Spring 2019

Bridging ACT-R and Project Malmo, developing models of behavior in complex environments

David M. Schwartz
Bucknell University, dms061@bucknell.edu

Bridging ACT-R and Project Malmo, developing models of behavior in complex environments

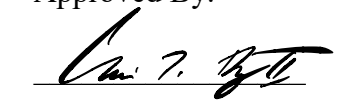
By

David M. Schwartz

A Thesis Presented to the Faculty of Bucknell University

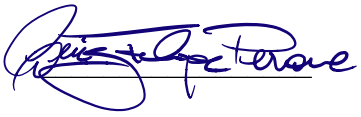
4/12/19

Approved By:



Christopher L. Dancy

Thesis Advisor



L. Felipe Perrone

Second Reader, Chair of the Department of Computer Science

Contents

List of Figures

Abstract

Cognitive architectures such as ACT-R provide a system for simulating the mind and human behavior. On their own they model decision making of an isolated agent. However, applying a cognitive architecture to a complex environment yields more interesting results about how people make decisions in more realistic scenarios. Furthermore, cognitive architectures enable researchers to study human behavior in dangerous tasks which cannot be tested because they would harm participants. Nonetheless, these architectures aren't commonly applied to such environments as they don't come with one. It is left to the researcher to develop a task environment for their model. The difficulty in creating one prevents cognitive architectures from being utilized in more advanced studies. This project aims to address that issue by building a bridge between ACT-R and Project Malmo, an artificial general intelligence test suite. The bridge facilitates easy integration of new missions by allowing researchers to specify how to create the world and update it without worrying about the overhead of Malmo. Furthermore, this study analyses how well ACT-R's utility learning system will adapt in a complex environment. The Adaptive Gain Theory was implemented to improve how the system adapts by using task engagement, derived from measures of utility, to dynamically modify noise. The system was tested using a modified Symbolic Maze task. Tests revealed the parameters of the Adaptive Gain mechanism need to be refined to have a greater impact on model performance. Nonetheless, the bridge provides an interface for ACT-R to be used to study decision making in a complex environment. Improving the

bridge will enable more advanced experiments to be conducted whilst improving the Adaptive Gain Theory implementation will move us one step closer to understanding everyday intelligent behavior.

1 Introduction and Motivation

Cognitive architectures are software systems that simulate human cognition (Anderson 2007). They enable researchers to develop models of mental action and simulate decision making. By itself an architecture can be used to simulate action in isolation. However, action in an empty room provides little insight into how people make decisions in their everyday lives. Instead these architectures should interact with the world and tools people use to generate more realistic simulations.

The benefit of connecting a cognitive architecture to a realistic environment is shown by the US Army's employment of it. The Army uses a Synthetic Task Environment (STE) to train solders in simulated battles (Judson 2018). Cognitive architectures can be added to a STE to simulate human like agents. Traditionally, such agents have been used to validate traditional operating procedures and evaluate new ones (Ritter et al 2003). Therefore, cognitive models allow researchers to study and improve behavior in dangerous environments before subjecting people to risk. Also, the presence of agents which make decisions like humans increase the realism of a training simulation. Therefore, a cognitive architecture paired with a virtual environment enables studies of decision making in extreme conditions by providing a replacement to human subjects who could be injured in the study.

Another potential use of a cognitive architecture in the STE is to model the solder in training. The D2P2 tutoring system develops a model of its user and how they learn based

on their interaction with the software. The program inspects a cognitive model that replicates the knowledge of the user and identifies areas they need to study more. Then it curates a custom curriculum to enhance their knowledge (Hopps, Ritter, Morgan 2012). A similar system can be implemented inside the STE to provide soldiers with feedback on how to better handle dangerous scenarios. Therefore, connecting a cognitive architecture and a complex environment can be used to create tutors which improve the performance of people in real life scenarios.

Unfortunately, most architectures fail to provide mechanisms to connect models to a task environment. Researchers are required to modify or recreate the interface and environment their model interacts with. The more realistic the environment, the more representative of people the model actions are. At the same time, a more realistic environment takes much longer to develop. As such, researchers must abstract away aspects of the environment to make implementation feasible (Cooke and Shope 2017). However, this requires them to determine the critical aspects of a task; in other words, the features of a task which cause behavior. However, spending time on creating the environment reduces the time spent on the model, thereby inhibiting research into behavior in a complex environment.

This issue can be avoided by having a complex environment that is connected to a cognitive architecture. This study creates a bridge between the ACT-R cognitive architecture and Project Malmo, an artificial intelligence testing platform with a complex task environment. The bridge is tested with a Symbolic Maze experiment. Additionally, this study analyzes how integrating the Adaptive Gain Theory with ACT-R changes how

models manage the exploration exploitation trade-off. Background on ACT-R, Project Malmo, the exploration exploitation trade-off, and the Adaptive Gain Theory are discussed first. Then, various versions of the bridge are documented. Next the Symbolic Maze experiment, model, and results are discussed. Finally, the paper concludes with a discussion of potential improvements for both the bridge and the model.

2 Background

This section provides background on theories tools, and previous bridge designs. First, ACT-R and how it creates model of cognition is covered. Then Project Malmo, the task environment it will be connected to, is described. After that the exploration exploitation trade-off and how it impacts behavior in a complex environment is discussed. In addition, how ACT-R models manage the trade-off is examined. Then, the Adaptive Gain Theory and its relation to trade-off addressed. Finally, the section ends with a summary of previously considered and implemented bridge designs.

2.1 ACT-R

ACT-R is a cognitive architecture that provides a functional model of the mind. It is a functional model because it doesn't simulate how the brain works. Instead, it replicates brain function at a high level. In other words, it isn't a neural model. Instead, ACT-R represents information symbolically and uses various modules representing high level brain functions (Anderson 2007) to determine how they symbols are manipulated. The

core functionality of ACT-R comes from rule based expert systems (RBS) described below.

2.1.1 Relation to Rule Based Expert Systems

RBS systems have several key components: memory, a knowledge base, an inference engine, and an explanation engine. Memory holds facts the system knows, such as the number of days in a month. The knowledge base stores rules that trigger actions. Knowledge of how to play a piano would be stored there. The inference engine powers decision making by looking through the available rules in the knowledge base and facts in memory to decide what actions to take. The selected action gets passed to the explanation engine which displays what occurred and why. The relation between these systems is shown in Figure 1.

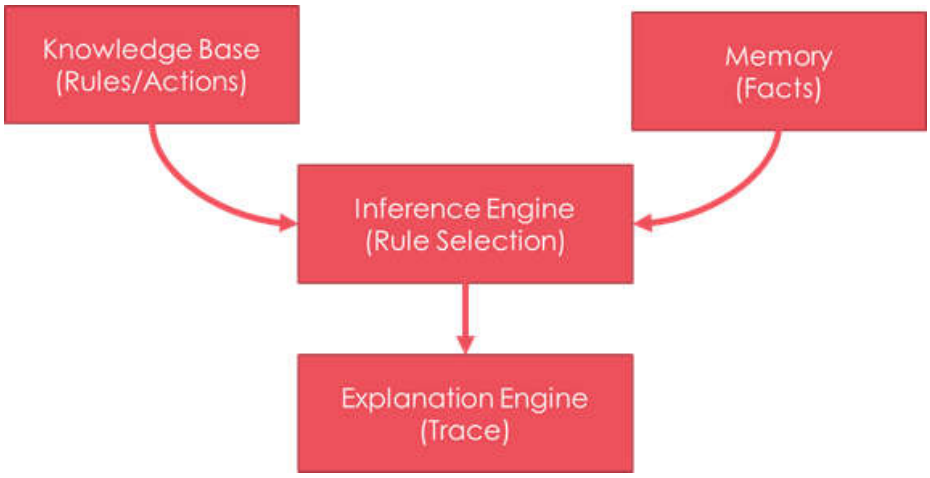


Figure 1. The structure of a rule based expert system. It includes two types of memory, one for facts and another for actions. It works by sequentially selecting rules to execute that satisfy constraints specified between memories.

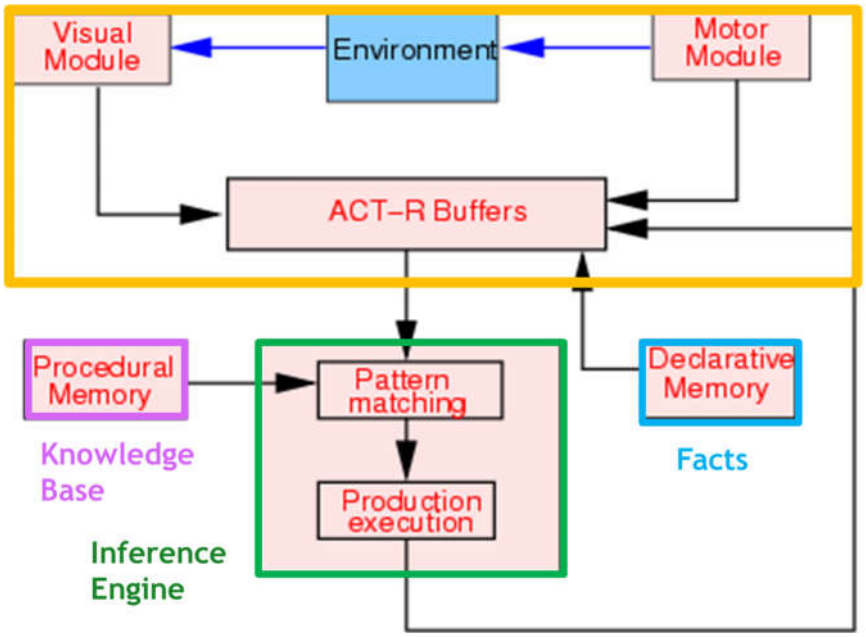


Figure 2. High level diagram of the ACT-R cognitive architecture. Relations to a RBS are marked on the diagram. Procedural memory is analogous to a knowledge base and declarative memory is factual memory. Production conflict and execution is the inference engine. The explanation engine isn't shown in the diagram. Lastly, ACT-R adds the ability to interact with an environment and modules with buffers that represent other high-level brain functions. Image adapted from Budiu 2013.

ACT-R inherits the core functionality of a RBS; Figure 2 shows the relationship between components in ACT-R and its RBS equivalent. Just like an RBS, ACT-R possesses two types of memory. Declarative memory stores facts in a structure known as a chunk. Information is stored in slot value pairs that each encode pieces of a fact. The fact "Bucknell is a University in Lewisburg," could be encoded with two slots: *university* which stores "Bucknell," and *location* which stores "Lewisburg." Procedural memory stores rules and actions as productions. They are encoded as if-then statements such as:

"if I want to graduate then I should go to class." The procedural module in ACT-R is equivalent to the inference engine in a RBS and is discussed more in Section 2.1.2.

ACT-R differs from a RBS by dividing functionality across modules. They can be split into three categories: central, perceptual, and response. The modules within these categories and their purpose are explained sequentially.

2.1.2 Central Processing Modules

The central category is responsible for decision making and memory. It contains the goal, declarative, procedural and utility modules. ACT-R assumes that agents are goal based and thus work towards some objective. Buffers are used as an interface to module functionality. The goal module has a buffer which stores the model's objective as a chunk. The buffer can modify the chunk and update the model's goals according to rules in the system.

The declarative module manages the recollection of information. It uses the retrieval buffer as an interface to recall chunks from declarative memory. Models can use the buffer to request a chunk matching some specification. For example, to recalling a chunk encoding where Bucknell is located could be done by requesting chunk with a *university* slot containing the value "Bucknell." The module will search for the chunk and place it in the buffer if it is found. A failure to recall a chunk puts the buffer into an error state. Note that this module only handles recollection; creation of chunks is the purpose of the imaginal module.

The imaginal module is used to encode new information that is not perceptually present. This module has a buffer which can create a new chunk and modify it to contain information representing an object or fact. The chunk can be placed into declarative memory by clearing the buffer. The module is used for non-perceptual information because perceptual information enters declarative method through another process called harvesting. Therefore, the imaginal and declarative modules together provide a way to generate and recall new facts or associations. However, the model must also be able to make decisions.

The procedural module in ACT-R is equivalent to the inference engine in a RBS. Periodically the module checks for a production to apply in a process called conflict resolution. The if conditions of every rule are analyzed to determine which productions are applicable. Out of the pool of suitable productions, the one with the highest utility is selected and applied. Utility a measure of how useful a production is and is learned over time.

The utility of a production in ACT-R is managed by the utility module. Utilities in are assigned via the temporal difference equation (Fu and Anderson 2006, Bothell 2019):

$$U_i(n) = U_i(n \quad 1) + \alpha[R_i(n) \quad U_i(n \quad 1)] \tag{1}$$

In Equation 1, $U_i(n)$ represents the utility of production *i* after its *n*th application. Alpha is a learning parameter set by the modeler. $R_i(n)$ represents the reward given to the production on its *n*th application and is defined as:

$$R_i(n) = r_i(n) \quad t_i \tag{2}$$

In Equation 2, $r_i(n)$ is the immediate reward given by the modeler for some action and t_i is the time difference between when the reward is received and when the production fired. As a model performs a task, it will receive rewards, increasing the utility of select productions, which boosts the likelihood they are selected during conflict resolution. In other words, the model learns to favor incentivized behavior.

2.1.3 Perceptual Modules

The perceptual category contains two modules: vision and audio. The vision module is the eyes of ACT-R. It has two subsystems that handle where an object is and what it is respectively. The where system is accessible through the visual-location buffer. It provides an interface to search for visual information matching some criteria specified by the model. Handling the query works analogously to the remembering a chunk, if a matching chunk is found it is placed in the buffer, otherwise the buffer enters a failure state. The chunk placed in the buffer will contain the object's position and a few details. For more specific information the object must be attended by the what system. This system handles the model's visual attention and is accessed through the visual buffer. A request to shift attention requires a location chunk extracted from the visual-location buffer. After a request, the model will focus on the object at the specified position and encode its properties into a new chunk placed within the visual buffer. Furthermore, the model will keep its attention locked on the object, updating its encoding if necessary. The model's attention will stay on the object until it disappears or a request is made to switch

attention or stop. The audio module is ACT-R's ears and is structured like the vision module. It has a where and a what system as well as audio-location and audio buffers which manage each system respectively. It handles requests just like the vision module too. Together, the vision and audio modules combine to give ACT-R eyes and ears, enabling it to interpret information from its environment.

2.1.4 Response Modules

Lastly, the response category contains the motor and speech modules which allow an agent to act within an environment. The motor module provides ACT-R with a set of hands. However, the architecture assumes that the model is using a computer. As such, the model can only type and use a mouse. Motor commands are invoked by sending a request to the manual buffer. The time it takes to complete an action is governed by Fitt's Law (Bothell 2018a):

$$T = b \quad \log_2\left(\frac{D}{W} + 0.5\right) \tag{3}$$

In equation one, *T* represents the time for an action to be completed. The coefficient *b* is a scaling parameter determined by the type of action and is stored within ACT-R. The variables *D* and *W* represent the distance to and width of the target the model is trying to hit; for instance, *D* could represent the distance from the model's hand to the 'A' key while *W* represents the width of the key itself. ACT-R uses a virtual keyboard and mouse to obtain values for the equation and signal actions. The speech module provides the model with the basic ability to talk. However, the module is only used as an output signal; it doesn't generate actual sound. Instead, ACT-R creates an output signal that

describes the phrase said in textual form. Requests to the module are handled through vocal buffer. Queries can be made to have the model talk to itself or to speak aloud. Together, the motor and speech modules provide ACT-R with a way of interacting with its surrounding environment.

2.1.5 ACT-R Summary

To summarize, ACT-R is a cognitive architecture that creates a function model of the mind through the collective usage of the various modules. The modules represent different high-level brain, perception, and motor functionality. The architecture uses two forms of memory: declarative and procedural. Declarative memory stores facts and is encoded as chunks whereas procedural represents actions and is encoded as productions. ACT-R models select and execute productions in order to complete some goal. Applying a production queues up actions in different modules, allowing the model to create and process information as well as interact with its environment.

2.2 Exploration and Exploitation

To navigate a complex environment an agent or person needs to manage the exploration exploitation trade-off. It concerns how information is gathered and utilized. For instance, think of buying a car. You could test drive every car on the lot and make an optimal decision with perfect knowledge. However, that is extremely time consuming and another customer could buy the best car while you finish testing the others. At some point you need to exploit the information available to make a satisfactory decision.

Managing the trade-off is an important part of daily life, so a cognitive architecture should be able to manage it too.

2.2.1 Exploration and Exploitation in ACT-R

ACT-R models deal with exploration and exploitation through utility learning. Recall rewards are given to a model to update production utilities and that the gaps in utilities enable it to learn beneficial actions. That works well for environments in which rewards are stationary; meaning the benefit of an action doesn't change. In such an environment the model can develop optimal behavior over time. However, the mechanism fails at solving reversal experiments in which correct and incorrect actions suddenly switch (Cohen, McClure, Yu, 2007). This is because utilities are difficult to adjust. Productions with high utilities will need to have their utilities diminished while productions with low utilities need to be strengthened. During the change, the high utility productions will continue to be selected, causing the agent to continually make poor decisions.

The typical solution to this problem is annealing. ACT-R selects the production to execute by looking at the sum of utility and noise, a random value generated at the time of the decision. Annealing increases the magnitude of noise within the system to improve the likelihood that a low utility production is selected; furthermore, that production is then eligible to receive rewards and increase its utility. Since only certain rules are applicable in any given situation, this increase in noise leads to exploration of available options and not just randomized behavior. However, annealing can only begin when it is determined that new learning must occur, a feature which is absent in ACT-R. As such,

ACT-R agents will have a difficult time adapting to a changing environment. To combat this, a mechanism must be added to ACT-R which can selectively activate annealing.

2.2.2 Adaptive Gain Theory

The Adaptive Gain Theory concerns how the Locus Coeruleus (LC) and Norepinephrine (NE) relate to the exploration exploitation trade-off (Aston-Jones and Cohen 2005). LC neurons are responsible for distributing NE throughout the brain.

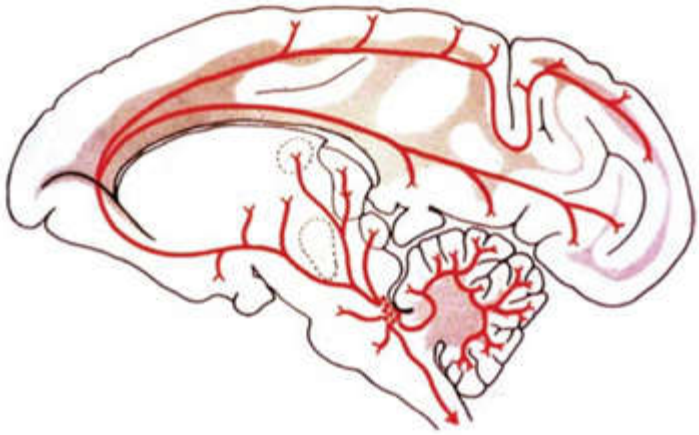


Figure 3. Locus Coeruleus connections, shown in red, in a primate's brain. The connections are believed to be similar to those in human brains. Image from Aston-Jones and Cohen 2005.

NE handles information filtering. How NE is dispersed changes depending on the model of LC: phasic or tonic. When LC is in phasic mode, bursts of NE remove irrelevant information making a person exploit certain facts. On the other hand, tonic mode has no bursts of NE, but a higher baseline amount. When this occurs, no

information is suppressed, providing more avenues to explore. Aston-Jones and Cohen have related this phenomenon to task engagement and provided a method to estimate it.

Task engagement is a proxy for the mode of LC and can be calculated by looking at long and short term utility. The intuition behind it is that an engaged person is focused and will filter out irrelevant information. As such LC is in phasic mode and the person exploits their knowledge. An unengaged person is more likely to be distracted or explore other avenues, hinting that LC is in tonic mode and exploration is occurring. Engagement will change with a person's performance, which is indirectly represented as utility. Task engagement can be calculated using Equation 4; a graph of the equation is shown in Figure 4.

$$E = \begin{pmatrix} 1 & logistic(U_{short}) \end{pmatrix} \quad (logistic(U_{long})) \tag{4}$$

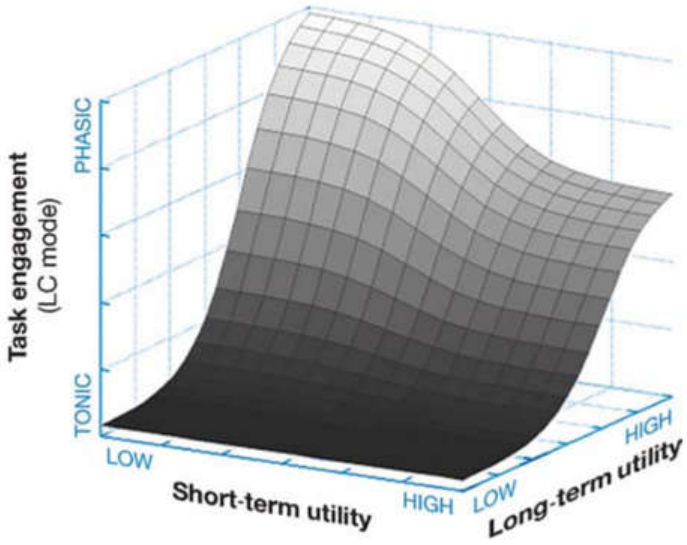


Figure 4. Graph of how task engagement changes with long and short term utility. Image from Aston-Jones and Cohen 2005.

2.3 Project Malmo

Project Malmo is a modification to the game Minecraft that allows it to be used as a testing suite for artificial general intelligence (AGI) (Johnson, Hofmann, et al. 2016). To understand Malmo and how it can be used, some knowledge of Minecraft is required. Minecraft is a game about survival. Players are placed in a three-dimensional world made of discrete blocks representing different material, like dirt, stone, and iron. In this world they must seek out food sources to nourish their character and fend off various enemies that come out at night. Players can make their lives easier by exploring the world and mining it for resources with which they can create tools and build shelters. While models being developed with cognitive architectures aren't aimed at general intelligence due to their explicit encoding of relevant knowledge, however, using an AGI test suite is attractive because it provides researchers with a tool to create complex and interactive environments. Specifically, Malmo allows researchers to alter the game whilst providing an interface to connect artificial agents.

Malmo modifies the game's world and mechanics by loading a mission from an extensible markup language (XML) file. XML is a human readable language that uses tags to define data elements. Malmo defines specific tags to modify the game world and functionality. There are several important tag types, the first of which are drawing tags. These allow blocks to be placed in the world, giving researchers control over the environment agents (and players) are in. Other tags control the game's time, weather, and under what conditions it ends. However, the most important tags are agent tags. These

define how many agents are present in a mission and where they spawn. Furthermore, they define how agents can move. It can be set to continuous, allowing for movement within blocks; discrete, for block by block movement, and absolute, allowing agents to teleport anywhere at any time. Agent tags are also responsible for creating observers, which controls what data about the world is sent to an agent. They can be used to send an agent the blocks around them, their distance to other characters (agents and enemies), and even information on their inventory. Those tags control what information gets sent to the agent through the application programming interface (API).

Malmo has APIs that enable data from the game to be sent to agents written in Python, Lua, C#, C++ or Java. Each API contains code to capture messages from the game and create data structures representing the world and agent. Furthermore, they have the ability to send messages back to the game, allowing agents to take action.

Another benefit of using Malmo is that it allows both players and agents to interact with the same mission. Players can perform the same experiment as an agent, generating a benchmark for performance. Furthermore, players and agents can connect to a mission at the same time, enabling researchers to study the collaboration of agents and people (Brill, Dancy, 2018). Even though Malmo is an appealing testing ground for cognitive models few people have used Malmo with them.

2.4 Malmo ACT-R Bridges

Now that both ACT-R and Malmo have been introduced, bridge designs can be discussed. The bridge has three responsibilities: data translation, data collection, and

updating mission logic. Data translation refers to the need to convert how data is represented between Malmo and ACT-R. Observations in Malmo have to be converted into chunks for ACT-R. Similarly, key presses and mouse movements in ACT-R must be converted to commands for Malmo. At the same time, the bridge updates the logic of an experiment and collects data on the agent's performance. Two designs which implement these functions are discussed in this section.

2.4.1 Ideal Bridge

The ideal bridge between Malmo and ACT-R creates a direct connection between the two. Thus ACT-R itself is responsible for converting data formats, collecting performance data, and updating mission logic. However, this would be difficult to construct and it wouldn't be officially supported by Malmo. Even though Malmo works over a network interface, it doesn't document the structure of messages used. Instead, it relies on special APIs which are only available in a few languages. Common Lisp, the language ACT-R is written in, is not supported. Therefore, a direct connection is only possible by decoding all possible message formats and implementing a custom API. Furthermore, this requires modifying both Malmo and ACT-R's source code. It is also unstable as updates to either tool could break the bridge entirely. To get around this, Son Pham proposed creating a bridge program that passes messages between Malmo and ACT-R using supported APIs.

2.4.2 First Bridge

Mr. Pham, under the supervision of Professor Dancy, created a bridge that connected Malmo to ACT-R 6 by using a third process. A diagram of the overall system is shown below:

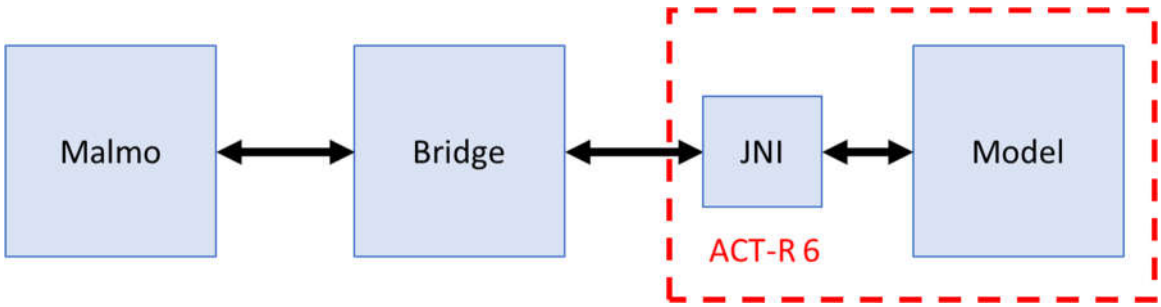


Figure 5. Diagram of the original Malmo to ACT-R bridge. It is a hierarchical client server model. Malmo is a server to the experiment, which is a server to ACT-R.

The bridge is its own program. It connects ACT-R and Malmo by creating several client sever relationships. Malmo acts as a server to the bridge script. At the same time the bridge is a server to ACT-R. The bridge was written in Python to take advantage of Malmo's existing API. It connects to ACT-R using the JSON Network Interface (JNI) (Hope 2013), an additional ACT-R module which enables it to connect to other programs. JNI works by sending data in JavaScript Object Notation (JSON) between ACT-R and the bridge over a network interface. JNI also has a Python API, which handled packaging, sending, and receiving data between the bridge and ACT-R. However, the bridge had several problems.

The first issue was that the bridge wasn't general purpose. It was designed for a specific mission and model. Thus, it had references to special observers present in that mission. In order to utilize it for another mission, the source code would have to be modified. That is dangerous because it means another researcher may accidentally break the bridge whilst trying to use it.

The second issue was time synchronization. Malmo, ACT-R, and the bridge each have their own notion of time. In Malmo it is Minecraft's game timer, the bridge uses real time, and ACT-R uses its own clock which has two modes of operation: real time and simulated time. None of these timers communicate with each other, which complicated how data should be collected. A person interacting with Malmo plays the game in real time. The bridge can capture the reaction time of them natively. An ACT-R model can also run in real time, but doing so drastically increases the time needed to collect data. Using simulated time allows for faster data collection, but invalidates any timing mechanisms as the bridge isn't aware of the ACT-R clock.

The third problem was a bug in JNI. A race condition, a situation where a program's output is dependent on the order in which events occur, was present in the code governing how JNI updated visual information. The bridge sends visual observations from Malmo to ACT-R as they are available. JNI receives this information and should replace the existing visual data so the model can make decisions on the current state of the environment. However, if the vision module is in use, the data replacement never occurs. Therefore, if a model continuously looks for something it will never receive new visual information, and be unable to react to changes in its environment. A better

mechanism would be to stop the model and then apply the visual update, thereby allowing it to always access to current information.

The biggest problem with the bridge was that it utilized outdated software. The code was written in Python 2.7. Its creators announced they will soon stop supporting it (Peterson 2008). Furthermore, some APIs the bridge depends on have not been upgraded to the new version. Therefore, in order to upgrade the bridge, underlying libraries would have to be rewritten. Similarly, the bridge only worked with ACT-R 6 and ACT-R 7 has been out for several years. Therefore, the bridge was becoming outdated and will soon be unusable.

3 New Bridge

For this thesis, the bridge was rebuilt from the ground up to support newer versions of Python, ACT-R, and Malmo. This was done to resolve issues and improve longevity of the bridge. Python 3.6 was used instead of 2.7. Similarly, a newer version of Malmo had to be used to support that. In addition, ACT-R 7.6+ is used instead of ACT-R 6. However, the internal structure of ACT-R changed dramatically between versions. It invalidated JNI, causing a change to the overall design of the bridge.

3.1 ACT-R Changes

ACT-R 7.6+ redesigned ACT-R to allow integration with other software in a manner similar to the JNI module (Bothell 2018b). The new version of ACT-R is based on

remote procedure calls (RPC) which enables a program to be split up between machines and processes. An RPC system consists of a server and one or more clients. Clients connect to the sever and send requests to execute a function with specific parameters. The server executes the requested action and sends the results back to the client.

ACT-R 7.6+ modifies this system by allowing client programs to register their functionality with the server, referred to as the dispatcher. If the dispatcher receives a request for a non-native command, it forwards the request to the client which registered it. The client services the request and sends the results to the server which forwards them to the origin of the request.

Additionally, the dispatcher has a monitoring system that chains commands together. Clients can add monitors by specifying a command to wait for and another to execute immediately after. This system is used to build interactions with the model. For example, ACT-R will send an *output-key* command to the dispatcher whenever the model presses a key. A client which needs that information, such as the bridge script, can create tell the dispatcher to monitor the *output-key* command and execute its own command to handle to keypress. Whenever the model presses a key, the dispatcher will fire the command for the keypress right after. The key pressed will be passed along, allowing the client to act on the model's input.

3.2 New Bridge Design

The dispatcher in ACT-R 7.6+ was incorporated into the bridge as a replacement for JNI. A diagram of the bridge is shown in Figure 6.

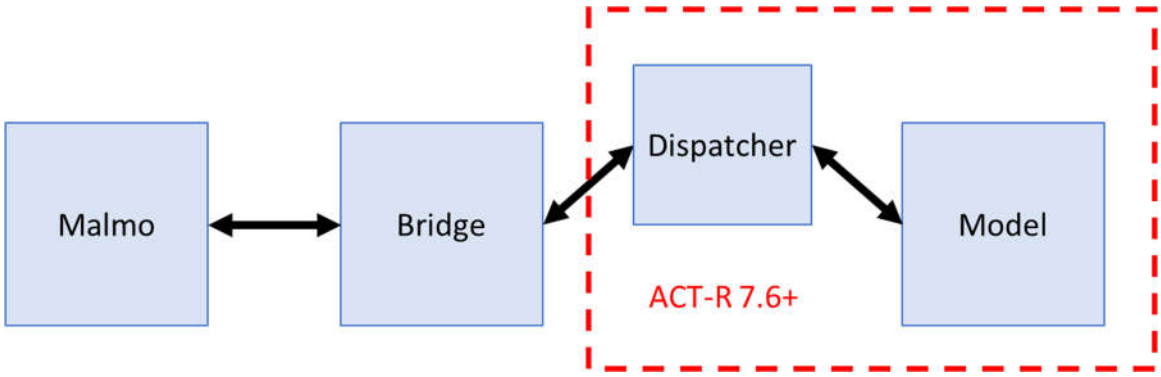


Figure 6. Diagram of the new bridge. Malmo acts a server to the bridge. The dispatcher is a server running remote procedure calls between the bridge and ACT-R.

The inclusion of the dispatcher creates extra overhead, but also solves several issues. In this bridge data being sent between ACT-R and Malmo goes through another process, thereby increasing the transfer time. However, the exclusion of JNI removes the race condition. New visual information is always received because the dispatcher temporarily stops ACT-R to update it. Also, the dispatcher allows the bridge to ask for information relating to ACT-R. Therefore, the bridge can request ACT-R's time, allowing it to correctly capture timing data from the model. Furthermore, recoding the bridge allowed a more general design to be implemented.

The new bridge defines a template for creating missions which allows them to be used interchangeably. The template allows researchers to modify five actions within the server. The first is the creation of a mission. Researchers can access mission properties before they are sent to Malmo and programmatically modify them; thus, they can create, objects, items, and observers through the bridge. Second is the setup of a trial in the mission. This provides researchers with the ability to initialize data and conditions in an

experiment that consists of multiple steps. Third is the mission's update logic, the code that governs how the experiment reacts to the agent. Fourth is called at the end of every trial and provides a coinvent place to log data. Last is the cleanup phase which is called once before the mission ends. Its is used to ensure the mission exits and all necessary data is retrieved from the program. Therefore, the bridge abstracts away the logic of a mission and provides researchers with the freedom to run any mission which satisfies the template.

4 Using the Bridge

This section documents the experiment that was used to test the bridge. It starts by describing the task and its implementation in Malmo. After that, the design of the ACT-R model which completes the task is described.

4.1 Symbolic Maze Experiment

I used a variant of the Symbolic maze task developed by Fu and Anderson (2006) to test the bridge and simulate the management of the exploration exploitation trade-off. The maze is setup in a tree structure (depicted in Figure 7).

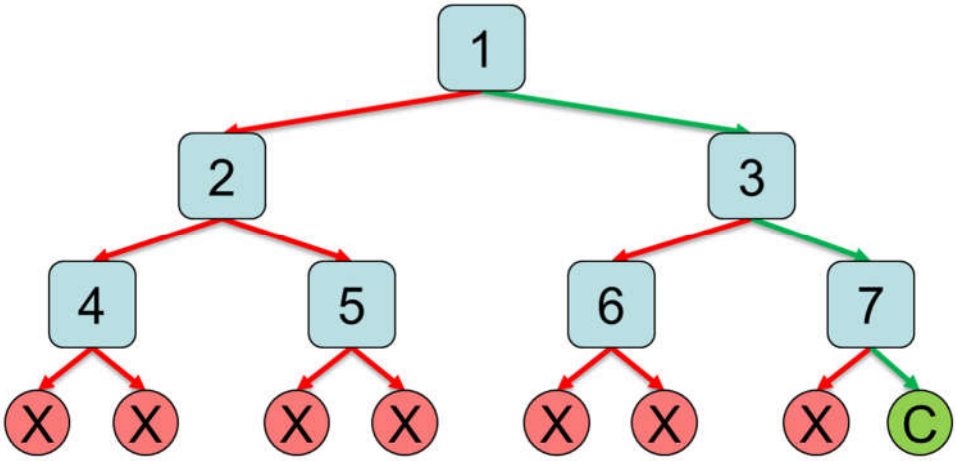


Figure 7. The structure of the Symbolic Maze. Green arrows depict correct decisions whereas red arrows depict incorrect decisions. Red circles are dead ends and the green circle is the exit. Adaptation of an image from Fu and Anderson 2006.

The player begins in room one and aims to reach the end of the maze. In each room, they are presented with stimuli and a set of options (Figure 8).

Figure 8. A maze room in the Malmo implementation of the maze. The center block is the stimuli that signals where the player is in the maze. The player stands on the ore to the left and right to move to the next room.

The stimuli in the room encodes which option is correct, however, the player must learn the association themselves. They move into a different room by standing on an option block. If they are correct they move closer to the exit. However, if they are incorrect they move towards a dead end. Upon reaching a dead end the player is placed in a reset room which contains a single wool block to stand on. After standing on it, they are reset to the point where they diverged from the correct path (shown in Figure 9).

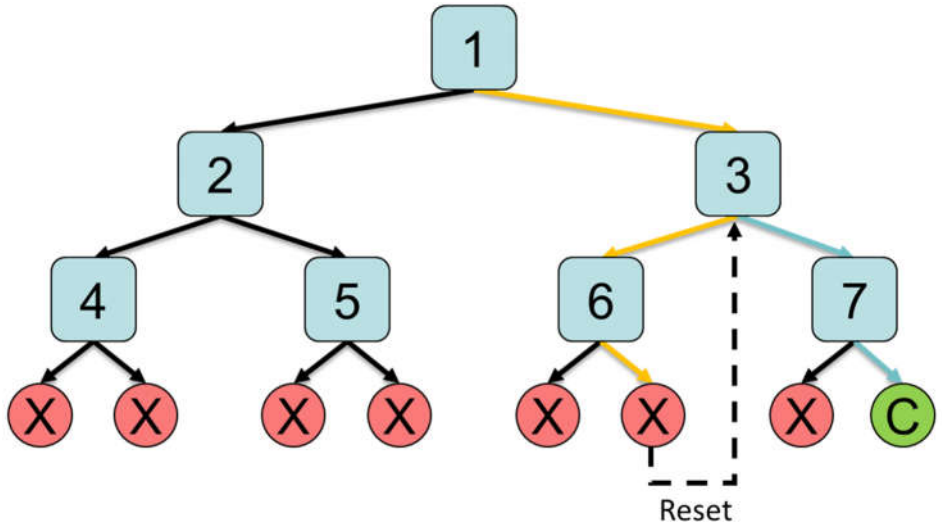


Figure 9. Resetting procedure. The yellow arrows represent the player's path before a reset. The light blue arrows show their path after. Upon hitting a dead end, they are brought back to room three and continue playing.

Furthermore, the configuration of the room is changed; each room has four possible configurations each with different stimuli and options. The player is only informed of whether they reach a dead end or the exit, so they have to determine which choices are correct.

The player must navigate the maze two hundred times to complete the experiment. To measure performance each trial, single navigation of the maze, is scored. The player begins with three points at the start of the trial. Every time they hit a dead end they lose a point. However, their score cannot go below zero in any trial. Furthermore, they must finish navigating the maze to complete a trial and continue the experiment.

The experiment's original specification only demonstrates the transition from exploring to exploiting. A slight modification is necessary to observe the opposite transition. After the player has travelled through the maze several times the correct path will be switched. This forces players who found the correct path to evaluate other options, thereby returning to an exploratory state. With the task specified, a model can be constructed.

4.2 ACT-R Model

Fu constructed a model to complete the task. However, his model worked with a different environment and an older version of ACT-R. Fu's design was reused in a new model written specifically to work with Malmo. At its core, the model performs three high level tasks: determine the composition of a room, decide which object to select, and lastly pick it. A flowchart of the model is presented in Figure 10.

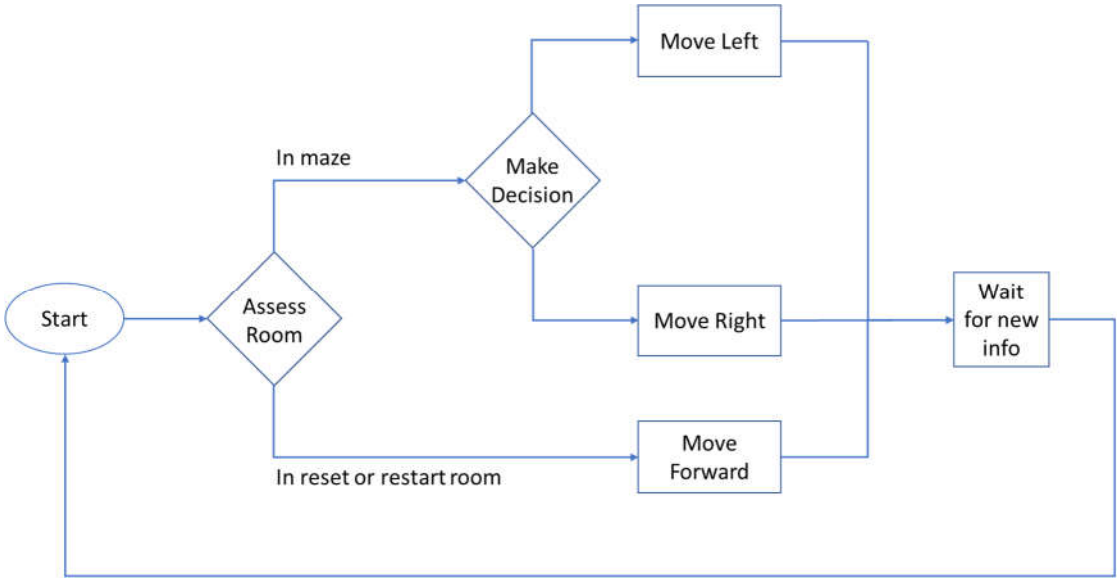


Figure 10. Flowchart of the model's decision making. It starts by determining what room it is in. Based on the room it determines how to move and then recalls what buttons should be pushed to cause that movement.

In Fu's experiment, the objects in the room were presented via text. In the Malmo version, the model focuses on various blocks throughout the room and encodes them into working memory. After that, the model can make a decision.

Decision making works the same as in Fu's model. A production exists for each option stimuli pair. Therefore, after encoding the room, two productions are eligible for selection, where each one selects one of the options. The utility of the production is the driving factor in the selection process. Thus, the correct path is encoded in the system by the productions' utilities. Lastly, the model's decision has to be translated into action.

Since the environment the Symbolic Maze is represented in has changed, so does the way the model interacts with it. In Fu's experiment, the options were represented as

buttons. Decisions were made by clicking on a button. However, in the Malmo version, options are encoded as tiles on the ground and decisions are made by standing on top of them. As such, the model switched from interacting with the mouse to using the keyboard. The result of the model's decision production yields a direction to move. After that, the model must remember what key is associated with movement in that direction. After recalling the key, the model presses it, causing the Malmo agent to move accordingly. The cycle repeats for as long as the agent is within the maze.

Two special case scenarios exist: restarting and resetting. These are handled with special productions to progress the game. In Fu's version the states brought up special screens with a singular button. The model had to press said button to continue. Similarly, in the Malmo version special rooms represent the restart and reset actions. The agent, upon realizing it is in such a room, has to stand on a singular block in the room. The Malmo version also implements a special mechanism to help it adapt to changes in the maze.

4.2.1 Modulated Annealing

Since answers within the maze constantly change, ACT-R's utility learning mechanism will have difficulty adapting. To compensate for this, the Adaptive Gain Theory (Aston-Jones and Cohen 2005) was added to the model. Assessments of task engagement (Equation 4, reproduced below for convenience) are used to modify the strength of annealing.

$$E = \begin{pmatrix} 1 & logistic(U_{short}) \end{pmatrix} \quad (logistic(U_{long})) \tag{4}$$

Short term utility, U_{short}, is the average utility of all productions executed within a short period of time whereas long term utility, U_{long}, is the average utility over a longer duration. Note that task engagement E is bound between zero and one. A value of zero is taken to represent no engagement and signals full exploratory behavior whereas a value of one is completely engaged and exhibits total exploitation. However, knowing the task engagement doesn't modify behavior.

To cause annealing to occur, the amount of noise within the model needs to change. ACT-R's temperature parameter controls the amount of noise added to a production's utility during conflict resolution. Higher temperature is associated with annealing. Thus, noise is increased, and the model is more likely to select a nonoptimal rule, which leads to exploratory behavior. The inverse is true for low temperature, noise will be lower and the model will tend to select the most optimal rule it knows. Thus, changing the temperature shifts the model's behavior. Task engagement is used to modify temperature within the model (Equation 5).

$$T = \begin{pmatrix} 1 & E \end{pmatrix} \quad T_{Max} \tag{5}$$

The temperature T will be bound between zero and T_{Max} since task engagement E is bound between zero and one. When task engagement is zero, the model is exploring and has the maximum amount of noise the model allows. When it is one, the model has no noise and chooses productions based on applicability and utility alone. This mechanism can be turned on and off in the model so it's impact on performance can be assessed.

5 Results

The model was run two hundred and fifty times without modulated annealing and two hundred and twenty-five times with it. The first model held the temperature parameter constant at the square root of two; the value was chosen to match the parameter from Fu's original model. The model that used modulated annealing calculated short term utility over three seconds and long term utility over one hundred and twenty seconds. The model began in an unengaged state, thus annealing started at full strength. The maximum temperature allowed in the model was also set to the square root of two. The scores for a trial were averaged across all iterations for each group and plotted to show how the models' performed over time (Figure 11).

Figure 11. Average score on each trial for each model. The red dashed line represents the model that used the task engagement system; the blue line is the model that did not.

The figure shows that the performance of the two groups was almost identical. Both models demonstrated an initial learning period between trial zero and twenty. At trial twenty performance drops off because answers in the maze are changed. The models relearn the maze until more changes occur; this pattern continues every time answers are changed until the end. Therefore, the task engagement mechanism had little impact on performance. To determine why the models were so similar, the average task engagement per trial was investigated (shown in Figure 12).

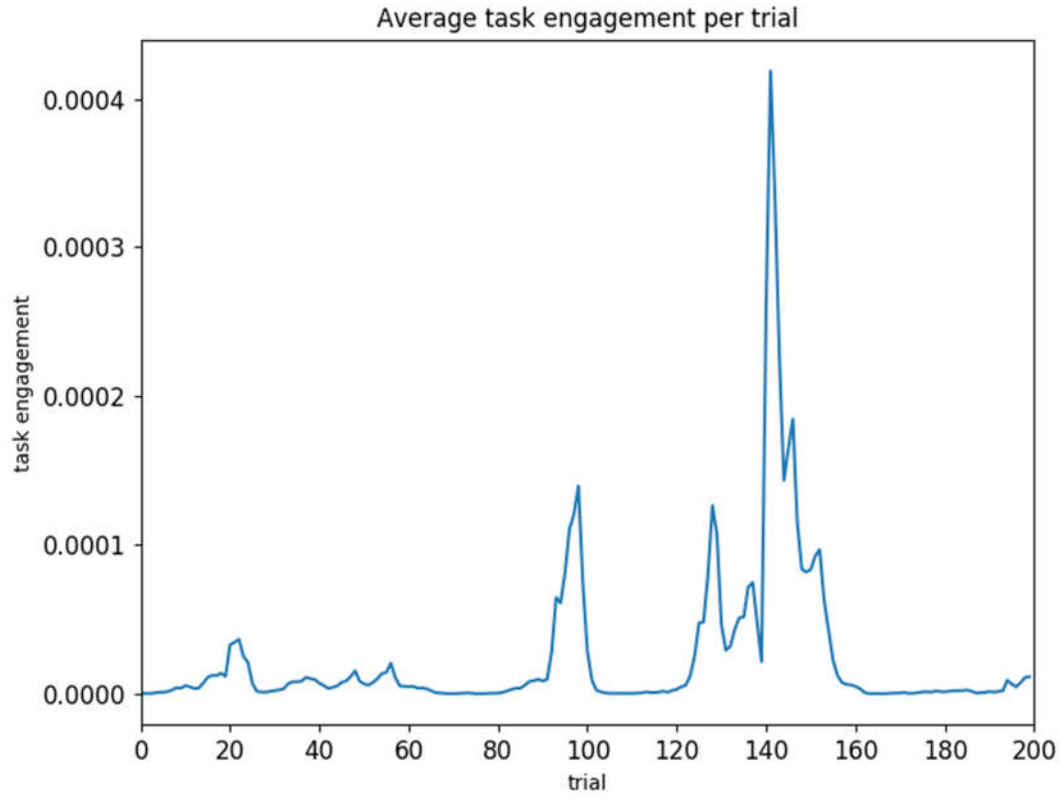


Figure 12. The average of the average task engagement per trial. The figure shows the model was never really engaged, so the system didn't have an impact on performance.

This graph depicts the average of the average task engagement per trial; the task engagement changes throughout a trial and the number of decisions within a trial vary, so averaging over the trials was the only method to compare task engagement. Task engagement does change, however the actual engagement remains very low. Even at its highest peak, the task engagement isn't high enough to vary noise that much. Therefore, the temperature in the modulated annealing model stayed close to the static temperature in the control model, which led to similar performance. In the future, the maximum

temperature should be higher to promote more thorough exploration. This may lead to larger changes in task engagement, promoting changes in temperature and thereby alter performance.

6 Next Steps and Discussion

This section covers future work to complete on the project as well as various improvements that can be made to both the bridge and model. The bridge is discussed first. The model is discussed last.

6.1 Bridge Improvements

Moving forward there are several improvements that can be made to the bridge. First of which is support for multiagent missions. Currently, the bridge only adds a single agent to a mission. All observations from that agent are passed to a single ACT-R model. Similarly, all ACT-R signals are passed to that one agent. However, both Malmo and ACT-R support multiagent simulations. They also both differentiate agents by giving them a name. Three changes have to be made to incorporate multiagent models. First, the bridge must be able to connect to multiple agents/models. Second, it needs to alter how data is stored such that it can be recalled by an agent's name. Lastly, observation collection and event handling must utilize the naming scheme to ensure data is passed between the correct agent and model. These changes still allow single agent missions to be run. The only difference from the current bridge is that the single agent must be

named. Adding support for multiagent missions makes the bridge more generic and will enable more studies to occur.

The bridge should also implement the default game controls. The current edition requires the mission programmer to specify how ACT-R motor signals, key presses and mouse movements, are translated into game actions. This was done to allow missions to utilize any of the three movement types Malmo supports; each use their own commands so the code for each is different. However, this doesn't accurately represent how a model would play the game. Minecraft normally functions under continuous movement controls and binds keypresses to specific commands. The bridge should support that functionality and give programs the opportunity to override it if they choose to utilize another scheme. Providing support for regular game interaction is no small task however. Players and agents have an inventory of items they can interact with. Mouse movement functions differently if the player is looking at their inventory. Thus, the controlling mechanism will have to manage multiple states representing what the agent is doing in order to correctly represent its action. Furthermore, these states need to be managed for each agent present. Malmo also doesn't report an agent's inventory by default. Therefore, a template mission XML must be generated that creates observers for inventory and menu management. However, supporting the default game controls simplifies the development of missions by allowing models to immediately interact with the game world.

6.2 Model Improvements

The bridge isn't the only component that can be improved. The model can be made more accurate and has to be validated. Fu's base model doesn't recreate the learning curve new players experience. At the start of the game, players would have to recall the rules. Furthermore, they won't have knowledge of the maze beforehand; meaning none of the decision productions in the model would exist. Instead, they would have to encode and recall rooms through declarative memory. Fu's model recreates how players function at a later state in the game where they are familiar with the rooms and the rules; though the model still must learn the correct answers. Players are expected to shift between using declarative to procedural memory as the game progresses, showing that they have learned the rules as well as the maze and can act more efficiently. ACT-R can model this change in behavior through a concept known as production compilation, which generates new productions by combining two productions into one. Furthermore, it can incorporate values from declarative memory into the queries of the new productions. Eventually, the model gets compiled into a form similar to Fu's model. Creating such a model would more precisely represent how players learn and how their reaction time changes.

Another major improvement that can be made to the model is incorporating a virtual body. A key component of cognition that ACT-R doesn't model is bodily influences. The body manages the health of the brain and can have effects on cognition. For example, students who lack sleep perform poorly on exams. Such a model could be developed with ACT-R/Φ (Dancy, 2013; Dancy et al., 2015) , a hybrid cognitive architecture that

combines the ACT-R theory of cognition with HumMod's physiological simulation (Hester et al., 2011) and theory from affective neuroscience. HumMod simulates anything from hormone concentration and regulation to blood pressure and flow. The components are connected, thus a change in the regulation of one hormone (or other value) effects the entire system. Furthermore, ACT-R/Φ modifies the functionality of declarative and procedural actions within ACT-R based on various hormones. Unfortunately, the architecture hasn't been updated to the newest version of ACT-R. Once the update occurs, the existing model can be used without modification in ACT-R/Φ, thereby giving the model access to a virtual body and how it influences cognition.

Lastly, the model would benefit from validation against human performance. Data needs to be collected from people playing the Symbolic Maze task. Those data can be compared to model performance to determine if the model is accurate and if the implementation of the adaptive gain theory had an effect. Behavioral data on what decisions players make will be logged, just as it was for the model. On top of that, eye tracking data and galvanic skin response could also be captured. Pupil dilation can be used as an indirect measure of the mode of LC neurons (Aston-Jones and Cohen 2005). This can be used as a mechanism to check the adaptive gain theory implementation. Galvanic skin response provides a proxy for arousal, a measure of cognitive alertness. ACT-R/Φ can compute the arousal for a model, thus providing another way to compare the model and human performance. Skin response also represents how stressed a subject is, meaning the data can be useful to studies of stress in the future. Overall, both the bridge and model can be improved. The data collection and analysis brings the project

together by demonstrating how well cognitive architectures can be used to model people in complex environments using simulations.

7 Conclusion

Cognitive architectures such as ACT-R provide a system for simulating the mind and human behavior. On their own they model decision making of an isolated agent. However, applying a cognitive architecture to a complex environment can yield more interesting results about how people make decisions in more realistic scenarios. To this end, a bridge between ACT-R and Project Malmo, a complex virtual world, was constructed. The system was built to facilitate easy generation of new missions by allowing researchers to specify how to create the world and update it without needing to worry about the overhead of Malmo.

The bridge was tested in a Symbolic Maze task. A model was constructed which implemented the Adaptive Gain Theory by modulating the strength of annealing. The goal was to improve how the model manages the exploration and exploitation trade-off and thereby enhance the model's ability to adapt in a complex environment. However, due to poor parameter selection, the mechanism failed to have a significant effect on performance. In its current state, the bridge provides an interface to an environment which can recreate complex and dangerous tasks, allowing simulations to provide insight into areas which subjects cannot ethically be used to study. Improving the bridge allows

more in-depth experiments to be conducted. Similarly, improving the model moves us one step closer to understanding what governs everyday intelligent behavior.

8 Resources/Citations

Anderson, J.R. (2007). How Can the Human Mind Occur in the Physical Universe? 198 Madison Ave, New York, New York: *Oxford University Press*.

Aston-Jones, G., & Cohen, J. (2005). An Integrative Theory of Locus Coeruleus-Norepinephrine Function: Adaptive Gain and Optimal Performance. *Annual Review of Neuroscience*, 28, 403-50.

Cooke, N. J. and Shope, S. M. (2017). Designing a Synthetic Task Environment. *Scaled Worlds: Development, Validation and Applications*, 263-278.

Bothell, D. (2018a). ACT-R Reference Manual. *ACT-R 7.6+ distribution.*

Bothell, D. (2018b). ACT-R Software Updates 2018. Retrieved from http://act-r.psy.cmu.edu

Bothell, D. (2019). Selecting Productions on the Basis of Their Utilities and Learning these Utilities. *ACT-R 7.6+ distribution.*

Brill, Z. and Dancy, C.L. (2018). Simulating Human-AI Collaboration with ACT-R and Project Malmo. *International Conference of Cognitive Modelling 2018 Proceedings.* 9-10.

Budiu, R. (2013) About ACT-R. Retrieved from http://act-r.psy.cmu.edu/about/

Cohen, J. D., McClure, S. M., & Yu, A. J. (2007). Should I stay or should I go? How the human brain manages the trade-off between exploitation and exploration. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 362(1481), 933–942. doi:10.1098/rstb.2007.2098

Dancy, C.L (2013) ACT-RΦ; A cognitive architecture with physiology and affect. *Biologically Inspired Cognitive Architectures*, 6(1), 40-45.

Dancy, C. L., Ritter, F. E., Berry, K. A., & Klein, L. C. (2015). Using a cognitive architecture with a physiological substrate to represent effects of a psychological stressor on cognition. *Computational and Mathematical Organization Theory, 21*(1), 90-114.

Fu, W., & Anderson J. R. (2006). From recurrent choice to skill learning: A reinforcement-learning model. *Journal of Experimental Psychology: General,* 135(2), 184-206.

Hester, R. L., Brown, A. J., Husband, L., Iliescu, R., Pruett, D., Summers, R., & Coleman, T. G. (2011). HumMod: A modeling environment for the simulation of integrative human physiology. *Frontiers in physiology, 2*(12).

Hobbs, J. N., Ritter, F. E., & Morgan, J. H. (2012). D2P/CLS: A Tutor for Combat Lifesavers. *In Proceedings of the 21st Conference on Behavior Representation in Modeling and Simulation. 12-BRIMS-043. 226-227. BRIMS Society: Amelia Island, FL.*

Hope, R.M, Schoelles, M.J, and Gray, W.D. (2013). Simplifying the interaction between cognitive models and task environments with the JSON Network Interface. *Behavior Research Methods*, 46(4), 1007-12.

Johnson, M., Hofmann, K., Hutton, T., & Bignell, D. (2016). The Malmo platform for artificial intelligence experimentation. *In proceedings of Twenty-Fifth International joint conference on artificial intelligence (IJCAI),* New York, NY, 4246-4247.

Judson, J. (2018). 25 bloodless battles: Synthetic training will help prepare for current and future operations. *DefenseNews*.

Peterson, B. (2008) PEP 373 -- Python 2.7 Release Schedule. Retrieved from https://www.python.org/dev/peps/pep-0373/

Ritter, F. E., Shadbolt, N. R., Elliman, D., Young, R., Gobet, F., & Baxter, G. D. (2003). Techniques for modeling human and organizational behaviour in synthetic environments: A supplementary review. Wright-Patterson Air Force Base, OH: *Human Systems Information Analysis Center.*