lidar com pequenas mudanças no meio ou mesmo serem implementadas para operar tarefas não-estacionárias (isto é, cuja distribuição estatística muda com o tempo) através da atualização em tempo real de seus vetores de pesos. No entanto é importante tomar cuidado com sua plasticidade, de modo a evitar problemas como sobre-treinamento.
- Resposta Evidencial: Em algumas tarefas, como a classificação de padrões, uma rede neural fornece não apenas a qual classe um padrão de entrada pertence, mas também informações da confiança sobre a decisão tomada, dado esse utilizado para melhorar a própria performance da rede.

- Capacidade de Aproximação Universal: Para uma região compacta de um espaço contínuo, existe uma configuração de rede capaz de, dado um conjunto suficientemente grande de dados e um número de iterações adequado, aproximar uma função desconhecida com erro ϵ arbitrariamente pequeno.

2.2.1 Redes MLP (*Multi-Layer Perceptron*)

Dentre todos os modelos já propostos, talvez o mais famoso, simples e direto seja o MLP (*Multi-Layer Perceptron*, em inglês), ilustrado na figura 1. Possuindo um tipo de estrutura conhecida como não-recorrente (*Feed Forward*) este modelo de rede realiza um mapeamento do espaço dos dados (R^n, onde `n` é o número de entradas da rede) para o espaço de soluções através de um processo não-linear de propagação de informação por meio de suas múltiplas camadas até a camada de saída.

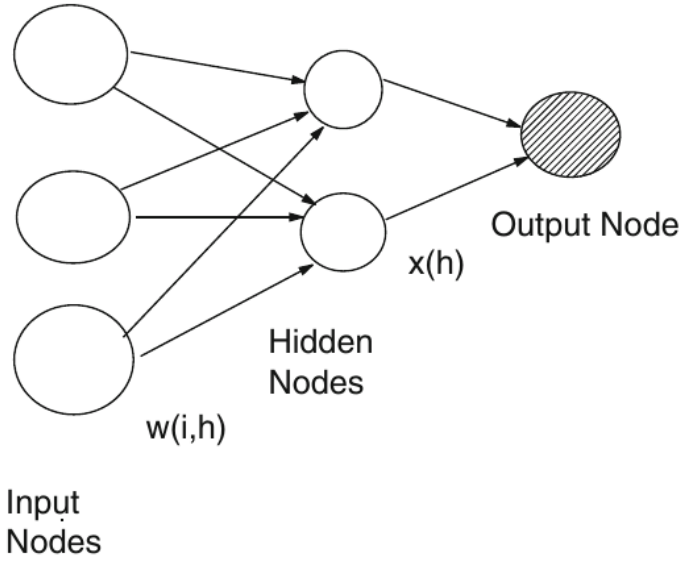

Figura 1 – Estrutura básica de uma Rede Neural do tipo MLP. Extraido de Gosavi (2015)

Entre as aplicações mais comuns para redes não-recorrentes destacam-se o uso como classificadores e como regressores. Em sua versão de classificação, isto é, de atribuição de rótulos já conhecido aos dados presentes, uma rede corretamente treinada é capaz de categorizar um padrão de entrada em uma ou mais das classes apresentadas, na forma da apresentação de um valor numérico nos neurônios correspondentes na camada de saída. Já em sua versão de regressão, uma MLP é capaz de aproximar uma função desconhecida com baixa perda de generalidade, ou seja, baixo erro em um conjunto de dados diferente daquele utilizado para treinar a rede, na forma de um valor real no neurônio (ou neurônios, no caso de múltiplas saídas) da camada de saída.

2.2.1.1 Estrutura

Uma MLP é composta das seguintes estruturas:

- Múltiplas unidades fundamentais conhecidas como neurônios, que possuem múltiplas entradas e uma única saída composta pela combinação linear entre estas entradas e seus respectivos pesos, sujeita a uma transformação pela função de ativação individual do neurônio.
- Uma camada de entrada, uma camada de saída e uma ou mais camadas intermediárias. Cada camada é composta por um determinado número de neurônios, os quais possuem suas entradas completamente conectadas aos neurônios da camada imediatamente anteriores e suas saídas àqueles da camada imediatamente posterior apenas, não possuindo conexões com neurônios da mesma camada ou de outras que não as especificadas.
- Uma função de ativação para cada neurônio. A função de ativação determina se e o quanto um neurônio esta ativo, isto é, atribui um valor para a combinação de entradas as quais a unidade esta sujeita. Apesar de existirem diversas propostas para funções de ativação, MLPs geralmente utilizam funções baseadas em uma categoria conhecida como *Ridge Function* devido ao seu formato. Entre essas funções destacam-se a Função Sigmoidal e a Tangente Hiperbólica, apresentadas na equações 2.1 e 2.2, respectivamente e ilustrada na figura 2.

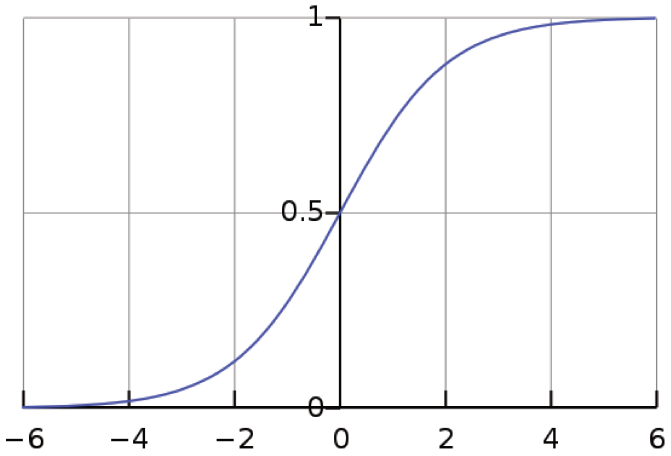

Figura 2 – Curva típica de uma função sigmoidal

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

$$g(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.2}$$

2.2.1.2 Aprendizado

O processo de aprendizado de uma Rede Neural se dá através da atualização dos valores de pesos sinápticos, valores que modulam como o nível de ativação de um neurônio influencia os outros neurônios com os quais este se conecta. Dessa forma, podemos entender que é nos pesos sinápticos que se armazena a informação em uma rede neural. Para modelar esse aprendizado, existe uma vasta gama de propostas de métodos, sendo o mais comum deles o Método de Gradiente Descendente, que utiliza o conceito de Retropropagação (*Backpropagation*, em inglês), que pode ser visto em (BOTTOU, 2012) e consiste em um método para calcular os gradientes necessários para o ajuste dos pesos da rede neural.

Vale lembrar ainda que o método em questão leva em conta um paradigma de aprendizado supervisionado, que é aquele onde a estrutura em questão aprende via exposição sucessiva de exemplos de pares de entrada e saída, adequando-se assim a uma estrutura que generalize o processo de mapeamento para o conjunto que lhe foi apresentado.

2.3 Aprendizado por Reforço

A aprendizagem é uma das capacidades cognitivas que deseja-se particularmente explorar neste trabalho. Conforme já mencionado anteriormente, aprendizagem é definida por Gross (2010) como o ato de adquirir ou modificar conhecimentos, comportamentos, habilidades, valores ou preferências preexistentes que podem levar a uma mudança na forma de sintetizar informação, profundidade de conhecimento, atitude ou comportamentos relativos ao tipo ou extensão de experiência. A aprendizagem é um processo que não ocorre todo de uma única vez, mas sim através do tempo, adaptando conhecimentos anteriores e seguindo um certo padrão convergente que chamamos de curva de aprendizagem. Note que apesar da definição tratar de aprendizagem em criaturas biológicas, pode-se traçar um bom paralelo entre esta categoria e aprendizagem em agentes autônomos.

De acordo com Sutton e Barto (1998), um tipo especial de aprendizagem de máquina é a assim chamada "Aprendizagem por Reforço" (*Reinforcement Learning*), método de aprendizado no qual um agente explora um espaço de estados-ação e, a cada espaço-ação que visitar, atribui um valor numérico associado à recompensa (ou punição) e, dessa forma, suas futuras ações ocorrerão de modo a maximizar um valor de recompensa numérica. Não é explicitado ao agente quais ações tomar, como na maioria dos métodos de aprendizado de máquina, cabendo ao próprio sistema a tarefa de descobrir quais ações levam à maior recompensa através de tentativa e erro. Em casos desafiadores, a ação tomada pode afetar não apenas a recompensa imediata, mas também a próxima situação e, portanto, todas as próximas recompensas. Essas duas características (busca por tenta-

tiva e erro e recompensa retardada) são as duas mais importantes marcas diferenciais do Aprendizado por Reforço.

De forma geral, em problemas dinâmicos ou de observabilidade parcial, abordagens que utilizam métodos de aprendizado supervisionado tendem a ser inadequadas, pois a aquisição de exemplos de comportamentos desejados que sejam tanto corretos como representativos para todas as situações que o agente tem que lidar é impraticável, em grande parte pela própria natureza exploratória do problema: não se conhece e nem se pode observar completamente o ambiente no qual o agente está inserido. Em um território não mapeado, por exemplo, um agente deve ser capaz de aprender a partir de suas próprias experiências.

Reforços e Punições representam as ferramentas centrais pelas quais o comportamento do agente é modificado, e seus termos são definidos de acordo com seus efeitos, sendo eles:

- Reforço Positivo: quando um comportamento é seguido por estímulos de recompensa, fazendo com que o comportamento apresente um aumento de frequência.
- Reforço Negativo: ocorre quando um comportamento é seguido pela retirada de estímulos negativos, fazendo com que o comportamento apresente um aumento de frequência.
- Punição Positiva: ocorre quando um comportamento é seguido por estímulos negativos, fazendo com que o comportamento apresente uma redução de frequência.
- Punição Negativa: ocorre quando um comportamento é seguido por retirada de estímulos de recompensa, fazendo com que o comportamento apresente uma redução de frequência.
- Extinção: ocorre quando um comportamento previamente reforçado positivamente não é mais efetivo, deixando de receber os estímulos de recompensa que anteriormente recebia, resultando em uma diminuição de frequência.

Note que a natureza do ”estímulo” esta ligada à natureza do agente: num agente biológico um estimulo de recompensa pode ser um alimento enquanto num criatura virtual pode significar um valor numérico maior que zero.

Um dos principais desafios em Aprendizado por Reforço é o compromisso entre explorar novas possibilidades e fazer bom uso do conhecimento já adquirido (SUTTON; BARTO, 1998). Para obter muita recompensa, o agente deve escolher ações previamente testadas e de eficácia comprovada. Mas, por outro lado, para conhecer essas ações e seus efeitos, o agente deve tentar executar ações não antes selecionadas. Nenhuma das duas

estratégias pode ser tomada de maneira isolada, sob pena de não ser capaz de atingir resultados adequados: o agente deve testar uma gama de ações e progressivamente favorecer aquelas que são consideradas as melhores. Provém das considerações anteriores que, devido ao mecanismo exploratório do espaço de ações, um agente submetido a esta forma de aprendizagem deve, desde o princípio, tomar decisões sob incertezas significativas. Atualmente técnicas mais aprimoradas de Aprendizado por Reforço (LITTMAN, 1994; SUTTON, 1996; KAELBLING *et al.*, 1996; KRISHNAMURTHY *et al.*, 2016) vêm sendo pesquisadas no intuito de tornar os métodos cada vez mais eficientes.

Existem diferentes maneiras por meio das quais o aprendizado por reforço pode ser utilizado. Uma delas envolve a existência de um instrutor externo, que por meio da escolha de sinais de reforço adequados visa condicionar um determinado agente a executar certas ações diante da presença de um conjunto de estímulos. Esse tipo de aprendizado por reforço é conhecido na literatura como condicionamento clássico (PAVLOV, 1960). No Condicionamento Clássico, o aprendizado ocorre da associação repetida de um estímulo condicionado, que normalmente não tem significado particular para o animal, com um estímulo não-condicionado, que tem significância para o animal e leva a uma resposta não-condicionada. Um outro tipo de aprendizado por reforço bastante diferente é o chamado aprendizado instrumental, também chamado de condicionamento operante. Nesse caso, não existe um instrutor externo como no condicionamento clássico, mas o agente recebe os sinais de reforço do próprio ambiente, a partir do resultado de suas próprias ações sobre este. Neste trabalho, estamos interessados em estudar somente o aprendizado instrumental, e como o mesmo pode ser integrado em uma arquitetura cognitiva. Note que apesar do termo Aprendizado pro Reforço ser amplamente conhecido na área de computação, as ideias presentes na técnica provem de um paralelo com conceitos de condicionamento vindo diretamente da psicologia.

2.3.1 Aprendizado Instrumental

Um dos aspectos importantes a se considerar na modelagem e implementação de agentes inteligentes que operam em ambiente parcialmente observável, isto é, aquele o qual não é possível (ou factível) obter informações completas, é a capacidade de prever as consequências de suas próprias ações. Ao aprender as relações de causa e consequência de suas ações sobre o ambiente, torna-se possível prever acontecimentos futuros. Neste contexto, modelos de condicionamento clássico e operante, originalmente desenvolvidos para criaturas biológicas e seus mecanismos cognitivos, podem ser utilizados como maneira de atingir tais objetivos de aprendizado.

Como vimos no início dessa seção, o aprendizado instrumental é aquele onde um agente (animal ou artificial) aprende com relativa autonomia a consequência de seus próprios atos. Mais especificamente, o agente aprende a exibir mais frequentemente um

comportamento que leva a uma recompensa e com menos frequência um comportamento que leva uma punição. Uma das principais diferenças entre o condicionamento clássico e o condicionamento operante é a existência de um instrutor externo. No condicionamento operante não existe esse instrutor externo, mas tão somente o agente e seu ambiente. O agente deve "operar" sobre o ambiente, decidir que ação tomar a cada instante de tempo e adaptar-se às consequências observadas.

O aprendizado instrumental vem sendo empregado em diversos trabalhos na área de robótica, com diversas finalidades (GAUDIANO *et al.*, 1996; GAUDIANO; CHANG, 1997; TOURETZKY; SAKSIDA, 1997; SAKSIDA *et al.*, 1997; ITOH *et al.*, 2005). Dentre outras aplicações, já foi utilizado no controle de um robô de duas rodas (HONG-GE; XIAO-GANG, 2009), em navegação autônoma (JING *et al.*, 2015) e no controle de pêndulo invertido (CAI; RUAN, 2009).

2.3.2 *Q-Learning*

Um dos mecanismos mais populares de aprendizagem instrumental é o mecanismo conhecido como *Q-Learning*, desenvolvido originalmente por Watkins (1989) em sua tese de doutorado. A ideia central do *Q-Learning* é reconhecer ou aprender a decisão ótima (também conhecida como Política Ótima) a cada estado visitado por um agente, por meio de tentativa e erro. Um vislumbre deste mecanismo pode ser visto na figura 3.

O funcionamento básico deste método é bastante simples: o agente executa uma ação, recebe um *feedback* sobre a decisão tomada e usa esta informação para atualizar sua base de conhecimento. Internamente, o agente mantém um valor chamado Fator-Q (*Q-factor* em inglês) para cada par estado-ação, isto é, para cada combinação entre estados e as ações possíveis a partir daquele estado. Quando o valor recebido por selecionar uma dada ação for positivo, esse Fator-Q é incrementado e, caso negativo, é decrementado. Esse valor recebido consiste na recompensa imediata mais o valor do próximo estado, que é dado pelo máximo Fator-Q em tal estado e representa uma estimativa de valor futuro.

Em termos matemáticos, o valor recebido pelo agente é dado por:

$$V(x_i, u(x_i)) = r(x_i, u(x_i)) + \gamma \max_{x_{i+1}} Q(x_{i+1}, u(x_{i+1}))$$

onde x representa um estado, u representa uma ação, r é a recompensa imediata, $Q(.)$ é o Fator-Q para um dado par estado-ação, $V(.)$ é o valor acumulado de um dado par estado-ação e $\gamma \in [0, 1)$ é um fator de desconto que regula o quão longo é o horizonte de estados ao qual esse fator remete.

Um dos pontos fundamentais do Algoritmo de *Q-Learning* é a seguinte função de atualização de valores:

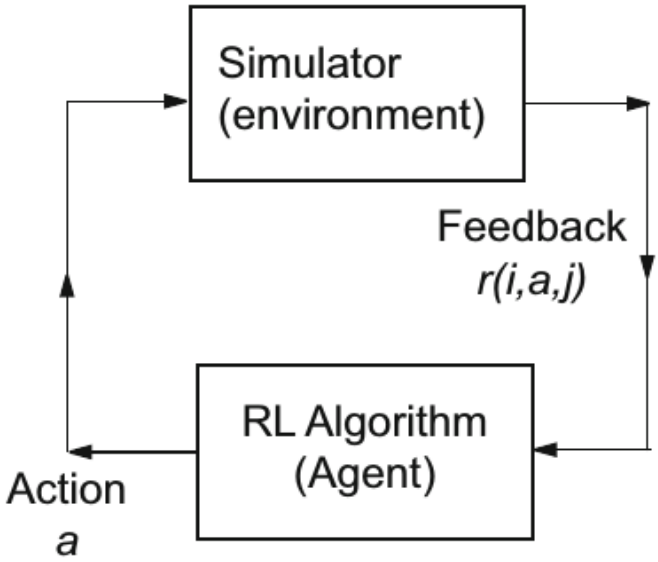

Figura 3 – Visão geral do mecanismo de aprendizado *Q-Learning*. Na imagem O Agente (*Agent*) atua (*Action*) no ambiente (*environment*) e obtém uma recompensa numérica (*Feedback*) do mesmo. Extraído de Gosavi (2015)

$$Q(x_i, u(x_i)) \leftarrow (1-\alpha)Q(x_i, u(x_i)) + \alpha V(x_i, u(x_i))$$

ou

$$Q(x_i, u(x_i)) \leftarrow (1-\alpha)Q(x_i, u(x_i)) + \alpha[r(x_i, u(x_i)) + \gamma \max_{x_{i+1}} Q(x_{i+1}, u(x_{i+1}))]$$

onde α é a taxa de aprendizado do sistema.

Apesar de apresentar muitas aplicações em aprendizado de máquina, o Q-learning original, com o simples uso de tabelas, é muitas vezes demasiado simples para descrever os processos cognitivos vistos em criaturas biológicas (TOURETZKY; SAKSIDA, 1997). Robôs com treinamentos baseados em algoritmos de Q-learning (CHEN *et al.*, 2008) não são capazes de superar a sofisticação e versatilidade de animais em termos de aprendizado. Assim, no intuito de buscar-se extrapolar as limitações de convergência dos métodos mais tradicionais, a utilização de outras técnicas baseadas em Condicionamento Operante (SKINNER, 1938) como o *chaining*, no qual comportamentos são construídos a partir de ações mais pontuais (TOURETZKY; SAKSIDA, 1997), formando um encadeamento de ações onde a ocorrência de uma ação leva a próxima, e que incorporam aspectos que não são trabalhados no Q-learning puro, representam boas alternativas. Neste trabalho, exploraremos algumas dessas extensões.

2.3.3 Redes Neurais como Aproximadores da Função Q

O uso de tabelas apresenta uma limitação bastante clara: quando se possui um grande número de pares de estado-ação não é factível guardar todos os Fatores-Q separadamente em uma tabela. Para ilustrar, em um cenário onde todas as ações possíveis possam ser tomadas independente do estado, o número de Fatores-Q a ser guardado é de $n_a.n_s$, onde n_a é o número de ações e n_s o número de estados. Esse valor pode rapidamente explodir conforme o cenário e o agente ficam mais complexos como no caso de um robô complexo navegando no mundo real: o número de estados é extremamente alto e o número de ações possíveis de serem escolhidas também é de ordem elevada. Esse problema é algumas vezes chamado de "Maldição da Dimensionalidade".

Para superar o problema da dimensionalidade, uma das possíveis alternativas é utilizar um aproximador de função. Esta solução consiste em encontrar uma função capaz de aproximadamente estimar o valor do Fator-Q utilizando como variável o estado atual e, em alguns modelos, a ação tomada. Porém esta mesma solução esbarra em outro problema: como conceber a priori uma função que estimasse os valores de estados em um ambiente desconhecido?

Neste ponto surge a utilização de Redes Neurais como aproximadores para esta função: uma rede neural utilizada como regressor receberia o estado atual como entrada e devolveria, em neurônios de saída separados, a estimativa de Fator-Q para cada ação tomada a partir do estado atual. Note também que a rede treinada seria capaz de estimar Fator-Q para estados não visitados por interpolação. Quando estes Fatores-Q precisassem ser atualizados, a rede neural teria seus pesos atualizados via algum método de aprendizado disponível na literatura, como a Retropropagação de Erros (*backpropagation, em inglês*) (HAYKIN, 1998). Para tal aprendizado, o sinal de erro deve seguir o método de atualização de Fatores-Q, isto é:

$$e_k(t) = \begin{cases} r(t) + \gamma \max_u Q_t(x(t+1), u) - Q_t(x(t), u_k), & \text{se } k = k' \\ 0, & \text{caso contrário} \end{cases}$$

onde k' representa o índice da ação $u(t)$ selecionada pelo agente no tempo t e $r(t)$ representa a recompensa imediata recebida pelo agente. É importante notar também que o Fator-Q, conforme expresso na equação de erro, representa a saída da rede neural para um dado estado, sendo cada nó k correspondente a uma possível ação.

De uma maneira ideal, o método simples de Retropropagação de Erros aplicado a uma rede MLP deveria ser capaz de adequar-se em quaisquer Fatores-Q, porém foi observado no presente trabalho e na literatura (FRASCONI *et al.*, 1997; ŠÍMA, 1996) que tal combinação requer uma cuidadosa escolha dos hiperparâmetros envolvidos tanto na

rede neural (como taxa de aprendizado e escolha da função de ativação), como da função de erro (como a escolha adequada do valor de γ), dado que não existe garantia de convergência de aprendizado para métodos de Aprendizado por Reforço com aproximadores de função, mesmo para aqueles com baixo grau de generalização.

2.4 Arquiteturas Cognitivas

A área de pesquisa em "Arquiteturas Cognitivas"começou a surgir como um *offspring* da Inteligência Artificial, no início da década de 90, a partir das bases lançadas anteriormente na década de 70 por Newell e Simon (1961) e Simon e Newell (1971), os sistemas especialistas, os sistemas de produção e os sistemas baseados em conhecimento. Duas arquiteturas cognitivas baseadas originalmente nessas teorias começaram a despontar. Baseando-se em sua teoria para a origem do conhecimento humano (ANDERSON, 1989), Anderson desenvolve a arquitetura cognitiva ACT-R (ANDERSON *et al.*, 2004; ANDERSON, 2009), ilustrada na figura 4, mais ou menos ao mesmo tempo em que baseando-se nas ideias de Newell *et al.* (1989), Laird lança as primeiras versões da arquitetura SOAR (LAIRD; ROSENBLOOM, 1996; LAIRD, 2012a).

Mas ao final da década de 90, todo um grupo de pesquisadores envolvidos na área de *Simulation of Adaptive Behavior* ajuda a desenvolver o conceito de arquitetura cognitiva como um conjunto essencial de estruturas e processos necessários para a geração de um modelo cognitivo-computacional, passível de ser utilizado em diversas áreas relacionando cognição e comportamento. Sloman (2002), por exemplo, começa a lidar com a questão “arquitetural” para representar conceitos mentais, e Sun (2004) começa a descrever os requisitos necessários para a construção de uma arquitetura cognitiva. Na sequência, o próprio Sun (2007) aborda questões e desafios no desenvolvimento de arquiteturas cognitivas. Langley *et al.* (2009) analisam diversas arquiteturas cognitivas, avaliando os progressos na área de pesquisa. Estes estudos indicaram que a principal vantagem em se caracterizar uma arquitetura como cognitiva é a possibilidade de se obter um *framework* concreto para a modelagem cognitiva, de tal forma que esses modelos pudessem ser testados de maneira concreta. Tal caracterização permitiria a definição de um conjunto de estruturas, módulos e processos básicos, criando uma linguagem comum para que diferentes pesquisadores pudessem explorar suas propostas de modelos cognitivos.

Desta forma, podemos entender uma arquitetura cognitiva como um sistema de controle de propósito geral inspirado em teorias científicas desenvolvidas para explicar o processo de cognição no ser humano e em animais. A ideia de se ter uma arquitetura é a de se decompor o fenômeno cognitivo em seu espectro de capacidades, tais como percepção, atenção, memória, raciocínio, aprendizagem, geração de comportamento e outros. Além disso, tais arquiteturas funcionariam também como modelos teóricos de processos

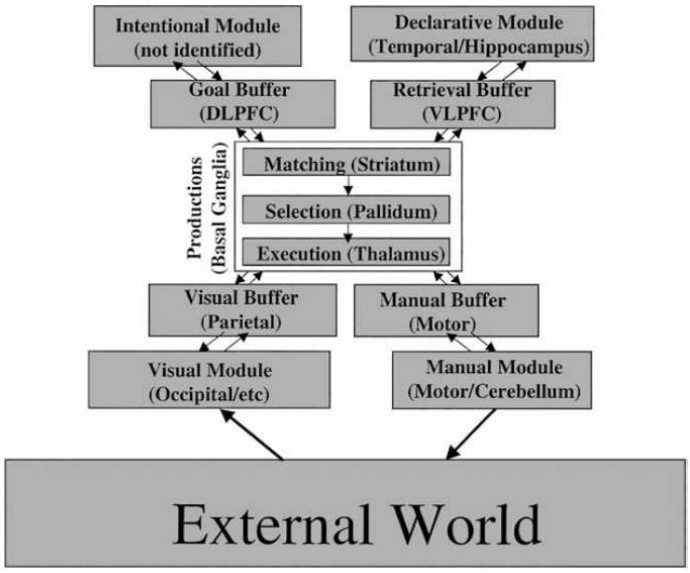

Figura 4 – Modelo estrutural da Arquitetura Cognitiva ACT-R 5.0. Adaptado de Anderson *et al.* (2004)

cognitivos, permitindo que tais teorias pudessem ser testadas e reutilizadas em diferentes aplicações.

Arquiteturas cognitivas vêm sendo empregadas nos mais variados usos, desde o controle de robôs a processos de tomada de decisão em agentes inteligentes. O uso de arquiteturas cognitivas facilita substancialmente os esforços para a construção de agentes artificiais inteligentes baseados em comportamentos humanos, em virtude da especificação de modelos cognitivos consistentes inspirados na cognição humana e animal (LAIRD, 2012a).

O estudo das arquiteturas cognitivas tem se tornado bastante popular mais recentemente, embora diversas questões ainda permaneçam em aberto. Por exemplo, o que é comum e o que é diferente entre as diversas propostas de arquiteturas cognitivas? Como o conhecimento é representado nas diversas arquiteturas? De maneira explícita (símbolos) ou de maneira implícita (redes neurais)? Quais os tipos de memória necessários (declarativa, não-declarativa, procedural, perceptual, episódica, semântica, de trabalho, sensorial, motora, etc.)? Quais seriam os sub-sistemas de memória essenciais? Existem processos que possam gerar uma cognição mínima ? Arquiteturas devem ser reativas, deliberativas ou híbridas? Como modelar os processos de formação de hábitos, adaptação e aprendizagem?

Apesar de tais questões permanecerem ainda inconclusas, ao longo dos últimos 20 anos, diversas arquiteturas cognitivas foram propostas (GOERTZEL *et al.*, 2014a). Em 2010, Samsonovich (2010) desenvolveu uma tabela comparativa apresentando uma revisão sistemática de um grande número de arquiteturas cognitivas conhecidas e documentadas. Diversas dessas arquiteturas são de propósito geral, possuem seu código-fonte

disponibilizado em *open-source* e estão disponíveis para download na Internet.

Atualmente, toda uma comunidade desenvolve pesquisas na área de arquiteturas cognitivas, sendo que os principais periódicos dedicados à área são o BICA journal – Biologically Inspired Cognitive Architectures e o AGI Journal – Journal of Artificial General Intelligence.

Nosso grupo de pesquisa, no DCA-FEEC-UNICAMP, desenvolve sua própria arquitetura cognitiva, o CST (Cognitive Systems Toolkit) (PARAENSE *et al.*, 2016). Em nosso trabalho, utilizamos o CST como base para a arquitetura cognitiva que controla o agente cognitivo que desenvolvemos.

2.4.1 Teoria do Processamento Dual (*Dual Process Theory*)

Conforme exposto na seção 2.4, as arquiteturas cognitivas buscam modelar computacionalmente teorias acerca da mente. Neste contexto algumas busca apoio em teorias como a do Processamento Dual. A Teoria do Processamento Dual (*Dual Process Theory*, em inglês) refere-se ao conjunto de teorias que, para modelar processos mentais, sugerem a existência de conjuntos de processos cognitivos que são rápidos, automáticos e inconscientes, distintos daqueles que são mais lentos, deliberativos e conscientes. Apesar de a natureza exata desses conjuntos depender do autor que os trata, segundo Osman (2004), esses dois sistemas possuem funcionalidades com papéis distintos que se integram para exercer os diferentes aspectos observados na mente humana, os quais, em seu trabalho, são denominados *System 1 e System 2*.

O *System 1*, aqui denominado Sistema de Baixa Cognição, é geralmente associado tanto a humanos como a outros animais, e representa um conjunto de subsistemas operando numa lógica de autonomia e paralelismo ,podendo, inclusive, estar relacionado a aspectos intrinsecamente ligados ao indivíduo. O *System 2*, aqui denominado Sistema de Alta Cognição, é considerado por alguns autores como sendo exclusivo da mente humana, e é o sistema responsável pela capacidade de abstração, construção de modelos mentais e tomadas de decisão, cujo funcionamento está intrinsecamente ligado a uma memória central chamada de Memória de Trabalho.

As contribuições presentes neste trabalho, como criação de expectativas e planos, focam principalmente no Sistema de Alta Cognição.

2.5 Memória Episódica

Uma das capacidades menos exploradas na área de agentes cognitivos é a assim chamada Memória Episódica. De acordo com (TULVING, 2002) a memória episódica é um dos tipos de memória do sistema de memórias que o ser humano possui, que permite que

seres humanos recordem de experiências passadas revivendo, na forma de uma simulação mental, trechos particulares no espaço e no tempo de sua autobiografia. Na classificação de Tulving, a Memória Episódica é um dos tipos de memória declarativa (o outro é a Memória Semântica), sendo uma memória de longo prazo, que armazena episódios sequenciais no tempo de forma linear, e permite tanto uma reprodução sequencial dos fatos acontecidos como saltos para pontos distinto no tempo, onde novas sequências são recuperadas. Esse fenômeno é chamado de *Mental Time Travel*, ou MTT.

Diversos experimentos realizados com seres humanos apontam para a evidência da existência de um sistema de memórias independente que armazena e permite a recuperação de experiências individuais de um ser humano. Uma dessas evidências é o caso de K.C., um indivíduo que, devido a um acidente de motocicleta, perdeu a capacidade de recordar-se de eventos, circunstâncias e situações passadas de sua vida enquanto manteve todos os elementos cognitivos associado a outros tipos de memória (TULVING, 2002), como habilidades com instrumentos musicais, conhecimento acadêmico etc.

Apropriando-se dos modelos cognitivos sobre memórias episódicas naturais, é possível desenvolver modelos computacionais de memória episódica. Dentre as questões importantes em modelos computacionais de memória episódica estão a questão da codificação (como a informação é passada para o armazenamento de memória), da recuperação (como e o que dispara a busca por episódios), bem como o próprio MTT, conforme discutido por Tulving e Thomson (1973).

Algumas arquiteturas cognitivas como o SOAR (LAIRD, 2012b) possuem módulos especializados implementando diferentes modelos de memória episódica, como mostrado na figura 5, onde se define como se dá o armazenamento, gerenciamento e recuperação de informações na memória episódica (DERBINSKY; LAIRD, 2009), sendo frequentemente usadas como ferramentas de teste e demonstração de teorias acerca da estrutura de tal memória ou mesmo de métodos computacionais eficientes baseados em modelos já existentes (LI *et al.*, 2015).

Conforme sugerido por Nuxoll e Laird (2007) e Castro e Gudwin (2013), a capacidade de memória episódica pode aprimorar significativamente as habilidades cognitivas de um agente inteligente, podendo trazer contribuições aos processos de percepção, aprendizagem, planejamento, tomada de decisões, seleção e execução de ações. Por exemplo, na percepção, o *recall* de episódios pode auxiliar no processo de detecção de situações repetidas. Além disso, a recuperação de um episódio pode trazer elementos importantes que tendem a se repetir (e que funcionam como predições perceptuais).

Na aprendizagem, a memória episódica pode auxiliar na revisão de experiências vividas, de tal forma que algo possa ser aprendido a partir de experiências positivas e negativas. Comparando múltiplos eventos simultaneamente, o sistema de aprendizagem

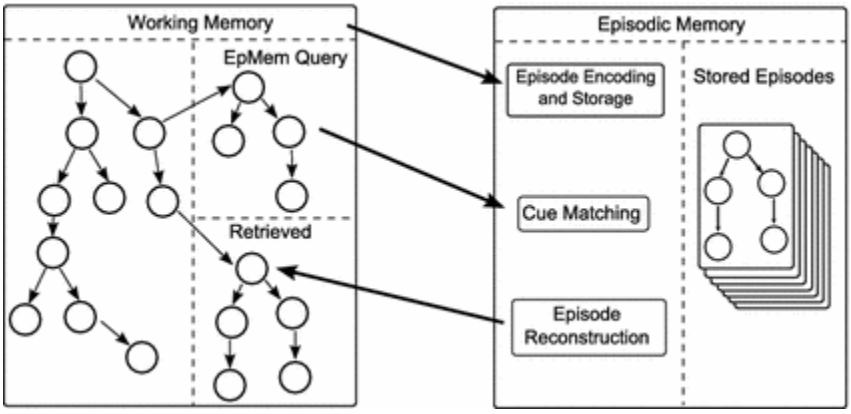

Figura 5 – Relação entre memória de trabalho e memória episódica na arquitetura do SOAR

tem a possibilidade de generalizar o conhecimento vivenciado. Da mesma maneira, é possível recuperar situações de falha e/ou sucesso, o que pode ser útil futuramente nos processos de planejamento e tomada de decisões. Esse mesmo processo de planejamento pode ser auxiliado, uma vez que a recuperação de episódios do passado pode permitir a predição dos resultados de diferentes ações. Pode-se tanto revisar o próprio passado do agente, como de outros agentes que possam ter sido observados. Decisões empregadas no passado podem se mostrar úteis para resolver situações no futuro. Decisões que não tiveram sucesso podem ser evitadas.

Na seleção e execução de ações, a memória episódica pode ser utilizada para se efetuar o *tracking* dos progressos obtidos até o momento na execução de um plano, e com isso propiciar o gerenciamento de metas de longo prazo. Utilizando-se a memória episódica, o sistema pode saber quais partes específicas de um plano já foram executadas, de tal forma que o algoritmo de seleção de ação possa definir quais os próximos passos a serem tomados na execução de um plano de longo termo.

Por fim, uma memória episódica permite ao agente o desenvolvimento de um senso de identidade, uma vez que os episódios criam aquilo que poderia ser visto como a história pessoal de um indivíduo. Essa história pessoal envolve os eventos que foram conscientemente percebidos pelo agente, bem como aqueles em que ele mesmo foi o protagonista. Particularmente o uso que nos interessa nas memórias episódicas é, conforme apontado por Castro e Gudwin (2013), o modo pelo qual a recuperação de episódios pode ser usada para fomentar o aprendizado (ao permitir a recuperação de experiências passadas), o planejamento e a tomada de decisão (provendo uma estimativa do resultado de um possível curso de ações), bem como a seleção e execução de ações (ajudando no monitoramento de objetivos de longo prazo).

2.6 Trabalhos Correlatos

Nesta seção, apresentamos alguns trabalhos presentes na literatura que possuem correspondência com alguns dos aspectos envolvidos nesta dissertação e que de algum modo podem servir de ponto de comparação, tais como Memória Episódica, Aprendizado ou mesmo o ambiente utilizado, o Minecraft.

2.6.1 Arquiteturas Cognitivas e Memória Episódica

Mesmo sendo relativamente pouco explorada no design de agentes inteligentes, a sinergia teórica entre os dois temas talvez torne a combinação de Arquiteturas Cognitivas e Memória Episódica a mais promissora entre as duas aqui apresentadas. Ainda que não exista uma teoria dominante sobre o funcionamento interno deste sistema de memória, alguns trabalhos vêm sendo feitos que exploram este aspecto da mente.

John Laird (LAIRD, 2012b), autor da já mencionada arquitetura SOAR, abstém-se de tentar propor modelos detalhados de como a Memória Episódica funciona em humanos. Ao invés disso, foca em obter os aspectos computacionais que melhor beneficiariam sistemas inteligentes. Assim, nesta arquitetura, o foco principal se dá em fornecer uma memória simbólica que suporte a recuperação de informação com base em informação parcial (um *cue*) e uma representação autobiográfica do agente. Especificamente, o SOAR trata cada Episódio como um retrato do estado da Memória de Trabalho num dado instante de tempo.

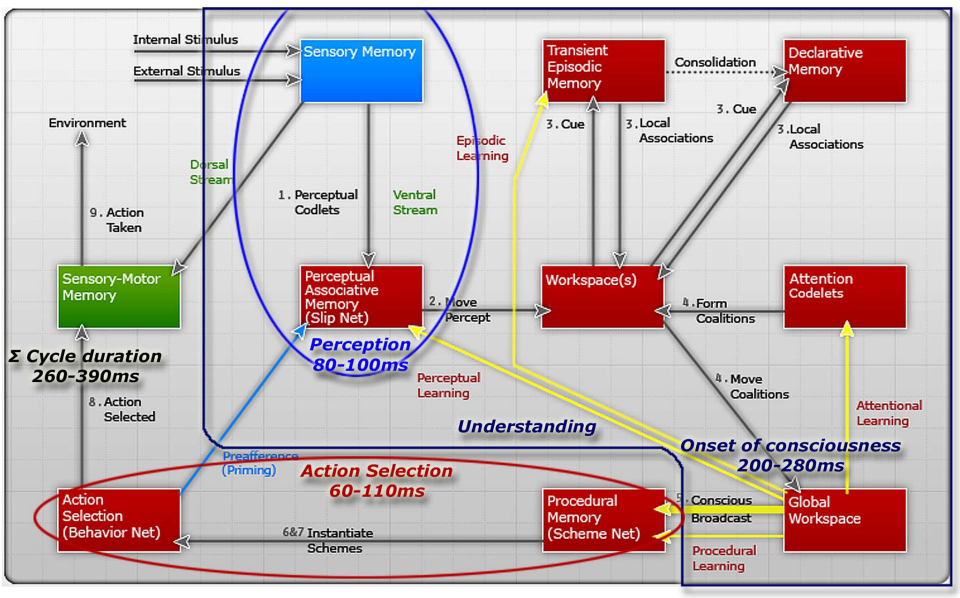

Figura 6 – Estrutura do LIDA. Note a relação entre Memória Episódica Transiente e Declarativa. Extraída de (MADL *et al.*, 2011)

A arquitetura LIDA, mostrada na figura 6, por sua vez, utiliza um conceito corre-

lato, porém distinto, o conceito de Memória Episódica Transiente, um sistema auxiliar de aprendizado a outros sistemas. Nessa proposta, a Memória Episódica Transiente captura a memória onde episódios são formados, com a finalidade de que tais episódios possam ser usados na Memória Declarativa. Este processo inicia-se com o armazenamento de informações na Memória Episódica Transiente via comandos vindos do Espaço de Trabalho Global. Então, para a passagem para a Memória Declarativa, um sistema de longo termo, é necessário o mecanismo de consolidação de memória. Apesar de esse mecanismo não ter seu funcionamento interno completamente entendido no cérebro humano, acredita-se que ele ocorra dentro de determinados períodos de horas ou dias. A teoria do LIDA propõe um tempo fixo, possivelmente equivalentes a ciclos de sono em humanos, no qual o conteúdo que permaneceu na Memória Episódica Temporária (aquele que não decaiu a ponto de deixá-la) é transferido para a Memória Declarativa (RAMAMURTHY; FRANKLIN, 2011).

Segundo Ron Sun em sua arquitetura CLARION (SUN, 2003), a Memória Episódica é antes de tudo um sistema que armazena experiências orientadas à ação, envolvendo estímulo, resposta e consequência, juntamente com a informação de tempo de quando o evento ocorreu. Assim, esse sistema de memória é intrinsecamente ligado ao Sistema Centrado em Ações (ACS, na sigla em inglês). Porém, ainda na teoria do CLARION, toda associação passada pelo ACS para o Sistema Não Centrado em Ações (NACS, na sigla em inglês), como também toda informação inferida pelo próprio NACS, gera um elemento na Memória Episódica. Assim como o LIDA, o CLARION prevê um decaimento temporal dos elementos presentes neste sistema de memória e a eliminação dos itens caso seus valores de ativação estejam abaixo de um determinado limiar. É ainda dito que o uso destes episódios pode auxiliar o aprendizado da mesma forma que qualquer outra experiência. Em termos de implementação, um Episódio nesta arquitetura consiste em uma estrutura com estado, novo estado, ação e reforço.

Diferente das propostas anteriores, o CLARION ainda conta com a Memória Episódica Abstrata, que resume as informações presentes na ME tradicional, permitindo que parte do conteúdo da ME esteja disponível ao uso sem a necessidade de examinar cada episódio individualmente. A MEA é modelada segundo duas redes de distribuição de frequência, a Rede de Frequência de Ação e a Rede de Frequência de Estados. A primeira mapeia cada estado para uma distribuição de frequência de ações e a segunda mapeia os pares de estado-ação em uma distribuição de frequência de sequência de estados e frequência de reforços imediatos. Uma visão geral da arquitetura CLARION pode ser vista na figura 7.

Outra abordagem presente na literatura é aquela apresentada por Ménager (2016) em sua arquitetura ICARUS. Nela, o módulo de Memória Episódica, que pode ser vista na figura 8, é uma estrutura composta por um *cache* episódico, uma árvore de generalização

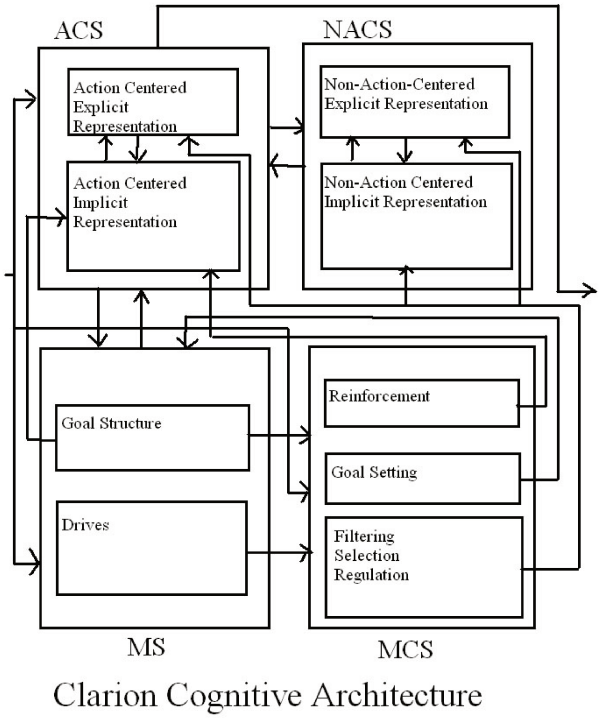

Figura 7 – *Framework* da arquitetura CLARION. Adaptado de (SUN, 2007)

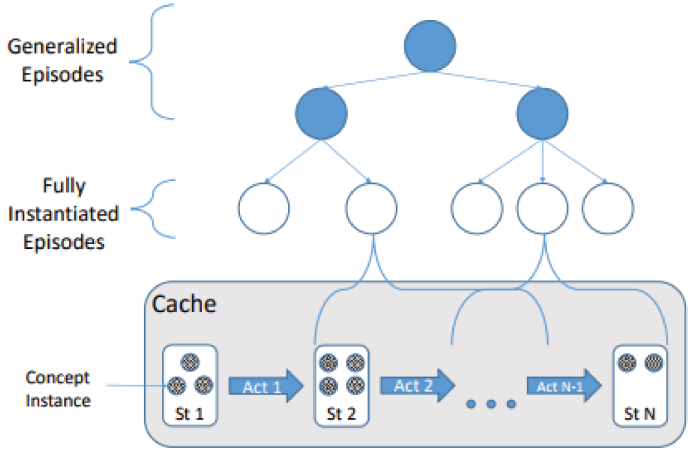

Figura 8 – Esquema de *cache* e Árvore de Generalização do Módulo de Memória Episódica presente na Arquitetura ICARUS. Adaptado de (MÉNAGER, 2016)

e uma árvore de frequência de conceitos. O processo de incorporação de episódios inicia-se no *cache*. Uma vez que os conceitos importantes de um episódio candidato tenham sido identificados, a arquitetura busca, por similaridade, encontrar algum episódio na Árvore de Generalização que assemelhe-se ao candidato. Caso encontre, o novo episódio é inserido na sub-árvore correspondente, caso contrário, ele é inserido como um nó no mesmo nível hierárquico dos Episódios dissimilares. Assim, a hierarquia dos episódios é posta de modo

a ocorrer uma aumento de especificidade conforme navega-se do nó raiz para as folhas.

Além da proposta de modelos, é possível ainda encontrar aplicações que põem à prova ambos os conceitos no campo da robótica (KAWAMURA *et al.*, 2005; KUPPUSWAMY *et al.*, 2006).

2.6.2 Arquiteturas Cognitivas e Aprendizado Instrumental (ou por Reforço)

Devido ao fato de já estarem mais bem estabelecidas, tanto em teorias como computacionalmente falando, e por representarem métodos mais diretos de adaptação comportamental, a presença de técnicas de Aprendizado por Reforço é mais marcante em estudos com arquiteturas cognitivas do que o par comentado na subseção anterior.

Dentre as arquiteturas que explicitamente incluem aprendizado por reforço em seu *core*, talvez a mais famosa seja a já citada SOAR, que possui um mecanismo interno de *Q-learning* embutido.

No SOAR, cada elemento representando um estado também possui uma estrutura de recompensa. Para permitir flexibilidade, essa recompensa deve ser ajustada via uso de regras específicas, podendo tanto copiar entradas externas como possuir regras próprias de atualização.

Ainda na mesma linha, tem-se Sandamirskaya e Burtsev (2015) com o NARLE e Chalita *et al.* (2016), que tem nesta forma de aprendizado grande importância para suas arquiteturas. Apesar disso, existem diversos estudos que incluem tal método em arquiteturas preexistentes, como em (PYNADATH *et al.*, 2014) e sua aplicação com arquitetura SIGMA para negociação entre dois agentes. Já o Aprendizado Instrumental vem sendo empregado com muito sucesso em aplicações com robôs como nos já citados (HONG-GE; XIAO-GANG, 2009; JING *et al.*, 2015; CAI; RUAN, 2009)

2.6.3 Agentes Inteligentes e o Minecraft

Como será mostrado no capítulo 4, o Minecraft vem sendo amplamente utilizado para experimentação acadêmica associada a inteligência artificial e agentes autônomos. Em sua maioria, essas aplicações lidam com o assim chamado *Deep Reinforcement Learning*, um esquema de aprendizado por reforço que utiliza uma rede neural profunda com imagens como entrada, como aproximador de valor.

Sendo talvez o trabalho que representa uma base para outros envolvendo o tema, Barron *et al.* (2016) apresentam uma técnica de Aprendizado por Reforço Profundo para navegação no ambiente 3D do Minecraft. No artigo, estão presentes testes com duas arquiteturas (duas e 16 camadas convolutivas) de rede diferentes, testes de estimativa de distância baseado puramente na imagem e teste exemplo com tarefas simples. Imagens

com exemplos de testes de estimativa de profundidade podem ser vistas na figura 9.

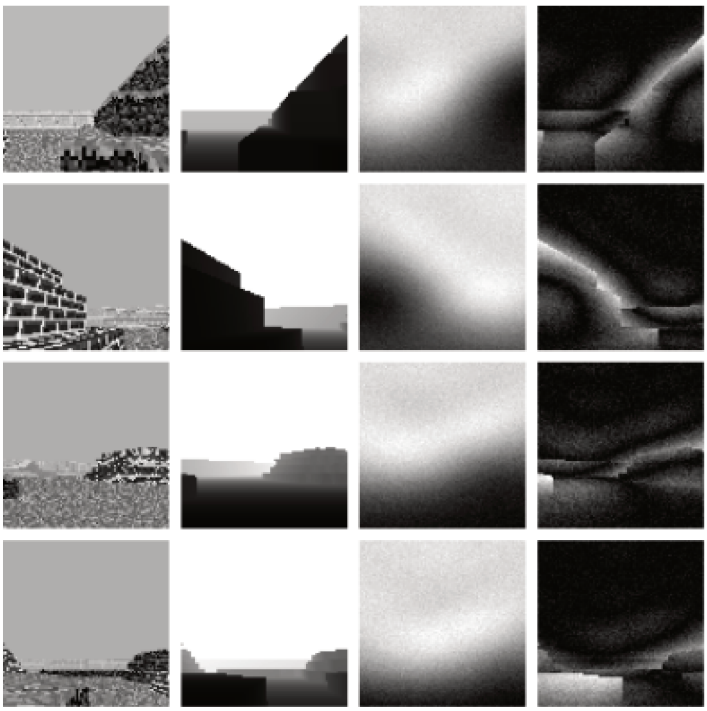

Figura 9 – Exemplos de teste de predição de profundidade. A primeira coluna mostra a imagem original em escalas de cinza. A segunda mostra as distâncias reais da imagem (quanto mais escuro mais próximo do agente). A terceira coluna mostra a predição de distância feita pelo agente e a quarta coluna mostra a superfície de erro, onde quanto mais escura a região, menor o erro. Extraido de (BARRON *et al.*, 2016)

Na Área de Aprendizagem profunda existem ainda diversos outros trabalhos que utilizam o Minecraft como ambiente virtual. Tessler *et al.* (2016), por exemplo, utilizam o Minecraft como plataforma de experimentação e propõem uma abordagem hierárquica, mas ainda dentro de Aprendizagem Profunda. É proposta em seu trabalho uma implementação de um esquema de aprendizado por reforço profundo com hierarquia (H-DRLN) no qual o conhecimento adquirido em uma tarefa é utilizado para o aprendizado de outras tarefas relacionadas, como ilustrado na figura 10. Para tal, são necessários uma retenção eficiente de conhecimento sobre as tarefas aprendidas, uma transferência seletiva do conhecimento adquirido e um bom grau de interação entre retenção e transferência. A estrutura de sua abordagem pode ser vista na figura 10. Como alternativa ao Aprendizado por Reforço, Bonnano *et al.* (2016) apresentam uma proposta para o aprendizado e seleção de subobjetivos de técnicas conhecidas de Aprendizado de Máquina ou diretamente dos *pixels*, utilizando redes bem estabelecidas, como a AlexNet, com pouco treinamento. Ainda em Bonnano *et al.* (2016), é apresentada uma implementação de uma Rede de Tarefa-Objetivo (Goal-Task Network), um tipo de mecanismo que decompõe tarefas complexas em subobjetivos.

Oh *et al.* (2016) propõem um mecanismo de Aprendizado por Reforço Profundo Baseado em Memória que visa suprir necessidades presentes em técnicas mais usuais de Aprendizado por Reforço Profundo. Nessa proposta, Oh *et al.* (2016) utilizam uma arquitetura que mantém um histórico recente de observações (memória), bem como um vetor de contexto usado tanto no processo de recuperação de informação quanto no processo de estimativa de valor para as ações. Para a estimativa desses valores, é utilizada uma rede neural MLP que recebe duas entradas: uma entrada de contexto (h), baseada diretamente em um produto vetorial entre dados devidamente codificados e uma matriz de pesos, e uma entrada que representa uma recuperação de eventos similares da estrutura de memória (o). A rede em questão é regida pela seguinte equação:

$$g_t = f(W^h h_t + o_t), q_t = W^q q_t$$

Ainda em Aprendizagem Profunda, Jaderberg *et al.* (2016) propõem uma estrutura de aprendizado por reforço que maximiza também valores de pseudo-recompensa associados a sinais de outras origens, muito comuns em ambientes complexos. Fontes de recompensa extrínseca costumam ser inexistentes em aplicações ligadas a situações que emulem o mundo real e, mesmo que presentes, as recompensas intrínsecas (ligadas a condições do agente, por exemplo) ainda estão presentes em abundância. Assim, o uso desses sinais intrínsecos como auxiliares para determinar a dinâmica do ambiente e prever recompensas mostra-se proveitoso no tocante ao aprendizado do agente.

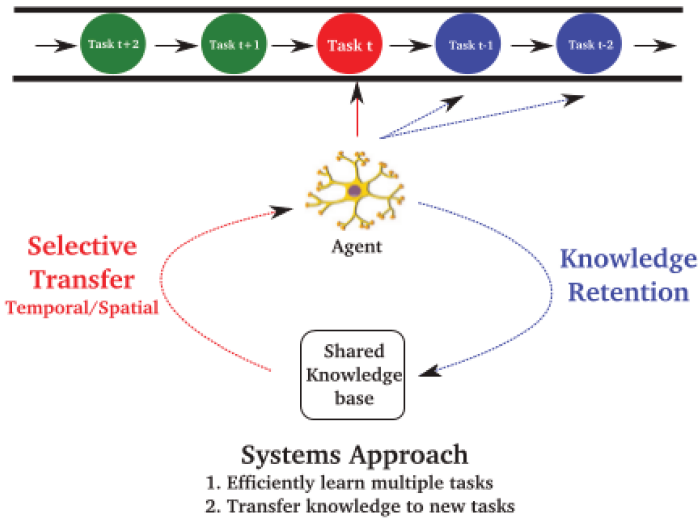

Figura 10 – Estrutura de abordagem sistêmica que retem e reutiliza conhecimento em tarefas diferentes daquelas de onde foram provenientes. Adaptado de (TESSLER *et al.*, 2016)

Siljebråt (2015), por sua vez, busca fazer uma conexão entre os métodos de Aprendizado por Reforço na computação e evidências neurobiológicas da literatura. Além des-

sas conexões, seu trabalho busca investigar como a adição do conceito de comportamento (animal) inato pode aprimorar os resultados de técnicas mais puras de Aprendizado por Reforço. Esses comportamentos inatos influenciam a tomada de decisão basicamente com a adição de condicionais que priorizam certas ações em detrimento de outras, algo similar aos *Codelets* comportamentais, que estão presentes na proposta de arquitetura. O algoritmo utilizado pode ser visto na figura 11

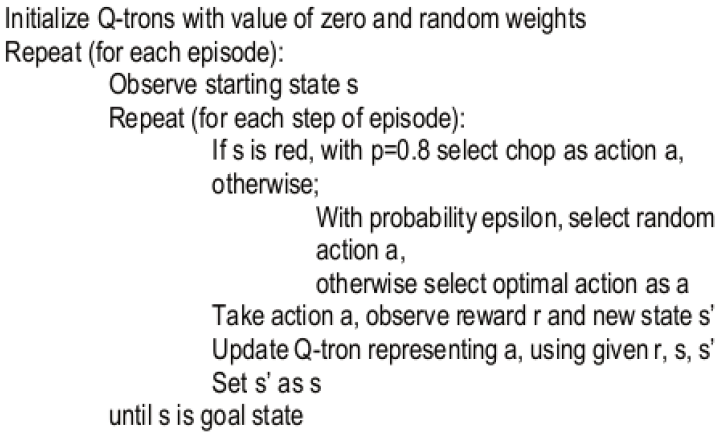

Figura 11 – Algoritmo de Aprendizado por Reforço com comportamento inato utilizado por (SILJEBRÅT, 2015)

Udagawa *et al.* (2016) apresentam um método de Aprendizado por Reforço Profundo com poucas ações que, puramente baseado nas imagens que recebe, permite que um agente dentro da plataforma Malmo aprenda a tomar ações que levam ao combate de criaturas hostis. Mirowski *et al.* (2016) mostram uma abordagem para navegação de um agente em um ambiente 3D dinâmico, onde o objetivo, conhecido do agente, muda constantemente de posição. Para tanto, são utilizados como auxiliares mecanismos de predição de profundidade e de laços de posição (isto é, prever estados repetidos de posição). Seu estudo mostra que o método atinge níveis humanos de performance.

Como um adicional a *frameworks* que utilizam Aprendizagem Profunda, a proposta de Sharma *et al.* (2017) estabelece um método que permite ao agente selecionar de modo mais refinado a repetição de suas ações, definindo, por exemplo, a escala de tempo que se dará essas repetições.

Abel *et al.* (2016) utilizam um esquema modificado de aprendizado por reforço com horizonte finito onde existe uma atualização de valor Q baseada na escolha de uma função `h` dentre um conjunto de regressores que minimize o valor da resultante da modificação da equação 2.3.3 para a inclusão do termo `h`. No caso da aplicação envolvendo o Minecraft, o agente navega com dados puramente visuais.

Fugindo um pouco de Aprendizagem Profunda, Denoyelle (DENOYELLE *et al.*, 2016) aplica o seu *Brainy-Bot* na tarefa de controle de uma criatura virtual. Essa proposta

apresenta explicitamente conceitos de Memória Associativa e de Categorização de Estados, isto é, a capacidade de extrair informação simbólica a partir de valores quantitativos e qualitativos.

Aluru *et al.* (2015) propõem em seu trabalho tanto uma plataforma para experimentação baseada no Minecraft anterior ao Malmo (BURLAPCraft) quanto um software para controle de agentes inteligentes para a validação do ambiente. Nos experimentos realizados, foi utilizado um esquema de aprendizado por reforço para a assimilação de conhecimento em tarefas de caminho mínimo e de aprendizado de linguagem.

2.7 Resumo do Capítulo

Neste capítulo, foi apresentada a fundamentação teórica de diversas técnicas utilizadas na proposta de nossa arquitetura, conforme relatadas na literatura. Primeiramente foi dada uma introdução sobre agentes inteligentes. Em seguida, falou-se sobre Redes Neurais, mais especificamente sobre Redes Não-Recorrentes e seus principais usos. Falou-se ainda sobre Aprendizado por Reforço e Aprendizado Instrumental. Em sequência, apresentamos os tópicos de Arquiteturas Cognitivas, uma breve introdução sobre Teoria do Processamento Dual e Memória Episódica. Então, uma seção de trabalhos correlatos encerra o capítulo. No próximo capítulo, complementamos essas técnicas com outras técnicas utilizadas em nosso trabalho, mas que foram desenvolvidas pelo nosso grupo de pesquisa, e que portanto merecem um capítulo em separado.

3 O CST (Cognitive Systems Toolkit)

3.1 O Toolkit e a Arquitetura de Referência

O *Cognitive Systems Toolkit* (CST) é um toolkit para o desenvolvimento e construção de arquiteturas cognitivas sendo desenvolvido pelo nosso grupo de pesquisa na Universidade Estadual de Campinas, e que foi utilizado neste trabalho para a construção da arquitetura cognitiva proposta. A concepção do CST envolve ideias oriundas de diversas arquiteturas cognitivas, tais como Clarion (SUN, 2015) e LIDA (FRANKLIN *et al.*, 2014). Diversos conceitos apresentados originalmente nessas arquiteturas foram aproveitados no CST. O objetivo principal do projeto CST é permitir a construção de diferentes arquiteturas cognitivas, incorporando os modelos cognitivos mais interessantes encontrados nas diversas arquiteturas cognitivas existentes, sem a necessidade de que o projetista seja obrigado a optar forçadamente por uma única arquitetura cognitiva, uma vez que uma grande parte das arquiteturas cognitivas é monolítica e obriga muitas vezes que um desenvolvedor adote certos modelos cognitivos, pois eles estão engessados a outros que poderiam ser desejáveis. Em sua estrutura de toolkit, a proposta do CST é que o projetista possa escolher os modelos cognitivos que achar importantes, integrando-os em uma arquitetura personalizada e flexível. Dessa forma, o CST impõe um conjunto mínimo de restrições sobre as arquiteturas cognitivas construídas com seu uso.

A figura 12 ilustra os principais conceitos do *Core* do CST, que permeiam todo o desenvolvimento do toolkit. Esse *Core* do CST não deve ser confundido com o toolkit em si. Ele corresponde a um conjunto de princípios arquiteturais, e são utilizados de maneira generalizada em qualquer arquitetura cognitiva desenvolvida com o auxílio do CST. De uma certa maneira, atender essas demandas do *Core* são a única restrição imposta em modelos construídos com o CST. Os conceitos introduzidos no *core* do CST são tão abstratos, entretanto, que podem induzir um leitor desprevenido a julgar o CST como sendo apenas um framework computacional para computação distribuída em geral, com barramentos e memórias locais e globais. Entretanto, de forma a podermos entender as estruturas mais complexas do CST, o leitor deve em um primeiro momento ser capaz de compreender os conceitos do *core*, que constituem os blocos básicos a partir dos quais estruturas mais complexas podem ser construídas dentro do CST (PARAENSE *et al.*, 2016).

Os conceitos mais importantes dentro do *core* do CST são os conceitos de *Codelets* e *Objetos de Memória*. Na histórica arquitetura *Copycat*, desenvolvida por Hofstadter e Mitchell (1994), estes definiram sua visão para uma arquitetura cognitiva como baseada na

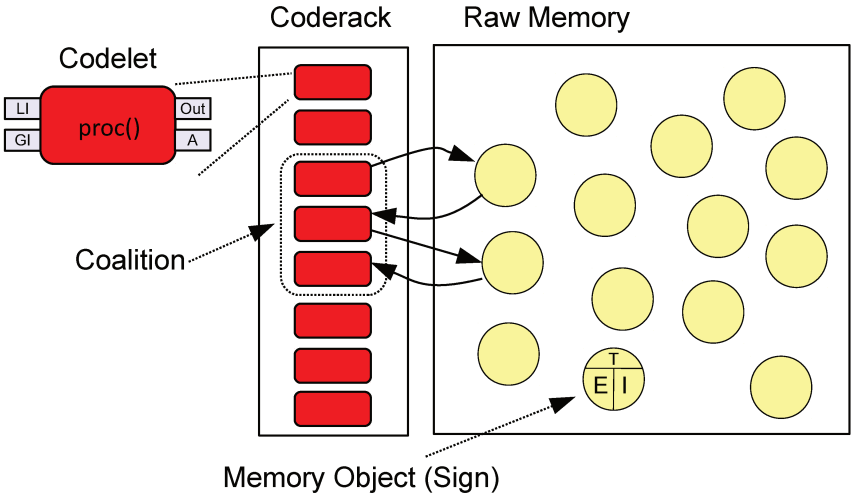

Figura 12 – O Core do CST - adaptado de (PARAENSE *et al.*, 2016)

interação de diversos micro-agentes que eles chamaram de *Codelets*, que eram responsáveis por todo tipo de ações dentro de sua arquitetura. Dessa forma, *Codelets* podem ser vistos como pequenos pedaços de processamento não bloqueante, cada um destes executando uma tarefa simples e bem definida. A melhor abstração para o que é um *Codelet* é pensar em um trecho de código de programa que idealmente é executado de maneira contínua e cíclica, repetidas vezes, sendo responsável por gerar o comportamento de um componente de um sistema rodando em paralelo. Essa noção de *Codelet* foi posteriormente aperfeiçoada por Franklin *et al.* (1998) e usada de maneira disseminada na arquitetura cognitiva LIDA (FRANKLIN *et al.*, 2014). Um dos princípios fundamentais do *core* do CST é que qualquer arquitetura cognitiva construída utilizando-se o CST deve ser orientada a *Codelets*, uma vez que todas as funções ou capacidades cognitivas devem ser implementadas na forma de *Codelets* ou grupos de *Codelets* interagindo entre si. Isso significa que de um ponto de vista conceitual, qualquer sistema implementado com o CST é conceitualmente um sistema paralelo e assíncrono, onde cada componente é modelado por um *Codelet*. Os *Codelets* do CST são implementados de maneira similar aos da arquitetura cognitiva LIDA, e são comparáveis aos processadores de propósito específico descritos na Teoria de Workspace Global de Baars (BAARS, 1988; BAARS; FRANKLIN, 2007).

Dessa forma, para que o sistema funcione, algum tipo de coordenação deve existir entre esses *Codelets*, de tal forma que os mesmos possam formar coalizões, isto é, uma união de dois ou mais *Codelets*, por meio de alguma interação coordenada, e possam dessa forma implementar as funções cognitivas atribuídas à arquitetura. Essa coordenação nos impõe algumas restrições quando queremos implementar o conceito de *Codelets* em computadores seriais. Em uma arquitetura cognitiva baseadas em Codelets, pode haver um grande número de Codelets, alguns deles que são mais rápidos e que necessitam ser chamados de maneira mais frequente, e outros que são mais lentos e que podem ser chamados

de maneira mais espaçada. Dessa forma, algum mecanismo de *scheduling* específico deve ser utilizado para garantir que Codelets mais críticos sejam chamados mais frequentemente e que outros menos críticos possam ser chamados de maneira mais intercalada. Outra restrição em implementações seriais é que, frequentemente, como alguns Codelets podem depender da informação gerada por outros Codelets, a ordem em que os Codelets são chamados é importante. De nada adianta chamar um Codelet se a informação que ele necessita ainda não foi calculada. De forma a contornar essas questões, o CST provê uma opção para se escolher entre o uso de Codelets implementados como threads, ou Codelets implementados como pseudo-threads, onde uma sequência pré-definida para a chamada dos Codelets é instituída.

O segundo elemento vital para a compreensão da infraestrutura do *core* do CST é a noção de *Objeto de Memória*. Um *Objeto de Memória* é um tipo de *container* para informações de ordem genérica, atuando como uma representação interna, responsável por armazenar qualquer informação auxiliar ou perene, necessária para a arquitetura cognitiva realizar seu comportamento.

Codelets e Objetos de Memória estão intrinsecamente acoplados um ao outro, uma vez que os objetos de memória armazenam a informação necessária para os Codelets poderem processar, e ao mesmo tempo armazenam as informações processadas por esses mesmos Codelets. A principal propriedade de um Objeto de Memória é exatamente essa informação (I). Essa informação pode ser simplesmente um número, ou armazenar estruturas complexas como listas, árvores, grafos ou mesmo redes. Na implementação computacional do CST, a informação I é um objeto Java qualquer. o que significa que virtualmente qualquer coisa pode ser representada internamente como um objeto de memória. A opção por usar um objeto Java genérico como a entidade básica de representação de conhecimento foi uma escolha deliberada, de forma a trazer flexibilidade ao usuário do toolkit. Internamente, o toolkit disponibiliza diversos outros mecanismos de representação, como proposições, predicados, regras, da mesma forma que estruturas do tipo rede, tais como redes neurais, redes Bayesianas, ou até mesmo grafos completos como as Redes de Comportamento do LIDA, os WMEs (Working Memory Elements) do SOAR, ou os *Atoms* e *Atom Spaces* do OpenCog (GOERTZEL *et al.*, 2014b). De forma que diferentes módulos em diferentes contextos possam trabalhar conjuntamente, soluções utilizando strings JSON podem também ser utilizadas.

Além da informação I, objetos de memória possuem ainda dois outros tipos de meta-informação: uma *tag* de tempo (T), que marca o objeto indicando sua última atualização, e um parâmetro de avaliação (E - de *Eval*), que pode ser utilizado de diversas formas pelo designer. Esta avaliação é simplesmente um número, que pode ser usado, por exemplo, como um fator apraisivo em um módulo emocional, ou simplesmente um rótulo para diferenciar entre dois objetos de memória de um mesmo tipo. Esta meta-informação

pode ser simplesmente ignorada na maioria dos Codelets, ou ser útil quando se deseja implementar algum tipo de modelo cognitivo mais sofisticado.

Como pode ser visto na figura 12, todos os objetos de memória são armazenados em um *container* chamado de *Raw Memory*. Todos os Codelets do sistema são igualmente armazenados em outro *container* chamado de *Coderack*. Codelets interagem entre si por meio de objetos de memória. Desta forma, um objeto de memória que é uma saída de um Codelet pode ser utilizado como entrada para outro Codelet. Um conjunto de Codelets interagindo entre si devido a um conjunto comum de objetos de memória é chamado de uma *coalizão*. Um Codelet possui duas entradas principais, uma entrada local (LI - Local Input), e uma entrada global (GI - Global Input), que são utilizadas em conjunto para implementar um mecanismo de Workspace Global (BAARS; FRANKLIN, 2007), implementando um modelo cognitivo de consciência, se for necessário.

A entrada local corresponde a uma lista selecionada de objetos de memória do *RawMemory*, onde o Codelet pode acessar a informação necessária para realizar seu processamento *proc()*. A entrada local de um Codelet é sua fonte preferencial e padrão para a entrada de informações. A entrada global é usada pelo Codelet para obter informações a partir do Workspace Global, um subconjunto virtual do *Raw Memory* que está constantemente sendo atualizado por Codelets especiais, denominados Codelets de Consciência, implementando o mecanismo de consciência preconizado por Baars. De acordo com a Teoria do Workspace Global, os Codelets que estão no Workspace Global fazem um broadcast de suas informações produzidas para todos os outros Codelets do sistema, por meio da entrada global. Assim, a informação adquirida por meio da entrada global é variável a cada instante de tempo, e usualmente corresponde a um sumário, um filtro executivo que seleciona a informação mais relevante na memória a cada instante de tempo. As duas saídas de um Codelet são a saída padrão, que é usada para modificar ou criar uma nova informação na *Raw Memory*, e o nível de ativação, que indica a relevância da informação disponibilizada na saída, e é usada pelo mecanismo de Workspace Global (Codelet de consciência), de forma a selecionar a informação a ser direcionada para o Workspace Global.

Ao mesmo tempo que introduzem o toolkit CST, Paraense *et al.* (2016) também descrevem a assim chamada Arquitetura de Referência CST, uma meta-arquitetura que pode ser instanciada de diferentes maneiras para a criação de uma nova arquitetura cognitiva por meio do CST. Essa arquitetura é chamada de arquitetura de referência, pois é uma visão abstrata de como organizar um conjunto de Codelets e um conjunto de memórias, de forma a construir uma nova arquitetura cognitiva compatível com o CST. Nessa arquitetura de referência, os Codelets são organizados em grupos, onde cada grupo é responsável por implementar um modelo cognitivo de uma função cognitiva determinada. O toolkit CST provê implementações padrão para alguns desses grupos. Outros Codelets

podem ser criados pelo arquiteto cognitivo, e facilmente incorporados utilizando-se as funções do *core* do CST. A figura 13 pode ser vista como uma visão refinada da figura 12, ondes grupos de Codelets são indicados. Da mesma forma, a *Raw Memory* é dividida em diversas memórias diferentes, de tal forma que os objetos de memória são divididos por entre essas memórias, responsáveis por armazenar diferentes tipos de conhecimentos.

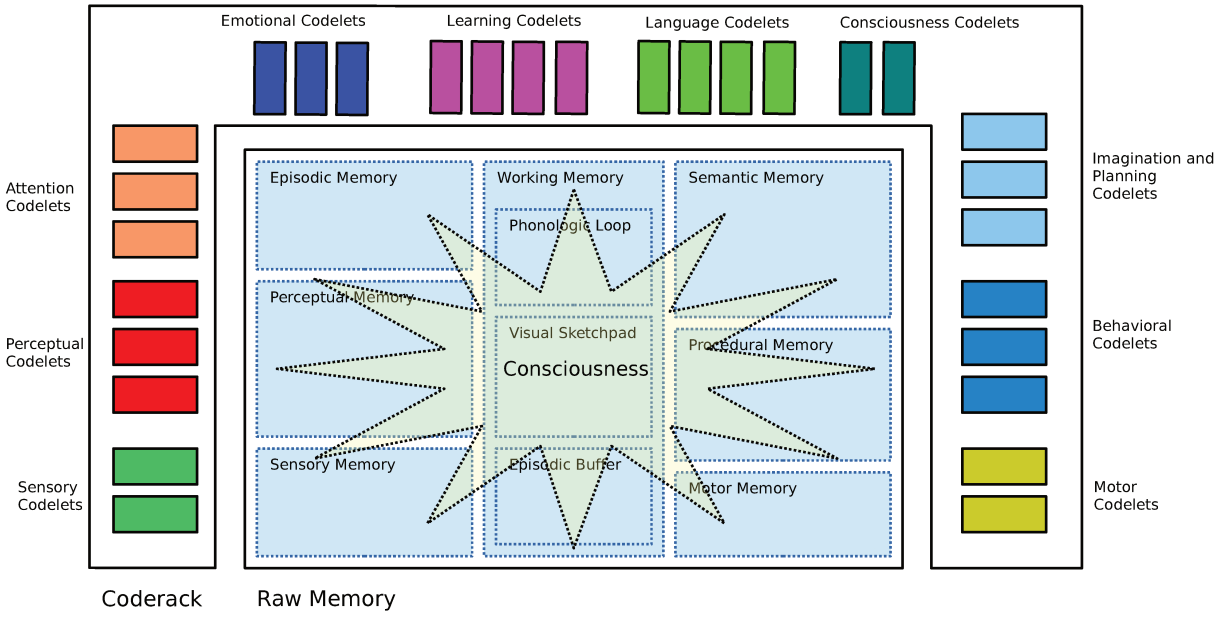

Figura 13 – Visão Geral da arquitetura do CST - adaptado de (PARAENSE *et al.*, 2016)

A *home page* do Projeto CST Project se encontra no endereço <http://cst.fee.unicamp.br>, e seu código fonte está disponível em formato open-source no GitHub: <https://github.com/CST-Group/cst>.

3.2 O MECA (*Multipurpose Enhanced Cognitive Architecture*)

Dentre as arquiteturas e ideias presentes nas referências deste trabalho, a maior de todas as fontes de inspiração foi o trabalho desenvolvido por Gudwin *et al.* (2017). A arquitetura proposta neste trabalho baseia-se fortemente nas ideias apresentadas no MECA.

O MECA, sigla para *Multipurpose Enhanced Cognitive Architecture*, foi construído com muitas ideias vindas da Teoria do Processamento Dual, Subsunção dinâmica (ARKIN, 1998), Espaços Conceituais (GÄRDENFORS, 2014) e Cognição Fundamentada (BARSALOU, 2010), utilizando o CST como *toolkit* base. Utiliza ainda, ideias presentes em diversas arquiteturas bem consolidadas como LIDA, CLARION e SOAR.

Primeiramente, o MECA é uma implementação da Teoria do Processamento Dual, isto é, assim como proposto em Osman (2004), o MECA está dividido em dois grandes

subsistemas, denominados *System 1* e *System 2*. O *System 1* contem 3 tipos de memória: Memória Sensorial, Memória Perceptual e Memória Motora. Essas memórias são acessadas por diversos subsistemas como os subsistemas comportamental, perceptual e motivacional no *System 1*. Já o *System 2* possui em sua estrutura subsistemas de mais alto nível, como o de planejamento, o de expectativas e motivacional do *System 2*. A figura 14 mostra a topologia da arquitetura e explicita os subsistemas e memórias presentes nos *System 1 e 2*.

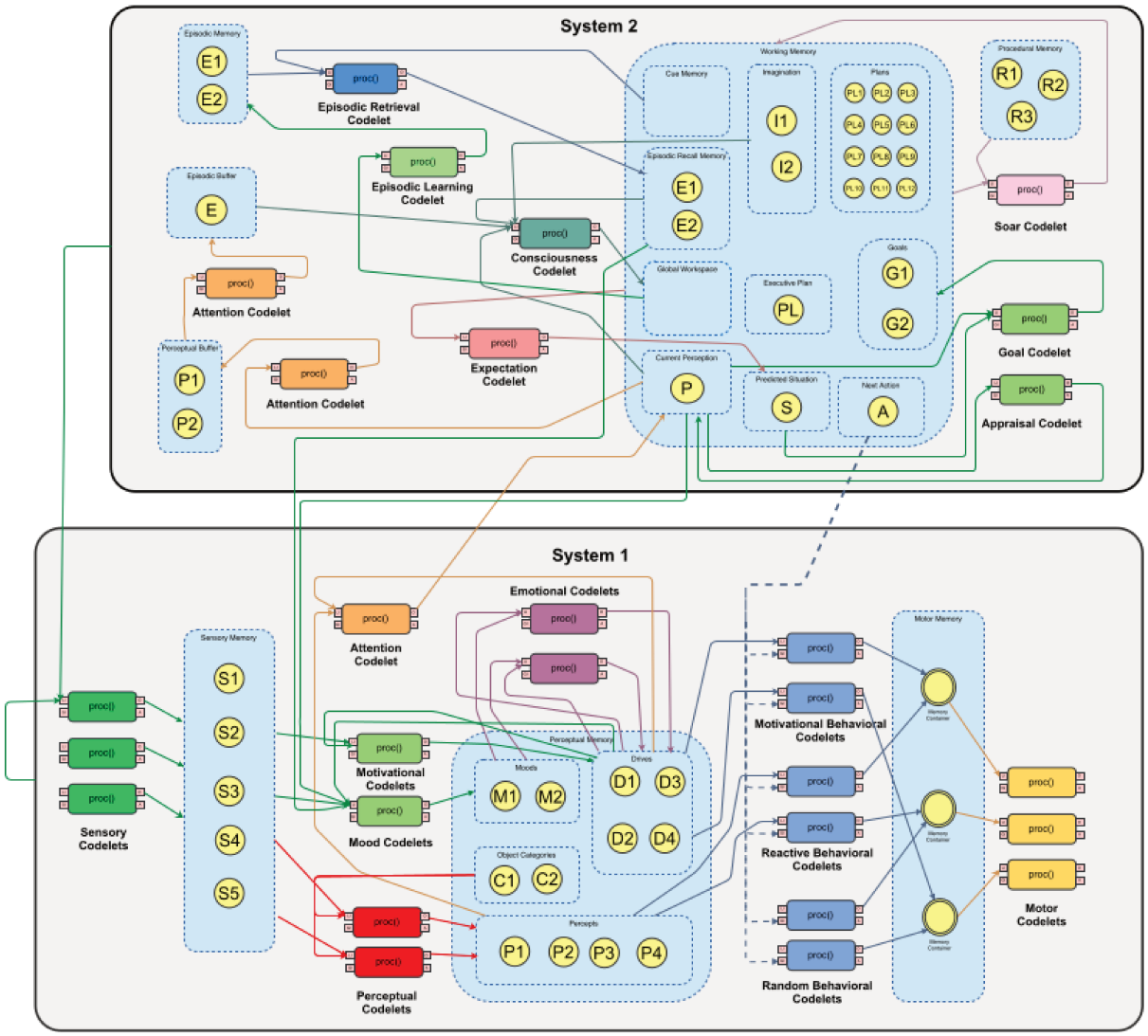

Figura 14 – Visão geral da estrutura da arquitetura MECA. Adaptado de (GUDWIN *et al.*, 2017)

Na arquitetura, o *System 1*, assim como previsto na Teoria de Processamento Dual, exerce o papel de lidar com a parte rápida, inconsciente do processo cognitivo. É principalmente composto por comportamentos reativos, sendo possível também a existência de comportamentos motivacionais, análogos à ideia de "instintos". O design básico deste sistema é uma arquitetura de Subsunção Dinâmica. As entradas pra esse mecanismo

podem vir diretamente da memória sensorial, mas usualmente passam por algum tipo de tratamento perceptual, isto é, a geração de estruturas de Percepts, que correspondem a abstrações do dados sensoriais, que podem ser utilizados em outras partes do MECA. Em especial, no *System 1*, esses Percepts são utilizados pelos *Codelets* comportamentais. Dentre esses *Codelets*, existe um subgrupo especial, os *Codelets* de Comportamento Motivacional, que integram neles o subsistema motivacional do *System 1*. Os detalhes de implementação do *System 1* podem ser encontrados em (GUDWIN *et al.*, 2017).

O *System 2*, ainda segundo a Teoria de Processamento Dual, é responsável pelo lento processo consciente de raciocínio deliberativo. Ele é majoritariamente um processo sequencial baseado em regras, operando com símbolos e considerando não apenas o presente, mas também o passado e o futuro. É nesse sistema que ocorrem os processos de planejamento, responsável por simular o futuro e executar planos de ações visando atingir objetivos de alto nível (*Goals*), e imaginação, utilizado como auxiliar para testar possíveis cursos de ações e avaliar suas pertinências. É nesse Sistema também onde, através do subsistema de Expectativas, ocorrem tentativas de previsão do futuro a curto prazo e aprendizado a partir de possíveis divergências entre o previsto e o atingido. Os detalhes de especificação deste sistema também podem ser encontrados em (GUDWIN *et al.*, 2017).

3.3 Ontologia de Representação de Conhecimento

Para a representação adequada do conhecimento acerca do ambiente, seus elementos constituintes (incluindo elementos não fixos, como o próprio agente) e suas propriedades foi usada a ontologia representacional OWRL (sigla, em inglês para *Object World Representation Language*) apresentada em (GUDWIN *et al.*, 2017). Tal uso foi motivado pela adoção de uma linguagem de representação padrão que pode ser utilizada em outros trabalhos como forma de fornecer suporte para possíveis interações entre diferentes programas e dispositivos que utilizem a mesma linguagem.

De acordo com a ontologia do OWRL (vide figura 15), o ambiente é representado como um mundo composto por *AbstractObjects*. Esses Objetos podem apresentar-se como formas primitivas que constituem uma unidade, partes integrantes de um outro *AbstractObjects* ou mesmo como um conjunto de partes agregadas, que permitem que um coletivo de objetos possa ser tratado como um *AbstractObject*. Nessa descrição, objetos primitivos são descritos por um conjunto de *Properties* e eventualmente *Affordances*, ou seja, possíveis ações que podem ser realizadas sobre os objetos. Os objetos, vistos como partes de outros objetos, podem eles mesmos possuir subdivisões com suas próprias partes integrantes e agregadas. Uma propriedade é definida por uma ou mais *Quality-Dimension*, descritores ligadas a uma visão mais específica da propriedade. Por exemplo, a propriedade "Posição"pode ter as *Quality-Dimension* "X"e "Y"como constituintes que, apesar de

possuírem significados distintos (valores no eixo X e Y respectivamente) pertencem ao mesmo conceito de "posição no plano". *Affordances* por sua vez modelam possibilidade de ações sobre um determinado *AbstractObject* e permitem que estes objetos possuam uma certa dinâmica temporal, implicando uma dinâmica em alguma mudança em uma ou mais Propriedades, ou que novas partes integrantes ou agregadas sejam criadas ou destruídas. O instante da realidade capturado pela percepção de um agente é visto como uma *configuração*, isto é, um *AbstractObject* que reúne todos os objetos do ambiente percebidos, assim como suas partes agregadas em um determinado instante de tempo, de maneira análoga a uma "foto"do ambiente, que pudesse ser tirada em um determinado instante de tempo.

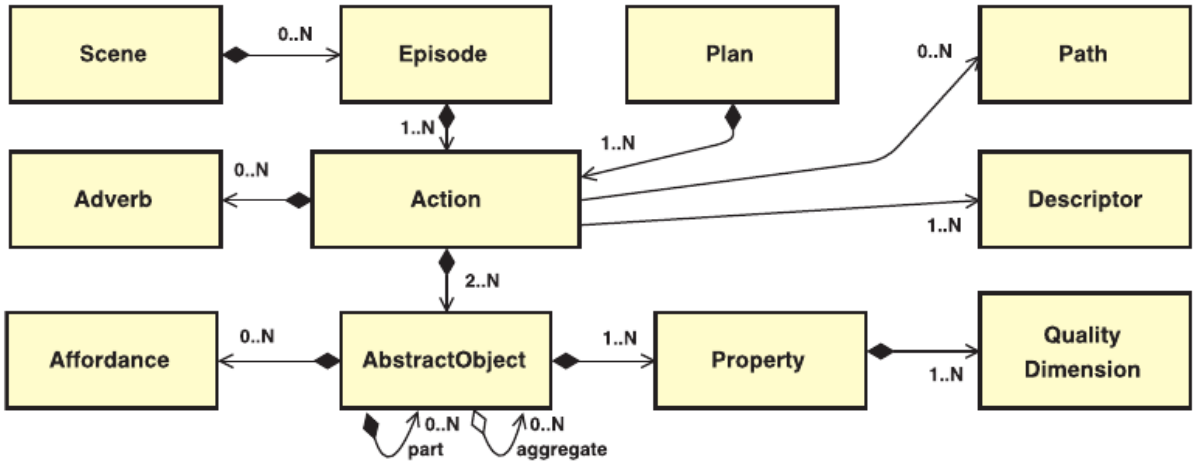

Figura 15 – Representação em UML dos conceitos do OWRL - retirado de (GUDWIN *et al.*, 2017)

Pode-se, dentro desta representação de mundo, ilustrar a passagem de tempo como uma sequência de *configurações*, amostradas periodicamente a uma determinada taxa. Uma sequência de *configurações* com mudanças significativas entre si é tida como uma Ação que, alternativamente, pode ser representada por um Descritor ou por um *Path*. Um Descritor é uma lista de comandos em OWRL do qual pode-se obter uma certa Configuração final a partir de uma Configuração inicial. Ações podem possuir modificadores, modelados nesta ontologia como Advérbios. Uma sequência de ações pode definir um Episódio e, de acordo com a situação, pode ocorrer de muitos Episódios estarem em curso ao mesmo tempo. Vê-se ainda a união de todos esses Episódios como uma Cena e o Plano como um simples sequenciamento de Ações. Note que Ação dentro deste contexto pode ir desde uma atuação elementar, como ativar um atuador, até aquelas com descrições mais elaboradas como “ir da posição atual até o objeto X”, que envolvem várias ações menores.

3.4 Resumo do Capítulo

Neste capítulo, apresentamos um conjunto de ferramentas sendo desenvolvidas pelo nosso grupo de pesquisa no DCA-FEEC-UNICAMP, que têm um impacto significativo neste trabalho, o CST e o MECA, respectivamente o *toolbox* utilizado para construir a Arquitetura proposta e a principal inspiração para o trabalho. Aqui, as bases sobre a qual o projeto sustenta-se (como orientação a *Codelet*) foram apresentadas e tratadas com certo detalhamento. Ao final, apresentamos a ontologia dos tipos de conhecimento que podem ser representados internamente no MECA, e que utilizamos também em nosso trabalho. No próximo capítulo, começamos a apresentar os detalhes principais do nosso experimento, e da arquitetura de controle que desenvolvemos neste trabalho.

4 Materiais Utilizados

4.1 Minecraft e Project Malmo

Com lançamento da versão completa em Novembro de 2011, o *Minecraft* é um jogo eletrônico 3D de jogabilidade não-linear, isto é, não apresenta enredo ou objetivos fixos, com alto grau de interação com o ambiente permitindo, por exemplo, ao jogador interagir com quaisquer elementos no mundo e construir estruturas e objetos utilizando recursos disponíveis ao redor, sejam itens que o personagem pode carregar ou construções fixas, como casas. O jogo ainda dispõe de elementos que tornam o mundo virtual mais interativo e interessante, como animais, monstros, climas e terrenos diferentes, ciclos de dia e noite e interação com outros personagens, sejam eles outros jogadores ou *bots*. Uma imagem que exemplifica a visão do jogador neste ambiente é mostrada na figura 16.

Figura 16 – captura de tela do jogo *Minecraft*. Notam-se os gráficos minimalistas, um contraponto à liberdade de ações presente no ambiente.

Devido à proposta inovadora - um jogo de interação livre onde o jogador lida com o jogo em primeira pessoa - o Minecraft se tornou rapidamente popular entre pessoas de todas as idades, sendo palco inclusive de competições internacionais de construção de estruturas temáticas. Mas a popularidade do jogo não ficou restrita apenas ao entretenimento: dadas as suas características o jogo é uma grande oportunidade para experimentos com agentes inteligentes pois desonera o pesquisador de lidar diretamente com o mundo real (que geralmente envolve custos e cuidados significativos) ou mesmo de desenvolver ele mesmo ambientes virtuais, poupando tempo e dispondo de comunidades de alcance mundial às quais pode recorrer em momentos de dificuldades.

Neste contexto de experimentação de sistemas inteligentes, a Microsoft lançou

em Julho de 2016 uma modificação do jogo intitulada de *Project Malmo* que conta com diversas facilidades para a interação com o Minecraft, constituindo uma conveniente camada de abstração e facilitando, assim, a implementação de tais sistemas (JOHNSON *et al.*, 2016). Entre as facilidades oferecidas pela plataforma, incluem-se métodos de controle da criatura (mover, rotacionar, pular etc.) e informações retornáveis em formatos padronizados, a exemplo do formato JSON, acerca do ambiente no qual o agente está imerso, podendo, inclusive, apresentar diferentes graus de especifidades, indo dos *pixels* puros conforme exibidos em tela até detalhes completos de áreas inteiras centradas no personagem. A plataforma oferece suporte a diversas linguagens de programação como C, Python, Lua e Java. Na figura 17 é mostrado um *screenshot* de uma aplicação utilizando a plataforma.

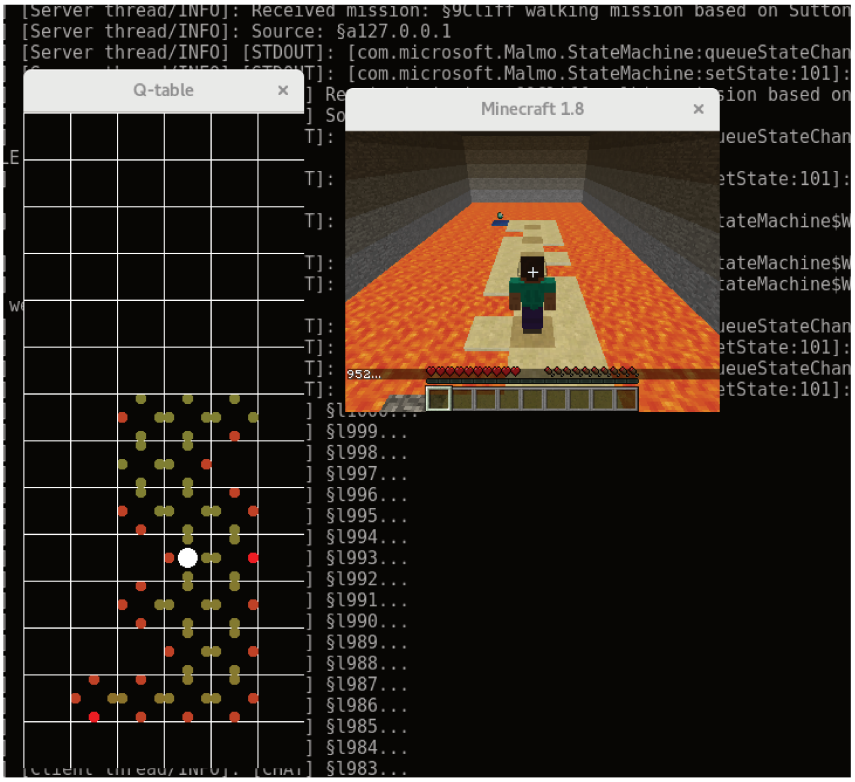

Figura 17 – Exemplo de aplicação de aprendizado por reforço usando a plataforma Malmo. O agente em questão aprendeu a navegar corretamente por um terreno hostil.

Dentre os diversos recursos disponíveis ao programador destacam-se quatro categorias de especial interesse neste trabalho e em outros que envolvam sistemas inteligentes. Tais recursos encontram-se descritos com maiores detalhes nas sub-seções a seguir.

4.1.1 *Mission* e *Mission Record*

Dentro do vocabulário utilizado pela Plataforma Malmo para definir objetos, ações e informações, uma Missão significa um conjunto bem definido de configurações

como tempo total, ações permitidas, recompensas e suas condições, condições de término, ambiente, etc., para uma dada execução do programa. Assim, o Malmo fornece ferramentas prontas para construir experimentos, desonerando em parte o designer do programa.

Um aspecto muito vantajoso de se utilizar o Minecraft/Malmo como ambiente virtual é a possibilidade de exercer alto controle sobre a configuração dos experimentos, desde o subconjunto de ações permitidas (dentre todas as ações que existem no jogo) até a definição completa do ambiente, passando pelas várias informações que o agente pode capturar. Pode-se ainda gerar relatórios com informações trocadas entre o jogo e a plataforma tais como Observações, Recompensas e até mesmo gravar em vídeo toda a Missão.

4.1.1.1 Definindo a Missão via código

Um dos modos de se definir aspectos dos experimentos é via código na própria linguagem do programa principal. Para isso, é oferecido um conjunto de funções, classes e métodos (de acordo com a linguagem ... a partir daqui serão utilizados os termos específicos para a linguagem Java) de modo a possibilitar desde opções de grande impacto até ajustes mais refinados da Missão. A classe `MissionSpec` possui 47 métodos que modificam diretamente a Missão dentre as quais pode-se destacar:

- `timeLimitInSeconds`: define o tempo limite, em segundos, de duração da missão. A conexão criada com o Minecraft irá terminar imediatamente após esse tempo, impossibilitando novas trocas de informação e gerando relatórios finais.
- `createDefaultTerrain`: método que gera um mundo completamente plano (*Flat World*) e vazio.
- `setWorldSeed`: Método que força a utilização de uma determinada semente de números aleatórios para a geração do mundo pela plataforma.
- `drawItem`: Método que coloca um determinado item (passado via argumento) na posição especificada do espaço.
- `drawBlock` e similares: A plataforma possui diversos métodos de conveniência para o arranjo de blocos específicos e outras formas geométricas formadas pela composição de 2 ou mais blocos, como linhas, quadrados, círculos, cubos ou esferas.
- `startAt / endAt`: Métodos que definem respectivamente o ponto inicial do agente no mundo e um ponto onde a missão é forçadamente terminada.
- `requestVideo`: Fornece ao agente dados das imagens exibidas em tela.

- `rewardForReachingPosition`: Método que passa ao agente uma recompensa numérica ao atingir uma posição especificada.
- `observeRecentCommands` e similares: Métodos que passam para o agente informações diversas sobre o ambiente, desde comandos enviados até o estado do próprio agente. As observações permitidas serão tratadas em subseções posteriores.
- `allowAllContinuosMovementCommands` e similares: Métodos que delimitam o subconjunto de ações permitidas ao agente. São esses subconjuntos Movimentos Contínuos, Discretos e Absolutos. Essas definições serão melhor tratadas na subseção *Handlers*.

4.1.1.2 Definindo a Missão via XML

Apesar de existirem tais classes para definição da Missão e de as mesmas permitirem uma flexibilidade no ajuste do experimento, o principal método para construção de uma Missão é via arquivo XML. Não apenas este método possui mais recursos, podendo, por exemplo, fornecer recompensas por situações mais amplas que as especificadas acima, como permite que um programa possa ser executado em diferentes situações sem a necessidade de alteração no código e recompilação. A lista de possibilidades adicionais via XML é extensa, mas pode ser encontrada em <http://microsoft.github.io/malmo/0.30.0/Schemas/Mission.html>. Porém, como desvantagem, uma vez que se carregue a Missão via XML, perde-se o conceito de *default* e toda e qualquer informação necessária não presente no arquivo irá impossibilitar o funcionamento correto do experimento. Um exemplo de código XML pode ser visto na figura 18.

```
<AgentSection mode="Survival">
  <Name>Arthur Abdel Simpson</Name>
  <AgentStart>
    <Placement x="4.5" y="46.0" z="1.5" pitch="30" yaw="0"/>
  </AgentStart>
  <AgentHandlers>
    <DiscreteMovementCommands/>
    <ObservationFromNearbyEntities>
            <Range name="nearbyEntities" xrange="10" yrange="1" zrange="10"/>
    </ObservationFromNearbyEntities>
    <ObservationFromGrid>
            <Grid name ="floor3x3">
                    <min x="-1" y="-1" z="-1"/>
                    <max x="1" y="-1" z="1"/>
            </Grid>
    </ObservationFromGrid>
    <ObservationFromRecentCommands/>
    <ObservationFromFullStats/>
```

Figura 18 – Trecho de código em XML para descrição da Missão, conforme padrão da Plataforma Malmo.

4.1.2 *Handlers*

Uma parte importante da definição de uma Missão são os *Handlers*. Como pode ser visto na figura 18, o código em XML que define a missão possui duas seções para tal categoria: uma para lidar com permissões do agente e outra para lidar com permissões no servidor. São, em essência, essas entidades que definem os tipos de movimento, as informações percebidas pelo agente e as recompensas recebidas.

Para as recompensas, é possível sempre ajustar o valor da mesma, que é um valor real, em ponto flutuante. Dentre as recompensas permitidas encontram-se:

- `RewardForTouchingBlockType`: Recompensa por tocar certo tipo de bloco. É possível também especificar se a mesma é dada apenas uma vez, continuamente ou uma vez por período de tempo (e respectivo período).
- `RewardForSendingCommand`: Recompensa por enviar algum comando ao agente.
- `RewardForSendingMatchingChatMessage`: Recompensa por enviar mensagem que seja exatamente igual a uma certa expressão.
- `RewardForCollectingItem`: Recompensa por coletar um item de um tipo especificado.
- `RewardForDiscardingItem`: Recompensa por descartar um item de um tipo especificado.
- `RewardForReachingPosition`: Recompensa por encontrar-se em uma determinada posição.
- `RewardForMissionEnd`: Recompensa dada devido ao termino da Missão por um motivo específico, como morte ou vitória, por exemplo.
- `RewardForStructureCopying`: Recompensa dada por copiar determinada estrutura.
- `RewardForTimeTaken`: Recompensa dada proporcional ao tempo total da Missão. Pode ser fornecida a cada período de tempo ou acumulada, no final.
- `RewardForCatchingMob`: Recompensa por encurralar uma certa entidade em um canto.
- `RewardForDamagingEntity`: Recompensa dada toda vez que o agente causa dano em alguma entidade do jogo como, por exemplo, animais.

Entre as possibilidades permitidas de forçar o fim de uma missão encontram-se:

- `AgentQuitFromTimeUp`: Término da Missão devido ao tempo máximo atingido.
- `AgentQuitFromReachingPosition`: Término devido ao agente ter alcançado determinada localização.
- `AgentQuitFromTouchingBlockType`: Término devido ao agente tocar um bloco de certo tipo.
- `AgentQuitFromCollectingItem`: Término devido ao agente coletar um item especificado.
- `AgentQuitFromReachingCommandQuota`: Término devido ao agente atingir o número máximo de comandos escolhido.
- `AgentQuitFromCatchingMob`: Término devido ao agente ser bem sucedido em encurralar uma entidade.

Os *Handlers* de movimento e de observações serão tratados em sub-seções separadas.

4.1.3 Ações

As ações permitidas ao agente pelo sistema dependem diretamente de qual *Handler* de movimento foi especificado no arquivo de configuração. Essa entidade e os comandos permitidos são especificados abaixo.

`ContinuosMovementCommands` permite o envio de comandos de movimento contínuo. Esses comandos tratam o espaço como contínuo e precisam ser enviados apenas uma vez, que continuarão a ser executados até que um comando de mesma categoria com argumento diferente seja enviado.

- `move`: Comando que permite que o agente desloque-se para a frente ou para trás segundo um argumento real entre -1 e 1, onde o primeiro representa marcha a ré com velocidade máxima e o segundo, avanço frontal também com velocidade máxima. Um argumento de valor 0 significa parar completamente a ação.
- `strafe`: Similar ao comando `move` mas realizando um deslocamento lateral. Valores negativos representam deslocamento para a esquerda e positivos para a direita.
- `pitch`: Comando que desloca o ângulo de visão do agente para cima ou para baixo. O deslocamento ocorre de acordo com um argumento no mesmo intervalo que aquele do comando `move` e com respectivas considerações de velocidade. Valores negativos representam um ajuste para cima e positivos para baixo.

- `turn`: Comando que faz o agente girar em seu próprio eixo, sem deslocar-se. Novamente o argumento do comando é similar ao do `move`. Valores negativos representam giros anti-horários e positivos no sentido horário.
- `jump`: Comando com argumento binário que faz com que o agente inicie o comportamento de pular (caso 1) ou cesse este comportamento, caso o mesmo esteja ativo (caso 0).
- `crouch`: Comando de argumento binário que faz com que o agente agache-se (1) ou retorne a posição em pé (0).
- `attack`: Comando de argumento binário que faz com que o agente inicie comportamento de atacar (1) ou cesse o mesmo, caso ativo (0).
- `use`: Comando de argumento binário que faz com que o agente inicie comportamento de utilizar o item segurado em mãos (posição 0 no inventário) (1) ou cesse o mesmo, caso ativo (0).

`DiscreteMovementCommands` permite o envio e tratamento de comandos de movimento discreto. Os movimentos enviados são executados apenas uma única vez e cessam.

- `move`: Comando que move o agente um bloco para a frente (argumento 1) ou para trás (argumento -1).
- `movenorth 1`, `movesouth 1`, `moveeast 1`, `movewest 1`: Comandos que movem, respectivamente, em direção ao norte, ao sul, ao leste e ao oeste.
- `strafe`: Comando que desloca o agente lateralmente um bloco para a esquerda (-1) ou para a direita (1).
- `use 1`: Comando que faz o agente utilizar o item em mãos uma única vez.
- `jump 1`: Comando que move um bloco acima.
- `jumpuse 1`: Comando que simultaneamente move o agente um bloco acima e utiliza o item segurado em mão uma vez.
- `jumpstrafe`: Comando que move um bloco acima e um bloco seguindo a mesma lógica do comando `strafe`.
- `jumpmove`: comando que move um bloco acima e um bloco seguindo a mesma lógica do comando `move`.
- `jumpnorth 1`, `jumpsouth 1`, `jumpeast 1`, `jumpwest 1`: Comandos que movem um bloco acima e, respectivamente, em direção ao norte, ao sul, ao leste e ao oeste.

- `turn`: Comando que rotaciona o agente 90º no sentido anti-horário (-1) ou horário (1)
- `look`: Comando que ajusta ângulo de visão do agente 45º para cima (-1) ou para baixo. (1)
- `attack 1` Comando que faz o agente golpear e destruir o que quer que esteja no foco de visão.

`AbsoluteMovementCommands` permite o uso de comandos que deslocam o agente sem percorrer a distância entre a origem e o destino ("teletransporte").

- `tp`: Com argumentos reais representando as coordenadas do destino final nos eixos X, YeZ, este comando posiciona o agente exatamente no ponto especificado sem alterar outros parâmetros, como *pitch* e *yaw*.
- `tpx`: Comando que posiciona o agente no ponto de mesmas coordenas das YeZ porém com X conforme o valor especificado como argumento.
- `tpy`: Similar ao comando `tpy` mas para reposicionamento segundo a coordenada Y.
- `tpz`: Similar aos comando `tpy` e `tpx` mas para reposicionamento segundo a coordenada Z.
- `setYaw`: O agente automaticamente assume o *yaw* passado no argumento do comando.
- `setPitch`: O agente automaticamente assume o *pitch* passado no argumento do comando.

Existem ainda outros *Handlers* para comandos envolvendo o inventário, chat, criação simples de itens, término da missão e comandos em Missões baseadas em turnos, porém não são de interesse deste trabalho. A documentação completa pode ser encontrada no mesmo endereço citado para a missão.

4.1.4 *Observations*

Um dos aspectos fundamentais em aplicações de sistemas inteligentes que envolvem diretamente o conceito de agente é a aquisição de informação.

A plataforma Malmo permite que o usuário escolha quais informações seu agente receberá do ambiente, permitindo diversos tipos de experimento onde o agente pode variar desde a onisciência possuindo, por exemplo, informações absolutas de posição e estado de outras entidades, até a navegação puramente pelos *pixels* exibidos em tela.

São *Handlers* válidos de observação:

- `ObservationFromRecentCommands`: Retorna observações que indicam quais comandos foram recebidos desde a última observação.
- `ObservationFromHotBar`: Indica qual o conteúdo de cada elemento de atalho de inventário.
- `ObservationFromFullStats`: Retorna um conjunto de diversas informações, como tempo que permaneceu vivo, dano causado e sofrido, vida, ar, alimento, posição, pontuação etc.
- `ObservationFromFullInventory`: Fornece informações sobre os elementos presentes em cada espaço do inventário.
- `ObservationFromSubgoalPositionList`: Informa a direção a ser seguida para atingir o próximo subobjetivo.
- `ObservationFromGrid`: Retorna informações completas sobre a estrutura $NxMxO$ de blocos centrada no agente. Tais dimensões são escolhidas pelo usuário.
- `ObservationFromDistance`: Informa a distância até uma certa localização especificada no XML.
- `ObservationFromDiscreteCell`: Indica a posição nos eixos XeZ do bloco atual do agente.
- `ObservationFromChat`: Retorna informações sobre as mensagens (conteúdo e agente) que foram enviadas no chat.
- `ObservationFromNearbyEntities`: Informa diversas informações (como posição, tipo etc) de todas as entidades dentro de uma região limitada especificada que é centrada no agente.
- `ObservationFromRay`: Fornece informações sobre o bloco ou entidade que se encontra na linha de visão do agente, isto é, que encontra diretamente uma linha imaginária que passa pelo ponto central da visão do agente.
- `ObservationFromTurnScheduler`: Retorna informações importantes para o uso de Missões baseadas em turnos, como por exemplo a chave para passar o turno para outro agente.

Apesar de seu recente lançamento a público, a plataforma já é ferramenta de estudos em várias áreas de concentração em Inteligência Artificial, como aprendizado por

transferência de conhecimento em sistemas multiagentes (GEIGER *et al.*, 2016), reconhecimento visual e seleção de subobjetivos para resolução de problemas (BONNANO *et al.*, 2016), validação de aprendizados com *Deep Reinforcement Learning* (UDAGAWA *et al.*, 2016) ou, mais próximo do objetivo do presente trabalho, aprendizado de habilidades de necessidade contínua (TESSLER *et al.*, 2016; BARRON *et al.*, 2016; OH *et al.*, 2016).

Porém o Malmo não é o o primeiro *Mod* do famoso jogo. O interesse no ambiente virtual é anterior à plataforma e já era explorada com diversos fins como educacionais (BAYLISS, 2012; SHORT, 2012; IAN *et al.*, 2016) ou mesmo similares aos do Malmo, como o BURLAPCraft (ALURU *et al.*, 2015), onde validou-se, entre outros, o aprendizado de linguagem natural via comandos.

4.2 A Mente do Agente: O Modelo Estrutural de Funcionamento

Segundo Franklin (1995), podemos fazer uma analogia entre o sistema de controle de um agente inteligente e a mente humana. Assim, podemos dizer que tal sistema corresponde a uma mente artificial. Essa analogia nos permite transpor mais facilmente modelos cognitivos aplicáveis, em princípio, somente à mente humana, a modelos computacionais utilizando arquiteturas cognitivas.

A proposta dessa dissertação constitui-se em desenvolver uma arquitetura cognitiva para a mente artificial que controla um agente no jogo *Minecraft*. O projeto desta mente artificial busca conciliar diversas técnicas presentes na literatura a fim de se obter resultados eficientes e em tempo aceitável para que arquiteturas similares possam ser usadas para uma vasta gama de aplicações, incluindo aquelas no mundo real, que possam se beneficiar de aprendizado autônomo como *drones*, carros autônomos e interfaces cérebro-computador.

Construída utilizando o já citado *toolkit* CST, esta arquitetura apresenta, assim como qualquer outra construída com o CST, uma estrutura fundamental orientada a *Codelets*. A figura 19 apresenta a estrutura topológica da proposta. Nela são apresentados os *Codelets* e macroestruturas presentes e necessárias para seu funcionamento. Essas estruturas funcionam contínua e paralelamente enquanto a mente artificial continuar ativa. Como o leitor poderá perceber, nossa arquitetura foi fortemente inspirada na arquitetura MECA, onde alguns *codelets* foram subtraídos e outros acrescentados. Alguns *codelets* são definidos na proposta de arquitetura por fazerem parte da concepção teórica proposta mas não serão utilizados na implementação específica dos controladores estudados neste trabalho. Entretanto, achamos importante colocá-los na arquitetura conceitual, pois os mesmos serão importantes posteriormente, considerando-se os trabalhos futuros que pretendemos empreender. Seguindo a divisão apresentada na seção 2.4.1, bem como diversas ideias presentes em (GUDWIN *et al.*, 2017), nossa arquitetura pode ser dividida em um

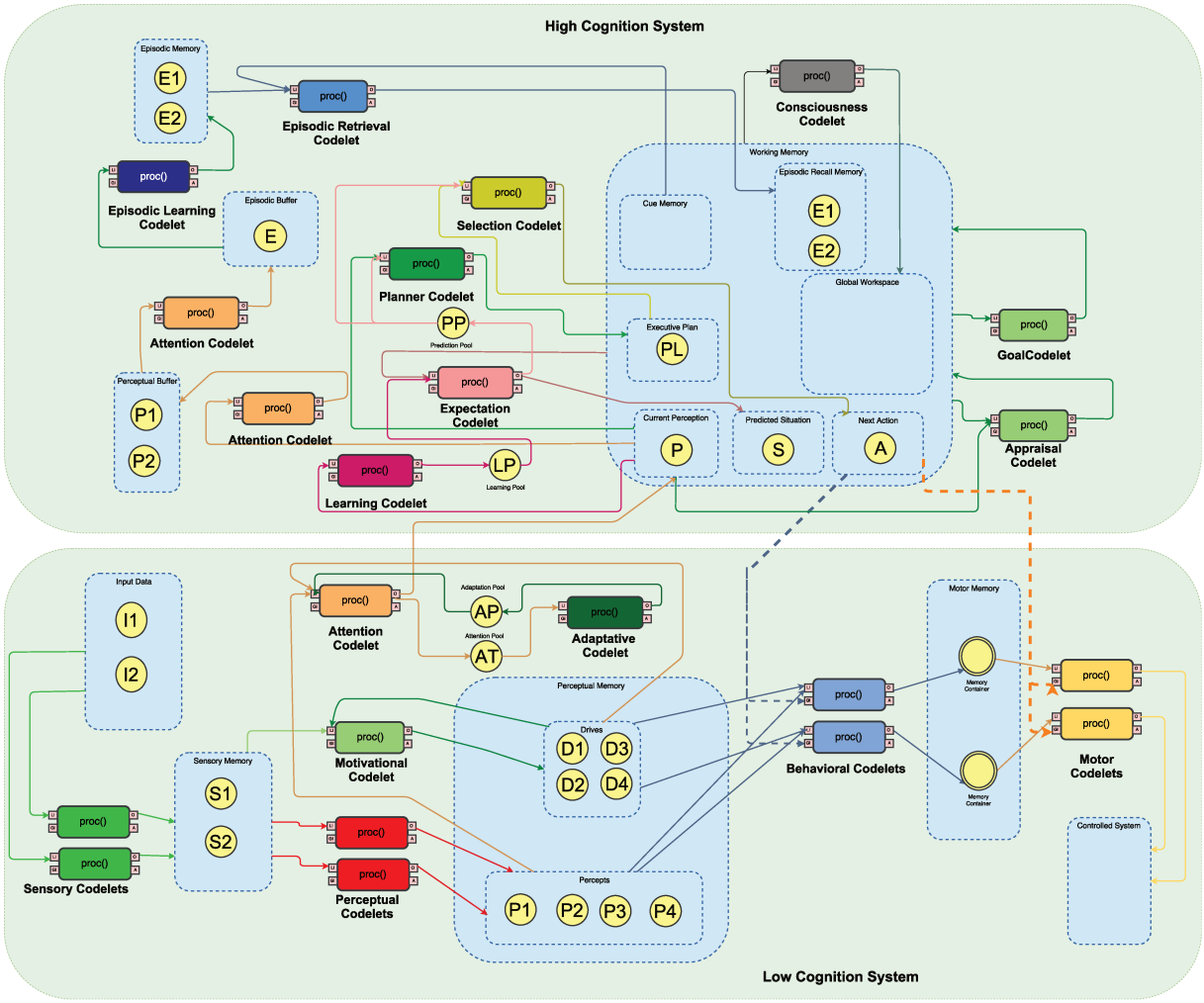

Figura 19 – Arquitetura Cognitiva Utilizada no Projeto. Nem todos os *Codelets* estão presentes no controlador final.

sistema de Baixa Cognição e um sistema de Alta cognição, interconectados entre si.

A parte da arquitetura em destaque na figura 20 representa o Sistema de Baixa Cognição. Nele é esperado que comportamentos simples, determinados com informações dos perceptos, sejam processados e suas respostas devolvidas ao sistema controlado através do módulo motor.

A parte em destaque na figura 21, correspondente ao sistema de Alta Cognição, é o sistema responsável por processamentos mais sofisticados em controladores desenvolvidos com a Arquitetura. Nele, é esperado todo tipo de aparato computacional que possa auxiliar no aprendizado e adequação de comportamento do agente. Em especial, destacam-se o uso de Redes Neurais como preditores e aproximadores, métodos de Aprendizado Instrumental e Memória Episódica pois, como visto em Kuppuswamy *et al.* (2006) e Castro e Gudwin (2013), esta deveria trazer uma melhora significativa no desempenho do agente, dotando-o de melhorias cognitivas importantes. Utilizaram-se ainda elementos especiais capazes de

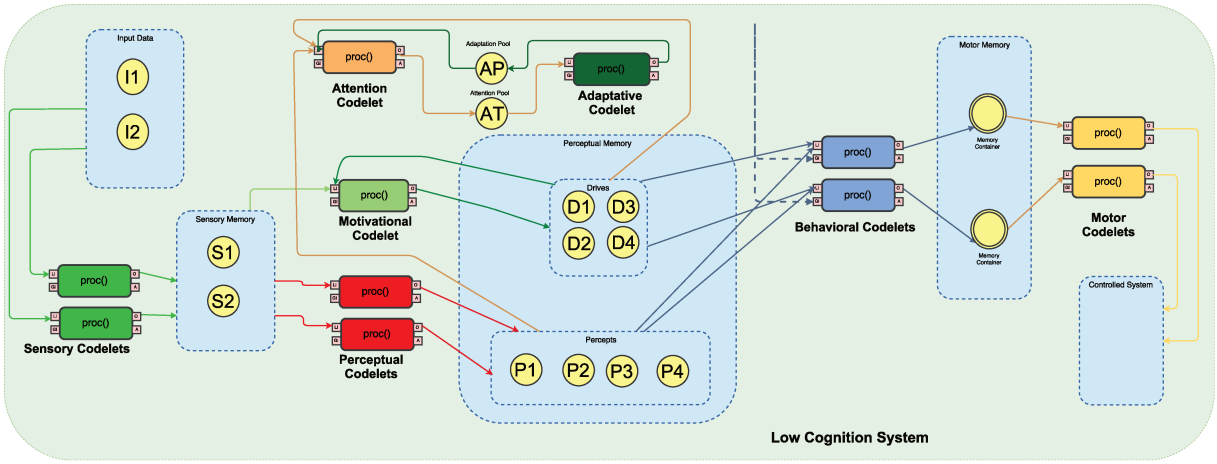

Figura 20 – Subsistema de Baixa Cognição presente na Arquitetura apresentada na figura 19

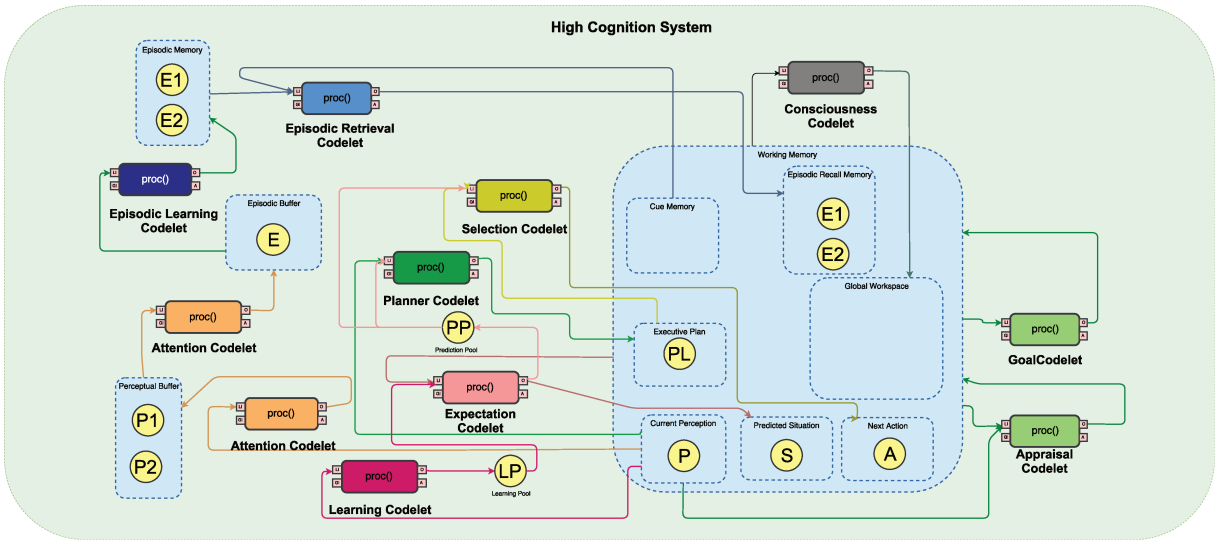

Figura 21 – Subsistema de Alta Cognição presente na Arquitetura apresentada na figura 19

emular conceitos de planejamento.

Na Arquitetura, estão presentes os seguintes *Codelets*:

- *Episodic Retrieval Codelet*: responsável por, utilizando gatilhos da *Working Memory* (*Cue Memory*, descrita mais abaixo), retornar episódios por critérios de similaridade. Como critério, foram utilizados tanto métodos generalizados baseados na estrutura do conhecimento adotada na arquitetura (OWRL) como métodos específicos para os experimentos descritos no capítulo seguinte. O algoritmo mostrado no pseudocódigo 4.1 representa um método independente de aplicação que foi implementado.

- *Learning Codelet*: responsável por armazenar informação e fornecê-la para a geração de conhecimento segundo o método utilizado no *Expectation Codelet* que, no caso especifico deste trabalho, significa a atualização dos pesos da rede neural. Para isso utiliza informações de percepções guardadas vindas do *Current Perception*, descritas mais abaixo.

- *Selection Codelet:* utilizando políticas definidas pelo usuário e podendo variar de aplicação para aplicação, este *Codelet* é responsável por atualizar o elemento de memória *Next Action* a fim de enviar aos *Motor Codelets* as informações relevantes. Utiliza informações vindas do *Expectation Codelet ou Planner Codelet*, conforme o caso.

- *Adaptative Codelet*: responsável por estabelecer regras especificas de pares condição-ação de modo a desonerar os processos ligados às tarefas cognitivas mais sofisticadas deste trabalho (efetivamente pulando todos os processos conscientes). Este procedimento visa simular a ideia de que algumas ações tomadas repetidas vezes (e geralmente de modo mecânico) tendem a ser tomadas de modo inconsciente ou de algum modo subconsciente. Neste trabalho tal elemento não será utilizado. Entretanto, pretendemos utilizar esse *codelet* em trabalhos futuros.

- *Appraisal Codelet*: Este *Codelet* avalia a qualidade de uma dada Configuração, estrutura de representação de mundo em OWRL conforme definida no capítulo 3, segundo parâmetros internos definidos pelo usuário, e o traduz em valores quantitativos. A aplicação presente neste trabalho não utilizará este *Codelet*. Entretanto, pretendemos utilizar esse *codelet* em trabalhos futuros.

- *Expectation Codelet*: *Codelet* que, através de métodos de inteligência computacional, avalia os resultados esperados para cada uma das possíveis ações dada a condição atual. No controlador mostrado na figura 24, esta estrutura utiliza-se de um esquema de aprendizado por reforço em conjunto com uma rede neural para gerar esses valores esperados.

- *Attention Codelets: Codelets* responsáveis por, agindo cada um em seus objetos de memória, selecionar informações que, segundo seus critérios, são importantes para a realização de outros processos.

- *Sensory Codelets: Codelets* nos quais as informações sensoriais chegam do ambiente, conforme percebidas pelo agente. Esta é uma das poucas partes da arquitetura (as outras são as que sintetizam funções motoras) que têm ligação direta com o sistema a ser controlado. Essas informações são então escritas em um Objeto de Memória onde fica a disposição dos *Codelets* Perceptuais.

- *Perceptual Codelets*: Responsáveis por, utilizando as entradas sensoriais captadas por outras partes da mente artificial, gerar estruturas chamadas Perceptos, que contêm as informações acerca do ambiente e/ou do próprio agente organizadas segundo uma lógica interna da arquitetura, distinguindo-se portanto de informações sensoriais puras.

- *Motivational Codelet*: Responsável por modificar os objetos de memória que contêm informações sobre as necessidades do agente (os *drives*), que irão guiar todo o seu comportamento. Para isso usa como entrada os próprios *drives* e as entradas perceptuais. Neste trabalho, estes *Codelets* não serão explorados. Entretanto, pretendemos utilizar esse *codelet* em trabalhos futuros.

- *Behavioral Codelets*: Nestes *Codelets* encerram-se os processos comportamentais dos quais a criatura é dotada. Normalmente, os comportamentos têm, em outras arquiteturas, suas estruturas básicas predeterminadas, sejam eles fixos ou adaptativos. As informações necessárias para a execução dos comportamentos são dispostas em Objetos de Memória para acesso e uso pelos *Codelets* Motores.

- *Motor Codelets*: Estrutura responsável por enviar à criatura (no caso específico o personagem no jogo Minecraft) as ações pontuais a serem executadas, conforme diretrizes. É importante salientar que estas estruturas enviam comandos bem específicos ao sistema controlado. Em um braço robótico, por exemplo, estes *Codelets* enviariam sinais para cada um dos servomotores, de modo a executar o movimento esperado.

- *Episodic Learning Codelet: Codelet* responsável por codificar e armazenar Episódios na estrutura da Memória Episódica. Na proposta, cada Episódio é definido como uma estrutura contendo um par de Configurações (representando um antes e um depois) e campos auxiliares, contendo ação tomada (que pode, inclusive, ser nula) e uma avaliação quantitativa do Episódio.

- *Planner Codelet: Codelet* ao qual é designada a função de traçar um curso de ações um determinado número de passos a frente. Tal estrutura permite uma exploração maior porém não explosiva do espaço de estados imediatamente próximo do estado atual do agente. Representa também uma transição entre um tipo de comportamento reativo para um deliberativo.

- *Goal Codelet*: Estrutura responsável pela assimilação, estruturação e cumprimento de objetivos de alto nível. Este *Codelet* não será explorado neste trabalho. Entretanto, pretendemos utilizar esse *codelet* em trabalhos futuros.

- *Consciousness Codelet: Codelet* responsável por fornecer a outros *Codelets* acesso direto a todos os *Global Inputs* da arquitetura. Esta estrutura não será trabalhada

nesta dissertação. Entretanto, pretendemos utilizar esse *codelet* em trabalhos futuros.

Além dos *Codelets*, a arquitetura possui algumas estruturas de memória, em sua maioria bem intuitivas e representando apenas estruturas conceituais, não envolvendo código propriamente dito. No entanto, vale detalhar a estrutura da Memória de trabalho:

- *Current Perception*: Estrutura responsável por receber a informação acerca das condições atuais, próprias e do ambiente, conforme o *Attention Codelet* presente no Subsistema de Baixa Cognição. Pode ser entendido como a "porta de entrada" do Subsistema de Alta Cognição.
- *Cue Memory*: Memória que, com dados vindos do *Attention Codelet*, fornece informações para a possível recuperação de episódios similares à situação atual. Na atual implementação, este elemento contém informações a mesma estrutura presente no *Current Perception*. No entanto, conforme desenvolvimentos futuros, pode-se necessitar de duas estruturas separadas.
- *Next Action*: Elemento cuja informação é repassada diretamente a todos as estruturas motoras e comportamentais e que, geralmente, representam diretrizes prioritárias sobre aquelas do Sistema de Baixa Cognição.
- *Predicted Situation*: Estrutura responsável por guardar para possíveis usos a situação esperada após a tomada de decisão engatilhada.
- *Episodic Recall Memory*: Memória na qual os episódios recuperados são dispostos pelo *Codelet* responsável.
- *Executive Plan*: Elemento de Memória no qual estão guardadas as diretrizes a serem seguidas na forma de plano.
- *Global Workspace*: Estrutura responsável por conter informação que será transmitida a todos os *Codelets* da arquitetura. Não será trabalhada nesta dissertação. Entretanto, pretendemos utilizá-la em trabalhos futuros.

Apesar da estrutura do CST em princípio funcionar como um conjunto de múltiplos agentes interagindo completamente em paralelo, diversos pesquisadores na área de arquiteturas cognitivas defendem que o estabelecimento de um ciclo cognitivo pode ser importante (MADL *et al.*, 2011). Esse ciclo é necessário porque existe uma certa direcionalidade, um certo fluxo de informações dentro da arquitetura cognitiva, onde a percepção antecede o planejamento de ações, que antecede a execução de ações motoras. No design de nossa arquitetura, o ciclo cognitivo previsto depende de quais *Codelets* estão ativos ou

não. Nas subseções seguintes serão apresentados os controladores criados com a arquitetura proposta e seus respectivos ciclos cognitivos. Note que, nas arquiteturas que serão apresentadas, o *Malmo Sensor* corresponde a um *Sensory Codelet* e os *Self Perceptual e Environmental Codelets* correspondem a *Perceptual Codelets* apresentados anteriormente, porém específicos da aplicação.

4.2.0.1 Controlador Cognitivo baseado em expectativas

Sendo a versão mais simples dos controladores propostos, a tomada de decisão neste controlador, ilustrado na figura 22 baseia-se puramente na geração de expectativas e escolha da ação que maximiza o valor de recompensa esperado, conforme o fluxo descrito abaixo:

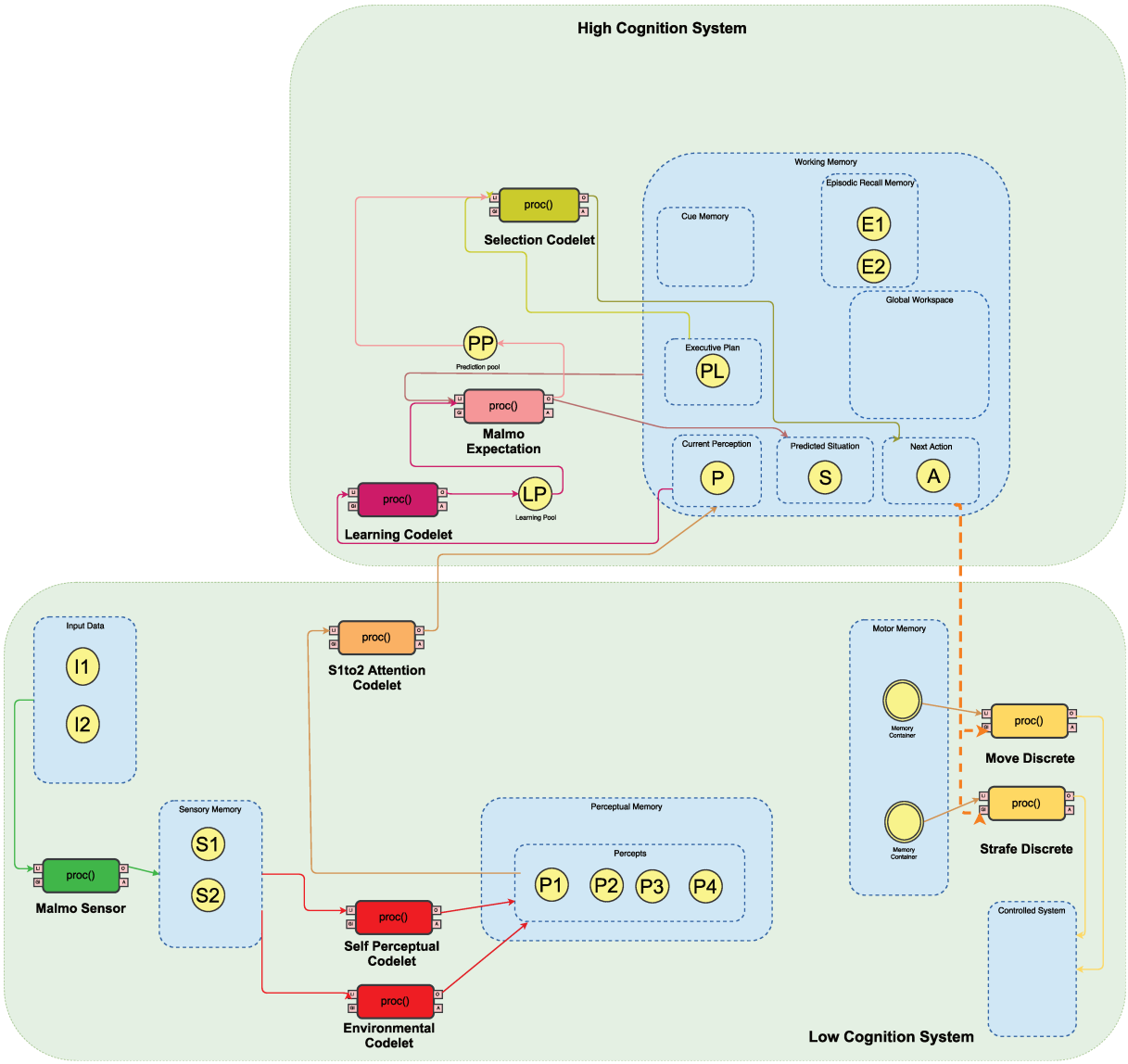

Figura 22 – Controlador construído a partir da Arquitetura apresentada na figura 19

1. As informações são percebidas do ambiente e sobre si mesmo, gerando Perceptos que são passados para um *Attention Codelet* que as repassa para a *Memory Current Perception*.

2. O *Expectation Codelet*, utilizando critérios internos de expectativa de recompensa que serão melhor explicados em subseções posteriores, atribui valores de expectativa para cada possível ação dado o estado atual do agente.

3. Utilizando as informações do *Expectation Codelet*, o *Selection Codelet* atualiza a próxima ação a ser tomada.

4. A ação (ou comportamento mais complexo) é então passada ao agente.

5. Num processo contínuo, a observação que também é referente ao próximo ciclo é aqui utilizada para avaliar as mudanças que ocorreram em decorrência da última ação tomada.

6. Através das diretrizes do *Learning Codelet*, a rede neural responsável pela função de aproximação para a geração de expectativas é atualizada para refletir o aprendizado sobre a tríplice condição-ação-efeito.

4.2.0.2 Controlador Cognitivo com Memória Episódica

Mostrado na figura 23, esta versão do controlador apresenta, além de tudo que já estava presente na versão anterior, um módulo de Memória Episódica. Este módulo adicional acrescenta duas etapas ao controlar, que consistem em gravar e recuperar informação, conforme mostrado a seguir:

- As informações são percebidas do ambiente e internamente sobre si mesmo, gerando Perceptos que são passados para um *Attention Codelet* que as repassa para a *Memory Current Perception*.

- Através do *Episodic Retrieval Codelet*, essas informações são testadas à procura de similaridade com episódios anteriores, a fim de se obter uma filtragem selecionando, de acordo com experiências passadas, uma expectativa mais acurada do efeito das possíveis ações para a atual situação.

- O *Expectation Codelet*, utilizando seus critérios internos de expectativa de recompensa devidamente modificados pelas informações advindas do *Episodic Retrieval Codelet*, atribui valores de expectativa para cada possível ação dado o estado atual do agente.

- Utilizando as informações do *Expectation Codelet*, o *Selection Codelet* atualiza a próxima ação a ser tomada.

- As etapas posteriores são similares às etapas 5 e 6 do controlador mais simples.

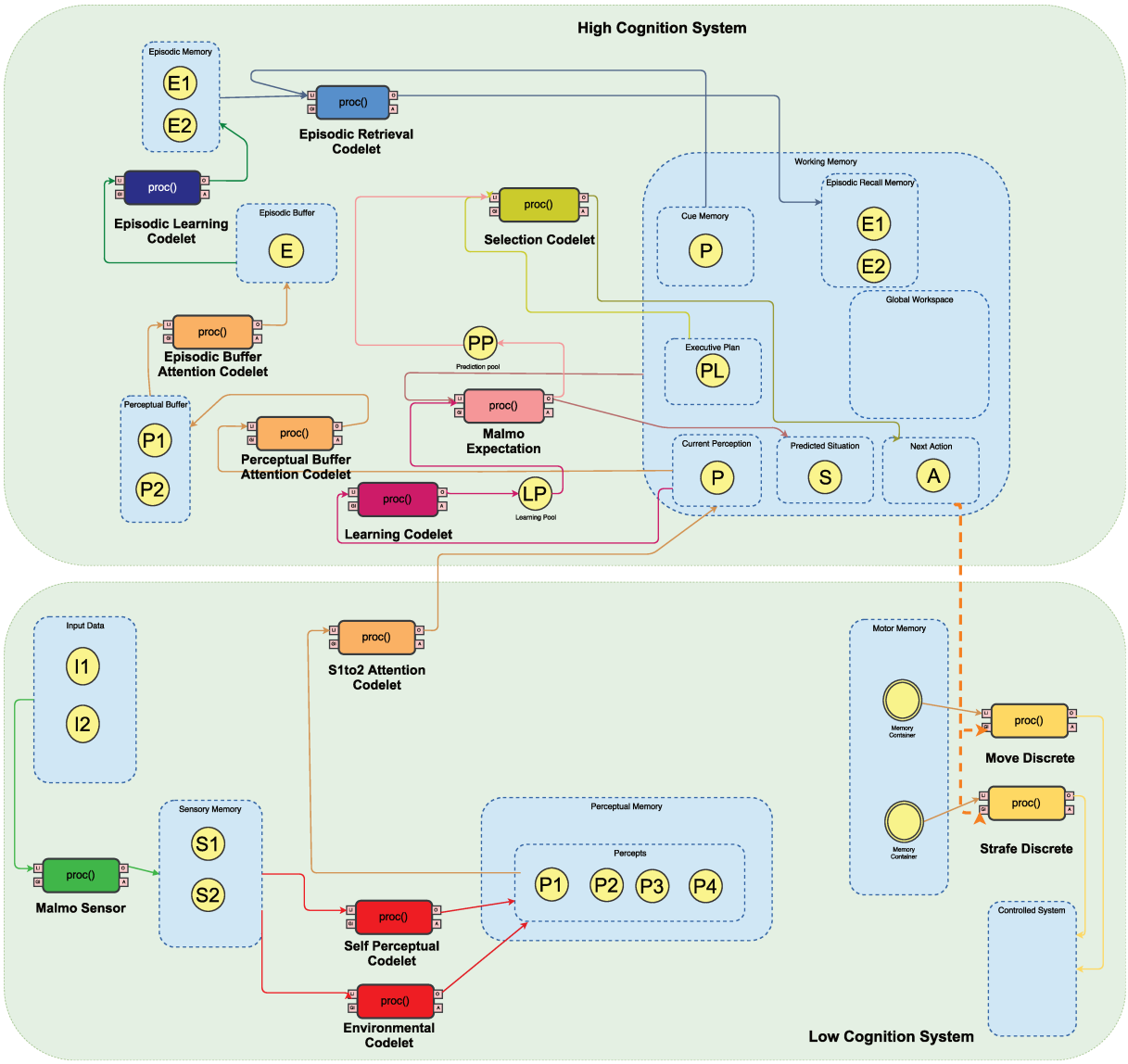

Figura 23 – Controlador construído a partir da Arquitetura apresentada na figura 19

4.2.0.3 Controlador Cognitivo com Memória Episódica e Exploração

Com topologia fundamentalmente equivalente àquela mostrada na figura 23, esta versão do controlador apresenta possui ainda uma rotina de exploração do ambiente, conforme mostrado a seguir:

- As informações são percebidas do ambiente e internamente sobre si mesmo, gerando Perceptos que são passados para um *Attention Codelet* que as repassa para a *Memory Current Perception*.

- Através do *Episodic Retrieval Codelet*, essas informações são testadas à procura de similaridade com episódios anteriores, a fim de se obter uma filtragem selecionando, de acordo com experiências passadas, uma expectativa mais acurada do efeito das possíveis ações para a atual situação.
- O *Expectation Codelet*, utilizando seus critérios internos de expectativa de recompensa devidamente modificados pelas informações advindas do *Episodic Retrieval Codelet*, atribui valores de expectativa para cada possível ação dado o estado atual do agente. Aleatoriamente, em 50% dos casos, o *Expectation Codelet* irá ainda atribuir valores específicos para ações que levem a estados não visitados de forma a favorecer a escolha dos mesmos em relação àquelas em que o resultado já é conhecido.
- Utilizando as informações do *Expectation Codelet*, o *Selection Codelet* atualiza a próxima ação a ser tomada.
- As etapas posteriores são similares às etapas 5 e 6 do controlador mais simples.

4.2.0.4 Controlador Cognitivo Completo

A figura 24 apresenta o controlador completo criado com base na arquitetura proposta. Note a existência do *Codelet* de Planejamento em sua estrutura.

Para a versão final do controlador, tem-se o seguinte ciclo:

- As etapas 1, 2 e 3 são similares às presentes na versão descrita na subseção anterior.
- Utilizando as informações do *Expectation Codelet*, o *Planner Codelet*, caso esteja ativo, define uma lista de ações a serem tomadas antes do próximo planejamento.
- As ações são passadas sequencialmente para o agente. Caso o *Planner Codelet* não esteja ativo, a ação será escolhida conforme mecanismos já descritos nas versões anteriores.
- As etapas posteriores são similares às etapas 5 e 6 do controlador mais simples.

Nas sub-seções a seguir detalham-se alguns dos módulos fundamentais ao funcionamento do controlador.

4.2.1 Módulos de Expectativas e Planejador

O subsistema de expectativas e planejamento representa o grande centro do controle inteligente proposto como prova de conceito. Nele está a estrutura capaz de estimar a recompensa recebida pelo agente devido a cada uma das ações tomadas a partir do estado atual, bem como a capacidade de deliberar sobre as sequências de ações a serem tomadas.

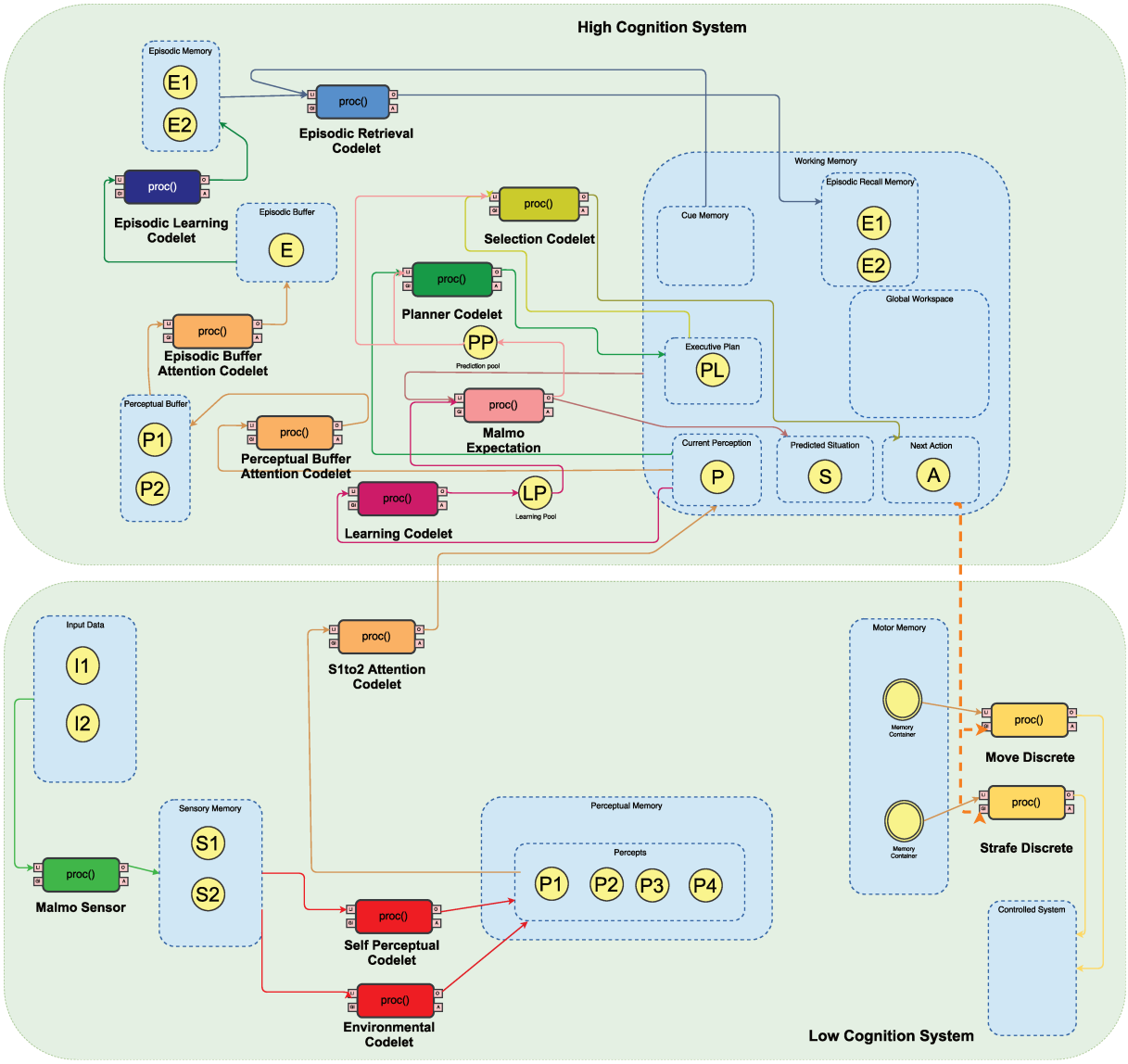

Figura 24 – Controlador construído a partir da Arquitetura apresentada na figura 19

Este módulo está basicamente dividido em dois *Codelets*: o Codelet de Expectativas e o Codelet Planejador, como mostra a figura 25.

4.2.1.1 *Expectation Codelet*

Conforme exposto no capítulo 2, a capacidade de inferência e generalização a partir de dados adquiridos constitui tanto uma vantagem como uma armadilha cognitiva para aqueles que a possuem. O *Codelet* de expectativas busca exercer justamente a vantagem já citada: criar uma expectativa do efeito das ações antes de efetivamente executá-las.

O funcionamento deste *Codelet* está majoritariamente baseado em uma rede neural artificial do tipo MLP ilustrada na figura 26. As entradas são informações do sistema a ser controlado, dependendo da aplicação. Cabe ao usuário definir quais informações são

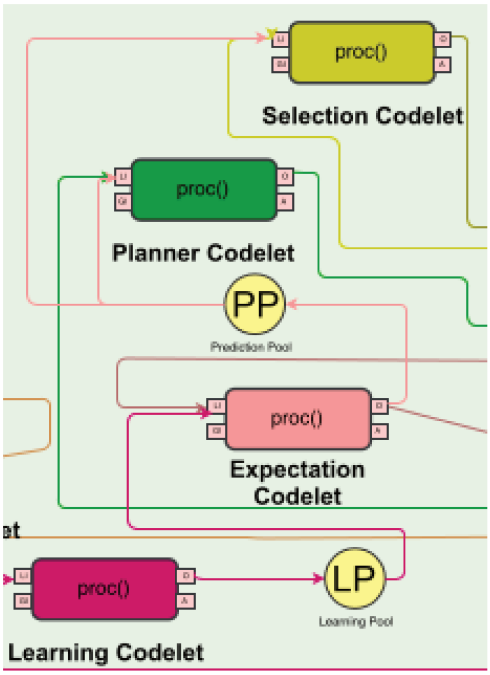

Figura 25 – Ênfase no Módulo de Expectativas do Controlador apresentado na figura 24.

relevantes e portanto devem ser passadas à rede. Especificamente em nossa aplicação, essas informações são um vetor com 9 números em ponto flutuante, representando diretamente a grade de blocos 3×3 centrada no agente, codificada segundo a posição em um vetor com todos os blocos possíveis na Plataforma. As saídas são as expectativas de recompensa para cada possível ação. Então, uma vez calculadas as expectativas, as mesmas serão passadas para um *Codelet* de seleção que, seguindo uma política definida pelo usuário, irá selecionar qual ação será tomada. No caso da aplicação apresentada, utiliza-se uma política conhecida como ϵ - *greedy* (SUTTON; BARTO, 1998), onde é selecionada a ação com maior valor, com probabilidade ϵ ou uma ação aleatoriamente escolhida, com probabilidade $1 - \epsilon$. Na aplicação, este parâmetro ϵ foi escolhido como sendo 0.99.

A rede é inicializada com pesos sinápticos aleatoriamente distribuídos ao redor de zero e, portanto, os primeiros resultados serão necessariamente imprevisíveis e as ações tomadas ao acaso. Conforme o agente explora o espaço de estados, informações de pares entrada-saída advindas do *Learning Codelet* são utilizadas para atualizar os pesos da rede neural em questão seguindo a função de erro mostrada na equação 2.3.3. Assim, a cada nova experiência, o agente melhora sua capacidade de prever o resultado de suas ações e, consequentemente sua capacidade de tomar boas decisões.

Tratando-se da aplicação, essa rede foi configurada para possuir 9 neurônios na camada de entrada, 20 na intermediária e 4 na de saída, camada cujos nós representam as possíveis ações do agente. Todos os neurônios da rede possuem função de ativação de natureza sigmoidal e foram treinadas com um algoritmo de gradiente descendente online

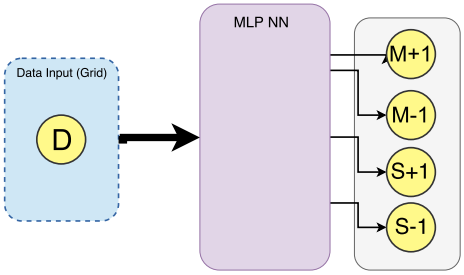

Figura 26 – Ilustração da rede neural responsável por gerar as expectativas por cada ação possível. Nesta imagem, a figura representa quatro saídas (ações), correspondente aos experimentos do capítulo 5, sendo elas `"move 1"` (M+1), `"move -1"` (M-1), `"strafe 1"` (S+1) e `"strafe -1"` (S-1). Essas ações seguem o padrão apresentado na subseção 4.1.3

com todos os exemplos de estados já visitados por 1000 iterações ou até o erro de validação parar de diminuir.

Em adição aos resultados devolvidos pela rede, este *Codelet* também utiliza informações da Memória Episódica para aprimorar suas decisões com base em situações que já viveu. Os Episódios relevantes são levados em conta alterando-se os valores de expectativa de recompensa fornecidos pela rede neural com os valores correspondentes às ações em que já se conhece efetivamente o resultado. Assim, apesar de ser capaz de estimar recompensas para suas ações, o agente beneficia-se em levar em conta resultados anteriores em situações similares, sendo capaz de prever situações nova e obter resultados exatos naquelas que já vivenciou. Os detalhes da do funcionamento do Módulo de Memória Episódica encontram-se na subseção 4.2.2.

4.2.1.2 *Planner Codelet*

Para a versão do controlador apresentada na subseção 4.2.0.4 como Controlador Cognitivo Completo, o processo de tomada de decisão deixa de ser meramente reativo e passa a ser deliberativo, à medida em que o agente é capaz de, utilizando-se de informações sensoriais adquiridas através de exploração, planejar um curso completo de ações antes de executá-las, podendo assim explorar de forma mais eficiente horizontes de decisão com melhores resultados a longo prazo, abandonando parcialmente imediatismos.

Ao longo do tempo, caso esteja ativo, o Planner Codelet responsabiliza-se por armazenar sequências de ações dentro de um período definido pelo usuário, formando assim um modelo de mundo na forma de estados e ações de transição. Após o fim deste período, a sequência candidata é otimizada de modo a se obter uma sequência mínima de ações e evitar a repetição de estados. O plano em questão é então armazenado como

melhor plano até que alguma outra sequência se prove mais eficiente (menor número de ações). No caso específico da aplicação, esse período consiste em todos os ciclos entre o inicio de uma dada execução e o momento em que o agente atinge o bloco desejado.

4.2.1.3 *Selection Codelet*

Além dos dois *Codelets* já citados pode-se incluir como parte importante do processo de decisão o *Selection Codelet*. Enquanto o *Expectation Codelet* é responsável pela geração de expectativas, ele não é, e nem precisa ser, a parte responsável pelo método de seleção de ações. Neste modelo, esse processo cabe ao *Selection Codelet*, que, com uma política definida internamente, escolhe a ação que será tomada a partir das informações fornecidas pelo *Codelet* de expectativas. Na aplicação em questão essa política é a já citada *ϵ-greedy*.

Já no caso do uso do *Planner Codelet*, deve-se garantir a execução da sequência de ações conforme determinado pelo plano. Para isso, cada ação é passada para o *Selection Codelet* sequencialmente, apenas uma por vez. Isso faz com que, independente da política escolhida, o plano seja seguido. Note que, apesar disto efetivamente inutilizar o *Codelet* de seleção, sua presença ainda é importante pois, caso não seja possível formular os planos (por inatividade programada do modulo planejador ou falta de informações suficientes), ele garantirá que o controlador comporte-se dentro do esperado.

4.2.2 Módulo de Memória Episódica

Outro módulo de grande relevância para o trabalho é a Memória Episódica. Como já visto no capítulo 2, este tipo de sistema de memória, ainda apenas identificada em humanos, confere ao agente uma organização dos fatos contextualizada no tempo. Essa organização contribui, dentre outros fatores, para melhorias no aprendizado, ao se conseguir traçar paralelos entre situações vividas no passado e situações presentes e na avaliação de desempenho na busca por um objetivo maior, ao se aplicar a mesma lógica para situações que ainda irão ocorrer.

Existem basicamente neste trabalho duas operações que envolvem a Memória Episódica, a aquisição e a recuperação de Episódios. Devido a seu funcionamento em tempo discreto, a aquisição de informação se dá de modo bem simples:

- Assim que a informação acerca do ambiente, estruturado formalmente como Configuração no padrão OWRL definido no capítulo 3, chega ao *Current Perception*, um *Codelet* de atenção repassa essa informação para um *Buffer* perceptual que tem como objetivo guardar temporariamente os estados obtidos.

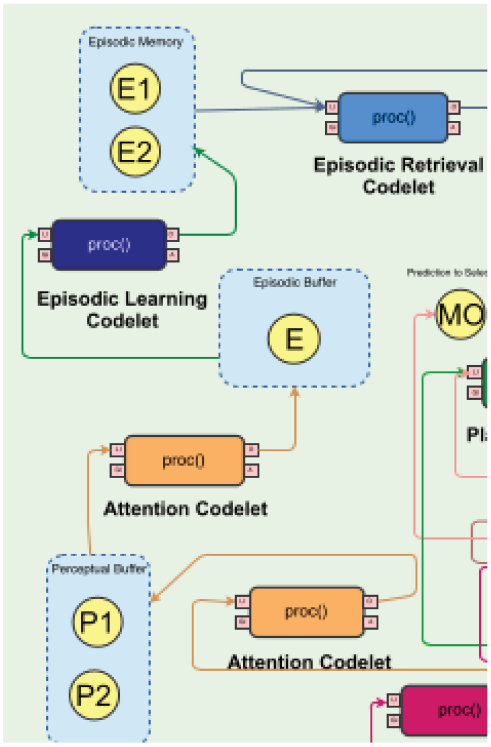

Figura 27 – Ênfase no Módulo de Memória Episódica do Controlador apresentado na figura 24.

- Do *Buffer* perceptual, essa informação vai para um *Codelet* de atenção episódica onde as Configurações são agrupadas aos pares e repassadas a um *Buffer* episódico
- Na terceira e última etapa do processo de aquisição, esses agrupamentos de Configurações recebem um processamento que deixa em evidência informações relevantes à sua posterior recuperação, como a ação responsável pela transição de configurações e uma avaliação da qualidade desta tomada de decisão, e são adicionadas definitivamente à Memória Episódica.

Já a recuperação de informações relevantes se dá por um critério de similaridade, aqui adotado como um percentual de semelhança de 80% entre a estrutura da Percepção Atual, definida como Configuração, e as memórias armazenadas. Esse método pode ser visto no pseudocódigo 4.1. O fluxo de informação correspondente a essa aquisição pode ser sintetizado da seguinte forma:

- Além de repassar a Configuração montada para o *Current Perception*, o *Attention Codelet* põe esta informação em um objeto de memória chamado *Cue Memory*, responsável por fornecer aos *Codelets* de recuperação de episódios relevantes um padrão a ser buscado na Memória Episódica.

- Com este padrão, o *Episodic Memory Retrieval Codelet*, seguindo um critério de similaridade mensurável, realiza uma recuperação de memória por conteúdo, escolhendo todos os Episódios onde a Configuração antes da tomada da ação assemelhe-se à percepção atual.

- Esta informação é, por último, repassada aos *Codelets* que a irão utilizar, como os de expectativas e planejamento para que estes possam fazer uso dos Episódios adequados de modo a aprimorar a performance geral. Na aplicação, desses Episódios são extraídas as ações tomadas em situações similares e as recompensas recebidas de fato são utilizadas como retificadoras de valores de expectativa.

Pseudocódigo 4.1 `evalSim (AbstractObject cue, AbstractObject epMemBefore)`

```
isSimilar = False
parts = 0
for cada componente na Configuração na Memória do
  if componente for um AbstractObject then
    for cada componente do
      similarPart = equals(cue.componente, epMemBefore.componente)
      if similarPart then
        parts++
      end if
    end for
    if parts dividido por n_total_partes ≥ threshold then
      isSimilar = True
    end if
  else
    if componente for textual and for exatamente igual ao equivalente na Percepção
    Atual then
      isSimilar = True
    else if componente for booleano and for exatamente igual ao equivalente na
    Percepção Atual then
      isSimilar = True
    else if componente for numérico and estiver dentro de uma margem de 10% do
    equivalente na Percepção Atual then
      isSimilar = True
    end if
  end if
end for
return isSimilar
```

Em termos de implementação, a Memória Episódica é uma estrutura do tipo *Memory* (conforme explicado no capítulo 3) que armazena internamente uma lista ordenada temporalmente, isto é, de acordo com o momento de aquisição do elemento. Esta lista é composta de outras estruturas do tipo *Memory* que guardam, cada uma, um Episódio.

Para o Episódio, foi adotada como estrutura uma implementação própria contendo algumas informações relevantes em um dado instante de tempo. Focada primariamente na tomada de ações mas não sendo limitada por ela, essa estrutura possui 4 elementos básicos: `Before e After` (Antes e Depois, em português), um par de Configurações (seguindo a Ontologia mostrada no capítulo 3, porém definida pelo usuário), `ActionTaken`, campo do tipo String indicando a ação responsável pela transição de Configurações e `Appraisal`, campo contendo uma estrutura que comporta valores quantitativos e/ou qualitativos que fornecem uma avaliação desta transição, de modo a classificar o quão boa ou ruim foi essa ação sob essas condições. Note que o campo relativo à ação tomada pode ser vazio, representando possivelmente transições em decorrência de ações tomadas no passado mas que ainda afetavam o mundo no momento descrito pelo Episódio. Na aplicação, a definição do `Apraisal` nada mais é do que a recompensa imediata recebida pelo agente após a tomada de ação, ou seja, a estrutura de recompensa recebida presente no `After`.

Para a recuperação, foi adotado um critério comparativo entre a estrutura da percepção atual (*Current Perception*) e o *Before* do Episódio. Nessa comparação, Episódios foram considerados similares aos atuais caso suas estruturas internas, que podem ser textuais, lógicas ou numéricas, fossem suficientemente parecidas: elementos textuais (do tipo *String*) e lógicos (*boolean*) deveriam ser exatamente iguais, enquanto elementos numéricos (*double*) deveriam estar dentro de uma margem ao redor do valor ao qual é comparado. Caso a similaridade supere um determinado valor (80%, como já mencionado), o episódio em questão é considerado similar à situação atual.

4.2.3 O Controlador do Malmo e seu funcionamento interno

O processo inicia com a coleta de informações fornecidas pela plataforma. As informações, de Grid, Status, Comandos e Recompensa, são recebidas em formato JSON e são transformadas em estruturas em OWRL separadas e enviadas à Memória Sensorial. Essas informações são adquiridas através dos *Handlers* de observação `ObservationFromGrid, ObservationFromFullStats e ObservationFromRecentCommands`, conforme explicado na seção 4.1. Em sequência, essas informações sensoriais são captadas pelos *Environment e Self Perceptual Codelets* e transformados em dois perceptos, o *SelfPercept e o EnvPercept*.

Como explicitado anteriormente, este trabalho foca no Sistema de Alta Cognição. Assim o único caminho a seguir por estes perceptos é o *S1to2 Attention Codelet*, responsável por montar a estrutura da Configuração, presente na figura 28, e torná-la disponível para o sistema em questão colocando-a na memória *Current Perception*. Essa Configuração, assim como ilustrada na figura em questão, possui um estrutura similar a um grafo, onde o nó raiz (*Configuration*) representa a o estado do mundo, como percebido

pelo agente, e seus nós filhos representam especificidades do mesmo. Assim, seguem dois nós, cada um representando um percepto e assim por diante até informações mais concretas como "vida", por exemplo, definida por um número em ponto flutuante. Uma vez no *Current Perception*, essa informação seguirá dois caminhos: através do Subsistema Episódico e para o Subsistema de Expectativas. No Subsistema Episódico seguirá conforme já explicado na subseção 4.2.2.

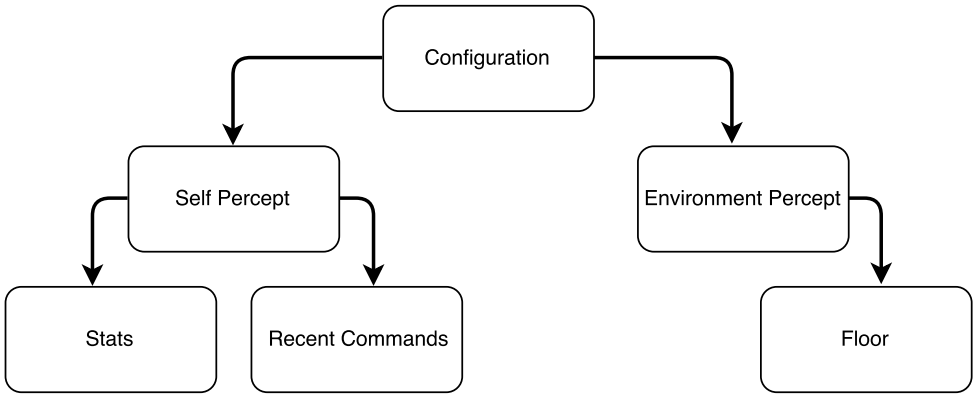

Figura 28 – Esquema de organização dos perceptos. Cada percepto possui ainda *Properties* não apontadas nessa imagem devido a quantidade

Seguindo pelo Subsistema de Expectativas, a informação acerca da situação atual (Configuração) é passada ao *Expectation Codelet*, onde então é codificada em decimais segundo um método fornecido pelo usuário e o trecho relevante para a aplicação, no caso os 9 valores representando o tipo de bloco do *Grid* 3×3 no qual o agente encontra-se no centro. Essa codificação se dá normalizando valores numéricos numa escala entre 0 e 198 (número total de tipos de blocos) para campos deste tipo e atribuindo valores (também seguindo esta escala) para campos categóricos segundo sua posição em um vetor com todas as categorias. A rede neural utilizada para gerar as expectativas é primeiramente atualizada através de um retreinamento com informações presentes no *Learning Codelet*, aprimorando assim seu comportamento. Os valores extraídos são então repassados à entrada da rede neural e quatro valores relativos às expectativas de recompensa das possíveis ações do agente são retornados pela mesma. Então, com informações de episódios similares recuperadas pelo *Episodic Memory Retrieval Codelet* (que são mais confiáveis), esses valores de expectativas são então corrigidos substituindo-se os valores devolvidos pela rede neural com valores de recompensa dos episódios em questão e uma tabela com as ações e respectivas recompensas segue para a próxima etapa. Note que, por mais que em um ambiente completamente explorado a rede neural perca sua utilidade, em ambiente parcialmente explorados (ou observáveis) ela permite a tomada de decisão baseada em informações anteriores e interpolações de estados não visitados.

Nos controladores sem o *Planner Codelet*, essa tabela vai diretamente para o *Selection Codelet*, onde um algoritmo que implementa a política $\epsilon - greedy$ com ϵ igual a

0.99 seleciona a ação com maior valor de expectativa. Essa ação então é passada ao *Next Action* que está diretamente conectado à entrada global dos *Motor Codelets*, ativando aquele correspondente à ação escolhida.

No caso com planejamento, nas situações em que o *Planner Codelet* esteja ativo, as informações de expectativas são ignoradas em favor da sequência de ações definida pelo componente. Dessa sequência, as ações são repassadas uma a uma, sequencialmente, para o *Selection Codelet*, de modo que essa única ação será a escolhida e repassada para o *Next Action* e então para os *Codelets* motores. Note que, caso este *Codelet* não esteja ativo, o fluxo usual de informações ocorrerá.

4.3 Resumo do Capítulo

Neste capítulo, introduzimos os elementos principais de nosso trabalho. O capítulo iniciou-se com uma introdução ao ambiente virtual Minecraft e à plataforma utilizada para experimentação. Foram mostradas em detalhes as possibilidades de modelagem de experimentos utilizando essa plataforma, o Projeto Malmo. Em seguida a estrutura da arquitetura proposta foi apresentada em detalhes, assim como o controlador criado a partir dela para os experimentos do capítulo seguinte. No próximo capítulo, apresentamos os resultados dos experimentos e uma análise desses resultados.

5 Metodologia e Experimentos

Neste capítulo, apresentamos maiores detalhes dos experimentos realizados, bem como seus resultados. Na seção 5.1 apresentamos as ideias por trás da concepção dos experimentos, bem como as métricas utilizadas para avaliar o desempenho do controlador e uma descrição de métodos alternativos utilizados para comparação. Na seção 5.1.1, apresentamos algumas considerações relevantes para o design dos experimentos. Os cenários utilizados nos experimentos encontram-se descritos na subseção 5.1.2. Na seção 5.2, apresentamos os experimentos e seus resultados, bem como uma discussão sobre esses resultados.

5.1 Metodologia

A Metodologia básica utilizada para os experimentos foi a de exploração do espaço de estados e verificação do aprendizado do agente através de indicadores de qualidade e velocidade de adaptação às condições encontradas no ambiente.

Conforme será explicado melhor na seção 5.2, cada experimento consiste em 10 rodadas de 50 execuções em um dado cenário e um dado controlador. Assim, o termo ”rodada” será aqui utilizado como o conjunto de 50 execuções e será utilizado ”execução” como, a partir das condições iniciais, o decorrer de uma única simulação (Missão).

A fim de validar a arquitetura e medir sua performance quando comparada a outras propostas, foram definidas métricas capazes de fornecer um panorama geral do aprendizado da criatura com cada um dos métodos apresentados. São essas:

- Número médio de vitórias por rodada: Métrica que aponta quantas vezes o agente foi capaz de atingir o objetivo preestabelecido da Missão. Devido ao fato de que existe a possibilidade da Missão ser finalizada por tempo, este valor avalia a capacidade de explorar corretamente o espaço disponível em busca de recompensas positivas.

- Tempo médio de vitória: Representa, em segundos, o tempo total necessário para que o agente atingisse o objetivo da Missão, nas execuções em que foi capaz de fazê-lo.

- Número de comandos até a vitória: Devido ao fato de os programas apresentarem estruturas e complexidades diferentes, nem sempre o tempo reflete completamente suas capacidades. Sendo assim, essa medida informa o número de comandos necessários para se atingir o objetivo, quando o mesmo foi completado com sucesso.

Acredita-se que estes critérios são adequados aos objetivos do experimentos, isto é, é possível observar uma adequação de comportamento mesmo em cenário com incertezas e aleatoriedades.

5.1.1 Considerações de Design de Experimento

Devido ao alto número de possibilidades de lidar com o agente e com o ambiente fornecidas pela plataforma utilizada, algumas considerações e definições precisaram ser feitas. Entre esses pontos, destacam-se:

- As simulações foram executadas com apenas um agente em detrimento a cenários com múltiplos agentes. O objetivo desses experimentos foi por a prova a eficiência da proposta. Um ambiente multi-agentes adicionaria questões que podem ser tratadas em trabalhos futuros.

- Mesmo sendo um mundo composto por blocos e ao contrário do isso possa sugerir, o Minecraft é fundamentalmente um mundo contínuo. Isto é, a mecânica original do ambiente não lida com o mesmo sendo "orientado a blocos"e sim de modo análogo ao mundo real. Com o objetivo de facilitar o tratamento do conceito de estados e lidar melhor com os comandos disponíveis, foi utilizado no *Handler* de comandos `DiscreteMovementCommands`, que basicamente discretiza o ambiente mantendo como coordenadas os pontos médios de cada bloco. Assim, o agente, durante os experimentos, enxerga o mundo como discreto.

- Das possibilidades de observações apresentadas no capítulo 4, foram escolhidas as seguintes:

  - `ObservationsFromFullStats`,
  - `ObservationsFromGrid` e
  - `ObservationsFromRecentCommands`

  Estas foram escolhidas por serem consideradas observações que forneciam as informações mais adequadas aos objetivos sem constituir “informação demais”, o que tornaria uma comparação com outros métodos inválida. Das observações de estado (”*full stats*”) foi utilizada apenas a recompensa. As outras observações foram utilizadas em sua totalidade.

- A Plataforma não fornece qualquer tipo de sincronia, nem mesmo da composição das Observações, isto é, suas partes são capturadas e estão disponíveis em momentos diferentes no tempo. Assim, para garantir a sincronia necessária para o funcionamento correto do sistema, vários *timers* e pontos de análise de sincronia foram colocados.

- As ideias apresentadas neste trabalho têm como principal foco o Sistema de Alta Cognição. Assim, foram deixados de lado comportamentos predefinidos, tais como "ir até um determinado ponto pelo menor caminho" ou "procurar" que, apesar de adicionar velocidade e bom desempenho, não seriam compatíveis com as demonstrações desejadas.

5.1.2 Cenários de Simulação

Foram explorados três cenários nos experimentos descritos a seguir.

5.1.2.1 Plataforma Contínua

Neste cenário mais simples, o agente parte de um bloco de granito (em cinza), na ponta esquerda de uma plataforma de arenito, rodeada por blocos de lava. Existe ainda um bloco de lápis-lazúli, também na margem esquerda da plataforma, mas mais ao final dela, que representa um ponto de alta recompensa, como pode ser visualizado na figura 29.

Figura 29 – Captura de Tela de um experimento com a Plataforma Contínua.

5.1.2.2 Plataforma Fragmentada

Este cenário apresenta uma estrutura similar à anterior, com a diferença de conter 10% de blocos de lava aleatoriamente colocados onde antes existiam blocos de arenito na plataforma. Esta situação, um pouco mais complexa, dificulta que o agente recaiam na escolha contínua de uma única ação. Aqui, cada rodada de experimentos irá apresentar uma configuração levemente diferente. Essa situação pode ser visualizada na figura 30. É importante ainda salientar que, na montagem dos experimentos, tomou-se o cuidado de não permitir a existência de situações onde não exista um caminho contínuo entre o ponto inicial e o bloco objetivo.

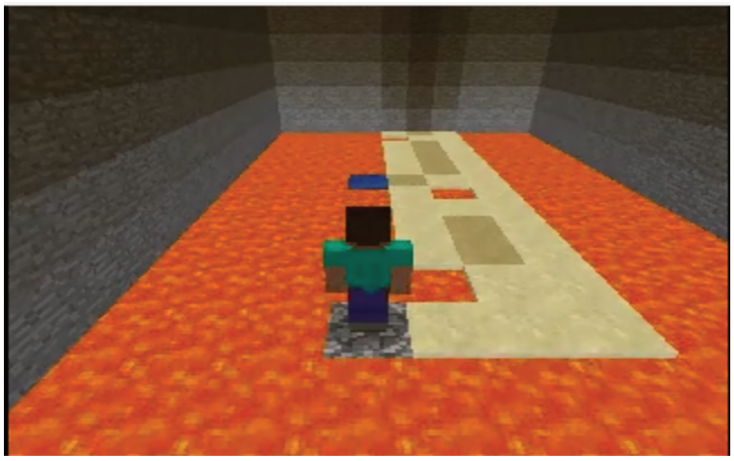

Figura 30 – Captura de Tela de um execução do experimento com a Plataforma Fragmentada. Note que os buracos na plataforma são determinados aleatoriamente e sua disposição varia de uma rodada para outra.

5.1.2.3 Plataforma Longa

Estrutura similar à plataforma contínua, porém maior, possuindo assim mais estados. O objetivo desta etapa é verificar como os métodos lidam com o aumento no número de estados. Essa situação pode ser visualizada na figura 31.

Figura 31 – Captura de Tela de um execução do experimento com a Plataforma Longa Contínua.

5.1.2.4 Plataforma Longa Fragmentada

Estrutura similar à plataforma fragmentada, porém maior. Assim como a etapa anterior, tem como objetivo verificar como os métodos lidam com o aumento no número de estados, porém em um ambiente levemente mais complexo. Essa situação pode ser visualizada na figura 32.

Figura 32 – Captura de Tela de um execução do experimento com a Plataforma Longa Fragmentada. Note que a disposição de buracos de lava varia de uma rodada pra outra.

5.2 Experimentos

Os experimentos se deram da seguinte forma:

- Cada rodada de simulações consiste de 50 execuções sequenciais, isto é, uma execução começa apenas quando a anterior termina, sob as mesmas condições de cenário. No entanto, o conhecimento adquirido em uma execução é repassado às execuções seguintes.
- O agente inicia sempre no mesmo ponto, em um bloco diferente dos outros, feito de granito ("Cobblestone").
- Como mencionado anteriormente, as informações de entrada do controlador são aquelas adquiridas através das Observações fornecidas pela plataforma Malmo, definidas no XML.
- Dentre as ações possíveis encontram-se *"move 1"*, *"move -1"*, *"strafe 1"* e *"strafe -1"*, ações discretas que seguem o modelo descrito no capítulo 4.
- O agente recebe uma recompensa negativa (-1) sempre que cair em um bloco de lava e uma recompensa positiva (+1) sempre que tocar um bloco de Lápis Lazúli. Todos os outro blocos são indiferentes.
- A execução finaliza, caso o tempo máximo seja excedido (60 segundos), o agente caia em um bloco de lava ou atinja o bloco especial de Lápis Lazúli.

Devido à natureza estocástica em alguns pontos do processo de decisão, tanto para o controlador proposto quanto para o sistema de aprendizado por reforço (como, por exemplo, a escolha completamente aleatória entre ações de mesmo valor de recompensa

esperado), cada rodada de simulações, que consiste em 50 execuções com uma versão do controlador em um determinado cenário, foi executada 10 vezes a fim de se obter uma significância estatística nos experimentos. Note que, apesar de haver acúmulo de conhecimento entre 2 execuções de uma mesma rodada, este conhecimento não é transferido entre uma rodada e outra.

Para cada um dos cenários apresentados na seção anterior, foram comparados os seguintes controladores:

- Controlador de Referência: baseado em Aprendizado por reforço com uma tabela Q, representando um método clássico de tomada de decisão, com convergência garantida. Essa tabela Q possui todos os valores iniciais em zero e cada campo é atualizado apenas uma vez, na ocasião da primeira visita do agente aquele par estado-ação.

- Controlador cognitivo puramente baseado em expectativas: modelo de controlador cujas decisões são simplesmente baseadas em uma estrutura de reforço com uma rede neural como aproximador para a função Q.

- Controlador Cognitivo com Memória Episódica: similar ao anterior, porém utilizando a Memória Episódica para aprimorar o processo de tomada de decisão, retificando os valores de expectativa produzidos pela rede neural.

- Controlador Cognitivo com Exploração: o mesmo que o anterior, incorporando o aproximador para a função Q e a Memória Episódica, porém, com uma dada porcentagem, o agente decide por explorar situações novas em detrimento de outras para as quais já tem conhecimento consolidado. Esse controlador foi simulado com 50% e 100% de situações de exploração.

- Controlador Cognitivo completo: além dos módulos apresentados nos itens anteriores, esta versão consegue guardar e otimizar parcialmente a sequência de ações que leva a um maior acúmulo de pontos de recompensa positiva, conforme explicado no capítulo 4. Como forma de obter-se um compromisso entre descoberta e benefício, o agente opta por seguir esse "plano" em 50% dos casos e por explorar novas possibilidades nos casos restantes (ou seja, quando não decide seguir o plano, ele se comporta como no caso do controlador com exploração).

Durante a realização dos experimentos, diversos problemas de assincronia não esperada na devolução das informações de percepção dadas pela Plataforma Malmo foram encontrados. Isso ocasionou alguns erros e paradas momentâneas súbitas nas ações do agente. Porém, devido à quantidade de execuções realizadas (10 rodadas com 50 execuções cada, para cada controlador), este fator foi mitigado e não deve apresentar grandes

discrepâncias de experimentos ideais, sem erros. Assim, pode-se considerar como resolvida esta questão.

Os resultados obtidos são apresentados e discutidos na subseção seguinte.

5.2.1 Resultados

Nos gráficos apresentados a seguir, são mostrados os resultados obtidos nos experimentos. Neles, cada controlador é apresentado por uma cor e uma legenda fixa, mantidas ao longo de todas as imagens. São elas:

- Controlador de Referência: Mostrado em cinza, sob a legenda de *simpleRL*.
- Controlador cognitivo puramente baseado em expectativas: Mostrado em roxo, sob a legenda de *wandercogSimple*.
- Controlador Cognitivo com Memória Episódica: Mostrado em azul, sob a legenda de *EpMem*.
- Controlador Cognitivo com Exploração: Mostrado em oliva, sob a legenda de *EpMemExploration*.
- Controlador Cognitivo completo: Mostrado em verde, sob a legenda de *Full*.

5.2.1.1 Plataforma Contínua

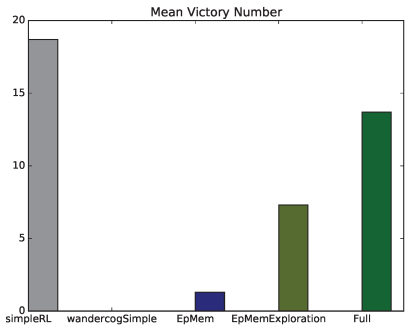

Figura 33 – Número médio de vitórias por rodada para o cenário da Plataforma Contínua. Neste gráfico quanto maior o valor, melhor é o resultado.

As figuras 33 a 35 mostram os resultados obtidos nos experimentos com plataforma contínua.

No primeiro gráfico (fig. 33), nota-se que a simplicidade do cenário favorece uma exploração mais rápida, como evidenciado pela vitória do Aprendizado por Reforço simples. Porém, no terceiro gráfico (fig. 35), vê-se uma discrepância entre o Aprendizado

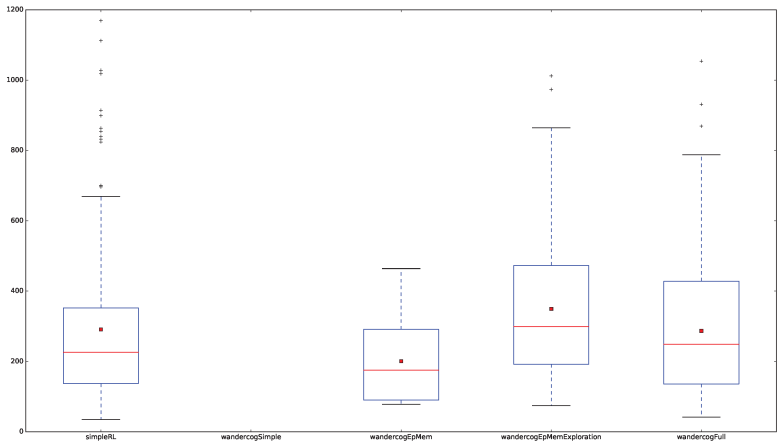

Figura 34 – Distribuição dos dados em um gráfico *boxplot* dos tempos (em *ticks* internos do Malmo) necessários para o agente atingir o bloco especial de lápis lazúli na Plataforma Contínua, quando o faz. As caixas representam respectivamente os controladores 'simpleRL', 'cogSimple', 'EpMem', 'EpMemExplorarion' e 'Full'. Os pontos vermelhos no meio da distribuição representam a média dos valores. Aqui, quanto menor é o valor melhor é o resultado.

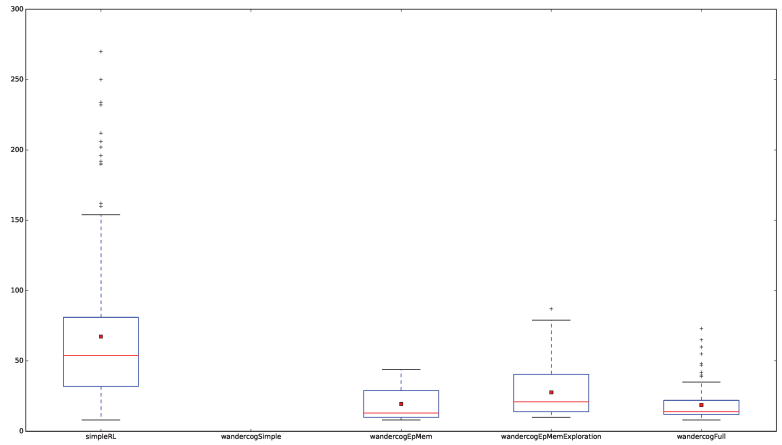

Figura 35 – Distribuição dos dados em um gráfico *boxplot* do número de comandos necessários para que o agente atinja o bloco especial na Plataforma Contínua, quando o faz. As caixas representam respectivamente os controladores 'simpleRL', 'cogSimple', 'EpMem', 'EpMemExplorarion' e 'Full'. Os pontos vermelhos no meio da distribuição representam a média dos valores. Um valor menor representa um resultado melhor.

por Reforço e os controladores propostos.Assim, apesar de ficar desde já clara a maior eficiência em termos de número de ações tomadas, para cenários simples como o destes experimentos, uma estratégia mais simples porém mais veloz se mostra mais efetiva.

5.2.1.2 Plataforma Fragmentada

Neste cenário, cujos resultados são apresentados nas figuras 36 a 38, a presença de um pouco mais de complexidade no ambiente, na forma de fragmentação aleatória da plataforma, favorece modelos com maior capacidade de generalização, que conseguem se adaptar de forma mais eficiente. No terceiro gráfico (fig. 37) a diferença de número de comandos quando comparado ao cenário anterior é reforçada.

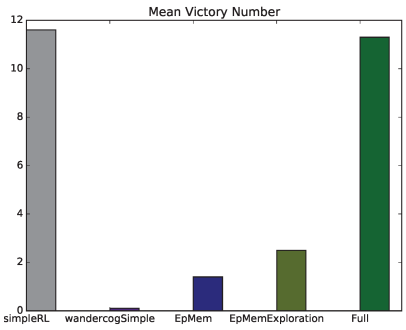

Figura 36 – Número médio de vitórias por rodada para o cenário da Plataforma Fragmentada. Neste gráfico quanto maior o valor, melhor é o resultado.

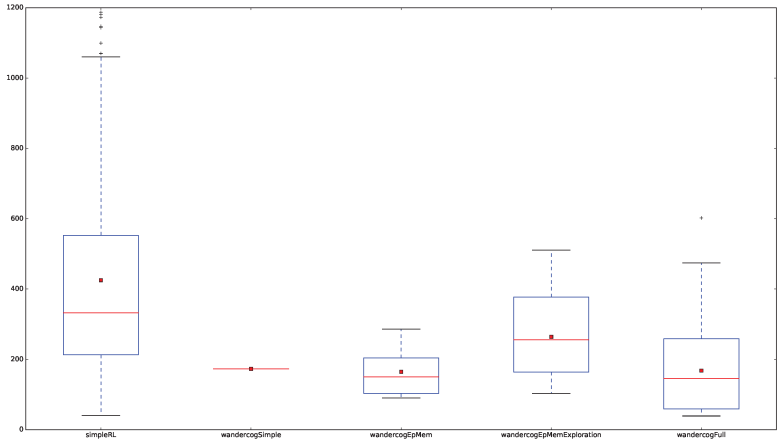

Figura 37 – Distribuição dos dados em um gráfico *boxplot* dos tempos (em *ticks* internos do Malmo) necessários para o agente atingir o bloco especial de lápis lazúli na Plataforma Fragmentada, quando o faz. As caixas representam respectivamente os controladores 'simpleRL', 'cogSimple', 'EpMem', 'EpMemExplorarion' e 'Full'. Os pontos vermelhos no meio da distribuição representam a média dos valores. Aqui, quanto menor é o valor melhor é o resultado.

5.2.1.3 Plataforma Longa

As figuras 39 a 41 mostram os resultados obtidos nos experimentos com plataforma longa contínua. Neste cenário fica evidente que, para um número razoavelmente

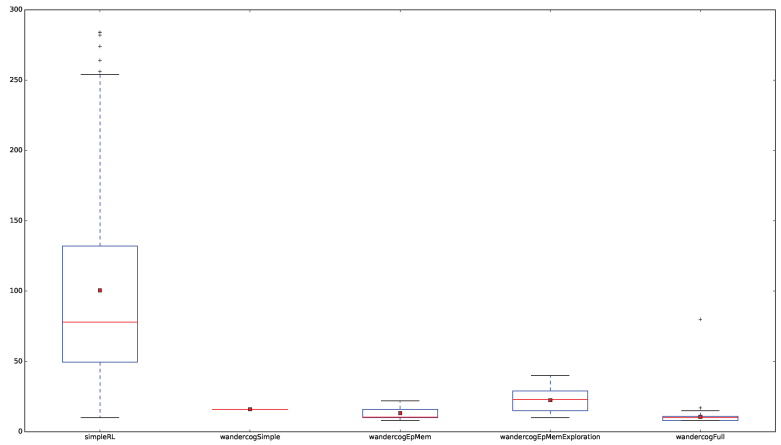

Figura 38 – Distribuição dos dados em um gráfico *boxplot* do número de comandos necessários para que o agente atinja o bloco especial na Plataforma Fragmentada, quando o faz. As caixas representam respectivamente os controladores 'simpleRL', 'cogSimple', 'EpMem', 'EpMemExplorarion' e 'Full'. Os pontos vermelhos no meio da distribuição representam a média dos valores, Um valor menor representa um resultado melhor.

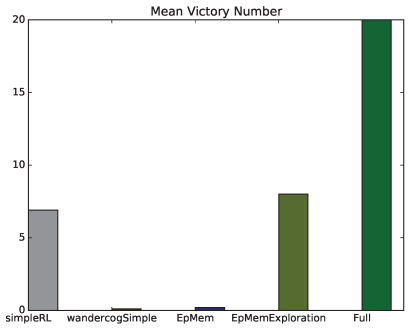

Figura 39 – Número médio de vitórias por rodada para o cenário da Plataforma Longa Contínua. Neste gráfico quanto maior o valor, melhor é o resultado. Note que apesar de similar a Plataforma Contínua o mero aumento no número de estados já favorece métodos um pouco mais generalizados que tratem de questões com exploração.

maior de estados, sendo similar a Plataforma Contínua em outros aspectos, métodos um pouco mais generalizados que tratam de questões como exploração apresentam melhores resultados, sendo o controlador completo proposto aquele com melhor desempenho e eficiência em termos de número de ações.

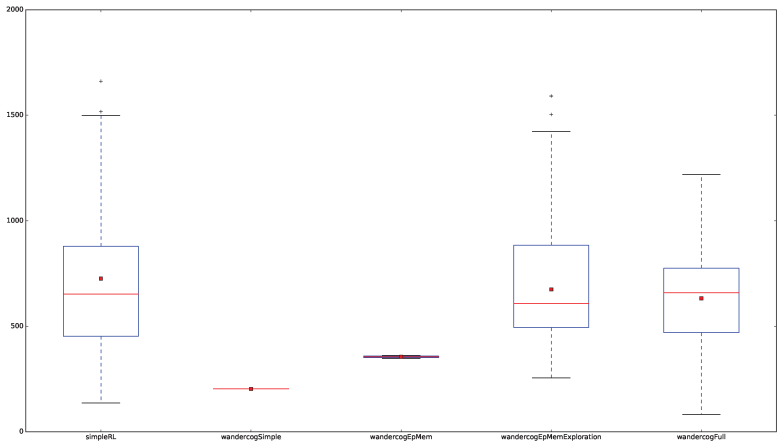

Figura 40 – Distribuição dos dados em um gráfico *boxplot* dos tempos (em *ticks* internos do Malmo) necessários para o agente atingir o bloco especial de lápis lazúli na Plataforma Longa Contínua, quando o faz. As caixas representam respectivamente os controladores 'simpleRL', 'cogSimple', 'EpMem', 'EpMemExplorarion' e 'Full'. Os pontos vermelhos no meio da distribuição representam a média dos valores. Aqui, quanto menor é o valor melhor é o resultado.

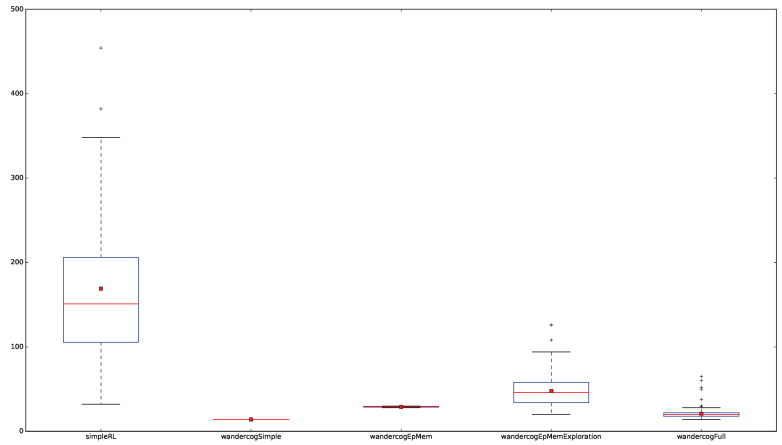

Figura 41 – Distribuição dos dados em um gráfico *boxplot* do número de comandos necessários para que o agente atinja o bloco especial na Plataforma Longa Contínua, quando o faz. As caixas representam respectivamente os controladores 'simpleRL', 'cogSimple', 'EpMem', 'EpMemExplorarion' e 'Full'. Os pontos vermelhos no meio da distribuição representam a média dos valores, Um valor menor representa um resultado melhor.

5.2.1.4 Plataforma Longa Fragmentada

As figuras 42 a 44 mostram os resultados obtidos nos experimentos com plataforma longa fragmentada. Neste cenário, mesmo reduzindo a performance de todos os

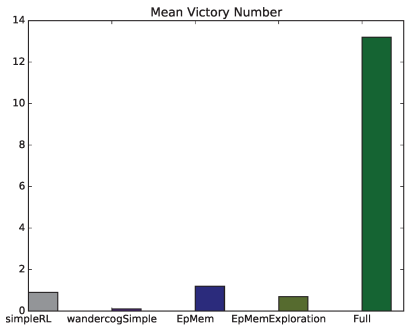

Figura 42 – Número médio de vitórias por rodada para o cenário da Plataforma Longa Fragmentada. Neste gráfico quanto maior o valor, melhor é o resultado.

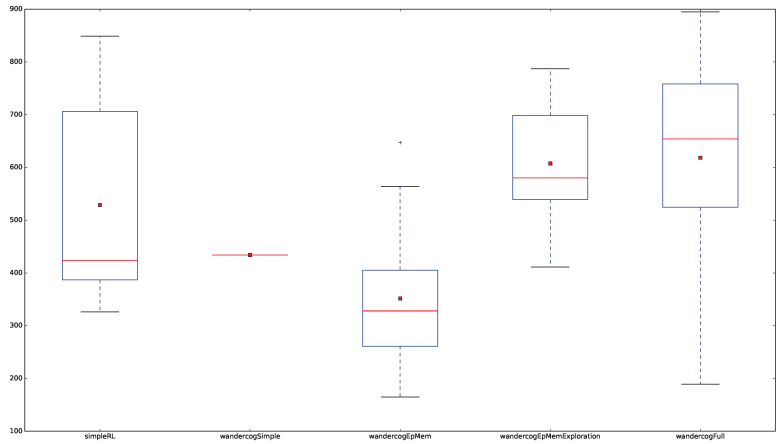

Figura 43 – Distribuição dos dados em um gráfico *boxplot* dos tempos (em *ticks* internos do Malmo) necessários para o agente atingir o bloco especial de lápis lazúli na Plataforma Longa Fragmentada, quando o faz. As caixas representam respectivamente os controladores 'simpleRL', 'cogSimple', 'EpMem', 'EpMemExplorarion' e 'Full'. Os pontos vermelhos no meio da distribuição representam a média dos valores. Aqui, quanto menor é o valor melhor é o resultado.

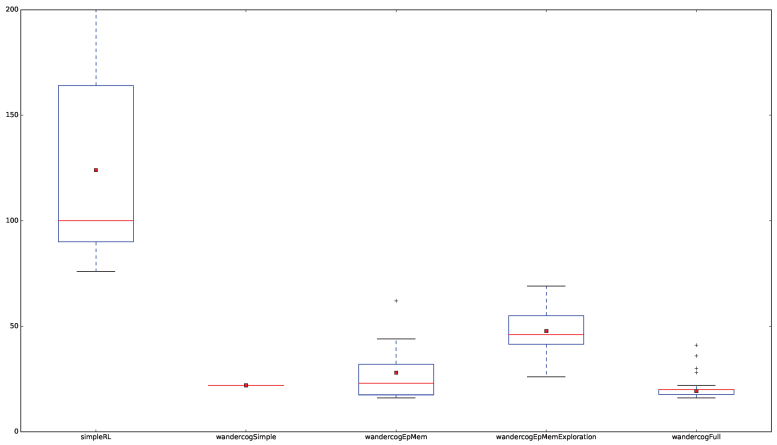

Figura 44 – Distribuição dos dados em um gráfico *boxplot* do número de comandos necessários para que o agente atinja o bloco especial na Plataforma Longa Fragmentada, quando o faz. As caixas representam respectivamente os controladores 'simpleRL', 'cogSimple', 'EpMem', 'EpMemExplorarion' e 'Full'. Os pontos vermelhos no meio da distribuição representam a média dos valores, Um valor menor representa um resultado melhor.

controladores, o maior número de estados e leve aumento de complexidade favorecem os métodos propostos e, novamente, controlador completo apresenta os melhores resultados, com grande discrepância entre o mesmo e o Aprendizado por Reforço.

5.2.2 Discussão

Conforme pode-se notar nos gráficos apresentados, o controlador proposto, em sua versão completa, apresenta um resultado conciso, superior em eficiência no tempo e quantidade de comandos desde os cenários mais básicos e, apesar de perder em número de vitórias em cenários simples, escala de forma muito mais adequada (em relação à referência) conforme o ambiente se torna mais complexo.

No primeiro cenário (plataforma contínua), o número de pares estado-ação relativamente baixo e a quantidade reduzida de pontos de recompensas ruins favorece uma exploração rápida, ainda que não apresente uma estratégia muito melhor que a aleatória ($\epsilon - greedy$), enquanto estratégias que demandem mais tempo e apresentem maior nível de generalidade não se deem tão bem como a mera verificação em tabela. Mesmo assim, o número de comandos necessários para a vitória apresentou distribuição de valores muito menos dispersa nos controladores propostos do que no controlador referência.

No segundo cenário, que apresenta mais pontos de recompensa negativa e maior complexidade, a exploração simples baseada na recompensa imediata de uma vizinhança

curta do espaço de estados mostra-se um pouco menos efetiva em lidar com o objetivo. Já o controlador completo, munido de recursos diversos, apresenta o mesmo número de vitórias que a referência, mas atingindo o objetivos em um tempo menor e com uma quantidade de ações mais baixa. Note ainda que, novamente, a dispersão do número de comandos para a vitória é bem estreita, padrão que se repete nos outros dois cenários.

O simples aumento do número de estados, mesmo que o ambiente mantenha-se essencialmente o mesmo, causou uma queda brusca na performance do método mais simples de aprendizado e, como pode-se notar na figura 39 o uso de memória episódica, devidamente ajustado para lidar com a exploração do espaço, foi capaz de fornecer um resultado superior ao Aprendizado por Reforço e, quando combinado com o módulo de planos, foi capaz de superar em muito o método de referência. Esta constatação foi reforçada no quarto cenário, no qual a performance do método de referência foi fortemente degradada, enquanto o controlador completo apresentou resultados satisfatórios.

Ponderando-se todos esses pontos, nota-se que, de acordo com a literatura (COULOM, 2002), a estabilidade de sistemas com Aprendizado por reforço com graus de generalidade é uma questão que não possui solução geral e que deve ser olhada caso a caso. Isto pode ser constatado no baixo desempenho do controlador cognitivo simples que, apesar de aprender a evitar decisões muito ruins, não foi capaz de atingir o objetivo na maioria dos caso. A mera adição de um sistema de Memória Episódica permite que o controlador supere em parte as limitações de sua versão anterior, apesar de ainda cair no problema de uma exploração não adequada do ambiente. O uso de um sistema de exploração, favorecendo parcialmente estados não conhecidos, melhorou a quantidade de vitórias do agente ao custo de mais tempo e um maior número de comandos. Por fim, a habilidade de guardar ações, gerar e seguir planos e mesmo assim explorar o ambiente apresentou bons resultados mesmo frente a cenários mais complexos.

Comparando o trabalho desenvolvido com a literatura, a arquitetura proposta consegue reunir diversos elementos cujo a combinação não foi encontrada em outros trabalhos. A proposta apresentada de Memória Episódica assemelha-se bastante àquela exposta em Sun (2015) com a diferença de que o campo "recompensa" é aqui substituído por uma entidade *Appraisal*, que permite, além de recompensas numéricas, apresentar informações adicionais, como uma avaliação qualitativa. O mecanismo de recuperação de informação por similaridade também, em menor nível, pode ser comparado ao esquema de Árvore de Generalização presente no ICARUS, onde os episódios são agrupados de acordo com similaridade. Assim, o processo de *recall* devolve múltiplos episódios, todos apresentando similaridade entre si. Ainda sobre a Memoria Episódica, similarmente ao SOAR e diferentemente do LIDA e CLARION, não adota-se nenhum tipo de decaimento temporal, sendo então um registro permanente.

Quanto à questão do Aprendizado, o uso do modelo com política fixa e aproxi-

mador neural pra Valores-Q constitui-se uma abordagem não totalmente explorada na literatura, possivelmente devido a sensibilidade do sistema, mas não apresenta grandes saltos de paradigma.

Por fim, os trabalhos relacionados envolvendo o ambiente do Minecraft focam-se majoritariamente no Aprendizado Profundo e captação de informação diretamente dos *pixels*, uma abordagem diferente da apresentada. Entre os trabalhos que se assemelham, encontram-se Oh *et al.* (2016) e Siljebråt (2015), o que reforça a amplitude da arquitetura na medida em que engloba e sistematiza diversas técnicas e teorias de uma forma capaz de ser facilmente reutilizada em outras tarefas.

5.3 Resumo do Capítulo

Dedicado inteiramente em relatar os experimentos realizados, este capítulo iniciou-se com uma apresentação dos objetivos dos experimentos e todas as considerações e decisões tomadas em suas montagens. Em seguida os cenários e controladores utilizados foram apresentados. Por fim os resultados e uma discussão dos mesmos fechou esta parte do trabalho. No próximo capítulo, apresentamos as conclusões finais do trabalho, bem como os trabalhos futuros.

Conclusão

Esta Dissertação de Mestrado teve como objetivo a proposição de um modelo estrutural de cognição que ao mesmo tempo apresentasse inspirações biológicas e fosse computacionalmente capaz de exibir comportamento complexo e inteligente e aprendizado eficiente, mesmo em situações generalizadas ou mais complexas, e estivesse em acordança com técnicas bem estabelecidas da literatura.

Para tal, apoiou-se em um conjunto de teorias presentes na literatura, tanto em termos de ideias já bem estabelecidas como aquelas ainda pouco exploradas. Buscou-se analisar o uso de técnicas de aprendizado de máquina e inteligência computacional, em geral no auxílio da adaptação de comportamento em um agente inteligente, como aprendizado por reforço e redes neurais. Foi usado para isso um ambiente virtual com rica complexidade em termos de dados sensoriais disponíveis ao agente.

Pode-se perceber que casos simples são melhor tratados com estratégias simples porém velozes, explorando o espaço de estados rapidamente. Notou-se também que, assim como previsto, mesmo uma adição baixa de generalidade pode comprometer a estabilidade de predição do sistema. Este e outros problemas, como o compromisso entre o uso de conhecimento já incorporado e a exploração do ambiente para aquisição de novos conhecimentos, foram progressivamente tratados adicionando-se recursos eficientes mas que ainda mantinham inspiração biológica.

Em situações com maior grau de complexidade, um maior grau de generalidade mostrou-se mais interessante, na medida em que conseguia lidar com situações menos favoráveis e mais complexas. Mostrou-se ainda os benefícios que o uso de Memória Episódica pode trazer para um agente virtual.

Em termos comparativos com outros trabalhos presentes na literatura, a arquitetura proposta alinha em um único *framework* diversas teorias e técnicas presentes em outros trabalhos e mostra uma implementação satisfatória dos mecanismos ao qual se dispôs a trabalhar, possuindo assim potencial competitivo com as arquiteturas existentes hoje.

Conclui-se que a arquitetura proposta é adequada como base para o design de controladores de sistemas inteligentes, sendo capaz de lidar com questões que algoritmos clássicos não lidam de maneira satisfatória. Conclui-se também que o uso de sistemas de Memória Episódica como retificadores em predição mostra-se eficiente para processos de tomada de decisão, inclusive sob incerteza, à medida em que consegue suprir certas deficiências presentes em métodos puramente baseados em predição numérica que, por

sua vez, representam alternativas a métodos baseados em tabelas.

Trabalhos Futuros

Desde que foi idealizada, a Arquitetura Cognitiva proposta sofreu diversas alterações, algumas delas devido a problemas encontrados ao longo do trabalho. Infelizmente, o tempo disponível para o desenvolvimento de um trabalho de mestrado é bastante exíguo, o que impediu que pudéssemos tratar de todos os pontos que originalmente nos inspiraram, visto que muitas das ideias consideradas ainda não puderam ser exploradas. Assim, parte dos primeiros trabalhos futuros recaem justamente em expandir a Arquitetura para comportar tais ideias:

- Compilação de comportamentos em tempo de execução: Mencionada de passagem no item que descreve o *Adaptative Codelet*, esta funcionalidade ainda não observada em outras propostas significaria uma maior e mais completa interação entre os dois grandes subsistemas cognitivos. A ideia é justamente criar (e eventualmente modificar e destruir) comportamentos no Sistema de Baixa Cognição que apresentem resultados similares àqueles conseguidos custosamente pelo Sistema de Alta Cognição, desonerando-o e "automatizando" determinado comportamento reativo.
- Formalização do subsistema de planejamento. A ideia aqui é fornecer um protocolo que permita ao usuário da arquitetura um uso sem preocupações com o funcionamento interno do processo de planejamento.

Além disso, ao longo do trabalho, diversas decisões de design foram tomadas. Devido ao custo em tempo de se explorar as vias alternativas àquilo que foi decidido, essas outras técnicas foram excluídas completamente. Deseja-se então explorar técnicas alternativas às presentes nesta dissertação:

- Outros modelos de redes neurais: No módulo de expectativas, foi escolhido como modelo de rede a MLP por ser simples, bem conhecida e apresentar resultados satisfatórios para determinadas classes de mapeamentos. Existe, porém, uma vasta gama de possibilidades de rede, como *Extreme Learning Machines* e *Echo State Networks*, muitas das quais não foram exploradas no presente trabalho.
- Codificação. Um dos pontos críticos em alguns pontos do projeto, como o processo de armazenamento e recuperação de informação na Memória Episódica e entrada da rede neural, uma codificação ruim pode afetar negativamente um método enquanto uma codificação boa pode salvá-lo. Deseja-se então, explorar diferentes formas de codificar a informação.

Pretende-se ainda utilizar a arquitetura para outras aplicações, como controle de tráfego urbano e controle de robôs.

Referências

ABEL, D.; AGARWAL, A.; DIAZ, F.; KRISHNAMURTHY, A.; SCHAPIRE, R. E. Exploratory gradient boosting for reinforcement learning in complex domains. *arXiv preprint arXiv:1603.04119*, 2016. Citado na página 40.

ALURU, K.; TELLEX, S.; OBERLIN, J.; MACGLASHAN, J. Minecraft as an experimental world for ai in robotics. In: . [S.l.: s.n.], 2015. Citado 2 vezes nas páginas 41 e 60.

ANDERSON, J. R. A theory of the origins of human knowledge. *Artificial intelligence*, Elsevier, v. 40, n. 1-3, p. 313–351, 1989. Citado na página 29.

ANDERSON, J. R. *How can the human mind occur in the physical universe?* [S.l.]: Oxford University Press, 2009. Citado na página 29.

ANDERSON, J. R.; BOTHELL, D.; BYRNE, M. D.; DOUGLASS, S.; LEBIERE, C.; QIN, Y. An integrated theory of the mind. *Psychological review*, American Psychological Association, v. 111, n. 4, p. 1036, 2004. Citado 3 vezes nas páginas , 29 e 30.

ARKIN, R. C. Behavior-based robotics (intelligent robotics and autonomous agents). The MIT Press, 1998. Citado na página 46.

BAARS, B. J. *A Cognitive Theory of Consciousness*. [S.l.]: Cambridge University Press, 1988. ISBN 0-521-30133-5. Citado na página 43.

BAARS, B. J.; FRANKLIN, S. An architectural model of conscious and unconscious brain functions: Global workspace theory and ida. *Neural Networks*, Elsevier, v. 20, n. 9, p. 955–961, 2007. Citado 2 vezes nas páginas 43 e 45.

BARRON, T.; WHITEHEAD, M.; YEUNG, A. Deep reinforcement learning in a 3-d blockworld environment. In: . [S.l.]: Deep Reinforcement Learning: Frontiers and Challenges, IJCAI, 2016. Citado 4 vezes nas páginas , 37, 38 e 60.

BARSALOU, L. W. Grounded cognition: Past, present, and future. *Topics in cognitive science*, Wiley Online Library, v. 2, n. 4, p. 716–724, 2010. Citado na página 46.

BAYLISS, J. D. Teaching game ai through minecraft mods. In: *2012 IEEE International Games Innovation Conference*. [S.l.: s.n.], 2012. p. 1–4. ISSN 2166-6741. Citado na página 60.

BISTRICKY, S. L. Mill and mental phenomena: Critical contributions to a science of cognition. *Behavioral Sciences*, Multidisciplinary Digital Publishing Institute, v. 3, n. 2, p. 217–231, 2013. Citado na página 18.

BONNANO, D.; ROBERTS, M.; SMITH, L.; AHA, D. Selecting subgoals using deep learning in minecraft: A preliminary report. In: *Working Notes of the IJCAI-16 Workshop on Deep Learning and Artificial Intelligence*. NYC, NY, USA.: [s.n.], 2016. Disponível em: <publications/bonnanoEtAl16.dlaiws.selectingSubgoals.pdf>. Citado 2 vezes nas páginas 38 e 60.

BOTTOU, L. Stochastic gradient descent tricks. In: _____. *Neural Networks: Tricks of the Trade: Second Edition*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 421–436. ISBN 978-3-642-35289-8. Disponível em: <https://doi.org/10.1007/978-3-642-35289-8_25>. Citado na página 23.

CAI, J.; RUAN, X. Self-balance control of inverted pendulum based on fuzzy skinner operant conditioning. In: *2009 International Conference on Information Technology and Computer Science*. [S.l.: s.n.], 2009. v. 2, p. 518–521. Citado 2 vezes nas páginas 26 e 37.

CASTRO, E. C.; GUDWIN, R. R. A scene-based episodic memory system for a simulated autonomous creature. *International Journal of Synthetic Emotions*, IGI Global, v. 4, n. 1, p. 32–64, jan 2013. Disponível em: <https://doi.org/10.4018%2Fjse.2013010102>. Citado 3 vezes nas páginas 32, 33 e 61.

CHALITA, M. A.; LIS, D.; CAVERZASI, A. Reinforcement learning in a bio-connectionist model based in the thalamo-cortical neural circuit. *Biologically Inspired Cognitive Architectures*, v. 16, n. Supplement C, p. 45 – 63, 2016. ISSN 2212-683X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2212683X16300159>. Citado na página 37.

CHEN, C.; LI, H. X.; DONG, D. Hybrid control for robot navigation - a hierarchical q-learning algorithm. *IEEE Robotics Automation Magazine*, v. 15, n. 2, p. 37–47, June 2008. ISSN 1070-9932. Citado na página 27.

COULOM, R. *Apprentissage par renforcement utilisant des réseaux de neurones, avec des applications au contrôle moteur*. Tese (Doutorado) — Institut National Polytechnique de Grenoble-INPG, 2002. Citado na página 92.

DENOYELLE, N.; CARRERE, M.; POUGET, F.; VIÉVILLE, T.; ALEXANDRE, F. From biological to numerical experiments in systemic neuroscience: a simulation platform. In: *Advances in Neurotechnology, Electronics and Informatics*. [S.l.]: Springer, 2016. p. 1–17. Citado na página 40.

DERBINSKY, N.; LAIRD, J. E. Efficiently implementing episodic memory. In: _____. *Case-Based Reasoning Research and Development: 8th International Conference on Case-Based Reasoning, ICCBR 2009 Seattle, WA, USA, July 20-23, 2009 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 403–417. ISBN 978-3-642-02998-1. Disponível em: <http://dx.doi.org/10.1007/978-3-642-02998-1_29>. Citado na página 32.

FRANKLIN, S. *Artificial minds*. 1. ed. [S.l.]: The MIT Press, 1995. v. 1. ISBN 0-262-06178-3. Citado na página 60.

FRANKLIN, S.; GRAESSER, A. Is it an agent, or just a program?: A taxonomy for autonomous agents. In: MÜLLER, J. P.; WOOLDRIDGE, M. J.; JENNINGS, N. R. (Ed.). *Intelligent Agents III Agent Theories, Architectures, and Languages*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997. p. 21–35. ISBN 978-3-540-68057-4. Citado na página 19.

FRANKLIN, S.; KELEMEN, A.; MCCAULEY, L. Ida: A cognitive agent architecture. In: IEEE. *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*. [S.l.], 1998. v. 3, p. 2646–2651. Citado na página 43.

FRANKLIN, S.; MADL, T.; D'MELLO, S.; SNAIDER, J. Lida: A systems-level architecture for cognition, emotion, and learning. *Autonomous Mental Development, IEEE Transactions on*, IEEE, v. 6, n. 1, p. 19–41, 2014. Citado 2 vezes nas páginas 42 e 43.

FRASCONI, P.; GORI, M.; TESI, A. Successes and failures of backpropagation: A theoretical. *Progress in Neural Networks: Architecture*, Intellect Books, v. 5, p. 205, 1997. Citado na página 28.

GÄRDENFORS, P. *The geometry of meaning: Semantics based on conceptual spaces*. [S.l.]: MIT Press, 2014. Citado na página 46.

GAUDIANO, P.; CHANG, C. Adaptive obstacle avoidance with a neural network for operant conditioning: experiments with real robots. In: IEEE. *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*. [S.l.], 1997. p. 13–18. Citado na página 26.

GAUDIANO, P.; ZALAMA, E.; CHANG, C.; CORONADO, J. L. A model of operant conditioning for adaptive obstacle avoidance. *From Animals to Animats*, v. 4, p. 373–381, 1996. Citado na página 26.

GEIGER, P.; HOFMANN, K.; SCHöLKOPF, B. Experimental and causal view on information integration in autonomous agents. In: *6th International Workshop on Combinations of Intelligent Methods and Applications*. [s.n.], 2016. p. 21–28. Disponível em: <https://www.microsoft.com/en-us/research/publication/experimental-causal-view-information-integration-autonomous-agents/>. Citado na página 60.

GOERTZEL, B.; PENNACHIN, C.; GEISWEILLER, N. Brief survey of cognitive architectures. In: *Engineering General Intelligence, Part 1*. [S.l.]: Springer, 2014. p. 101–142. Citado na página 30.

GOERTZEL, B.; PENNACHIN, C.; GEISWEILLER, N. *Engineering General Intelligence, Part 2: The CogPrime Architecture for Integrative, Embodied AGI*. Atlantis Press, 2014. (Atlantis Thinking Machines). ISBN 9789462390294. Disponível em: <https://books.google.com.br/books?id=a9-RngEACAAJ>. Citado na página 44.

GOSAVI, A. *Simulation-Based Optimization*. Springer US, 2015. Disponível em: <https://doi.org/10.1007/978-1-4899-7491-4>. Citado 3 vezes nas páginas , 21 e 27.

GOTTFREDSON, L. S. Mainstream science on intelligence: An editorial with 52 signatories, history, and bibliography. *Intelligence*, Elsevier BV, v. 24, n. 1, p. 13–23, jan 1997. Disponível em: <https://doi.org/10.1016/s0160-2896(97)90011-8>. Citado na página 18.

GROSS, R. *Psychology: The Science of Mind and Behaviour*. Hodder Education, 2010. ISBN 9781444108316. Disponível em: <https://books.google.com.br/books?id=QIQfQAAACAAJ>. Citado 2 vezes nas páginas 18 e 23.

GUDWIN, R.; PARAENSE, A.; PAULA, S. M. de; FRÓES, E.; GIBAUT, W.; CASTRO, E.; FIGUEIREDO, V.; RAIZER, K. The multipurpose enhanced cognitive architecture (meca). *Biologically Inspired Cognitive Architectures*, Elsevier, v. 22, p. 20–34, 2017. Citado 7 vezes nas páginas , 16, 46, 47, 48, 49 e 60.

HAYKIN, S. *Neural Networks: A Comprehensive Foundation*. 2nd. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998. ISBN 0132733501. Citado 2 vezes nas páginas 20 e 28.

HOFSTADTER, D.; MITCHELL, M. The Copycat Project: A Model of Mental Fluidity and Analogy-making. In: _____. *Fluid Concepts And Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*. 10, East 53rd Street, New York, NY 10022-5299: BasicBooks, 1994. p. 205–268. Citado na página 42.

HONG-GE, R.; XIAO-GANG, R. Self-learning of robot based on skinner's operant conditioning. In: *2009 International Conference on Information Technology and Computer Science*. [S.l.: s.n.], 2009. v. 1, p. 175–178. Citado 2 vezes nas páginas 26 e 37.

IAN, C. W.; ONG, Y.-S.; FERNANDO, O. N. N. Introduction of artificial intelligence through fuzzy logic in minecraft. In: . Singapore: Global Science and Technology Forum, 2016. p. 14–20. Copyright - Copyright Global Science and Technology Forum 2016; Document feature - Tables; Diagrams; Photographs; Graphs; ; Last updated - 2016-10-08. Disponível em: <https://search.proquest.com/docview/1826881115?accountid=8113>. Citado na página 60.

ITOH, K.; MIWA, H.; MATSUMOTO, M.; ZECCA, M.; TAKANOBU, H.; ROCCELLA, S.; CARROZZA, M. C.; DARIO, P.; TAKANISHI, A. Behavior model of humanoid robots based on operant conditioning. In: IEEE. *Humanoid Robots, 2005 5th IEEE-RAS International Conference on*. [S.l.], 2005. p. 220–225. Citado na página 26.

JADERBERG, M.; MNIH, V.; CZARNECKI, W. M.; SCHAUL, T.; LEIBO, J. Z.; SILVER, D.; KAVUKCUOGLU, K. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016. Citado na página 39.

JING, H.; XIAOGANG, R.; YAO, X.; XIAOPING, Z.; XIAOYANG, L. Operant conditioning model in autonomous navigation. In: *2015 IEEE Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. [S.l.: s.n.], 2015. p. 1015–1019. Citado 2 vezes nas páginas 26 e 37.

JOHNSON-LAIRD, P. Mental models in cognitive science. *Cognitive Science*, v. 4, n. 1, p. 71 – 115, 1980. ISSN 0364-0213. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0364021381800055>. Citado na página 20.

JOHNSON, M.; HOFMANN, K.; HUTTON, T.; BIGNELL, D. The malmo platform for artificial intelligence experimentation. In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*. [s.n.], 2016. p. 4246–4247. Disponível em: <http://www.ijcai.org/Abstract/16/643>. Citado 2 vezes nas páginas 17 e 52.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. *J. Artif. Int. Res.*, AI Access Foundation, USA, v. 4, n. 1, p. 237–285, maio 1996. ISSN 1076-9757. Disponível em: <http://dl.acm.org/citation.cfm?id=1622737.1622748>. Citado na página 25.

KAWAMURA, K.; DODD, W.; RATANASWASD, P.; GUTIERREZ, R. A. Development of a robot with a sense of self. In: *2005 International Symposium on Computational Intelligence in Robotics and Automation*. [S.l.: s.n.], 2005. p. 211–217. Citado na página 37.

KRISHNAMURTHY, R.; LAKSHMINARAYANAN, A. S.; KUMAR, P.; RAVINDRAN, B. Hierarchical reinforcement learning using spatio-temporal abstractions and deep neural networks. *CoRR*, abs/1605.05359, 2016. Citado na página 25.

KUPPUSWAMY, N. S.; CHO, S. h.; KIM, J. h. A cognitive control architecture for an artificial creature using episodic memory. In: *2006 SICE-ICASE International Joint Conference*. [S.l.: s.n.], 2006. p. 3104–3110. Citado 2 vezes nas páginas 37 e 61.

LAIRD, J. E. *The Soar cognitive architecture*. [S.l.]: MIT Press, 2012. Citado 2 vezes nas páginas 29 e 30.

LAIRD, J. E. *The Soar Cognitive Architecture*. [S.l.]: The MIT Press, 2012. ISBN 0262122960, 9780262122962. Citado 2 vezes nas páginas 32 e 34.

LAIRD, J. E.; ROSENBLOOM, P. The evolution of the soar cognitive architecture. *Mind matters: A tribute to Allen Newell*, Mahwah, NJ: Lawrence Erlbaum, p. 1–50, 1996. Citado na página 29.

LANGLEY, P.; LAIRD, J. E.; ROGERS, S. Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, Elsevier, v. 10, n. 2, p. 141–160, 2009. Citado na página 29.

LI, F.; FROST, J.; PHILLIPS, B. J. An episodic memory retrieval algorithm for the soar cognitive architecture. In: _____. *AI 2015: Advances in Artificial Intelligence: 28th Australasian Joint Conference, Canberra, ACT, Australia, November 30 – December 4, 2015, Proceedings*. Cham: Springer International Publishing, 2015. p. 343–355. ISBN 978-3-319-26350-2. Disponível em: <http://dx.doi.org/10.1007/978-3-319-26350-2_30>. Citado na página 32.

LITTMAN, M. L. Markov games as a framework for multi-agent reinforcement learning. In: *IN PROCEEDINGS OF THE ELEVENTH INTERNATIONAL CONFERENCE ON MACHINE LEARNING*. [S.l.]: Morgan Kaufmann, 1994. p. 157–163. Citado na página 25.

MADL, T.; BAARS, B. J.; FRANKLIN, S. The timing of the cognitive cycle. *PloS one*, Public Library of Science, v. 6, n. 4, p. e14803, 2011. Citado 3 vezes nas páginas , 34 e 65.

MÉNAGER, D. Episodic memory in a cognitive model. 2016. Citado 3 vezes nas páginas , 35 e 36.

MIROWSKI, P.; PASCANU, R.; VIOLA, F.; SOYER, H.; BALLARD, A.; BANINO, A.; DENIL, M.; GOROSHIN, R.; SIFRE, L.; KAVUKCUOGLU, K. *et al.* Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016. Citado na página 40.

NEWELL, A.; ROSENBLOOM, P. S.; LAIRD, J. E. *Symbolic architectures for cognition*. [S.l.], 1989. Citado na página 29.

NEWELL, A.; SIMON, H. A. *GPS, a program that simulates human thought*. [S.l.], 1961. Citado na página 29.

NUXOLL, A. M.; LAIRD, J. E. Extending cognitive architecture with episodic memory. In: *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 2*. AAAI Press, 2007. (AAAI'07), p. 1560–1565. ISBN 978-1-57735-323-2. Disponível em: <http://dl.acm.org/citation.cfm?id=1619797.1619895>. Citado na página 32.

OH, J.; CHOCKALINGAM, V.; SINGH, S. P.; LEE, H. Control of memory, active perception, and action in minecraft. In: *ICML*. [S.l.: s.n.], 2016. Citado 3 vezes nas páginas 39, 60 e 93.

OSMAN, M. An evaluation of dual-process theories of reasoning. *Psychonomic Bulletin & Review*, v. 11, n. 6, p. 988–1010, Dec 2004. Disponível em: <https://doi.org/10.3758/BF03196730>. Citado 2 vezes nas páginas 31 e 46.

PARAENSE, A. L.; RAIZER, K.; PAULA, S. M. de; ROHMER, E.; GUDWIN, R. R. The cognitive systems toolkit and the {CST} reference cognitive architecture. *Biologically Inspired Cognitive Architectures*, v. 17, p. 32 – 48, 2016. ISSN 2212-683X. Disponível em: <//www.sciencedirect.com/science/article/pii/S2212683X1630038X>. Citado 7 vezes nas páginas , 17, 31, 42, 43, 45 e 46.

PAVLOV, I. *Conditioned Reflexes: An Investigation of the Physiological Activity of the Cerebral Cortex, by I. P. Pavlov. Translated and Edited by G. V. Anrep*. Dover Publications, 1960. (Dover Publications. S). Disponível em: <https://books.google.com.br/books?id=WQTqngEACAAJ>. Citado na página 25.

PYNADATH, D. V.; ROSENBLOOM, P. S.; MARSELLA, S. C. Reinforcement learning for adaptive theory of mind in the sigma cognitive architecture. In: SPRINGER. *International Conference on Artificial General Intelligence*. [S.l.], 2014. p. 143–154. Citado na página 37.

RAIZER, K.; PARAENSE, A. L. O.; GUDWIN, R. R. A COGNITIVE ARCHITECTURE WITH INCREMENTAL LEVELS OF MACHINE CONSCIOUSNESS INSPIRED BY COGNITIVE NEUROSCIENCE. *International Journal of Machine Consciousness*, World Scientific Pub Co Pte Lt, v. 04, n. 02, p. 335–352, dec 2012. Disponível em: <https://doi.org/10.1142%2Fs1793843012400197>. Citado na página 17.

RAMAMURTHY, U.; FRANKLIN, S. Memory systems for cognitive agents. In: *Proceedings of Human Memory for Artificial Agents Symposium at the Artificial Intelligence and Simulation of Behavior Convention (AISB'11)*. [S.l.: s.n.], 2011. p. 35–40. Citado na página 35.

RUSSELL, S. J.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 2. ed. [S.l.]: Pearson Education, 2003. ISBN 0137903952. Citado na página 19.

SAKSIDA, L. M.; RAYMOND, S. M.; TOURETZKY, D. S. Shaping robot behavior using principles from instrumental conditioning. *Robotics and Autonomous Systems*, Elsevier, v. 22, n. 3-4, p. 231–249, 1997. Citado na página 26.

SAMSONOVICH, A. V. Toward a unified catalog of implemented cognitive architectures. *BICA*, v. 221, p. 195–244, 2010. Citado na página 30.

SANDAMIRSKAYA, Y.; BURTSEV, M. Narle: Neurocognitive architecture for the autonomous task recognition, learning, and execution. *Biologically Inspired Cognitive*

Architectures, v. 13, n. Supplement C, p. 91 – 104, 2015. ISSN 2212-683X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2212683X15000341>. Citado na página 37.

SHARMA, S.; LAKSHMINARAYANAN, A. S.; RAVINDRAN, B. Learning to repeat: Fine grained action repetition for deep reinforcement learning. *arXiv preprint arXiv:1702.06054*, 2017. Citado na página 40.

SHORT, D. Teaching scientific concepts using a virtual world–minecraft. *Teaching Science*, v. 58, n. 3, p. 55–58, September 2012. ISSN 1449-6313. Disponível em: <https://www.learntechlib.org/p/91796>. Citado na página 60.

SILJEBRÅT, H. Maia: The role of innate behaviors when picking flowers in minecraft with q-learning. 2015. Citado 4 vezes nas páginas , 39, 40 e 93.

ŠÍMA, J. Back-propagation is not efficient. *Neural Networks*, Elsevier, v. 9, n. 6, p. 1017–1023, 1996. Citado na página 28.

SIMON, H. A.; NEWELL, A. Human problem solving: The state of the theory in 1970. *American Psychologist*, American Psychological Association, v. 26, n. 2, p. 145, 1971. Citado na página 29.

SKINNER, B. F. *The behaviour of organisms: An experimental analysis*. [S.l.]: D. Appleton-Century Company Incorporated, 1938. Citado na página 27.

SLOMAN, A. Architecture-based conceptions of mind. In: *In the Scope of Logic, Methodology and Philosophy of Science*. [S.l.]: Springer, 2002. p. 403–427. Citado na página 29.

SUN, R. A tutorial on clarion 5.0. *Cognitive Science Department, Rensselaer Polytechnic Institute*, 2003. Citado na página 35.

SUN, R. Desiderata for cognitive architectures. *Philosophical Psychology*, Taylor & Francis, v. 17, n. 3, p. 341–373, 2004. Citado na página 29.

SUN, R. The importance of cognitive architectures: An analysis based on clarion. *Journal of Experimental & Theoretical Artificial Intelligence*, Taylor & Francis, v. 19, n. 2, p. 159–193, 2007. Citado 3 vezes nas páginas , 29 e 36.

SUN, R. *The CLARION Cognitive Architecture*. Oxford University Press, 2015. Disponível em: <https://doi.org/10.1093/oxfordhb/9780199842193.013.11>. Citado 2 vezes nas páginas 42 e 92.

SUTTON, R. S. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In: *Advances in Neural Information Processing Systems 8*. [S.l.]: MIT Press, 1996. p. 1038–1044. Citado na página 25.

SUTTON, R. S.; BARTO, A. G. *Introduction to Reinforcement Learning*. 1st. ed. Cambridge, MA, USA: MIT Press, 1998. ISBN 0262193981. Citado 3 vezes nas páginas 23, 24 e 71.

TESSLER, C.; GIVONY, S.; ZAHAVY, T.; MANKOWITZ, D. J.; MANNOR, S. A deep hierarchical approach to lifelong learning in minecraft. *CoRR*, abs/1604.07255, 2016. Disponível em: <http://arxiv.org/abs/1604.07255>. Citado 4 vezes nas páginas , 38, 39 e 60.

TOURETZKY, D. S.; SAKSIDA, L. M. Operant conditioning in skinnerbots. *Adapt. Behav.*, MIT Press, Cambridge, MA, USA, v. 5, n. 3-4, p. 219–247, jan. 1997. ISSN 1059-7123. Disponível em: <http://dx.doi.org/10.1177/105971239700500302>. Citado 2 vezes nas páginas 26 e 27.

TULVING, E. Episodic memory: From mind to brain. *Annual Review of Psychology*, Annual Reviews, v. 53, n. 1, p. 1–25, feb 2002. Disponível em: <https://doi.org/10.1146%2Fannurev.psych.53.100901.135114>. Citado 2 vezes nas páginas 31 e 32.

TULVING, E.; THOMSON, D. M. Encoding specificity and retrieval processes in episodic memory. *Psychological Review*, American Psychological Association (APA), v. 80, n. 5, p. 352–373, 1973. Disponível em: <https://doi.org/10.1037%2Fh0020071>. Citado na página 32.

UDAGAWA, H.; NARASIMHAN, T.; LEE, S. Fighting Zombies in Minecraft With Deep Reinforcement Learning. dez. 2016. Citado 2 vezes nas páginas 40 e 60.

WATKINS, C. J. C. H. *Learning from delayed rewards*. Tese (Doutorado) — King's College, Cambridge, 1989. Citado na página 26.