A Divide and Defend Firewall

Walter B. Gress V

Drexel University

Friday, June 7 2013

This concept of operations details a “Divide and Defend” Firewall. Recently there has been surge of malware attacks, those attacks typically being orchestrated through a Botnets. Modern enterprise and personal firewalls are not designed by default to handle massive number of attacks at once. D&D is designed with that very thought in mind.

# introduction

As said in the abstract, botnets have become more and more of a threat. Instead of taking the offense in this situation, I suggest taking a lightweight approach to the botnet problem until the proper authority can be contacted or

the attacks (bot) are cut off from their ISP, or some other form naturally occurring termination. The lightweight procedures includes a divide and conquer, modular, procedure, making it the host’s firewall more difficult to crash the firewall. Also, it allows for a crashed component to be recovered and made up and functional once again.

# feature list and ITERATIVE PROPERTIES OF THE D&D FIREWALL

This section consists of a series of requirements for the D&D Firewall in the form of several bulleted notions.

* The central concept is to have the firewall and its components in linked modules, making it harder to attack.
* The Firewall Core (hereto after referred to as the Core), the Main Firewall controlling software is isolated from the rest of the network, inaccessible through any outside ports.
* The D&D firewall uses micro or mini firewalls to block incoming attacks. These firewalls are designed to run lightweight threads and to be quick to spawn, despite performing all of the duties of a firewall. Being small and lightweight allows them to not only react quickly, there are multiple firewalls to handle multiple attackers, and should they crash they are lightweight enough that a new mini firewall can be thrown up again for the Core as quicker than would a typical firewall architecture.
* A data abstraction layer exists between the Core and the “mini” firewalls that are created to defend against attacks. The Core spawns the micro-firewalls through the DAL and out into the web where they can block botnet attacks. IT is a one-way pipeline. The Core can allow spawning of the micro-firewalls through the DAL, but the micro-firewalls cannot communicate back to the Core.
* Each thread spawned through the DAL has firewall software running or “attached to” the thread.
* When one mini firewall goes down, the Core (main firewall) immediately respawns a mini-firewall through the DAL.
* Since the mini-firewalls take up little resources, they can be spawned, respawned, and so on again and again taking much effort to crash the host computer.
* When a mini-firewall is generated, it generates according to the hostname and port currently being attacked. If other hostnames and ports are attacked, additional mini firewalls are spawned.
* Since the Main Firewall (Core) software managing the mini firewalls is isolated, the enemy is unable to easily penetrate and attack the Core and in essence, for an analogy, “swats away flies”, the flies being the bots attempting atbreakinginto the system, the flyswatter(s) being the mini-firewalls.

# general structurE of firewall

The firewall consists of three primary things. 1) The Monolithic Core which is designed to be implemented as small and fast as possible.

2) The Data Abstraction Layer, meant to “confuse” the enemy malware by implementing the “proxy” like behaviour of a firewall. That is, it means it the frontdoor so to speak, or gateway. The DAL is loosely coupled so that it may change its ip address of on the fly. Should a botnet be attacking 101.1268.1.1, the DAL will immediately shifts its ip address to another available (possibly stored similar to how proxy server software does present day).So now its ip address, whose only purpose is to proxy, sitting on the gateway, has changed, and the botnet no longer knows where its attacking.

Also, as mentioned earlier the Core of the firewall, that is, the most of the firewall functionality we administer we put in the Core, allowing the the mini/micro firewalls to be very lightweight.

As said, the Firewall spawns micro-firewall from monolithic core to block enemy attack on the dummy host ip address and then shifts the dummy ip address to another (these dummy ip addresses are the address info for the Data Abstraction Layer). If a new dummy host attack occurs, the main firewall generates another micro-firewall and shifts the DAL’s dummy ip, and so on….This part I’m not sure of how to do: how to shift the DAL’s ip address. I considered adding another proxy, one proxy only outgoing and the other tying the two proxies together. Instead of changing the ip address of the DA, change the connection between the DAL and this new, internal proxy. That way should the attacker try to attack , he wouldn’t get beyond the dummy proxy. (DAL)

3. When not under attack, the DAL operates as a normal gateway between the internal and external networks.

The main firewall core control all of the mini/micro firewalls that are spawned as threads from it.

havior for the robocup bot.

# short psychologiy background

For those with interest in the comparison of this project, the subsumption architecture, with the psychology of C.G. Jung and Sigmund Freud I’ve elaborated briefly in this section. For those not interested you may skip to section 5 without any loss of information. This is simply optional background material and not necessary to understand the project in general. In order to introduce the modifications made to the subsumption architecture’s stack, some references can be made to the models of the mind generated by early psychologists who actually founded the field. Both Sigmund Freud and Carl Jung helped create the field we know today and each had their own idea of a model of the brain. Sigmund Freud’s model is known as the “iceberg” model in that he viewed the conscious mind to be like a tip of an iceberg, with a superego and ego exposed. The superego being in charge with high-level functions like planning, setting goals, and self-image and the ego being the conscious mind capable of everyday thought. Below the “waters” of this so called iceberg existed a subconscious, where the super ego also extended to but in which also existed a part of the mind known as the id: responsible for more animalistic and primitive functions such as desire for food, pain, reflexes, instincts, etc. The combination of the id and superego formed the motivations for the actions of the conscious mind, the ego.

Carl Jung, a pupil of Sigmund, developed his own theory of the unconscious that contained a model similar to Freud’s model, but also contained archetypes or complexes, as they are sometimes referred to. These archetypes were present, Jung reasoned, because they existed across all cultures. Jung also delved into some mysticism in his theories of a collective unconscious, but those are outside the scope of this paper. For an image of the iceberg model see **Figure** 1 in the appendix.

# subsumption architecture and the unconscious

The subsumption architecture developed by Rodney Brooks resembles a somewhat tapered stack. From here on out we will assume it has the properties of both a priority queue, allowing us to move behaviors up and down the “stack.” The subsumption architecture shares a few common attributes with Freud’s iceberg model. Both models, despite the fact Brook’s model is for simple robots of intelligence not much greater than that of an insect, share common parallels. At the lower level of Freud’s iceberg exist the mind’s most basic and necessary needs for life and survival. The same occurs in Brooks’ subsumption architecture, where the reflex like behaviors are on the bottom of the stack and will takeover in cases of danger or other cases where reflex like actions is necessary. Brook’s architecture has the higher-level functions of the robot’s “mind” (to use the term loosely), can be subsumed by the lower parts, and sit higher in the architecture. The higher functions of both the robot’s mind in Brook’s model and the higher functions of the human’s mind in Freud’s model both are subsumed by lower levels of the stack in times of emergency and the two models are very closely coupled when taken to comparison.

Brook’s model however is static, where Freud’s iceberg model allowed for the suppression and release of certain thoughts or behaviors. This is not present in Brook’s model and is the first modification this work will apply. An example why this is necessary can be presented in the frame of reference of RoboCup soccer. Let us assume that two bots are approaching the goal. The original plan for the first bot is to attempt to shoot to make a goal. However, suddenly the second bot comes into range and it is more advantageous to pass to this player. However, because Brook’s architecture is static, several different levels may separate the two behaviors and one will not be able to subsume the other. Using the concept of “suppression” that Freud introduced in his iceberg model, we could push the one behavior for the bot downwards and suppress it, and “release” the other behavior, bringing it upwards along the stack. This movement up and down the stack, suppression and release of thoughts and behaviors, makes the stack dynamic.

Also, we should consider the additions of archetypes made by Jung to the idea of the unconscious. Although Jung hypothesized his archetypes to occur in the unconscious, it seems from the subsumption model that using them in a different fashion may be more advantageous. For example, treating these archetypes, such as the “Mother”, or the “Child”, or a leadership type, as aggregates of weights on known/planned behaviors. For example, the archetype of Child may have a heavier weight on its “following” traits in the architecture, making it more likely to follow other players than say, a leadership archetype that would spend more time chasing after the soccer ball than assisting other players. The archetype of “Mother” may essentially “mother” another player, spending more time on playing defense of a single or multiple specific players. *These archetypes are primarily the decision makers and* *represent* *both personality and behavior.* An archetype will personify a certain series of traits, such as with the Mother archetype, more heavy weights in the defense parameters. The personality traits are determined by the weights of certain parameters. *When it is time to make a decision, these archetypes and their* *weighted behaviors, with each archetype representing a behavior, all contribute to the central decision-making process*. *The convergence of this decision making process among archetypes forms the conscious decision is the action the agent takes.*

In the simulation there are multiple parameters that compose the behavior of each player, and the Jungian archetypes essentially give names and inspiration for different strategies for each internal agent to adopt. Also, more heavily weighted behaviors or strategies would carry a heavier priority when moving upwards or downwards along the stack. These archetypes could also be more simple, defined as simple behaviors, such as “chase the ball”, “block player x”, or “shoot to goal.” The complexity of the archetypes vary per archetype. See **Figure 2** for an illustration of Brook’s architecture in the appendix.

# hypothesis

The hypothesis presented is that that because of the use of the archetypes, each robot will present itself differently and with a slightly different “personality,” than other bots playing. Varying the assigned archetypes will result in changing the weights on certain behaviors, and furthermore cause different social behaviors to emerge. Social behaviors being, for example where there may be many “Child” archetypes a follow the leader type environment where the Child archetypes follow one bot, playing defense to protect that one bot when it has the ball, and mimicking enemy bots behavior when they have possession. Another example would be a “Leader” archetype along with child archetypes, and mother archetypes. I predict the mother archetypes will cause the agent to play strong defense over the child archetypes, and the child archetypes will follow the leader archetypes. A stated in section 4, an archetype will represent cooperative decision makers in order to solve the internal conflicts experienced in subsumption architecture. The archetypes themselves contain both personality and behavior. For more information on the archetypes and their possible behaviors see the end of section 4.

# DESIGNING ARCHETYPE STACK

Note that, like in Freud’s Iceberg model, the conceptualization as we progress from the lower layers to the upper layers become more complex and as in a human being, the reactive layers subsume (take over) the higher level behaviors if necessary (pain reactions, fight/flight behavior, etc.) Except, it begs the question, where is the “conscious” and “subconscious” in the subsumption architecture? Or is it even possible for this type of division.

The first modification made to the subsumption architecture is the ability to move behaviors from layer to layer, that is, suppress or resurface behaviors. This eventually resulted in the feed-forward nature of the archetype stack. Take for instance the case of two robots playing soccer on the same team. Consider a configuration where one robot approaches the goal with the ball and is ready to shoot. Then suddenly, the team mate has approached the goal and is closer and has a better shot. We would want the behavior to pass to the team mate to override the behavior to kick the ball to the goal. This is not possible in a static architecture. Therefore, we grant the “behaviors” as we will call them for the time being to be able to be suppressed or released along the stack.

One of the major flaws of Brook’s architecture is the periodic deliberation that occurs because of conflicts within layers. Suppose the architecture is not composed of individual behaviors, but of simple agents that can communicate with each other. This is similar to Jung’s model of the collective unconscious with its Archetypes. Assuming each level has multiple Archetypes that are able to communicate and interact, each with its own personality and ability to make decisions. These Archetypes would be able to make decisions through archetype to archetype communication, passing those decisions upward along the stack.

Supposing each of these Archetypes has its own behaviors and its own personality based off of a series of universal parameters. When it is time to make a decision, all the Archetype agents cooperate to come to a final decision. In fact, the structure of the communication and compromise between the agents is based off of a feed-forward network.

For an illustration of the archetype stack and its functionality, see **Figure** 3 in the appendix.

# DESIGN

## Archetypical Stack

In the archetypical stack diagram (Figure 3) each circle represents an instance of an Archetype. The interactions between the archetypes, resulting in the archetype on a higher level of the stack, displaying its “feed-forward” nature and also the mediation between behaviors. The archetypes communicate with each other to arrive at decisions; lower, more reactive and primal behaviors in a higher multitude than upper, more complex and abstract behaviors. The most important thing is the red circle at the very top, which represents the convergence point of all the decisions; the so-called “conscious” decision the stack has arrived at. This singular decision is the result of all the compromises of subconscious archetypes.

## Negotiating Between Archetypes

Archetypes must negotiate between themselves in order for their input to be passed further up the stack to the top in the feed-forward manner. The method of this negotiation is based off of feed-forward neural networks and the intrinsic weights built into the archetypes. The bottom layer archetypes receive input signals. Those input signal are multiplied between archetypes, their weights magnified by the input. When two archetypes match up so that the combination of their magnified weights surpasses the threshold, then it is said that the two archetypes, along with their characteristics, have completed negotiations and fire upward to the next archetype on the stack.

For example

Archetype 1: Offense 0.3, Defense 0.4, Leadership 0.1

Archetype 2: Offense 0.1 Defense 0.2, Leadership 0.2

Magnifier input 2 and Threshold 1

Then archetypes one and two negotiate, since Archetype 1 Defense is 0.4 magnified to 0.8 and Archetype 2 Defense is magnified to 0.4, there sums being 1.2, greater than the threshold 1.

## Archetypical Stack Peak and Behavior Production

Once the feed-forward network has reached the top of the stack, the peak, the decision is made to perform a complex behavior, based off of the parameters that have been averaged and forwarded to the peak archetype.

This complex behaviour is engaged, and consists of a complex behaviour function that it consists of several simple behaviours to accomplish this behaviour. These simple behaviours are forwarded to ATAN (the Robocup Library that was used in this article to simplify the command execution to the Robocup server) to be executed in the soccer simulation and the decision making process is at a point of completion.

## Archetype Parameters

Each archetype simply consisted of the following six parameters:

1. Defensiveness: How strong is the bot’s sense of duty to defend his other players.
2. Offensiveness: How strong is the bot’s sense of offense; that is, what is its drive towards making a goal.
3. Aggression: Similar to offensiveness, but behaviourally is more related to interfering with the opponent and the opponent’s play.
4. Teamwork: How strong the bot’s sense of duty towards his other players is and has a direct correlation on strategy and how often the player will pass the soccer ball.
5. Follower: How much the soccer player will mimic other players, both on the offense and defense, to simulate learning.
6. Directing: Leadership. How often the bot will direct other players and like teamwork, work together with other bots.

## Archetypes

The Archetypes are loosely based off of C.G. Jung’s Archetypes from his book, “The Archetypes of the Collective Unconscious.”

These “pure” Archetypes (unmixed with any others through negotiation) live at the bottom of the archetype stack and are the first to negotiate.

They are the following:

1. Leader: Tends to direct other players
2. Mother: Tends to defend own players
3. Chid: Tends to mimic players
4. Shadow: Extremely aggressive
5. WiseOldMan: Calculated, doesn’t rush into things.
6. Maiden: Very defensive
7. Anima: Both very defensive and very offensive
8. Trickster: Directing, Offensive, Defensive, Low Teamwork.

## Complex Behaviors

The following are the complex behaviours that are acted out according to the parameters of the top of the archetype stack and their aggregate simple behaviours that are executed in ATAN.

1. Chase the soccer ball
   1. Turn Towards Ball
   2. Dash Quickly
2. Defend player x on your team
   1. (Can See Nothing) Turn 180 degrees
   2. If (InfoSeePlayerOwn) is true
   3. Dash Quickly
3. Defend player y on enemies Team
   1. (Can See Nothing) Turn180 degrees
   2. If (InfoSeePlayerOther) is true
   3. Dash Quickly
4. Pass to player x on own team
   1. If (InfoSeePlayerOwn) is true
   2. Kick ball at a certain rate
5. Mimic behavior of player x on your team
   1. If (InfoSeePlayerOwn) is true
   2. Store current action of teammate in cache
   3. When mimic is triggered, perform cached actions
   4. Reset cache
6. Mimic behaviour of player y on other team
   1. If (InfoSeePlayerOther) is true
   2. Store current action of other player in cache
   3. When mimic is triggered, perform cached actions
   4. Reset cache
7. “Hold” ball, dribble to goal and pass to a striker
   1. If (CanSeeBallAction) and if (CanSeePlayerOwn) and
   2. distanceOwnGoal is short, kick to PlayerOwn
8. Play midfield, try to disposses defending team from ball (left midfield and right midfield.
   1. If (CanSeeBallAction) and if (CanSeePlayerOther) dash towards player
   2. Try to kick ball
9. Play heavy defense (crowd defending player with ball)
   1. If (CanSeeOtherAction) and (CanSeeBallAction) move towards player.
   2. Communicate to other players to play (9)
10. Play heavy offense (all players charge towards goal)
    1. Commmunicate to other player to play 10
    2. Move all players towards goals

# related work

Much work has been done in RoboCup, developing intelligent soccer playing agents. The majority, Everest as an example, use either a finite-state machine approach, or a learning ANN style approach. In order to prepare for the designs of this project, I researched the rules of soccer, feed-forward neural networks, and went through the heavy documentation on the protocol Robocup uses to communicate between agents and the server. Most of the communication between the server and the agents was done using a Java Wrapper library called ATAN, which is one library amongst many similar, and was provided with a skeleton Robocup agent to which I could build my Archetype agent. The skeleton agent was known simply as “Simple,” and its primary behavior was to swarm after the soccer ball, pushing it towards the opposing goal using as many players as possible.

# empirIcal approach

The ultimate goal was to measure the success in ridding the architecture of internal, debilitating, conflict. Initially, we had hoped to script a large number of matches, each game sampled for the following statistics. Unfortunately, a mechanism for automating the start of matches compatible with the library (ATAN) being used could not be found in the given time. Therefore the flow of the game could not be measured quantitatively and only qualitatively. Because of this loss of experimental data, video clips of the matches at crucial points have been included for demonstration and to compensate.

For 9 matches, the following statistics were gathered:

* The results of three games of the simple agent of which our agent was built overtop. The Simples Team agent came with the ATAN library.
* The results of three games against the very Simple Team that came with the ATAN library and that our team was built upon.
* The results of three games with our team played against a professional team, in this case, Team Everest: <http://www.cs.cmu.edu/~guyang/Everest/>
* The flow of the game. How many robots display a “shakey” behavior, that is, how often it freezes deliberating its next decision. Often in robocup this manifests as the bot turning in place in circles for no apparent reason.
* Number of times counted of how often robots appear to act in a strategic manner. This will be captured on video.
* Number of times counted of how often robots appear to take any other form of emergent behavior. This will also be captured on video.
* Team Cooperation. This is another subjective quality to be captured on video.

# metrics

Statistics to gather for each of the sample runs (but not limited only to these statistics):

* Number of games won by home team
* Number of deliberations occurred per game
* Number of observed strategies
* Number of observed emergent behaviour
* Team cooperation

Observed strategies and behaviours will obviously have to be done a per sample basis and use human judgement. Video will be taken to record and support this portion of the data.

Team Cooperation is a metric based off of a sum of how many times the soccer ball is passed from one player to another. This is another internal statistic. The more times the ball is passed between players, the higher the team cooperation. Higher team cooperation values are desirable.

# results

## Simple vs. Everest

As a control, the Simple Team that the project was built on top of by ATAN played a professional Robocup 2D Simulation League Team, Team Everest.

Out of the three matches, Simple showed “shakiness” (deliberation) seven times. It also showed one player every game that simply wandered the field, adding another three instances of deliberation for a total of ten. Simple’s only strategy was to swarm after the soccer ball, no matter where on the field Simple was or where on the field the ball or Everest was, even swarming when the ball was close to Simple’s goal and a goalie should have been defending the goal. The only emergent behaviour that could be seen was the swarming behaviour, that was Simple’s only real behaviour. There was one exception, during the second match Simple briefly guarded its goal while the rest swarmed against the other player. This seemed to be more of a fluke than anything else.

Everest’s behaviour consisted of pushing the ball forward, passing back and forth between players, no deliberation, but also no emergent behaviour among agents. Everest’s strategy was clearly hard-coded and finite state machine like.

Simple is very stupid and will not even approach midfield for a kick off. Often Simple will swarm after the ball, leaving Simple’s own goal free, even when the offensive team is close to Simple’s goal. Everest’s strategy easily outwits Simple’s.

## Archetype vs. Simple

Our team, Archetype, is built on top of Simple, using the Archetype Stack to generate unique behaviors in game. Clearly Archetype improves upon Simple. Archetype did not simply swarm after the ball like Simple, but showed instances of strategy and emergent behaviour.

Archetype however had some difficulties dribbling the ball and targeting the goal, but these are based off of constants that would simply need to be tweaked in the complex behaviour portion of the stack. Also, Archetype does not pass well, again a problem that could be easily remedied by tweaking the constants in the complex behaviour portion of the stack. The fact that passing actually occurs, which does not occur in Simple, is significant. The first match was lost simply because of the pure aggressiveness of Simple. Archetype clearly demonstrated strategy and emergent behaviour, leaving behind a few players to defend the goal and the others to swarm and play offense. This was not directly programmed in Archetype but a result of the archetype stack’s behaviours.

Also, often, when the ball comes close to the home goal, an agent will circle around the ball, in essence defending it in an unusual way in which could be considered a form of emergent behaviour.

Archetype also has trouble with kicking the ball out of bounds and with incomplete passes. These two areas were not a major focus when designing the various behaviours of the player agents, and can be improved in the future somewhat easily as well.

In the final match, the game went into multiple overtimes, ending with me stopping the game because there wasn’t code implemented to handle penalty kicks.

Archetype was often able to keep the ball on the opposing teams side of the field and Archetype would crowd its own goal when the opposing player would approach the goal with the ball. On the last match, Archetype did not perform as well, having less teamwork with some agents standing still on the field waiting for something to do when they should have been moving around to try and find the ball. Also Archetype would often gain possession of the ball but because of the amount of force assigned to its kick procedure would perform incomplete passes to other players. Also, Archetype had trouble aiming at the goal when performing a goal kick. These are all problems that can be resolved in the behaviours, the implementation of ATAN’s library code.

Overall Archetype seemed to deliberate very little, but often would wander, most likely when the agent’s sensors were not picking up the ball or an opposing player and the agent did not know what to do. Emergent behaviour-wise was the protection of the goal and the ball when the opposing player was close to the goal. This behaviour was not programmed in but was a result of the archetype stack implementation.

## Archetype vs Everest

In match one of Archetype vs Everest,

two Everest players deliberated for a few seconds. Archetype, still would perform different strategies, sometimes placing all players to defend the goal, sometimes splitting the number of players between guarding the goal and playing offense against Everest. Sometimes all players would play offense. Unfortunately, since the behaviours could be more complete, Archetype does not know how to handle a free kick, which puts Archetype back significantly and yielding an advantage to Everest. Archetype is not a full-fledged soccer agent at this point, and was not designed to be, but was designed to not produce deliberation or difficulty in negotiating a decision. The goal is to eventually develop a full soccer-playing agent using the archetype stack method. Also during the second match Archetype displayed an emergent behaviour of crowding the goal when Everest approached. It performed this behaviour multiple times, making it difficult for Everest to score. Archetype also does not have functionality built in for goal kicks as well, another advantage given to Everest, a full-fledged professional soccer client.

Archetype in the second match against Everest often times had Everest score on them, even though Archetype had a goalie in and several others surrounding. This problem is due to the fact that Archetype needs to look around when one of its bots are idle, otherwise as was the case, they do not sense the ball when it approaches.

## Score Table

|  |  |  |
| --- | --- | --- |
| **Game** | **Simple** | **Everest** |
| 1 | 0 | 26 |
| 2 | 0 | 32 |
| 3 | 0 | 22 |
| **Game** | **Archetype** | **Simple** |
| 1 | 0 | 1 |
| 2 | 1 | 0 |
| 3 | 0 | 0 |
| **Game** | **Archetype** | **Everest** |
| 1 | 0 | 35 |
| 2 | 0 | 27 |
| 3 | 0 | 33 |

## Metric Table

(Larger Version of Table is available in Appendix)

# conclusions and future work

Over all Archetype behaved in a unique manner, often times exhibiting behavior that was not programmatic. Despite the lack of programmed behavior, Archetype was still able to function on the field and had only two major downfalls. One being that it was not programmed how to handle certain rules of the game such as off-sides kicks and penalty kicks, and second the tweaking of the kicking and looking parameters, where sometimes Archetype would try to pass but not kick the ball hard enough or in the correct direction and other times not looking around to see if the opposing team was close by. For the actual little amount of direct behavior coding in Archetype, especially compared to Simple, Archetype I consider to be a definite success and I the future I plan to implement the prior mentioned features that had caused faults and disadvantages in game play, especially the disadvantages that were apparent in playing professional teams such as Everest. Archetype does have the advantage, I believe, in the fact that its players can and will alter their own strategies. The constant change of strategy should serve to give Archetype an advantage, hypothetically.

I am certain if Archetypes game play issues were corrected, Archetypes ability to play would improve, if not exceed some professional teams like Everest in the 2D Simulation League.

Full match videos are available via the following link on youtube at: <http://www.youtube.com/user/wgres101>

# References

“Association Football”, Wikipedia

“Feed Forward Neural Networks”, Wikipedia

RoboCup Soccer Server Manual

RoboCup Soccer Client Manual

Prescott, Tony et. al. “Layered Control Architecture in Robots and Vertebrates” *Adaptive Behavior*, 1999

Jung, C.G. , Hull, R.F.C. “The Archetypes and the Collective Unconscious” Aug. 1, 1981

# appendiX

## Freud’s Iceberg Model (Figure 1)

Super-ego

Id: Primal/Reflexes

Repression

Resurfacing

## Brook’s Subsumption Architecture (Figure 2)

Simple, reactive behaviors (reflexes)

…..

.

Mapping

Wandering

## Archetype Stack (Figure 3)

Increase in Priority Queue

Decrease in Priority in Queue