**Термины**

Термины пишутся с заглавной буквы.

**Стимуло**м будем называть активацию дерева автоматизма по действиям с Пульта оператора или по изменению гомеостатического состояния. Стимул – это активный образ действий (если оператора – то ActionsImage, а если по изменению состояния, то - образ текущего состояния BaseStateImage)

**Ответ**ом будем называть сформированные действия Beast на данный Стимул.

Ответ всегда бывает только в виде образа действий ActionsImage.

**Эффект**ом будем называть результат Ответа: -1 негативный (неудачный), без изменения – 0 и 1 - позитивный.

При общении на какую-то тему возникают тройки:

Стимул – Ответ – Эффект, которые фиксируются в качестве Правил.

**Правило**м будем называть зафиксированные в TriggerAndAction значения тройки Стимул – Ответ – Эффект.

**Общая картина начинается с Дерева особенностей ситуации (дерево понимания)**

Это дерево (understanding\_tree.go) имеет относительно небольшое и ограниченное число веток, а дерево автоматизмов (automatism\_tree.go) может иметь неограниченное число веток из-за новых фраз.

Сначала активируется дерево автоматизмов и сразу вслед – дерево понимания. Так что на каждую активную ветку первого может прийтись одна из веток второго.

А по активности дерева понимания запускается функция consciousness (understanding.go) – в режиме объективной активации (activationType=1), т.е. по активности объективного Стимула.

На 4-й стадии развития происходит более разумное формирование Ответа на Стимул с помощью функции осмысления consciousness, в которой есть четыре уровня:

1. *есть ли штатный мот.автоматизм и нужно ли его менять*

2. *попытка использования примитивных Правил*

3. *попытка найти решение, используя всю текущую инфрмацию с учетом срочности*

4. *доминанта нерешенной проблемы - только если нет срочности (Творчество)*

Consciousness вызывается как объективным Стимулом, так и субъективным, т.е. по инициативе в самой Consciousness (при срабатывании ментальных автоматизмов). В случае субъективного вызова (только после объективного) возникает цикл рекурсии, по результату которого возвращается bool функции. Результат рекурсии – всегда запуск моторного автоматизма (при запуске ментального с таким действием) или переактивации системы.

Если не сработал первый уровень рекурсии, делается попытка найти последний фрагмент последовательности Правил в Эпизодической памяти, в которой подряд пишутся все активации consciousness в виде Правил.

Функция consciousness названа осмыслением потому, что в ней реально с каждым вызовом происходит обновление информационной картины текущего состояния (о котором говорил Дж.Тонони), что является контекстом осмысления. Вызов объективный происходит по внешнему ориентировочному рефлексу, а вызов ментальный можно зазвать внутренним ориентировочным рефлексом (так же пишется эпиз.память и еще Кратковременная память shortTermMemory).

*После объективной активации (activationType==1) начинается рекурсивный субъективный вызов (activationType==2)  
- цикл обдумывания (субъективный ориентировочный рефлекс), каждой шаг которого основывается на информации, даваемой предыдущим шагом с целью найти подходящие действия для данной ситуации, что дает возможность снова сориентироваться.*

При каждом осмыслении становится доступной новая информация, получаемая с помощью ментальных автоматизмов, но сам процесс ее получения оказывается субъективно недоступен (нет записи промежуточных данных).

При срабатывании ментальных автоматизмов перезапускается Consciousness для того, чтобы организовать цикл обдумывания с помощью ментальных автоматизмов, который можно прервать следующей объективной активацией Consciousness.

Т.е. Consciousness может активироваться субъективно множество раз между двумя объективными активациями. Т.о. осмысление происходит очень быстро, но если прервалось, то состояние на момент прерывания остается доступным для следующего осмысления.

При объективной активации сначала нужно *посмотреть, есть ли прерванные цепочки осмысления и, если есть, выбрать, с какой продолжить осмысление.*

В основе организации осмысления лежат цепочки последовательных вызовов ментальных автоматизмов, которые всегда начинаются с одного из узлов активной ветки дерева понимания, что фиксируется в массиве goNextFromUnderstandingNodeIDArr (поиск по узлу дерева понимания).

От одного узла дерева понимания может начать расти множество цепочек в зависимости от активного узла дерева автоматизмов. Такие цепочки НЕ могут ветвиться на каждом из звеньев т.к. всегда продолжаются для текущей активности дерева автоматизмов: если сменится активная ветка, то станет активной и другая цепочка.

Но все оказывается связанным (обобщенным) через Кратковременную память последовательности запусков звеньев всех цепочек.

Цепочки огранизуются с помощью стркутры ветвителя goNext:

type goNext struct {  
 ID int  
 MotorBranchID int *// узел AutomatizmNode* FromID int *// ID родительского goNext* NextID int *// ID дочернего goNext* AutomatizmID int *// id ментального автоматизма*}  
*// по ID goNext -> \*goNext*var goNextFromIDArr=make(map[int]\*goNext)

*/\* по узлу UnderstandingNode.ID -> goNext  
Может быть множество goNext, привязанных к узлу UnderstandingNode.ID  
Карта сохраняет первые вхождения всех цепочек от самого их начала и создается  
при формировании первого автоматизма от узла дерева понимания.  
ПРИМЕР uID - это ID узла дерева понимания:  
uID=15 -> goNextFromIDArr[goNext.ID=1] -> goNextFromIDArr[goNext.ID=2] - цепочка для MotorBranchID==1  
 -> goNextFromIDArr[goNext.ID=10] -> -> goNextFromIDArr[goNext.ID=11] - цепочка для MotorBranchID==2  
Для каждого MotorBranchID создается своя линейная цепочка!*

*К узлу uID дерева понимания пристегнуты 2 goNext с разными MotorBranchID,  
а к тем пристегнуты по 2 других goNext с разными MotorBranchID  
goNextFromUnderstandingNodeIDArr[1]=goNext.ID=1  
goNextFromUnderstandingNodeIDArr[2]=goNext.ID=10  
  
goNextFromUnderstandingNodeIDArr представляет собой память о том, как шли размышения до конечного звена: создания успешного моторного автомтаизма.  
При этом ранее созданные успешные могут стать неуспешными к каких-то условиях и построение цепи продолжится. Но последовательности Правил в эпиз.памяти хранять случаи успешного действия и они могту использоваться. Так что необходим отслеживающий режим, который выявляет, какое именно Правило нужно применить в данном случае.  
\*/*

Каждое звено типа goNext позволяет направлять поиск к следующему или предыдущему звену, обеспечивая целостность и направление развития цепочки. Фактически так определяются последовательности ментальных автоматизмов цепочки размышления.

**На втором** уровне Consciousness происходит примитивное использование Правил.

*Функция* getRulesFromEpisodicsSlice п*озволяет найти последнее известное Правило по цепочке последних limit кадров эпиз.памяти (шаблон решений)  
с учетом шаблона уже реализованных правил.  
  
Последовательность шаблона использует прежний опыт цепочек Стимул-Ответ-Эффект  
с ожиданием очередного Стимула для последнего нахождения подходящего Ответа.  
  
Каждый раз, находя последнее правило в данной ситуации,  
оно может использоваться для того, чтобы перейти к следующему известному правилу  
или, если такого нет, начать формировать новое решение  
ментальным автоматизмом получения инфы с запуском ментального осмысления.  
  
Получается ветвление дерева решений (основанного на эпиз.памяти) по каждому Стимулу  
  
Такой поиск одинаково работает как для кадров объективной эпизод.памяти,  
давая решение для запуска моторных автоматизмов,  
так и для кадров субъективных - давая решения запуска ментальным автоматизмам.  
  
Чем больше limit тем маловероятнее найти совпадения,  
так что можно вызывать getRulesFromEpisodicsSlice постепенно уменьшая limit*

**На третьем** уровне Consciousness происходит сложное использование Правил с учетом всей текущей информации окружения. Причем здесь используются не только Правила, полученные объективно, но полученные субъективно, что оптимизирует процесс мышления.

При этом каждый ментальный автоматизм определяется действием в виде

type MentalActionsImages struct {  
 ID int *// идентификатор данного сочетания пусковых стимулов* activateBaseID int *// активация настроения* activateEmotion int *// активация эмоции  
 /\* Разнообразие заготовленных инфо-функций дает больший потенциал  
 разных ментальных действий, поначалу случайных, но оптимизирующихся по эффекту Правила.  
 \*/* activateInfoFunc int *// вызов инфо функции* activateMotorID int *// запуск моторного автоматизма по результатам инфо-функции создания автоматизма*}

Так что в ментальных Правилах в качестве образа действия используется MentalActionsImages.

Осмысление на третьем уровне вызова Consciousness происходит с помощью цепочек ветвления goNextFromIDArr, начинающихся от узла дерева понимания: goNextFromUnderstandingNodeIDArr. Осмысление – активация последнего звена цепочки и выбор: если там есть уже хороший моторный автоматизм, то просто запустить его (хотя испытанный хороший автоматизм и сам вызовется т.к. запишется штатным для данного узла дерева автоматизмов в Периоде ожидания), если нет, то начинать формировать следующий ментальный автоматизм и помещать его в звено продолжения цепочки.

Как уже говорилось, запуск ментального автоматизма сопровождается обязательным перезапуском consciousness().

Целью процесса на третьем уровне является нахождение наиболее подходящего к текущим условиям обсуждения действия моторного автоматизма - Ответа.

Тогда он запускается с периодом ожидания.

По результату Периода ожидания afterWaitingPeriod() записывается ментальное Правило из всего цикла осмысления (var saveFromNextIDcurretCicle []int). Если хорошо, то данная цепочка так и заканчивается запуском моторного автоматизма (MentalActionsImages.activateMotorID), если плохо – формируется следующая цепочка (.NextID) с выбором другого MentalActionsImages и т.д. Т.е. формирование следующего звена цепочки идет С УЧЕТОМ ОПЫТА (MentalActionsImages) ПРЕДЫДУЩИХ.

Из-за того, что у мент.автоматизмов нет brangeID и NtxtID, они составляют общий набор решений и могут участвовать в разных звеньях цепочек.

Поиск MentalActionsImages для следующего .NextID идет по ментальным Правилам или по функции случайного или прописанного в редакторе подбора с учетом опыта предыдущих (Переактивации Базового состояния или эмоций выбираются на 5-й ступени развития).

Структура мент Правила MentalTriggerAndAction начинается или с мент.действия MentalActionsImage или с моторного ActionsImage, потом - Ответ MentalActionsImage и обычный Эффект.

После срабатывания инфо-функции (.activateInfoFunc) информация добавляется к текущему информационному окружения в виде общей глобальной структуры mentalInfoStruct (для каждой инфо-функции в mental\_automatizm\_INFO\_structs.go создается общая переменная currentInfoStructId == ID инфо-функции.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Постановка проблемы**

И-за наследственного образования Инфо-функции имеют конечное число реализаций. Не может быть процесса создания новой инфо-функции в ходе осмысления. Можно использовать только существующие. Поэтому важно продумать, какие из них обеспечат главное: возможность находить решения в виде моторных автоматизмов для проверки при запуске.

Цикл осмысления – рекурсивный вызов consciousness для вызова новых ментальных автоматизмов с разными информационными функциями или пере активацией системы до того, пока не будет найден перспективное действие для моторного автоматизма.

Важно понимать, что практически все шаблоны решений заимствуются (отзеркаливаются) у предшественников, а собственные решения – очень редкий случай. Как правило, собственные решения возникают случайно по ассоциации с чем-то далеким от цели Доминанты.

Все попытки создать алгоритм использования универсальных правил для нахождение нового решения оказались безрезультатными (например, ТРИЗ), в лучшем случае удавалось применить схожее в чем-то, но уже имеющееся решение.

Итак, задача – в создании Инфо-функций, в том числе – для создания новых ментальных автоматизмов продолжения нахождения решения. Решение – найденная последовательность Правил и затем – организация цикла провокационных для оператора действий для того, чтобы спровоцировать его на очередной Стимул в данной цепочке Правил. Именно на это и нужно затачивать Инфо-функции.

Кроме этого, нужен некий набор Инфо-функций для генерирования совершенно новых действий по какой-то аналогии, в надежде, что это сработает. Но это – для стадии Творчества – для решения Доминанты (в которой определена Цель), не раньше.