CABot1: Technical Report

Chris Huyck, Emma Byrne

September 28, 2009

This report describes CABot1, a games agent built using simulated neurons. CABot 1 is the version of the project finished in July 2007. The agent "lives" in a very simple game, built using the Crystal Space game engine. It is hoped that this report is sufficiently detailed to allow the reader to reproduce the agent from the report. CABot1 is an agent that supports a user in a simple, Crystal Space 3D navigation game. It takes one of several commands from the user, and parses the command to set the goal. The goals are pursued following a partial implementation of a spreading activation net. The agent also views the environment of the game and recognizes two types of object, pyramids and stalactites. A control system is used to alternate between parsing, goal setting, goal fulfilling and awaiting user input.

The parser performs well on the 21 sentences in its repertoire. The performance of nets varies, but a random net usually gets around 98% of parses correct as gauged by semantic frames. Results of tests of the CABot1 agent shows that it succeeds 75% percent of the time in performing the correct action given a user's command.

The agent itself does almost all of its processing in fLIF neurons. The network is divided into 21 subnetworks. The structure of these subnetworks is largely for programming convenience. Neurons coordinate their activity by connections within and between cell assemblies (see Section 9). This report will explain how to download and install CABot1(Section 1). It will give a walkthrough for a typical CABot1 run (Section 2). It will give an overview of the code, and each of the functional areas of the CABot1 network (Sections 3 to 7). It will discuss the fLIF neuron model (Section 8), the cell assembly model (Section 9) and the CABot1 model (Section 11).

1 Downloading and installing CABot1

The CABot1 code can be downloaded from:

http://www.cwa.mdx.ac.uk/CABot/cabot1/cabot1CANT.tar.gz

CABot1 extends several of the classes in CANT23 (also included in the download.) CABot1 and CANT23 is known to run under JRE1.6.0_07. CABot1 talks to the Simple2 game via shared files.

You need to compile `CANT23, fastbind, utils, and cabot1`. There are `.bat` files for each of these: `compilebase.bat`, `compilefastbind.bat`, `utils/compileutils.bat`, and `cabot1/compilecabot1.bat` respectively. You will probably have to change the paths in these bat files, or you can use your favourite development environment. The paths to the files used for communicating between CABot1 and Crystal Space are set in `CABot1.java` with the methods:
`bmpReader.setFilePath(String path)` and
`experiment.setFilePath (String path)`.

You then run the CABot1. There is a bat file provided, `runcabot1.bat`. The paths to the files used for communicating between CABot1 and Crystal Space need to be set here also. Note that CABot1 takes up a lot of memory and the -mx flag has to be set as in the bat file (-mx400200000). Once the 21 CABot nets are up, hit the start button on the base net, and you should be able to run.

To run the agent you'll need the Simple2 Crystal Space game that we developed. Simple2 is based on the Simple2 tutorial that ships with Crystal Space with some modifications. The environment consists of a single room (with no boundary detection) with one object, a user, and the agent.

We found that the most difficult part of getting CABot1 installed is getting Crystal Space running. If you've done it before move on. If not:

- Get the crystalspace3d.org tar.gz file from Crystal Space:
- http://www.crystalspace3d.org/main/Download
- Follow the directions on the building and installing manual link
- Go to the specific instructions for your platform. We have been using Windows and cygwin with all devel and libs.
- Get the CABot1 specific code for Simple2 from www.cwa.mdx.ac.uk/CABot/cabot1/csCabot1.tar
- Place `simple2.cpp` and `simple2.h` files in `CS/apps/tutorials/simple2` Keep a copy of the originals if you like.
- In the CS directory, do a `make simple2` (you should be able to use the build process that comes with CS, but we're old fashioned and did it this way).
- Put in the sprites. The objects in the scene (a pyramid or a stalactite) are not correct. We've built two and they're included in the tar file above. Go to `CS/data` and add the two sprites (chrispyramid and chrisstalagmite) and add them to the standard.zip file.

- Setup the paths. CS communicates with the CABot agent via a series of files. These files have to be stored in a directory that the two systems agree on. I usually put them in `../CANT23/cabot1/data`.
- Modify the Crystal Space vfs.cfg file. The tar file includes my version of it, but it is probably best to edit the one you already have. The relevant line to add is the VFS.Mount.cantAgent line about 20 lines down.
- You should now be able to run simple2 by simply doing ./simple2.exe from the command prompt in CS.

When Simple2 comes up you should see the agent and an object (pyramid). The user has a body (which you can't see but the agent can), and you can move the agent around with the arrow keys. You can look at the agent's view by using the F2 key. If you type in text commands, they get passed onto the agent when you hit return. Commands that work include 'Move forward.', 'Move backward.', 'Turn left.', 'Turn right.', 'Go left.', 'Go right.', 'Turn toward the pyramid.', 'Go to the pyramid.', 'Turn toward the stalactite.', and 'Go to the stalactite.' (note the terminating periods.) These commands should make the agent move if the CABot1 agent is running correctly. CABot1 takes around 1000 CABot cycles to parse a command.

2 Using CABot1

First, ensure that the paths in lines 15 and 16 of `CABot1.java` are set to the path that Crystal Space will write and read from. Build and run Simple2. Then build and run CABot1.

In Simple2 type one of the 10 commands shown in Table 2. On pressing enter the command will appear in an alert box. This command is also passed to CABot1 via the file `csToCabotText.txt` in the path set above. N.B. because of limitations in the way in which Simple2 handles keystrokes, there may be some "stutter" in the typing of the command. CABot1 can handle many mistyped commands, but if it fails to recognise the input id defaults to the command "move backwards."

The Control net ignites, as does the assembly in the base net for the first word of the command. If this is a verb (e.g. "turn"), then the corresponding assembly in the verb frame ignites also. Activation continues to spread through the nets in order for the input sentence to be parsed (see Section 5).

In the meantime, the vision area is permanently on and receiving images via jpeg files, again in the same path as was set above. This input is updated periodically, and is received in the visual input net as a 50×50 array of neurons. These fire if the pixels in the image that correspond to the area of each neuron are darker than a certain threshold. Section 7 details how this visual input is processed by CABot's visual system.

Section 6 explains how the parsed textual input and the processed visual input are used to determine the next action that the CABot agent should perform. Once an action has been decided on, CABot1 writes the command to a text file (`cabotToCSText.txt`) that is read by Simple2.

3 CABot1 Code

The main method for CABot1 is found in `CABot1.java`. The system first calls the method `makeNewSystem()`. This method initialises the network of networks `CABot1Net nets` using the network parameters in `cabot1.xml`.

Al nets are created and the method `connectAllNets` is called. This first creates the other subnetworks (see Table 1). It then calls two methods, `connectParseNets()` and `connectVisionNets`. The `BaseNet` in the network is used to control the operation of the CABot network as a whole via the start, step and parameter controls.

The networks in CABot 1	
The *Base Net*	The *Instance Net*
The *Verb Net*	The *Noun Net*
The *Other Net*	The *Stack Top Net*
The *Stack Net*	The *Test Net*
The *Push Net*	The *Pop Net*
The *Erase Bound Net*	The *Erase Net*
The *Rule Net*	The *Visual Input Net*
The *Retina Net*	The *V1 Net*
The *V2 Net*	The *Control Net*
The *Fact Net*	The *Module Net*
The *Action Net*	

Table 1: The subnetworks in the CABot1 network

The parse nets are those networks that are involved in parsing, that is, `inputNet, nounNet, otherNet, verbNet, instanceNet, stackTopNet, stackNet, popNet, pushNet, eraseNet, eraseBoundNet, testNet` and `ruleNet`. The planning and goaling networks are the `controlNet, factNet, moduleNet` and `actionNet`. The vision nets are `retinaNet, v1Net, v2net`. These networks are connected with connection Methods such as the one in Figure 1.

Neurons in subnetworks may send activation (or inhibition) to neurons in other subnetworks. It is in this way that the processes are executed in CABot1. The code shown in Figure 1 shows how the Control net is connected to the Stack network (labelled `eraseNet` in the method definition).

The first loop makes excitatory connections between excitatory neurons in positions 200 to 399 of the control net to the first 300 neurons in the stackNet. These

```
public void connectControlToStack(CABot1Net eraseNet) {
  //the second control CA stimulates stack 1
  for (int neuronNum=200; neuronNum<400;neuronNum++)
    {
    if (!neurons[neuronNum].isInhibitory())
      for (int synapse = 0; synapse < 5; synapse++)
        {
        int eraseOffset = (int)(Math.random()*300);
        neurons[neuronNum].addConnection(eraseNet.neurons[eraseOffset],1.0);
        }
    }
  //The third control CA suppresses the stack
  for (int neuronNum=400; neuronNum<600;neuronNum++)
    {
    if (neurons[neuronNum].isInhibitory())
      for (int synapse = 0; synapse < 105; synapse++)
        {
        int eraseOffset = (int)(Math.random()*300);
        neurons[neuronNum].addConnection(eraseNet.neurons[eraseOffset],-2.0);
        }
    }
}
```

Figure 1: Sample connection method: `connectControlToStack(CABot1Net eraseNet)`

connections ensure that activation in neurons 200-399 leads to activation in the stack-top. Each (non-inhibitory) neuron in this portion of the control net is connected (with a weight of 1.0) to five randomly chosen neurons in positions 0-299 in the stack net.

The second loop makes inhibitory connections between inhibitory neurons in positions 400-599 in the control net to the first 300 neurons in the stack. This implements part of cell assemblies 2 and 3 in the control net (see Section 4.) Cell assembly 2 turns the first stack item on and cell assembly 3 turns it off. Each inhibitory neuron is connected to 105 of the first 300 neurons in the stack net at random, with an inhibitory weight of -2.0.

3.1 CANT

A CANT network is formed from one or more subnets. Constants such as firing threshold θ and fatigue rate F_c are associated with each subnet and are shared by all neurons in the subnet. Subnets allow easier software engineering by providing a logical way to subdivide the full network.

Since the connectivity determines the behaviour of neurons (see Equation 1), the connectivity determines the behaviour of the network. CANT allows the connectivity to be programmed. By programming the connectivity, CANT can be used to implement different subnets with different functions[1].

[1]The base CANT23 class allows several subnets. The leak, fatigue and firing threshold parameters for each subnet are set independently, and the values for these parameters are stored in an XML file, *ExperimentXMLFile*. Each subnet is an instance of the class CANTNet. Each net has an array of neurons, and these neurons are instances of the class CANTNeuron. Neurons are either inhibitory or excitatory, and this value is stored in the $isInhibitory$ variable. CANT23 has several functions

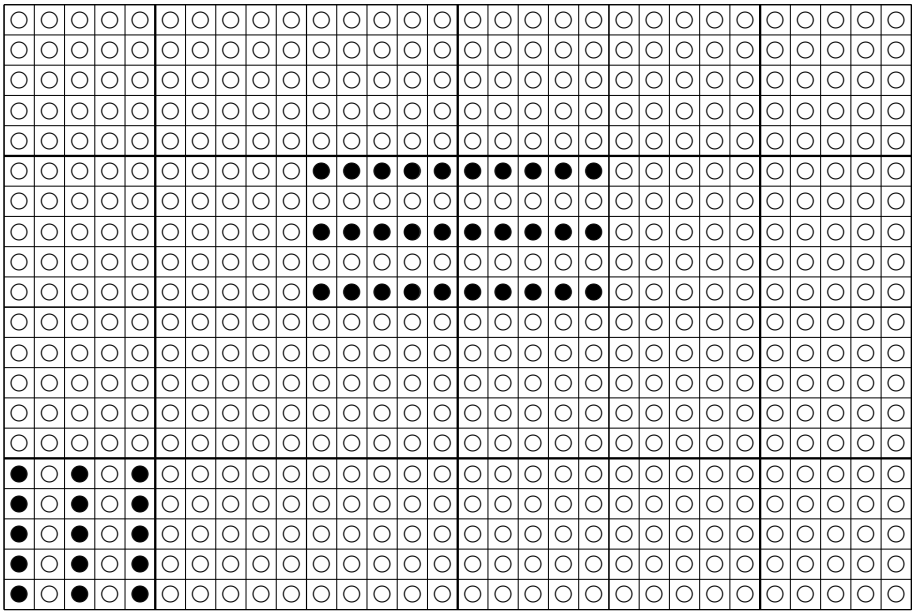

Figure 2: A Sample CA Network.

An example net is shown in figure 2, which is a 30×20 subnet containing 600 neurons. Each circle represents a neuron; if the activation of a neuron surpasses threshold θ (see Equation 5), the neuron is activated or fired (shown in black dot), otherwise it does not fire (shown in white circle). Each of these neurons is a fLIF neuron (see Section 8). The fired neuron spreads its energy to other connected neurons, then loses all energy.

In the mammalian neural system, synaptic connections differ in strength, and any given neuron is unlikely to connect to any other particular neuron [6]. In CANT, the value of the weight between neurons is used to represent their connections; when two neurons have a stronger connection, the weight value is bigger.

3.2 Cell Assembly Topologies

CABot1 is a network made up of 21 specialised subnetworks. Using specialised subnets makes programming more manageable and also makes it possible to set some parameters, such as decay and fatigue, separately.

In each subnet (with the exception of the visual input, retina and V1 subnets) there are several cell assemblies. Cell assemblies are groups of strongly interconnected neurons, such that partial activation of the assembly leads to sustained,

for adding synapses (connections). A synapse is associated with the neuron that it comes from (the pre-synaptic neuron). The pre-synaptic neuron has an array of synapses; these are instances of class Synapse. Each synapse also has a pointer to the post-synaptic neuron.

widespread activation. In CABot1, cell assemblies in all but the V2 subnet are orthogonal (that is, each neuron belongs to only one cell assembly). This need not be the case but again, it makes the design and management of the CABot1 agent more straightforward.

Cell assemblies may also cross subnet boundaries. For example the Noun and Noun Instance cell assemblies that correspond to the same word are essentially the same cell assembly, in that they both ignite in response to the same input, but they play functionally different roles.

3.3 The CABot1 Process and Architecture

The CABot1 agent consists of 21 subnets. These are responsible for:

1. The Base subnet acts as an input layer for words. Individual words in the sentences to be parsed are represented as cell assemblies in this subnet.

2. The Verb subnet has four cell assemblies - one each for the verbs "follow" "move", "turn" and "go". Each one of these cell assemblies acts as a verb frame (see Section 5 below). Each verb frame cell assembly contains three sets of 60 fast bind neurons for short term binding. These bind to actor, objects and location words in the noun or "other word" subnets.

3. The Noun subnet has nine cell assemblies for the words "me", "left", "right", "forward", "back", "stalactite", "pyramid", "door" and "it". Each of these also excites the corresponding Instance cell assembly. The Stack subnet can bind to the noun cell assemblies.

4. The Instance subnet has 11 cell assemblies - each one represents either a word from the noun net, or the prepositions "to" and "toward". The cell assemblies in the Instance subnet have fast bind neurons. These allow nouns to be bound to prepositions in order to generate prepositional phrases.

5. The OWord subnet has cell assemblies for "the", "toward", "to" and the period. "The" is not parsed: a rule discards the determiner when it is encountered. When the "Period" cell assembly ignites, this triggers the rule that completes the parsing of the sentence. The prepositions "to" and "toward" are bound to nouns via the instance net.

6. The Stacktop subnet contains five cell assemblies. When CABot1 initialises, the 0th cell assembly ignites in response to external activation. The active cell assembly indicates the highest occupied position in the stack. When the Stacktop subnet receives activation from the Push subnet, the next cell assembly ignites, and the previous cell assembly is inhibited. When activation is received from the Pop subnet, the previous cell assembly ignites and the

currently active cell assembly is inhibited. In this way the Stacktop subnet works as a counter.

7. The Stack subnet contains four cell assemblies, each one being a position in the stack. Each of these cell assemblies is made up of 150 standard fLIF neurons, which account for the "core" of the cell assembly's activity. Each one also contains 150 fast bind neurons that bind to active cell assemblies in the Noun, Verb and OWord subnetworks.

8. The Test subnet tests if a parsing rule will work, by activating the items in a stack in turn, twice. These stack positions send activation to semantic items, which may ignite parsing rules (see Section 5).

9. The Push increments the stacktop position.

10. The Pop subnet decrements the stacktop position.

11. The Erase Bound subnet prevents items from being erased that are below the stacktop when the erase net runs.

12. The Erase subnet contains 18 cell assemblies that ignite in turn - cell assembly 1 excites cell assembly 2, which in turn excites cell assembly 3 (and inhibits cell assembly 1). Thus, over several time steps, activation 'chains' through the subnet. The Erase subnet is used as a timer that suppresses activation in the stack subnets in order to allow the weights of the fast bind neurons to decay.

13. The Rule subnet contains six cell assemblies, each representing a different parsing rule, such as VP $->$ VP Period (a verb phrase is made up of a verb phrase followed by a period). The Noun, Verb or OWord cell assembly that is bound to the top of the stack sends activation to rule cell assemblies.

14. The Visual Input subnet is not made up of cell assemblies, as the neurons that fire do not belong to coordinated groups. Rather, the visual image that the agent sees in the Crystal Space environment is translated into a 50×50 neuron grid. Where the pixel in the image is darker than a given threshold, the neuron is on. Where the pixel is lighter, the neuron is off. As such, the 2500 neurons act as simple photoreceptors.

15. The Retina subnet contains 6 50×50 grids of neurons. These act in a manner similar to human retinal ganglion cells with varying size on-centre and off-centre receptive fields (see Section 7.2), which respond to "patches" of differing sizes in the visual field.

16. The V1 subnet has neurons that integrate the activity in the Retina subnet. Each of these CAs detects a line or an edge. See Section 7.3 for more details.

17. The V2 subnet is divided into six modules, each of which recognises one of two shapes (a stalactite or a pyramid) at one of three sizes (small, medium, or, large). Each of these modules also contains cell assemblies that indicate where the stalactite or pyramid is in the visual field. See section 7.4 for more details.

18. The Control subnet controls which subnets are active. For example, it allows parsing by suppressing activation in the fact subnet in order to allow activation to take place. See Section 4 for more details.

19. The Fact subnet contains cell assemblies for each of the possible goals that CABot1 may execute, plus information about the content of the visual scene, such as whether there is an object in the scene and if so, where is it.

20. The Module subnet contains cell assemblies for each of the possible goals that CABot1 may execute.

21. The Action subnet contains a cell assembly for each of the primitive actions (turning right and left, going forwards and backwards and two error states) that the agent may carry out in order to fulfil the goals in module net.

Figure 3: The CABot1 gross topology.

These subnetworks are divided into two input regions and four functional regions (see Figure 3). The input regions simply accept text and black/white representations of what CABot1 can see in the game. The four functional regions are described in the next sections.

4 The System Control region

The Control subnet controls the activation of various subnetworks by means of inhibitory and excitatory connections with other networks. There are eight cell assemblies in the Control subnet. These are orthogonal and each contains 200 fLIF neurons. The activation spreads through the subnet from CA to CA. The CAs also have inhibitory and excitatory connections with other subnets.

Control CA 1 (neurons 0 to 199) begins when parsing takes place. This CA 1 inhibits the Fact subnet and excites the CA 2 in the control subnet. It receives an initial 'jolt' of stimulation in the form of activation to 50 of its neurons, when CABot1 is first run. Simultaneous to this the stack top receives external activation, which begins the parsing process. Whilst parsing is ongoing, and stacktop is active, CA 2 in the control subnet is inhibited. Once the rule VP $-$ $>$ VP Period is

encountered, parsing is complete (see Section 5). The stacktop is cleared and the inhibition to CA 2 ceases. Activation in the control net moves on to the next cell assembly.

CA 2 sends activation to CA 3 and inhibition to CA 1 in the control net. It also sends activation to the stack net, which turns the stack on. At this point there is only one item on the stack: the parsed sentence that is result of the parsing process. Activating the parsed sentence causes activation in the fact net, thus setting a goal (see Section 6). Activation in the fact net supresses activation in CA 2 and CA 4 in the control net. Thus, activity in the control net moves to CA 3.

CA 3 in the control net excites CA 4 and inhibits the Stack, Instance and Verb subnets. It remains active as long as an assembly in the fact net remains active. Eventually the goal in the fact net will lead to activation in the module net, which will inhibit the assembly in the fact net (see Section 6). Once the assembly in the fact net is suppresed, CA 4 is no longer inhibited, and CA 4 becomes active.

CA 4 - CA 7 turn the erase net off and on twice. The Erase subnet is used to support the erasing of the fast-bind neuron connections. When activated, the erase net reinforces the fast-bind connections between the CAs that should remain connected (those at or beneath the top of the stack - see Section 5). The Erase subnet activates each of the pairs of CAs that must remain bound three times in succession. This reinforces the weights in the fast bind connections. The Skip subnet prevents the Erase network from reactivating those pairs of CAs that no longer need to be bound (those above the stacktop), thus the weights of the fast-bind neurons that are not reactivated decay to zero.

CA 4 sends activation to CA 5 and to the Erase subnet. CA 5 waits for the erase to complete. CA 5 sends some activation to CA6. When the erase subnet completes its operation, it too sends activation to CA 6. CA 6 activates erase and CA7. CA 7 sends some activation to CA 8, which is fully activated when erase is no longer active.

CA 8 waits for a new sentence to become available from the Simple2 game. If the `commandAvailable` flag in `CABot1Experiment.java` becomes true then activation is sent to CA 1 and the stacktop, the cycle begins again.

5 The language processing (parsing) region

CABot1 has a natural language parser built using cell assemblies of fLIF neurons that parses natural language input and generates the meaning of a sentence given the words and the sentence structure. The parser shows that a traditionally symbolic task, natural language parsing, can be implemented in relatively biologically accurate simulated neurons. The parser can be found at:
http://www.cwa.mdx.ac.uk/CABot/CANT.html and is described in [18]. The parsing system contains 13 CAs. The topology of the parsing system is shown in Figure 5.

‘move forward’	‘move back’
‘turn left	‘turn right’
‘go left’	‘go right’
‘turn toward the stalactite’	‘turn toward the pyramid’
‘go to the pyramid’	‘go to the stalactite’

Table 2: The sentences that CABot1 successfully parses

```
public void connectInputToVerb(CABotlNet verbNet) {
   connectInputWordToVerb(0,0,verbNet); //Follow to V1
   connectInputWordToVerb(5*150,1*480,verbNet); //Move to V2
   connectInputWordToVerb(4*150,2*480,verbNet); //Turn to V3
   connectInputWordToVerb(3*150,3*480,verbNet); //Go to V4
}

private void connectInputWordToVerb(int inputStart, int verbStart,
  CABotlNet verbNet) {
  int verbOffset;
  for (int inputNeuron = inputStart; inputNeuron < inputStart + 150; inputNeuron ++)
    {
    for (int synapse = 0; synapse < 7; synapse++)
      {
      verbOffset = (int)(Math.random()*300);
      verbOffset += verbStart;
      if (!neurons[inputNeuron].isInhibitory())
        neurons[inputNeuron].addConnection(verbNet.neurons[verbOffset],0.35);//.5
      }
    }
}
```

Figure 4: The connection between text input and the verb cell assemblies is created by these methods in CABot1Net.java

The parser in CABot1 is a neural parser, like that implemented by Knoblauch and Palm [20]. However, unlike Knoblauch and Palm’s parser, the CABot1 parser can interpret context free languages. The parser interprets the text commands that the CABot1 agent receives from the user. These commands are drawn from a repertoire of ten sentences (see Table 2).

Text arriving from the game activates neurons in the Input (Base) subnetwork. These are connected to the nouns, verbs and other words in the lexicon. For example, cell assemblies 0, 5, 4 and 3 are connected to verbs 0, 1, 2, 3 (‘follow,’ ‘move,’ ‘turn’ and ‘go’) in the verb net. Each neuron in the input cell assemblies synapses with seven random neurons in the matching verb cell assembly (See Figure 4).

The parser has a lexicon (implemented as noun, verb and “other word” subnets containing one cell assembly per word) and a small grammar (with storage in verb frames) (see Table 3). Each verb CA in the verb subnet acts as a verb frame. The assembly consists of 240 standard neurons that constitute the verb symbol, three sets of 60 fast bind neurons that bind to an actor, location or object CA in the noun

Nouns	Verbs	Other
‘me’	‘follow’	‘period’
‘left’	‘move’	‘the’
‘right’	‘turn’	‘to’
‘forward’	‘go’	‘toward’
‘back’		
‘pyramid’		
‘stalactite’		
‘door’		
‘it’		

Table 3: CABot1’s lexicon

subnets respectively [2]. These fast bind neurons are activated in response to rules (see Section 3.3), and 60 fast bind neurons that synapse to these three slots.

The parser is a partial implementation of a Marcus parser [22]. It has a stack, preferences for rule selection, limited look-ahead and grammar rules that combine syntax and semantics in the form of verb frames. The stack allows the parser to interpret context free languages.

The noun subnet contains nine cell assemblies, each of which corresponds to one of the nouns listed above. When a noun is read in, the noun cell assembly ignites and is bound to an element in the stack. The corresponding instance cell assembly is also ignited.

Whilst parsing, words (in the form of active cell assemblies in the word subnets) are bound to stack positions. Verbs are bound to verb frames using fast bind neurons in the Verb subnet (see Section 8.4). In the Instance subnet, fast bind neurons bind noun instances to prepositions in order to represent noun phrases. The parser is a Push Down Automaton: the state of the parsing system changes depending on the current state and inputs. The parsing algorithm is as follows:

1. Start by pushing a word onto the stack
2. Repeat
   (a) Test rules
   (b) If no rule succeeds,
      i. Push new word on stack
   (c) Else (a rule succeeds)
      i. Apply the rule

[2]The grammar used in CABot1 is such that no Actor objects are ever encountered, thus these fast bind neurons are not used in the current implementation.

Parameter	Verb Subnet	Noun Subnet	OWord Subnet
θ Threshold	4	4	4
τ Decay Rate	1.5	1.5	1.5
F_c Fatigue	0.1	0.2	0.1
F_r Recovery rate	0.8	0.8	0.8
η Learning rate	0.2	NA	NA

Table 4: Parameter values for the Verb, Noun and Object Word subnets. Learning rates are not given for the noun and other word subnets as these subnets contain no fast bind neurons.

Parameter	Stack Subnet	Stacktop Subnet	Instance
θ Threshold	4	4	4
τ Decay Rate	2	1.5	0.5
F_c Fatigue	1	1	0.01
F_r Recovery rate	3	3	0.011
η Learning rate	0.2	NA	NA

Table 5: Parameter values for the Stack and Stacktop. A learning rate is not given for nets that contain no fast bind neurons.

ii. Pop stack

3. Until VP -> VP Period rule applied

The grammar is defined by the rules in the parser. Rules are stored as CAs in the rule subnet. When one of these rules fires it ignites CAs in the word subnets and the stack operation subnets. The rules can construct verb phrases and prepositional phrases from verbs, nouns and prepositions. Determiners are discarded via a rule that pops them from the top of the stack.

If a preposition is at the top of the stack, the PP->PP noun rule fires. The instance subnet forms prepositional phrases by binding instance cell assemblies (representing noun phrases) together via fast bind neurons. The CAs in the instance net for the noun and the next preposition in the stack are bound together via fast

Parameter	Rule Subnet
θ Threshold	4
τ Decay Rate	1.05
F_c Fatigue	0.3
F_r Recovery rate	0.6
η Learning rate	0.1

Table 6: Parameter values for the Rule subnet

```
private void connectVerbToV_NPPLoc(CABot1Net ruleNet) {
for (int verb = 0; verb < 4; verb++)
  {
  for (int verbNeuron = 0; verbNeuron < 240; verbNeuron ++)
    {
    int fromNeuron = verb*480+verbNeuron;
    for (int synapse = 0; synapse < 15; synapse++)
      {
      int ruleOffset = (((verbNeuron*5)+synapse)%300) + 1200;
      if (!neurons[verbNeuron].isInhibitory())
        neurons[fromNeuron].addConnection(ruleNet.neurons[ruleOffset],0.02);
      }
    }
  }
}
```

Figure 5: The connection between verb cell assemblies and a rule is created by this method in CABot1Net.java

bind neurons. The noun and preposition are popped from the stack and the bound prepositional phrase, in the form of a filled noun frame, is left on the stack.

If a noun phrase is at the top of the stack the VP->VP NPObj rule may fire. Slots in verb frames are filled when a noun phrase is at the top of the stack, immediately preceded by a verb phrase. The nouns in the noun phrase are attached to the slots in the verb frames via fast bind neurons. The noun phrase is then popped from the stack. This is done by the rule firing the object slot neurons in the verb frame. As the slot neurons and the noun frame are simultaneously firing, they bind.

If a full stop is encountered at the top of the stack, the parser tries to apply the VP -> VP Period rule. At this point the stack is halted and the VP and control passes to the planning and goaling system (see Section 6). Connections between the verb and rule nets are instantiated by the method shown in Figure 5. Each neuron in a verb cell assembly synapses with 15 neurons in the respective rule cells.

The parser was tested, in isolation from CABot1, on each of the 17 sentences using five grammar rules and 16 words that fall into five lexical categories. The best performing network was over 99% accurate on 32 presentations of each of the 17 sentences (the performance of the parser is 542/544=99.63%). The errors that did arise came from mistakes in the choice of slot fillers.

6 The Planning and Goaling region

Planning and goaling control the CABot agent's behaviour. Goals are set in response to natural language text commands from the user. When a command is parsed, assemblies in the verb and instance nets are active.

Spreading activation between sub-networks executes the appropriate plans for

these goals. The control net inhibits the fact net whilst parsing takes place, and the fact net inhibits the control net whilst goal-directed activity takes place. This ensures that processing is carried out one step at a time. The parameters of the subnetworks involved in planning and goaling are shown in Table 7.

Facts and goals are combined in the CABot1 FactNet. Goals may be simple, compound or object driven. The four primitive goals are:

- turn+left
- turn+right
- move+forward
- move+backward

There are two compound goals that are:

- go left
- go right

These goals consist of a turn to the left or right, followed by one step forward. The subsidiary goal "forward after turn" is used to activate the goal "go forward" once a turn to the left or right is accomplished. the goal "forward after turn" can not be set directly.

There are two object-driven goals:

- turn toward
- go to

There are two goal objects that can be combined with the last two compound goals: pyramid and stalactite.

There are six facts that may be active:

- pyramid in scene
- stalactite in scene
- object in left
- object in right
- object in centre
- object big

These facts are partially ignited by spreading activation from the visual system. They only become completely active when activation spreads within the fact net from a relevant goal that is also active. The steps in the planning and goaling process, as represented by populations of active assemblies, can be thought of as states in a finite state automaton (see Section 10). Activation in the verb, instance and vision cell assemblies act as preconditions to transitions between states. When all necessary "precondition" cell assemblies are fully active, the next assembly in the process becomes activated.

The control net inhibits the fact net whilst parsing is taking place. Once parsing is complete, the appropriate goal in the fact net will be ignited by spreading activation from a completed verb frame in the verb net and associated instance nets.

```
private void connectOneVerbToOneFact(int verbNum, int factNum,
  CABot1Net factNet) {
  for (int neuronNum=0; neuronNum<300;neuronNum++)
    {
    int fromNeuron = neuronNum+(verbNum*480);
    if (!neurons[fromNeuron].isInhibitory())
      for (int synapseNum=0;synapseNum < 4; synapseNum++)
        {
        int toNeuron = ((int)(Math.random()*100))+(factNum*100);
        neurons[fromNeuron].addConnection(factNet.neurons[toNeuron],0.25);
        }
    }
}
```

Figure 6: Verbs in the verb net are connected to the goals in the fact net via the `connectOneVerbToOneFact(int verbNum, int factNum, CABot1Net factNet)`

```
public void connectVerbToFact(CABot1Net factNet) {
  connectOneVerbToOneFact(2,0,factNet); //Turn to turn left
  connectOneVerbToOneFact(2,1,factNet); //Turn to turn right
  connectOneVerbToOneFact(1,2,factNet); //Move to move forward
  connectOneVerbToOneFact(1,3,factNet); //Move to move back
  connectOneVerbToOneFact(3,4,factNet); //Go to 'turnleft+go' of factNet
  connectOneVerbToOneFact(3,6,factNet); //Go to 'turnright+go' of factNet
  connectOneVerbToOneFact(2,7,factNet); //Turn to turn toward
  connectOneVerbToOneFact(3,10,factNet); //Go to goto
}
```

Figure 7: The method `connectVerbToFact(CABot1Net factNet)` creates the connections that spread activation between verbs and the fact net.

For example, verb 1 is "turn." Activation here sends activation to fact 1 "turn+left" and fact 2 "turn+right." `connectOneVerbToOneFact(int verbNum, int factNum, CABot1Net factNet)` sends four synapses from each of 300 neurons associated with verb `verbNum` to 100 neurons in the `factNum` assembly in the fact net, each with a weight of 0.25. Note that the total number of neurons devoted to each verb in the verb net is 480. The `verbNum*480` and `factNum*100`

ensure that the correct assemblies are connected.

Similar methods, `connectOneInstanceToOneFact(int instanceNum, int factNum, CABot1Net factNet)` and `connectInstanceToFact(CABot1Net factNet)` exist to connect instances to facts. For example the instance assembly for "left" connects to facts "turn left" and "go left."

Parameter	Fact subnet	Control subnet	Module Subnet	Action Subnet
θ Threshold	4	4	1.5	4
τ Decay Rate	1.5	2	1.5	2
F_c Fatigue	0.4	1	0.4	1
F_r Recovery rate	1.2	3	1.2	0.3

Table 7: Parameter values for the Fact, Control, Module and Action subnetworks.

The primitive goals spread activation to corresponding assemblies in the module net, which then spreads that activation directly to the action net. There is a one-to-one connection between the primitive assemblies in the fact, module and action nets for these goals.

The compound goals send activation to the module net. The module net then sends activation to a number of simple actions in sequence. For example, the compound goal"go left" sends activation to the "go left" module, which ignites the "turn left" and the "go forward after turn" assemblies in the module net.

The object driven goals send spreading activation to the module net. However, the module net also requires activation from the fact assemblies in order to ignite. For example: if the goal and object "turn toward" and "goal pyramid" are both active in the fact net, and the facts "pyramid in scene" and "object on left" are both active, then the module net will receive enough activation for the turn left module cell assembly to ignite. The module net will then determine which sub-goal is the most appropriate (in this case, "turn left"). This goal will be repeatedly activated, and the appropriate action activated, until the facts "pyramid in scene" and "object in centre" become active. The module net will then ensure that the action "go forward" is repeatedly executed until the fact "object big" is active. If the object "drifts" out of centre, the goal "turn left" or "turn right" will become active again.

If the object disappears from the agent's field of vision, then an error module ignites and turns the goal off. Once a simple or compound goal has been executed, or the object of a goal occupies the entire visual field ("object big" in the fact net), the goal net ceases to be active and the control net can switch state.

7 Vision

The visual system of CABot1 consists of four subnets: a visual input network, a retina and two visual processing areas, V1 and V2.

The retina and V1 share some similarities with their human counterparts, but are much simplified models of only a few of these elements. V2 is less biologically plausible, and does not mimic known mechanisms in the human visual system. It does however carry out two important functions of the visual system: the identification of *what* is seen and *where* it is in the visual field.

Parameter	Visual Input Subnet	Retina Subnet	V1 Subnet	V2 Subnet
θ Threshold	4	4	4	5
τ Decay Rate	1.2	2	2	1.1
F_c Fatigue	0.4	0.3	0.6	1
F_r Recovery rate	0.8	0.6	0.15	3

Table 8: Parameter values for the Visual Input, Retina, V1 and V2 subnets

7.1 Visual Input

The visual input subnet is a 50*50 network of fLIF neurons. As fatigue is switched off and the input is externally presented, this subnet acts as a constant stimulus until the agent's point of view changes. Activation levels in the visual input subnet are the direct result of the input received from the game and are not dependent on the activation of any other neurons. The activation of the visual input subnet is retinotopic: each neuron in the 50*50 subnet corresponds to an identically located "cell" in a 50*50 grid applied to the input from the environment.

At intervals, the Crystal Space game writes a 400 $\times$ 400 pixel JPEG file that shows the grey-scale image that the CABot1 agent can see in the game. This grid is translated into a 50*50 grid by taking every eighth row and column, each cell of which corresponds to one of the neurons in the visual input subnet. The CABot1 agent receives this as visual input via a CANT pattern. A CANT pattern is an array that stipulates the level of activation for each neuron in the visual input subnet. Where the RGB value of one of the cells in the grid is below a threshold (-9,000,000) the CANT pattern activates the respective neuron. Otherwise, no activation is received (see Figure 8).

7.2 Retina

The CABot1 retina subnet contains six 50*50 grids of fLIF neurons. Each of these neurons acts as either an on-centre or an off-centre retinal ganglion cell. In the human visual system, the retinal ganglion cells respond favourably to patches of light, with a dark surround (on-centre cells) or patches of dark with a light surround (off-centre cells), of differing sizes. Each CABot1 off- and on-centre cell has a square receptive field that receives activation from the visual input net. The level of activity in each of the retinal neurons is the result of feedforward connections from a 3*3, 6*6 or 9*9 grid of cells in the visual input subnet. The receptive fields

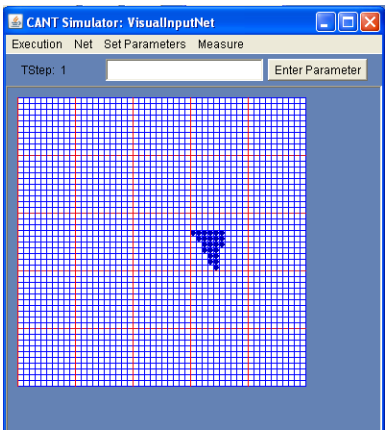

Figure 8: Activation in the visual input network. A single stalactite is in the scene.

are segmented into a centre and a surround: the 3*3 receptive fields have a single-cell centre, the 6*6 receptive fields have a 2*2 cell centre and the 9*9 receptive fields have a 3*3 cell centre.

Each of the 150,000 neurons in the retina receives input from an n*n grid of cells in the visual input subnet. The position of the neuron in the retina subnet is the same as the position of the central neuron in the n*n grid[3] in the visual input subsystem.

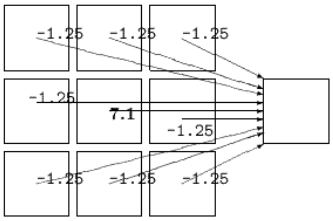

Figure 9: The connections to a 3 x 3 on/off centre surround cell (right) in the retina subnet from neurons in the visual input subnet (left).

An on-centre neuron is connected to the visual input subnet such that activation in the centre of the receptive field excites the retinal neuron, whilst activation at the edges of the retinal field inhibits the retinal neuron. An off-centre neuron is connected to the visual input subnet such that the reverse is true: excitation at the centre inhibits the retinal neuron and excitation at the edges excites it.

The on- and off-centre cells simulate processing in the late retina. This simu-

[3] in the 6*6 neuron receptive field the neuron is in the same position as the upper left neuron in the 2*2 cell centre of the receptive field.

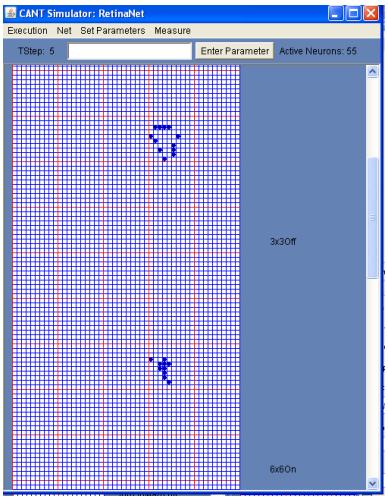

Figure 10: Activation in part of the retina network, in response to the input shown in Figure 8. Retinotopic is activation can be seen in the 3x3 off centre cells and 6x6 on centre cells.

lation of the retina makes a number of simplifying omissions with respect to the human visual system. For example there are no colour photoreceptors, there is no fovea (acuity is constant across the visual field) as these features are not required for an object identification and location task. The simulation of the on- and off-centre ganglion cells is sufficient for the requirements of the visual processing at subsequent stages, to recognise lines and shapes, and to determine location. Figure 10 shows the activation in two of the retinal cell assembles in response to the stalactite in Figure 8.

7.3 Area V1

In the V1 area of the human visual system there are neurons, known as simple cells, that are tuned to specific line orientations. These simple cells are location specific. In the CABot1 V1 subnet, model neurons exhibit this type of behaviour. These model neurons are selective for horizontal and acute and oblique angled lines and edges. There are also model neurons that identify compound shapes composed of these oriented lines - the 'and', 'or', 'less-than' and 'greater than' angles.

Weighted connections feed activation from on-centre and off-centre cells in the retina subnet. The horizontal and angled lines each accumulate activation from three 3*3 on-centre cells in the retina subnet. For example, the horizontal line detector fires when it receives activation greater than a threshold $\theta = 4$. If a V1 horizontally selective neuron (call it $H_{[0,0]}$) is at coordinates $[0, 0]$ then three on-centre cells with 3*3 receptive fields centred at $[-1, 0]$, $[0, 0]$ and $[0, 1]$ are connected to $H_{[0,0]}$, each with a weight of 1.4 (see Figure 9).

The neurons that identify angles accumulate activation from both on- and off-centre cells in the retina of multiple receptive field sizes. Figure 11 shows activation in two of the cell assemblies in the V1 net in response to the activation in Figure 10.

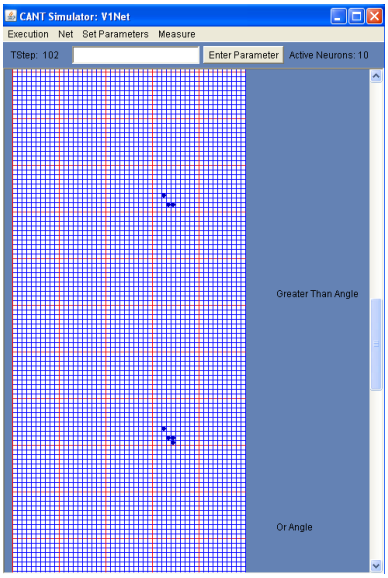

Figure 11: Activation in V1 in response to the activation in 10. A single stalactite is in the scene. V1 identifies a ‘greater than angle’ (the top left corner of the stalactite) and an ‘or angle’ (the bottom corner of the stalactite’

7.4 Area V2

Area V2 is the least biologically plausible of the visual subnets. It is able to recognise both ”what” and ”where” features of the visual stimulus. There are six modules in the in the V2 subsystem, made up of a number of overlapping cell assemblies. The six modules are specialised to recognise small, medium and large projections of an upright triangle(pyramid) or downward triangle(stalactite) respectively.

The “what” (object recognition) pathway of V2 is made up of object- and size-specific modules. Within these modules, cell assemblies ignite in response to the contents of the visual field, as mediated by the connections from the Retina and V1 subnets. For example, if a small stalactite is visible, a cell assembly in the small stalactite module will ignite.

Figure 12 shows the activation in the small stalactite cell assembly in response to activation in Figure 11.

The same modules also carry the “where” (position) information. Each module is a 50*50 grid of fLIF neurons, divided into a 5*5 grid of subsections (of 10*10

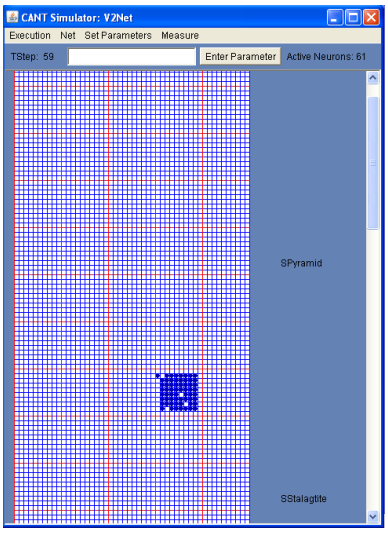

Figure 12: Retinotopic activation in V2 in response to the activation in V1: a small stalactite has been detected in the middle-left of the visual field.

neurons each). Each of the 25 grid sections corresponds to a retinotopic area of the visual field: the top left section corresponds to the top left area of the visual field and so on. Each of these sections functions as a "position" cell assembly. The cell assembly corresponding to the position of an object in the visual field is primed in each module. Extra activation from the "what" pathway is enough to ignite the position cell assembly in the module corresponding to the object in the visual field.

8 fLIF Neurons

The CABot1 network is built on the fLIF neuron model, an idealised model of a biological neuron that is an extension of the Integrate and Fire neuron model [23, 13]. There are a wide range of neural models, and the fatiguing Leaky Integrate and Fire (fLIF) model is one that is simple, relatively biologically faithful, and efficient to simulate. Most neural simulations based on neurons of this, relatively simple, complexity learn new things. For example, fLIF neurons have been used to learn hierarchical categories [16] and in real world categorisation tasks [14]. However, fLIF nets can also be used to implement programs without learning. This technical report describes how the CABot1 games agent is implemented using networks of fLIF neurons.

Integrate and Fire (IF) neurons are a long-standing and widely-used model of neural activity [23]. The IF neuron is a model of a spiking neuron - at a given time step, if the activation that reaches the neuron passes a certain threshold then the neuron fires. Maass and Bishop extended this model to include a leak component

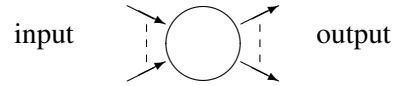

Figure 13: A Simplified Model of a Biological Neuron.

[21], based on the observations that some of the activation in a biological neuron 'leaks away' over time if the neuron does not fire. This model is more biologically plausible than the simple IF neuron, and it precludes firing caused by the accumulation of trivial amounts of activation over very long periods of time. The fatigue component chase the refs in [9] - esp TRACE models the observation that repeated firings lead to an increase in the threshold level of activation that a neuron must surpass in order to fire.

There are a number of biological features that the fLIF model does not address, such as the opening of ion transfer channels, or synaptic delays. However, these features are below the useful level of granularity for the purposes of this model. fLIF neurons represent processes that take place in around 10ms of biological time. A model neuron that fires at a given time step can be considered to have 'spiked'.

A fLIF neuron is described by three sets of equations that define:

1. Firing, in response to the integration of activation levels
2. The leaking of potentiation
3. The fatiguing of neurons due to their firing

Neurons fire in discrete cycles with time steps. As the model does not take synaptic delays or refractory periods into account, these time steps are tentatively mapped to 10ms of biological time. This also enables the system to perform efficiently enough to enable several 100,000 neurons to be simulated on a PC in real time.

8.1 Integrate and Fire

Figure 13 is a simplified model of a biological neuron. At its most basic level, a neuron receives activation (energy or "potentiation") from other neurons and fires once this activation passes a given level. Biological neurons that fire send activation on to other neurons via connections called *synapses*. The strength of the connection between two neurons is modelled using a simple weight coefficient.

Let E_i be the level of activation energy for neuron i. Let W_{ji} be the strength of the connection from neuron j to neuron i. Let P_j be a "potential flag". $P_j = 1$ if neuron j fires or $P_j = 0$ if neuron j does not fire.

$$E_i(t) = \sum_{j=1}^{n} W_{ji} \times P_j(t-1) \quad (1)$$

Thus the activation energy (E) of neuron i at time t is the sum of the activation passed from other neurons connected to i that fired at time $t-1$.

Biological neurons may spread either excitatory or inhibitory signals. Both convey information between neurons, the difference is that the inhibitory signals always reduce or remove action potential of a neuron, and excitatory signals always add action potential to a neuron. When a neuron receives inhibitory inputs, the neuron becomes less active. To model this, weights may have negative values.

Let θ be the firing threshold for the network. That is, for any neuron i, if $E_i > \theta$ then neuron i fires. All neurons in a given subnetwork (see Section 3.2 for the definition of a subnetwork) have the same value of θ.

Implicit in Equation 1 is the discrete nature of the simulation. The model and the code that implements it assumes that there are discrete cycles. All of the neurons have a chance to fire, and the activity is passed to other neurons for reintegration in the next cycle.

8.2 Leak

Biological neurons integrate activation over a period of time (several time steps in the model). However, some of that activation 'leaks away' after it arrives at the neuron. Terms are added to represent the accumulation of activation and the leaking of activation if the neuron does not fire.

In Equation 2, $E_i(t)$ is the sum of the activation passed from other neurons connected to i that fired at time $t-1$ and the existing activation energy of neuron i at time $t-1$ reduced by a decay constant. Let d be a decay constant such that $d > 1$. All neurons in a given subnetwork have the same value of d. The activation energy of neuron i at time t is now given by the following equation:

$$E_i(t) = \frac{1}{d} E_i(t-1) + \sum W_{ji} \times P_j(t-1) \quad (2)$$

There is an exception to this: if a neuron fires, all of its activation leaks away. Thus if neuron i fires at time $t-1$, the activation carried over to time step t is 0, rather than $E_i(t-1)$.

8.3 Fatigue

Neurons fatigue after firing. Immediately after firing there is a brief refractory period of 2-3ms. This element of fatigue occurs at $\langle$10ms intervals and so is not relevant to the model. After repeated firings, the neuron experiences longer term fatigue in which the response of that neuron diminishes. This feature of the biological cell is modelled with a *fatigue level*, that uses a *fatigue constant* and a *fatigue*

recovery constant. The higher the fatigue level, more energy the neuron needs in order to fire. Thus it is possible to model the reduction in the spiking rates of a fatigued neuron.

Let $F_i(t)$ be the fatigue level of neuron i at time t. Let F_c be a fatigue constant, which increases the fatigue level if a neuron fires. Let F_r be a recovery constant that decreases the fatigue if a neuron does not fire. The overall fatigue level has a lower bound of 0.

If neuron i fires:

$$F_i(t) = F_i(t-1) + F_c \tag{3}$$

If neuron i does not fire:

$$F_i(t) = max\{0, F_i(t-1) - F_r\} \tag{4}$$

Note that F_c and F_r are positive and may take identical values. The ratio between the fatigue constant and the fatigue recovery rate determines the maximum proportion of the neurons in a cell assembly that may be firing on average at any time. For the purposes of the model, all neurons in a subnetwork have the same values for F_c and F_r, with the value of $F_i(t)$ entirely dependent on the firing behaviour of neuron i in previous time steps.

A cell assembly is made of a finite number of fLIF neurons. If these all fatigue at the same time then the activity in the cell assembly will spontaneously extinguish. If these neurons do not fatigue then the cell assembly will continue to fire indefinitely. Cell assemblies must oscillate in their activity in order to remain active for a sustained period. Having a proportion of neurons in the assembly that are not firing at any given time step allows these to recover as the rest of the cell assembly remains active. Different subnets in CABot1 have different ratios between their F_c and F_r values and therefore have different persistence levels.

Taking fatigue into account in the model, a neuron i fires if and only if:

$$E_i(t) - F_i(t) > \theta \tag{5}$$

8.4 Variable binding with fast bind fLIF Neurons

In section 8, W_{ji} was defined as the weight variable that represents the connection strength from neuron j to neuron i. It is possible for these weights to be plastic, thus allowing the network to learn. In CABot1, long term links are created manually, so no long term learning takes place. Decaying Short Term Potentiation (STP) is used to make short term modifications of connections between neurons in CABot1. These STP links form the basis of the bindings used in processing. For a more complete explanation see [15].

Biological systems make use of short term potentiation (STP) as well as long term potentiation (LTP) [12, 1]. In LTP, when a neuron fires, the connection between that and other neurons that are simultaneously firing is strengthened. Conversely, the connection between a firing neuron and all neurons that are not firing is

weakened. These processes are referred to as Hebbian learning and anti-Hebbian learning respectively. With STP, as with LTP, simultaneous firing of two neurons increases the synaptic weight between them. However, with STP the synaptic weights decay to their initial strengths relatively soon after the firing.

STP is a plausible mechanism for binding [17]. There is evidence that STP in the human brain provides support for LTP by reinforcing the activation levels in the emerging CA [19]. STP has also been proposed as a basis for working memory [8]. In CABot1, STP is used to bind concepts. STP acts as a means of temporarily associating concepts in order that processing may take place. A class of neurons called "fast bind neurons" permits short term connections. Certain CAs have fast bind neurons that connect to cell assemblies in other subnets. Ordinarily the weight of these connections is 0. When two neurons are active at the same time the weight of these connections increases, thus binding the two cell assemblies. At each time step that the two cell assemblies do not fire together, the weights decay until they reach 0. For example, the stack subnet has a learning rate of 0.2, that is, when the pre and post synaptic neurons are both firing together, the weights of the fast bind synapses increases by 0.2 per time step.

STP via fast bind neurons is used in CABot1 to permit variable binding. Variable binding is used in parsing, to bind nodes (existing CAs representing words) to elements in the stack, and to bind slot fillers (words or phrases) to verb frames. For example, if the command "Turn right" is received via the Crystal Space interface the CA for the word "Turn" ignites in the verb network as does the 0th element in the stacktop subnet. The 0th element in the stacktop subnet ignites the 0th node in the stack subnet. As these the stack and verb CAs are firing together, fast bind neurons in the stack are activated and STP increases the weights connecting them. If fast bind neurons are sufficiently high in weight, ignition of one stack CA will cause the bound word CA to ignite. If the fast-bind synapses are still of sufficiently high weight then it is possible to retrieve the word at stack position 0 by activating the stack CA (causing the word CA to ignite).

As STP is based on weights that automatically decay, and as the LTP weights in CABot1 are not changed by learning, these bindings are automatically erased.

9 Cell Assemblies

A cell assembly (CA) is a set of neurons within a network that have high mutual synaptic strength. As a result when a few of the neurons in the assembly fire, mutually reinforcing activation tends to propagate to the rest of the CA. The CA will then "reverberate", maintaining the activation pattern, even in the absence of external stimuli. This reverberation serves not only to allow patterns of activity to persist, but also facilitates the strengthening of links between neurons, aiding learning.

Hebb suggested the CA as a basic unit of neural processing thus: "[A] repeated

stimulation of specific receptors will lead slowly to the formation of an 'assembly' of association-area cells which can act briefly as a closed system after stimulation has ceased; this prolongs the time during which the structural changes of learning can occur and constitutes the simplest instance of a representative process (image or idea)." [11, pp 60]. Hebb suggested two types of memory, a "dual trace" system. Instantly formed, evanescent memories (such as the short term memory of a sequence of numbers) may be the result of a "reverberatory trace" whilst an assembly is active. This trace will persist for a short time (on the order of half- to one second) even when the stimulus is removed or changed. After decay the memory will not be readily retrievable. In the second type of longer term memory, the reverberatory trace allows the memory to persist long enough for *physical* changes to take place at the synaptic level, thus leading to a long term change in the association between neurons. This has useful parallels with the psychological concept of short term or working memory and long term memory respectively.

10 Processing with fLIF neurons

Much of the processing in CABot1 relies on the fact that the sub-networks in CABot1, and indeed CABot1 as a whole, can function as a finite state automaton. A finite state automaton (FSA) is a quintuple $A = \langle \Sigma, S, S_0, T, nF \rangle$ where:
Σ is the input alphabet. S is a finite, non-empty set of states.
In general, $S_0 \subset S$ is the set of start states. If S_0 is not a singleton, then the automaton is non-deterministic. In CABot1, S_0 is always a singleton.
T is a set of transition rules such that $T : S \times \Sigma \rightarrow S$.
$F \subset S$ is a set of final states, such that no further transitions are applied.

In CABot1, the input alphabet is a set of text and visual inputs. The set of states are the cell assemblies in the subnets. Transition rules are the weights of the connections between these cell assemblies. In order to allow the transition between different states, depending on input, a CA may send excitatory input to a number of other CAs. This is enough to prime those CAs, but extra activation is required from the inputs in order to cause one of the CAs to ignite.

A sequence topology is a subnet, or set of assemblies that cross subnets. In these subnets, when a CA ignites it sends activation to the next CA and inhibition previous CA. Ignition then spreads from one CA to another in sequence. It is possible for the sequence to branch. A CA may send activation to two (or more) CAs. This activation is enough to prime those CAs, but is not enough to make either one ignite. One of these CAs will then receive activation from some other CA or direct from input which is enough to ignite that CA. This in turn inhibits the CA that was primed but did not ignite.

The general processing abilities of the Cell Assembly architecture are described in [7] and [2].

11 Discussion

The CABot1 agent is an agent that can parse text commands in order to navigate a simple, virtual 3D environment. The agent is novel in that its architecture is entirely based on cell assemblies of fLIF neurons. These biologically plausible ensembles of neurons permit "mental states" to arise and to persist for some time. Mental states (in the form of cell assemblies) cause other mental states by means of excitatory and inhibitory connections to other cell assemblies.

CABot1 is capable of parsing text commands. The stack permits it to parse context free languages. The agent also has a visual system that is capable of recognising objects and their positions within the scene. Parsed text commands and the representation of the objects in vision permit the agent to determine the actions that will allow it to carry out commands such as "go to the pyramid".

CABot1 does not include a capacity for long term learning. The cell assemblies are manually constructed and their interconnections hard coded to result in the desired behaviour. It is possible, however, to add the capacity to learn through the longer term alteration of weights. CABot1 is a relatively simple agent but it potentially forms the basis of a much more ambitious neuro-cognitive architecture that will allow a better understanding of embodied cognition via cell assemblies.

CABot1 is a software agent that is based solely on simulated neurons. The purpose of this agent is to assist the user in this environment. The agent is embedded in a 3D virtual game environment. The CABot1 agent is able to understand natural language instructions from a user. These may be simple commands, such as 'turn left' or higher level goals, such as 'go to the pyramid'. CABot1 is able to carry out appropriate actions in response to these instructions.

There is, however, a more fundamental reason for placing the CABot1 agent in a virtual environment. CABot1 is, in effect, an *initial step* towards a neuro-cognitive architecture, masquerading as a games agent. As such, the decision has been taken to make CABot1 an embodied agent with a neural architecture.

An embodied agent is able to learn from its interactions with its environment. CABot1 is 'virtually' embodied, in that is has no physical existence, but it receives and processes sensory data via its visual inputs, and the 'speech' of the user via text input. As a result, CABot1 is able to ground the symbols that appear in the user's speech in terms of the visual inputs it receives from the environment, thus addressing the classic symbol grounding problem (see Section 12) [5, 3, 10].

The neural architecture of CABot1 is based on biologically plausible fatiguing, leaky, integrate-and-fire (fLIF) neurons (see Section 8). Sets of neurons can be used as Cell Assemblies (CAs), the neural basis of concepts (see Section 9). Cell Assemblies were posited by Hebb [11] as the underlying structure of an element of thought. CAs are tightly connected groups of neurons that spread activation among themselves. Once a CA receives sufficient activation to 'ignite', the level of activity in the assembly spreads and persists until fatigue, or an inhibitory signal

from outside the assembly, causes its activation to cease.

CABot1 uses CAs built from fLIF neurons to implement a number of different modules. CABot1 has a modular architecture in order to facilitate development and analysis. We do not claim that these modules necessarily have counterparts in the human brain, however elements such as the retinal and V1 networks harness known features of human vision.

12 Related Work

The CABot1 agent was designed to address one of the underlying problems of language acquisition: the symbol grounding problem. According to Harnad [10], "the meaning of... symbols comes from connecting the symbol system to the world 'in the right way.' But it seems apparent that the problem of connecting up with the world in the right way is virtually coextensive with the problem of cognition itself."

Cangelosi and Harnad [4] propose two complementary methods of the acquisition of meaning for symbols: *sensorimotor toil* versus *symbolic theft*. Symbolic "theft" is the process of learning symbols by means of an interaction between a knowledgeable and a naive individual (let us call them K and N respectively). N can learn from K's propositional statements. These propositional statements provide N with information about a symbol's meaning. So N need never see a zebra in order to learn what one is if K is able to define the zebra as "a striped horse."

However, suppose our N does not know the meaning of "striped", or "horse". K may provide definitions such as "having a pattern of bars or lines of two or more alternating colours" or "a fast, domesticated mammal". Again, poor N may not know what a "pattern", a "line", a "colour" or a "mammal" is. The problem is one of infinite regress. This is, in short, the symbol grounding problem: symbolic theft is only possible if N has an existing repertoire of grounded symbols. These groundings can, therefore, only be obtained by sensorimotor toil.

CABot1 is, first and foremost, an agent that is designed to support users in 3D game environments. However, given that the CABot1 agent can see its environment and react to natural language commands, it is sufficiently embodied to form the basis for a system that can learn from its environment through sensorimotor toil. CABot1 does not do this at present, but there are plans to add the ability to learn symbols from the environment.

References

[1] Dean V. Buonomano. Distinct functional types of associative long-term potentiation inneocortical and hippocampal pyramidal neurons. *J. Neurosci.*, 19(16):6748–6754, aug 1999.

[2] Emma Byrne and Chris Huyck. Processing with cell assemblies. *Neurocomputing - Under Review*, 2009.

[3] Angelo Cangelosi, Alberto Greco, and Stevan Harnad. Symbol grounding and the symbolic theft hypothesis. *Simulating the Evolution of Language*, 2002.

[4] Angelo Cangelosi and Stevan Harnad. The adaptive advantage of symbolic theft over sensorimotor toil: Groundinglanguage in perceptual categories. *Evolution of Communication*, 4:117—142, 2000.

[5] Angelo Cangelosi and Domenico Parisi. *Simulating the Evolution of Language*. Springer, 1 edition, dec 2001.

[6] Patricia S. Churchland and Terrence J. Sejnowski. *The Computational Brain (Computational Neuroscience)*. MIT Press, new ed edition, mar 1994.

[7] Yulei Fan and Christian Huyck. Implementation of finite state automata using flif neurons. In *Seventh Conference on Cybernetics Systems*, 2008.

[8] Stefano Fusi. Neuroscience: A quiescent working memory. *Science*, 319(5869):1495–1496, mar 2008.

[9] H. Ghalib and C. Huyck. A cell assembly model of sequential memory. In *Neural Networks, 2007. IJCNN 2007. International Joint Conference on*, pages 625–630, 2007.

[10] Stevan Harnad. The symbol grounding problem. *Physica D*, 42:335–346, 1990.

[11] D.O. Hebb. *The Organization of Behaviour*. J Wiley and Sons, 1949.

[12] Chris M. Hempel, Kenichi H. Hartman, X.-J. Wang, Gina G. Turrigiano, and Sacha B. Nelson. Multiple forms of short-term plasticity at excitatory synapses in rat medial prefrontal cortex. *J Neurophysiol*, 83(5):3031–3041, may 2000.

[13] J.J. Hopfield. Neural networks and physical systems with emergent collective computationalabilities. *PNAS*, 79(8):2554–2558, apr 1982.

[14] C. Huyck and V. Orengo. Information retrieval and categorisation using a cell assembly network. *Neural Computing and Applications*, 14:282–289, 2005.

[15] Chris Huyck. Variable binding by synaptic strength change. *Connection Science*, to appear, 2009.

[16] Christian Huyck. Creating hierarchical categories using cell assemblies. *Connection Science*, 19(1):1–24, mar 2007.

[17] Christian Huyck. Variable binding by synaptic strength change. *In Preparation*, 2008.

[18] Christian Huyck and Yulei Fan. Parsing with flif neurons. In *Proceedings of Advances in Cybernetic Systems, 2007*, Dublin, 2007.

[19] Stephen Kaplan, Martin Sonntag, and Eric Chown. Tracing recurrent activity in cognitive elements (trace): a model oftemporal dynamics in a cell assembly - connection science. *Connection Science*, 3(2):179–206, 1991.

[20] Andreas Knoblauch and Günther Palm. Pattern separation and synchronization in spiking associative memories andvisual areas. *Neural Networks*, 14(6-7):763–780, jul 2001.

[21] Wolfgang Maass and Christopher M. Bishop. *Pulsed Neural Networks*. MIT Press, apr 2001.

[22] Mitchell P. Marcus. *Theory of Syntactic Recognition for Natural Languages*. MIT Press, 1980.

[23] Warren McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5(4):115–133, dec 1943.