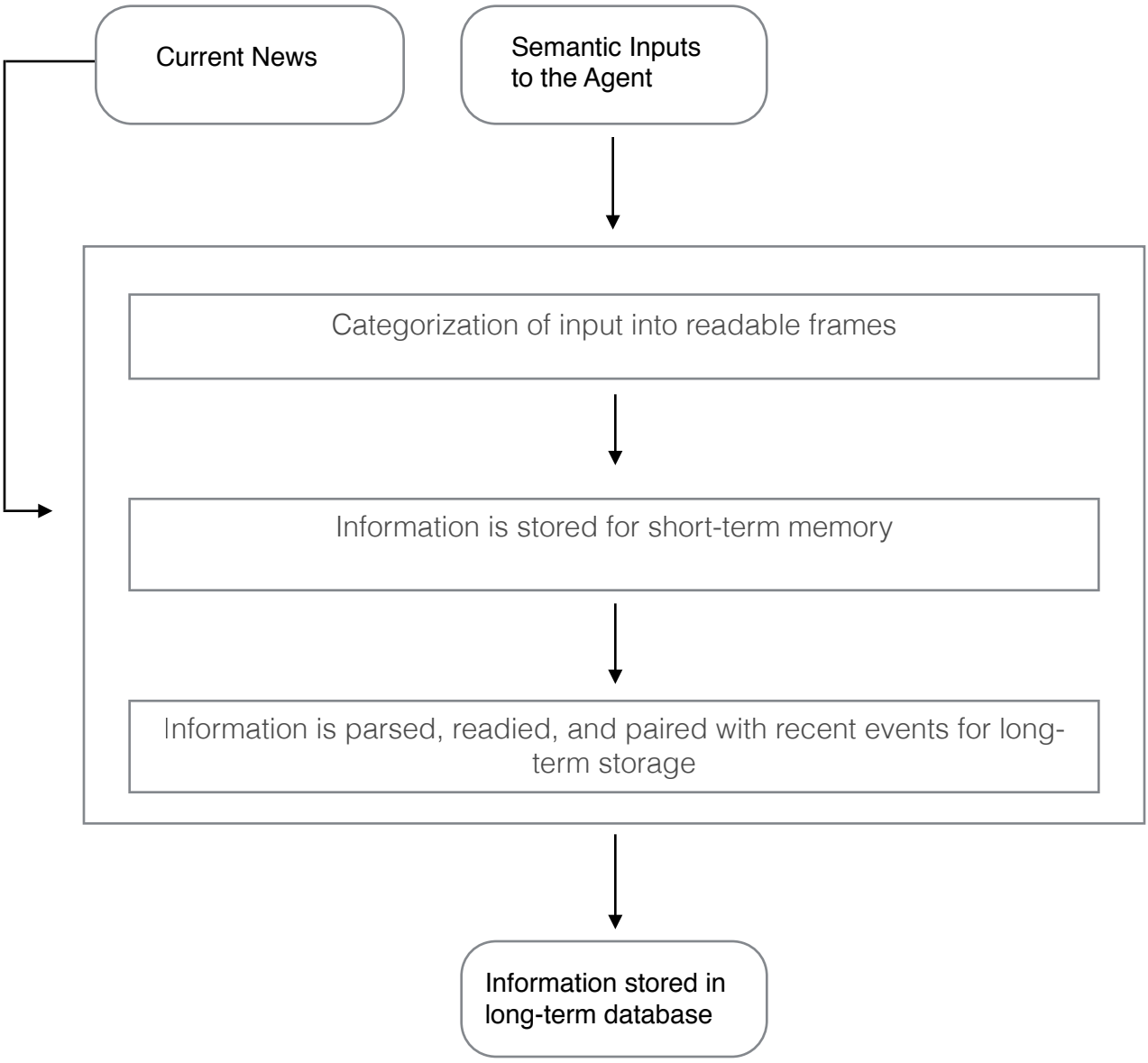
Brad Ware
KBAI Final Exam

Question 1

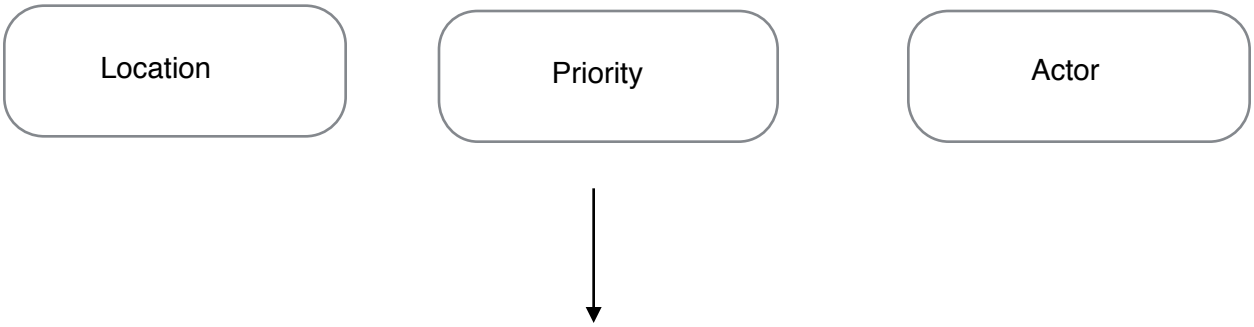
I will first start off showing a very high level view of the computational architecture of the intelligent assistant.



In this computational architecture, the first critical feature is being able to interpret the information by language processing. We are assuming our agent can interpret information and process it into the frame orientation in the example. From here, my agent will then begin to rank information in short term memory off of a priority system. For example, loads of information will be stored, such as everyday actions like me buying a coffee. However, any frame that triggers an action such as bought, killed, stole, broke, wrecked, etc will be flagged in the system for further analysis. Then, after a frame has been identified as an important action, it will be evaluated further to detail what other contributors make it actually a priority, or an even higher priority. This includes looking at the slots for object and instrument, as both of these are crucial and directly related to the action. Finally, if indeed the frame is labeled as a high priority to keep a watch on, the agent will group it with other frames that are first and foremost similar in location and time. This is to keep events current based on when they happened and where. After that, further grouping will occur based on the event and the actor. Also, if the agent at any time wants to see the history of an actor, it can quickly index by name and see a complete set of actions from the short term or long term. But let's say for instance that the event is not a high priority, that a small boy stole a piece of gum from a 'Toys R Us'. Our agent will still document the information in the short term memory by location, actor, etc. This will allow seemingly harmless actions to still be documented in case further leads are calculated at a different time. Here is a pseudo algorithm that takes given input and stores it correctly in the short term memory:

```
algorithm shortTermStorage(Frame input) {

        if(input.event is HIGH_PRIORITY) {
                int count = 0
                for each (slot, filler) in input:
                        if (slot, filler) is HIGH_PRIORITY:
                                count += 1

                if count > (constant_num):
                        storeAsHighPriority(input, count)

                else:
                        flagFrame(input)
                        storeAsLowPriority(input, count)

        }

        else:
                storeAsLowPriority(input, 0)

}
```

Because high priority frames are much more important and potentially dangerous, they will be indexed and stored together at a greater complexly. The algorithm will not just look at the location and actor, but the object and instrument as well. The 'rating system' will also help in the storage. Below is an example storage of short term memory and its transition to the database.



5	{ (Killed (TidMarsh, SecurityGuard, Atlanta, Wednesday, gun)), (Fought (TidMarsh, SecurityGuard, Atlanta, Wednesday, stabWound)), … }
4.5	{ (Bought (TidMarsh, gun, Atlanta, Monday, credit-card)), (Stole (TidMarsh, car, Atlanta, Friday, brokeWindow)), … }
4	…
3.5	…
…	…

In this example, a hash table is used to store each frame based on the priority of each event. This is just one way the agent will store information, as it will also cross reference with an Actor's name, location, and even object and instrument. This complex indexing with O(1) actions for look up will allow my agent to quickly search for given information.

A semantic network could even be built as well based on the action for each event. For example, stolen articles could all be linked, where it branches off from physical objects to virtual items, such as SSN and credit card information. If a giant semantic network was now located as well in each cell of the hash table, my agent could slowly build up information over time about different events occurring. For example, with social media information is spread quickly through the internet all over the country. My agent could track the similarities of crimes or actions throughout the US by semantically building networks in each location and cross referencing their similarity. Another way a semantic network would be extremely helpful is tracking the actions of an individual over time. This would link specific actions they repeatedly did or new one's to track their movements over time and detect if anything seemed out of the ordinary. In the example above this would help identify a suspicious activity if TidMarsh just randomly purchased a gun without ever having any activity linked to it before such as hunting. The semantic network would recognize that he visited a new bank recently and just purchased a

new gun, showing no interest previously about weapons or hunting. Since they were registered near each other in terms of time, there could be an alert for him at any public place.

John Doe	Semantic Network of his actions
Brad Ware	Semantic Network of his actions
…	…
…	…
…	…

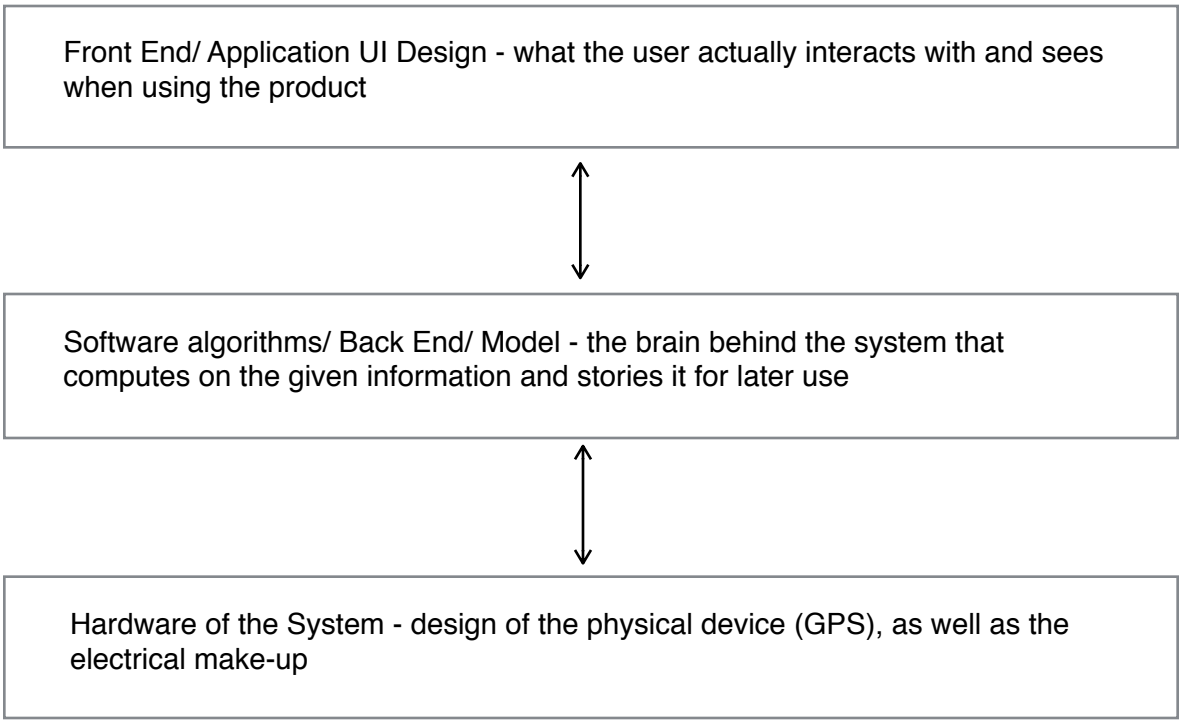
Even though neither of those actions are illegal separately, together they do show a plan to possibly rob a bank. There are so many ways to store the information about each individual event but a semantic network definitely makes it the easiest to transform and identify actions as they build on each other.

Finally, after information has passed after a certain amount of time, it will eventually transfer to long term memory storage. This will mimic the same style as the short term storage with indexes on location, actor, and event, but will not be as efficient. This is due to more information being stored (permanently) and the urgency of the actions is not as high. Long term storage will mostly be used to reference an actor. If they have repeatedly show interest in something (such as purchasing a gun) they will also be stored in the semantic network style, linking together events and objects to each actor. The key difference for long term storage is that it will also have the current events that actually accumulated (without speculation) to match the actions our agent perceived. This will help build the semantic network more accurately by allowing our agent to know what actually is a high priority, and how closely it should be watched. By having the actions that occurred stored in memory with each frame, this allows my agent to learn over time what is important to keep a watch over, and how to ‘score’ each event. Learning is a critical process for a situation like this as the world is ever evolving so too can my agent keep up with the change of time.

Through the storage of long and short term memory my agent will be able to build semantic networks of each actor, region, and object to identify and predict criminal acts. Long term memory will allow my agent to learn as the short term memory will be used to stop potential dangerous events from occurring, such as TidMarsh robbing a bank.

Question 2

First, I will define some of the higher level components of the architecture of this system, then I will dive in with a few examples to show how my system will work.



We will focus on the software/ Back End layer of this diagram as that pertains the most to the question and the concepts from this class. On assuming that our application has access to a mapping API such as MapQuest or Google Maps, we can leverage the shortest-path algorithm, which factors in traffic due to weather, car wrecks, and construction. Thus, to make a “smarter” and more personalized mapping application that does more than what the market offers, we have to have more information about the user. And thus I will begin at the Front End layer of the application.

On the Front End, we not only need a sleek design that displays in a map format how to get to the destination, but there also needs to be a profile format where the application understands the user on a more personal level. This can be filled out once after the user initially visits the site. Some categories may include:

Name: Brad Ware
Home Address: 770 Techwood Drive NW Atlanta, GA 30313
Work Address: 950 East Paces Ferry Rd NE, Atlanta, GA 30326
Type of Car: Silverado Truck, 2004
MPG: 18
Preferred Travel Highway/Backroads: Highway

This can be stored with the user's initial profile. Then, each time the user wants to map to a destination, more information can be gained to help the processing algorithms for mapping:

My scenario on the way to work at 7:56 am

Destination Address: Work Address
Method of Travel: Car
Stoping to Eat Breakfast: Yes
Specific Restaurant: No
Stopping by Cleaners: No
***Preferred Route: Fastest route*

**Other options would be avoiding traffic, stress-free, near many restaurants, etc.

After our application has initial information about your profile, as well as details about the trip, it can then take that data and use it for the software algorithms of mapping a trip.
So now that we've seen some details on the Front End, lets talk more about the actual brain of the application. For Case-Based reasoning logic, I think it would be best to represent each scenario in frames. Every time one tries to map to a destination a frame is produced to represent that request. A frame that is passed to the Back End might look something like:

Request

{
Profile: Brad Ware ———> references my profile
Destination Address: Work Address
*Method of Travel: His Car ——-> references frame of my car type, make, and MPG
*Stoping to Eat Breakfast: Yes
*Specific Restaurant: No
*Stopping by Cleaners: No
*Preferred Route: Fastest route
*Traffic: references Maps API near my Work Address

}

*All optional fields

with each slot referencing other frames for the fillers if needed. At the end of each trip, the application will prompt the user for feedback about the route, and save that with the associated frame. For the actual processing, our application needs to incorporate using both the original Maps API as well as Case-Based reasoning to use appropriate path recommendations. There is a generic algorithm on the next page for a mapping request from a typical user of the application. Notice that it takes in a data request as input, and leverages the details of each frame to properly route the user to the most preferred path. It also uses a database of similar Frames to try and apply Case-Based reasoning to analogically map the situations that had a positive conclusion to the current user's request.

```
algorithm mapRequestToRoute(Frame request) {

        location = request.getLocation
        destination = request.getDestination

        if(request.fields > 4) {
                similiarFrames = [ ]
                similiarFrames = searchForSimiliarCases(request, Database)
                similiarFrames.sort( )      //sort based on customer feedback of the route
                path = similiarFrames[0].route
        } else {
                path = MapQuest.route(location, destination)
        }

        return path
}
```

The most important piece of code is the searchForSimiliarCases function, which queries the database for mapping a similar situation to the current one. It's trying to use past cases that can be applied here to gain information on the best route for the user. The algorithm for that method might look something like this:

```
algorithm findSimiliarRoutes(Frame request) {

        frameList = [ ]
        if(request.user has previous frames):
                for each Frame in request.user past frames:
                        if(Frame.destination is similar to request.destination)
                                frameList.add(Frame)


        for each Frame in DB:
                if(Frame.destination is similar to request.destination)
                        if(Frame.userPreferences is similar to request.userPreferences)
                                frameList.add(Frame)

        return frameList
}
```

After you have a list of similar requests to the mapping that have feedback based on social media and from the application, you can then justifiably map it to the best route for the user. If the user did not wish to fill out any preferences and simply wants the shortest path to the destination, then my application will route using traditional Map APIs. The main aspect of this algorithm is that it combines using the shortest-path algorithms with Case-Based reasoning based off of social media feedback and user preferences to find the optimal route catered towards the user.

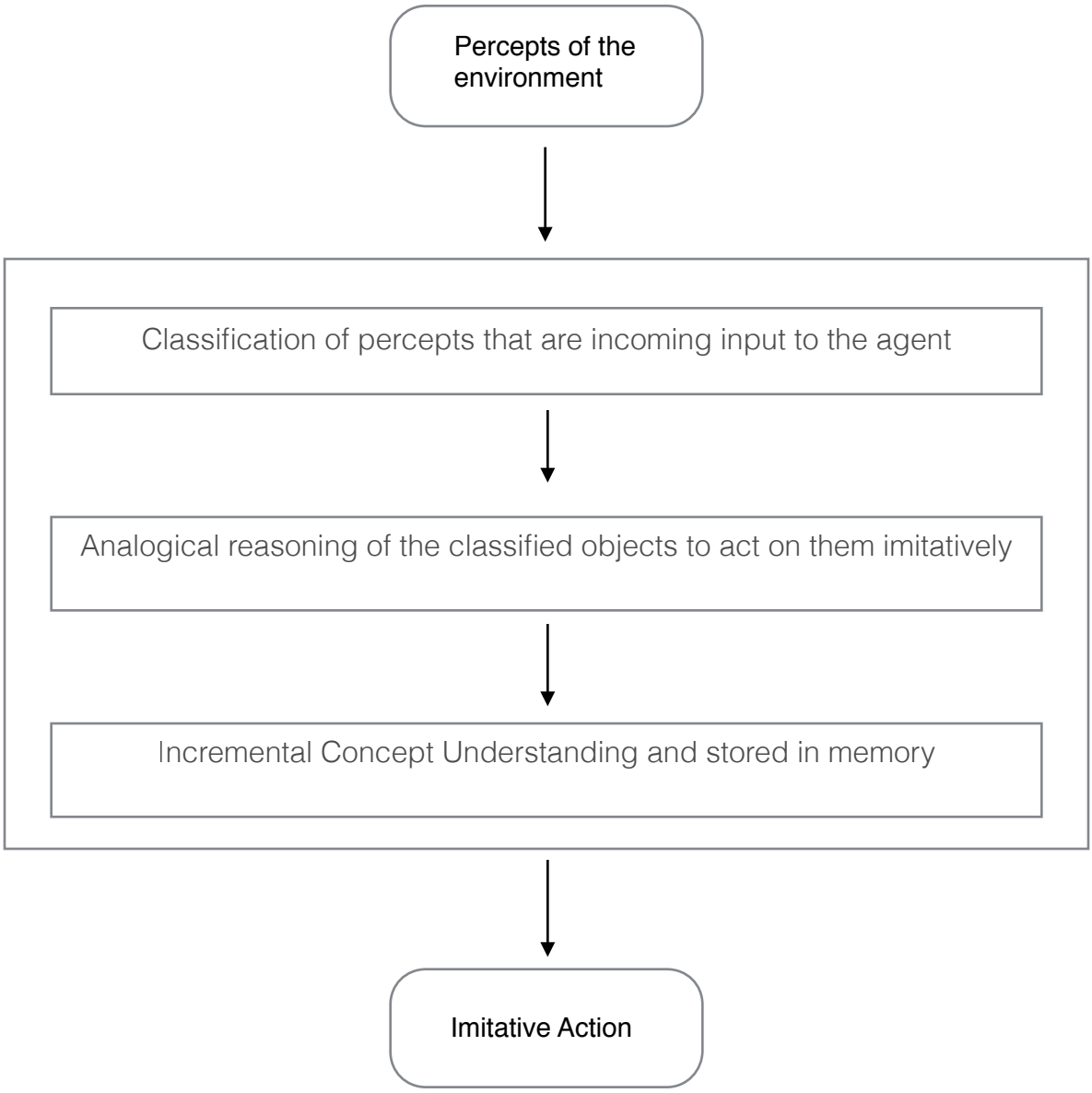
Let's go through an example to see how this might work. Let's use the input from above for the profile of Brad Ware, where he is trying to map to work from his home in the morning. He has put that he is driving his car, a Chevrolet Silverado, but also wants to stop on the way and eat breakfast. Therefore, my algorithm will first find all of the similar cases that the user has already mapped near the destination. This not only will find the shortest path but also will allow the application to learn from the user. It can then recommend paths that the user liked, or avoid the one's it didn't. However, it will then look through the database of frames that are near the destination and had preferences similar to my request. It will also find paths where the user stopped to eat to make sure it routes my request near plenty of breakfast restaurants. Ultimately for my request it will find the shortest route to work that avoids morning traffic but also passes a variety of breakfast restaurants where I can grab something before arriving to the office.

Another way to look at this situation would be me mapping to my friend's house in South Georgia in a very small town. In this town, there are no restaurants nearby my friend's house, and not many on the way. In my request I asked for a path from my residence to my friend's house, but still needed to stop and eat on the way. Therefore, my algorithms would have to process the closest restaurant near the shortest-path taken to my friend's house, calculating them both separately and combing the results. That way, I can navigate the shortest route to my friend's house using traditional API's, but also use Case-Based reasoning for restaurants near that route so that I can stop and eat.

There are definitely negatives about my algorithm. For one it depends on user-feedback a lot so that it can learn more about making the best path. But, if the user never answers any of the more detailed questions when requesting a route, my algorithm will just have to route a traditional Mapping API. It could leverage social media to gain information about cool restaurants in the area of the destination, or learn about updated traffic. This would then allow my application to index into previous cases that are similar to try and learn from experience on routes. There are infinitely many ways to solve this problem and ultimately I went the route of using feedback from others and the user's past routes to index recent cases. Ultimately, the goal of the application is to always provide the most optimal path depending on what the user values.

Question 3

For this problem, I will first describe the computational architecture that drives the overall process by learning through imitation.



In the architecture above, the agent takes as input percepts in its environment. These percepts most likely are the other actions being taken by the other agent that will eventually get imitated. These percepts are then filtered through a classification algorithm that is applied not only on the incoming objects, but also on the objects currently surrounding the agent. The agent needs to classify both sets of objects to properly understand what it currently has access too and also the agent it's trying to imitate has access too as well. After this the agent then needs to analogically map the actions of the agent it's imitating to the current objects it has. This means taking the

classified objects and performing the same actions on them as the target agent. Finally after all of these actions are taken, the agent can piece together what role each object plays and how actions performed on them can produce a desired result. The agent is basing this off of emotion, seeing the outcome from the imitated agent perform the action and then actually feeling the emotion after it performs the action itself.

After we have explained the overall architecture, let's dive a little farther into detail about the classification and analogical reasoning. In the classifyWorld pseudocode, the agent is looking through a list of percepts in the world around them and trying to categorize the objects. If they have already experienced the object then obviously they can just retrieve the name through case-based reasoning and a hash map type data structure. If they have not experienced the object then they can create a new category based on the visual and physical attributes of the new percept. After this, they then store the relation of the agent to the percept, gaining knowledge on the world as a whole.

```
algorithm classifyWorld(Percept [ ] percepts) {

        for each Percept p in percepts:

                category = searchKnowledgeBase
                if category is null:  //haven't seen the object yet
                        newCategory = p

                storeDistance(p, category, currentLocation)

}

algorithm mapObjects(Object [ ] objects) {

        agentObjects = [ ]
        otherObjects = [ ]
        for each Object obj in objects:

                if(obj.distance is not close):
                        otherObjects.add(obj)
                else:
                        agentObjects.add(obj)

        for each obj in agentObjects:
                for each obj2 in otherObjects:
                        if sameCategory(obj, obj2):
                                storeMapping(obj, obj2)



}
```

In the second algorithm, our agent is now storing objects that are similar to each other that are also far apart in distance to the agent. This is mapping objects together that are in different areas of the world. After the current environment is properly classified and mapped together, the agent then can start imitating. This imitation procedure will follow the other agent on the classified objects, using the similarity mapping of analogical reasoning that is based on appearance, size, functionality, and the actions the opposite agent takes toward the object.

Let's look at the example of the toddler, imitating her father to understand drinking from a cup. First, the toddler will analyze the room, taking in other objects such as her water cup, the coffee mug, the furniture in the living room, and even the clothes she is wearing. She also will most likely notice the other agent in the room, her father. After that, she will begin to classify the objects she is already familiar with - like her clothing and furniture she is sitting on, and other objects that she's not. This may include her water cup and most likely her father's mug. The most obvious knowledge representation for each object is a frame, storing the size, shape, and purpose of each object (see below).

Cup
{
size: medium
color: blue
hard/soft: hard
purpose: container
unique: no
relative: Mug, Glass, Bowl (reference to separate frames)

}

After classification she can then analogically map the action her father performs on each object that is similar to the one adjacent to her. This step can definitely be hard for a toddler to analyze the room and recognize duplicated functionality in similarity. But if the toddler already has experienced the object, she can just retrieve it from memory using Case-Based reasoning on her last experience with it. Finally, after she has mapped each object to it's counterpart, the toddler begins to learn by imitating the actions being performed by her father, and storing the effect of each. After each action is performed, she now knows the physical and emotional effect of what she just did, and can store that with each object frame. This is incremental learning, as she first sees that when she's thirsty she can tilt the cup towards her mouth. Then when the cup is empty, she realizes that the water within the cup is what she desires, not the cup. Therefore her knowledge will begin to expand on numerous ways to hydrate - a water bottle, fountain, sink, and even a camelback. As she puts more concepts together and performs actions the toddler will begin to understand the imitation at a much deeper level, allowing her to learn.

Soon, after the toddler matures and performs the new actions on a daily basis, she will teach others around her about drinking from a cup. This cycle continues not just with simple actions, but also complex material such as AI concepts we have learned in this course. Learning by imitation is a crucial ability humans use to develop themselves and their potential.