Jefferson TDR : Multi-object interactions

Jamie Doornbos
10/21/97
Revision 1.

Introduction

This document describes a number of new behavior language features and object properties that will enable doing two things at once, such as sitting and eating. New concepts include standard idle states, vertical routing preferences, proximity routing preferences, standard reaching animations, and internally issued sitting and standing.

No section numbering yet. Sorry.

Limitations

- Objects in slots (of table, furniture, etc.) may not be interacted with. They must be grabbed first and then interacted with. Example: food on a table must be picked up before it is eaten.
- Objects that cannot conform to this will not lie in slots. Example: stereo will sit on its own shelf when purchased.

Standard idle states

6 possible idle states. Standardized so that the vertical position of the person is fixed. The idle state is set by interaction trees that change the idle state and by the goto primitive described below. The idle state determines what suite of animations are used on the person, including resting animation and motive animations. These different suites are embedded in the animation table so that animation clients just provide an id and the appropriate animation is determined by the idle state.

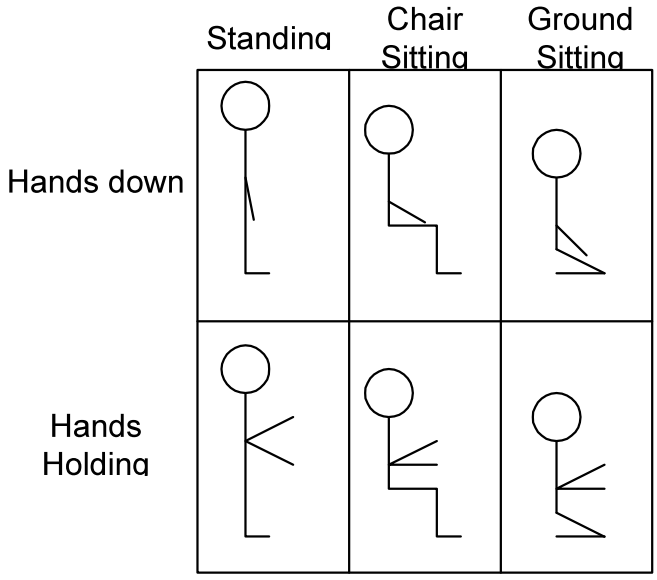

Figure: demonstration with stick figures of the idle states.

These positions may be modified to include more vertical positions, such as stool sitting or chair standing. The holding states are implied if the person is holding something.

TBD: can the person hold things specifically by right, left, and center, or is it always center? Perhaps holding in the left and right hands is allowed, but it is an error for the person to be idle while holding something there.

TBD: How does the person's rotation affect the rotation of the object being carried?

Chairs are special

Objects will have a sit down tree id field in the object definition, much like the portal tree id for doors. Objects with a sit down tree are considered members of a special category of objects: chairs. Such objects must also have a stand up tree id. Another definition field will tell if the chair is a standard chair, that is, if a person can be in the chair sitting idle state when sitting on it.

TBD: can a person be in their main loop in a non-standard chair? If so, how is the background animation determined for a person in a non-standard chair.

TBD: A non-standard chair cannot be used to watch TV since the sit tree is not allowed to complete.

Standard reach animations

For each vertical position, (standing, chair sitting, and ground sitting), a number of reach animations will be provided. The beginning idle state is one of the "hands down" states, and the ending idle state is one of the "hands holding" states. A reach animation will be provided for several destination rectangular prisms. The animation for a prism will begin in a hands down idle state, achieve a position where the hands are at the center of the prism, then retract to a hands holding state. When the hands are at the center of their prism, some kind of animation event will be issued, which will be used by the reach primitive (see below). The number of prisms will start out rough, with 4 different heights (see first figure), but can be easily increased if needed. For example, adding a left to right axis and a back to front axis would increase the number of prisms to 64, as in the second figure.

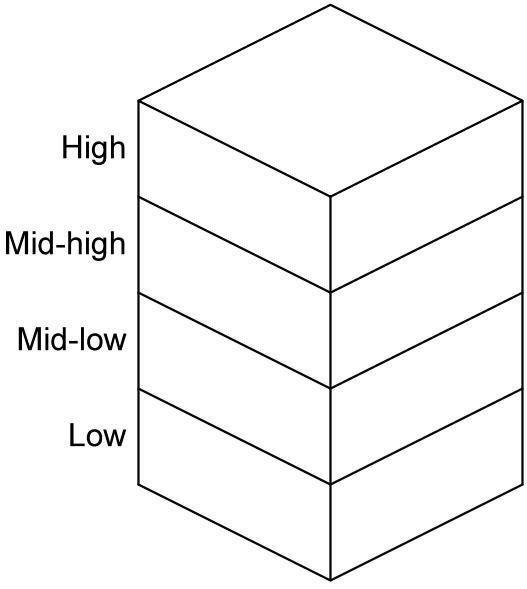

Figure: rough reaching regions.

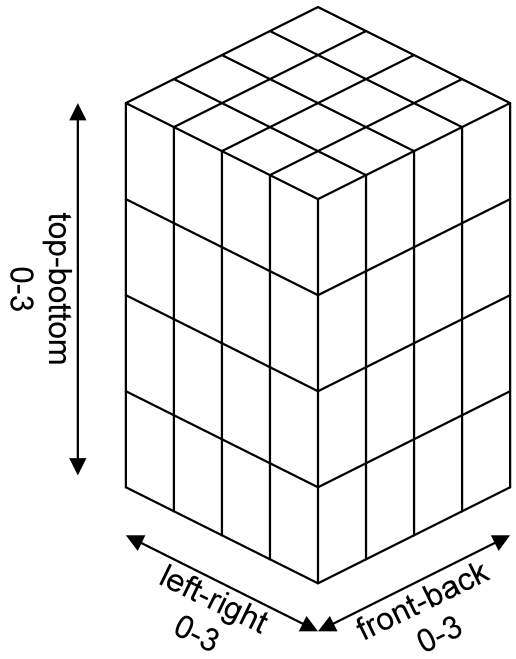

Figure: finer reaching regions.

The total number of animations necessary for reaching is computed as:
(# prisms) * (# vertical idle state positions) * (# skeletons)

New primitive: "reach"

Three parameters:

- Container object: object whose slot is being reached for. (e.g. table). Zero specifies that the ground is the destination.
- Slot number: slot number to reach for.
- Reach object id: object being reached for, or dropped.

Returns true if the process finished to completion. I.e. the reach object has been successfully transferred.

The reach primitive will use the idle state of the person and the coordinates of the destination to run the appropriate reach animation in the right scale and pick up or drop the object. Several conditions control the behavior of the reach primitive. If any of these conditions are true, the primitive executes as specified and returns true. Otherwise it returns false.

- **The person is not holding anything and the reach object lies in the specified slot of the container object.** The slot coordinates are used to find the reach animation. The animation is run forwards, and the object is picked up on the event.
- **The person is not holding anything, zero is specified as the container object, and the reach object lies on the ground.** The object's coordinates are used, the animation run forwards, and the object is picked up on the event.
- **The person is holding the reach object, and a valid, empty slot is specified.** The slot's coordinates are used, the animation is run backwards, and the object is dropped on the event.
- **The person is holding the reach object and zero is specified as the container object and the tile in front of the person is available.** The ground coordinate of 1 tile away is used, the animation is run backwards, and the object is dropped on the event.

Special reaching.

Some objects require state changes or special animations to reach a slot. One example is door-covered slots: the door must be opened before and closed after the reach animation. Objects will have a reach tree id field in their definitions. If the container id specified to the reach primitive refers to an object with such a tree, the tree will be called with the reach primitive parameters on its stack. Then the object can perform special reach logic to determine the animation to run. If the special reach tree calls the reach primitive, special reach tree will not be called again. The reach tree may assume that the routing preferences of the slot have been satisfied (see cascaded parameter below). If the reach tree fails, the calling reach primitive also fails.

New primitive : “goto relative vertical”

Name not yet decided. This new primitive will specify orientation, proximity, and preferred vertical positions.

Parameters:

- Orientation: 8 bit integer flags for each of 8 directions which are acceptable.
- Proximity max.: 8 bit integer which is the maximum distance in tiles that the person should route to.
- Proximity min: 8 bit integer which is the minimum distance the person should route to.
- Proximity optimal: 8 bit integer which is the optimal distance the person should route to. This value determines the gradient for scoring.
- Vertical preferences: 16 bit integer specifying the order of preference of the idle states: standing, chair sitting, and ground sitting. Each idle state corresponds to an enumerated value. The vertical preferences are just the first, second, and third preferred idle states that the person should finally be in. If an idle state is not preferred at all, it is considered “blocked”, which means the primitive should fail if the person cannot achieve that state.
- Destination slot number: number of the slot on the stack object that is desired. -1 specifies that this is just standard routing. This is used if the person is going to an object in order to drop something onto it.

It is assumed that the desired facing is towards the object. TBD: is this a valid assumption?

TBD: Ideally, all of the preference information would be imbedded into a destination scoring system controlled by the route planner. Destination scoring is distinctly different from route scoring and may not be possible with the A-Star algorithm. It is likely that the implementation will be one of scanning the possible destinations in order of preference, checking for validity, and trying the route planner on each one.

Cascaded “goto relative vertical” parameters of slots:

In addition to the values specified by the primitive, each slot will also have the same set of routing parameters. If the destination object is in a slot, the parameters of the slot (and all slots below) and those of the primitive call will be cascaded to the most restrictive set of parameters. For every two sets:

- Orientation will be only those allowed in both sets.
- Proximity max. will be the min of the two.
- Proximity min will be the max. of the two.
- Proximity optimal will be averaged and pegged to the min/max.
- To cascade vertical preferences, each idle state will be scored on its ordinal, 10 for first, 5 for second, and 2 for third, with a bonus point for the first choice of the original object to break ties. The final score for an idle state is the sum of its scores for each parameter set. The final preference will be based on the order of the scores. Blocked states get blocked without question. TBD: maybe a minor point, but vertical preferences should probably be done all at once in the case of multiple containers, instead of two-by-two, to get good precision.

New primitive: “Find Slot”

A primitive to locate the nearest slot at a given height.

Parameters

- Min height: 8 bit integer specifying the minimum height slot to return. TBD: units?
- Max height: maximum height slot to return.
- Open or closed or don’t care: 8 bit integer 0-2. Whether the found slot must be open, i.e. surfaces, or closed, i.e. containers. TBD TODO: this requires a new slot field for whether it is an open or closed slot.

Returns:

- Found container id: the object that was found with that slot.
- Found slot number: the number of the slot that matches the height.

Example 1: Eating an apple on a table

The example begins with the person having decided, for whatever reason, to eat the apple, that is, run the “eat” interaction of the apple object. The eat interaction specifies a goto primitive that allows any orientation, must have a 1 tile proximity, and first prefers standing (since its a snack), then chair sitting, then ground sitting:

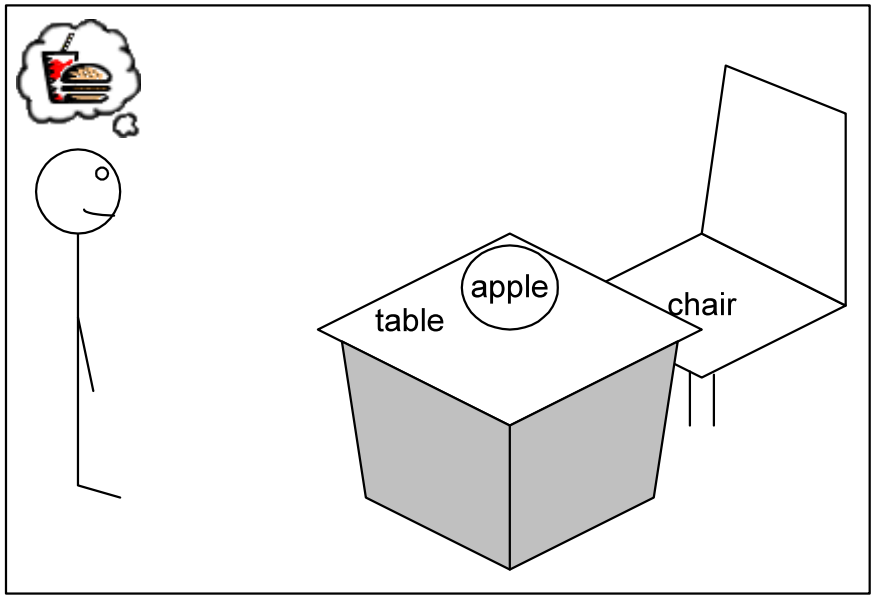

Figure: Person thinking of eating an apple on the table with a chair next to it.

The slot of the table allows any even orientation (no diagonals), requires a 1 tile proximity, and first prefers chair sitting, then standing. Recall that this blocks ground sitting. The parameters of both the primitive and the table’s slot are combined. The resulting proximity is 1 tile. The resulting orientation is any non-diagonal orientation. The preferred vertical positions are scored as follows:

- chair sitting: 5 from apple + 10 from table = 15
- ground sitting: blocked by table
- standing: 11 from apple + 5 from table = 16

Thus standing, then chair sitting are the preferences. These scores are assigned to adjacent tiles like this:

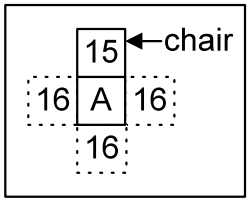

Figure: scoring of tiles around the apple.

The routing proceeds by choosing (TBD: at random?) one of the tiles with 16 as its score, and attempting a route. For this example, assume it is empty and the person routes there successfully, completing the goto primitive.

The interaction tree then calls the reach primitive in "get" mode with the table as the container object, 0 as the slot number (slot of the apple, available from apple's state vars.), and the apple as the reach object. Since the container object is specified, the reach primitive uses the slot coordinates and animates the person reaching to slot 0 of the table, where the base of the apple is. This is how things look in the middle of the reach primitive, just as the event happens:

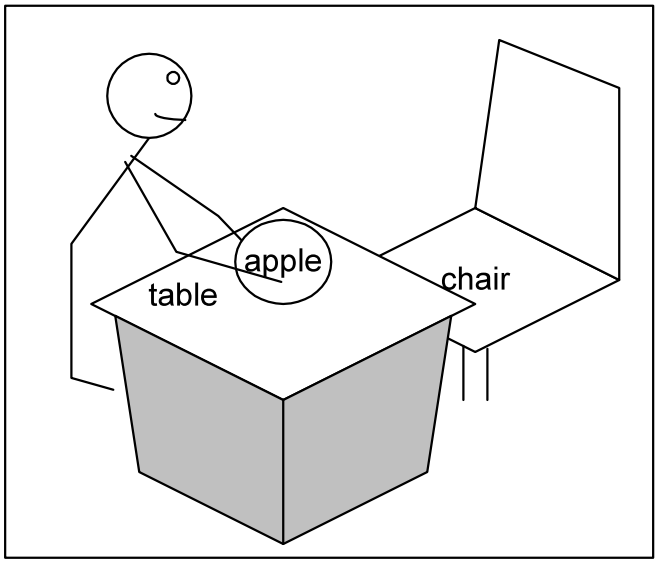

Figure: person grabbing the apple.

After the reach primitive is complete, the person is holding the apple and is in the "standing with hands holding" idle state. The interaction tree then runs animations like "bring hands to mouth" and plays sounds like "chomp" and "squirt" to show eating. Whether the apple's graphic is changed as it is eaten, and whether it is destroyed or if the person must throw it away is to be determined by the apple designer.

Example 2: Watching TV (from a standard chair).

The example begins with the person having decided, for whatever reason, to watch TV, i.e. to run the "watch" interaction of the TV object. Assume that the TV is already on. (Turning on the TV is a simpler, independent branch of the interaction.)

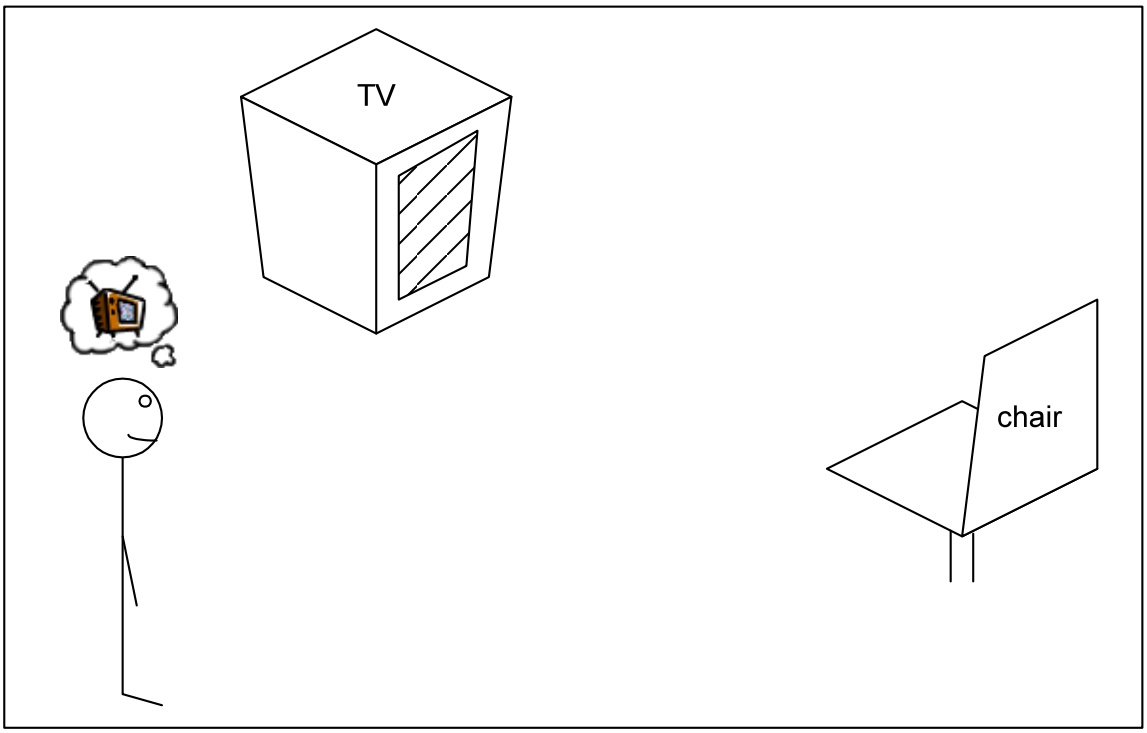

Figure: person thinking of watching TV.

The TV executes a goto primitive with a minimum proximity of 2 tiles, a maximum proximity of 8 tiles, or 24 feet, and an optimal proximity of 5 tiles. (These number are just for demonstration.) The valid orientations are 0, 1, and 7. These correspond to directly in front or along either of the diagonals just to the left and right of the front. For a proximity range, orientations are interpreted as a kind of rectangular cone area, shown below. The vertical preferences are first chair sitting, then ground sitting, then standing. The valid destination space is demonstrated as follows:

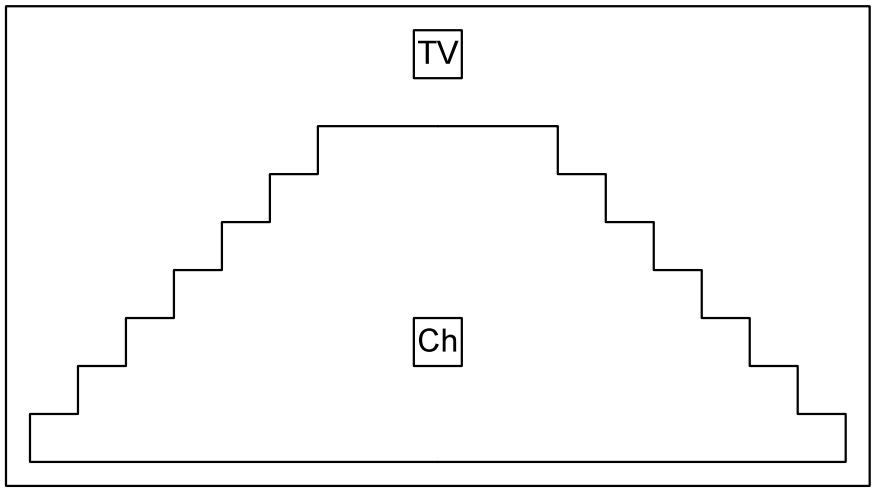

Figure: valid destination region for goto call.

The possible destinations are scored as follows:

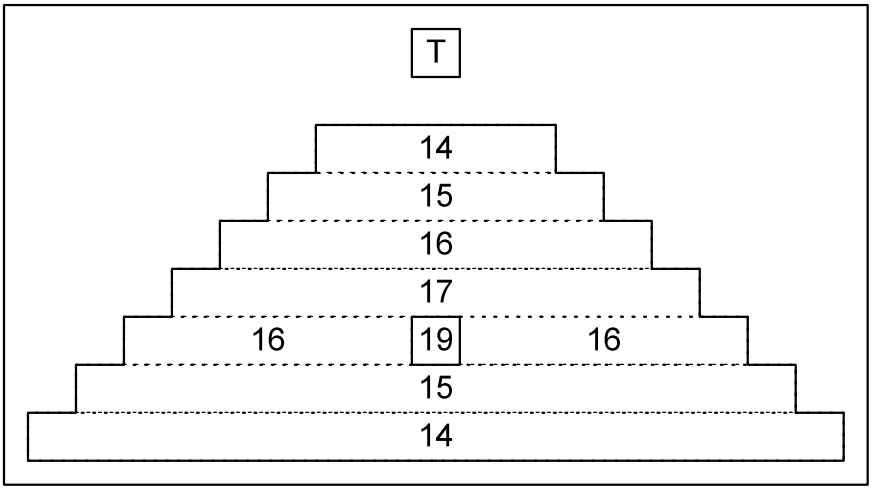

Figure: results of destination scoring algorithm.

The 5 tile line has a base score of 10 since it is at the optimal proximity. In addition, the tiles in that line are valid for ground sitting, or standing, which are preferred second and third, adding an additional 7 points to the score, for a total of 17. Empty tiles closer and farther away have scores that fall off at a rate of 1 until the min and max. tile lines are reached, which are not allowed. The chair in the middle gets an additional 10 since the first vertical preference is chair sitting. TBD: A chair on the left facing right or on the right facing left should still be allowed, but possibly scored lower. TBD: for these chairs, how can the person's head be made to turn towards the TV?

In order to proceed, each destination must be tried, starting with the highest-scored: the chair tile. Since it is not occupied, its sit down tree would be run by the goto primitive. This tree does a simple routing to the front of the chair, and runs a sit down animation. Then (possibly through another new primitive, but probably just a "set" of the state), the idle state of the person is set to chair sitting. This completes successfully the goto primitive, and control is back in the Watch TV interaction.

The interaction completes, and the person is left in the "chair sitting" idle state. The main tree of the TV should now broadcast entertainment to the person. The main tree of the person should keep them idling in the chair until some other interaction presents a better opportunity or the user clicks.

Example 3: Preparing Dinner

The example begins with the person wanting to interact with some food that is in the refrigerator, for whatever reason. The interaction that provides hunger satisfaction is "Prepare and eat". This interaction is broken up into several sub-tree parts. Any part can fail or succeed. If a part fails, the whole interaction fails.

Not finished {
The main interaction chains the parts together using a state variable attribute in the food. The state variable has values:

- 0: Brand new food.
- 1: On the counter. The food was abandoned on the counter. If this value is set upon entry into the interaction, the stove tree is run.

}

Get from fridge

This tree is essentially a goto and a reach. It assumes that the stack object is the food, and the person is holding nothing.

The first step is to get to the fridge. This is done using a goto primitive, specifies that the person should go to the food from any orientation at a distance of 1 tile. It also specifies the vertical preferences are standing, chair sitting, and ground sitting. Through the cascading mechanism, the fridge's slot will force a front-only orientation and require standing by blocking other preferences. If the goto fails, the tree fails.

If the goto succeeds, the reach primitive is called with the fridge as the container, the food as the reach object and the food's slot number. This primitive will invoke the reach tree of the fridge.

The reach tree of the fridge assumes the person has met the routing requirements of the slot, and is thus standing directly in front of the fridge at a distance of 1 tile. If the door is not open, the tree runs an animation of the person opening the door. This animation has events that cause the graphic of the door to change. The animation may sidle out a bit to allow the door to pass without hitting the person. Once the door is open, the reach primitive is called with the same parameters. This time, no reach tree is invoked since we are already in the reach tree. If the reach primitive fails, the reach tree fails, and the parent reach primitive fails.

Upon success of the second reach primitive, the person should be holding the food, and thus in the standing holding idle state. The tree then calls an animation for closing the door, and returns true. The reach tree completes successfully, so the original reach primitive completes successfully as well, and the get from fridge part of the interaction is done.

TBD: how is the door closed? A one-handed door close animation is easy enough to envision, but how the food is tied to the person's hands becomes important. Currently, people have a slot for each hand. With the new idle states, there will probably be a new slot for the center. Perhaps the fridge could just temporarily move the food into one hand or the other, run the animation, and move it back to the center.

Prepare at counter

Once the food is in the person's hand, he is ready to prepare it. The first thing to do is to find a counter top. This is done with the find slot primitive. The minimum and maximum height for this call depend on the flexibility of the preparation animations. The min and max. may be the same but should at least include the standard counter top height of 37 inches.

If a slot cannot be found, the tree fails.

If the slot is found, a goto primitive is executed with any orientation, 1 tile proximity, and a standing only preference, since the preparation animations are created specifically for standing. TBD: different skeletons—can children prepare food? In this case, the food also specifies the slot number to the goto primitive so that the cascading mechanism can be used. TBD: what to do if the slot is already full? (Sibling objects are getting more and more attractive as more examples are found.)

If going to the counter fails, the tree fails.

If it succeeds, the reach primitive is called to place the food on the counter. If reaching is successful, the graphic of the food is set to the preparation graphic, such as a cutting board with stuff on it, or a mixing bowl.

The tree then calls out preparation animations that were created specifically for the height range specified when the counter was found. These may include chopping, spicing, stirring, etc. The amount of time and possibly which animations are run depend on the cooking skill of the person. The tree may also alter the food graphic during this process to reflect the preparation.

Once the “preparation” is complete, the reach primitive is called again to pick up the food from the counter. This should succeed since the person just put it down, but if it fails for some reason, the tree must fail.

Prepare at stove

This tree starts by finding a stove. This is accomplished using a normal search loop to find the closes piece of furniture with its “can cook” flag set. This flag implies that things in its slots will have their temperature increased over time. TBD TODO: new attribute. If no such object can be found, the tree fails.

If a stove was found, the tree changes the graphic of the food to a pot to show that preparation is done. Then the person is made to walk to the stove using a goto primitive specifying any orientation (since the stove could actually be a barbecue), a 1 tile proximity and standing only. These parameters will be further restricted by the stove’s slot to be front-only.

TBD: how does the tree determine which slot to use on the stove. It could just find the first empty one.

If the goto fails, the tree fails. If it succeeds, the reach primitive is called to put the food on the stove. The reach primitive will invoke the reach tree of the stove.

The reach tree of the stove will first invoke the reach primitive to place the food on the stove. If this fails, the reach tree fails, consequently failing the original reach primitive. If reaching succeeds, the tree will then make sure the stove is on, running an “turn-on” animation if necessary.

Once the food is on the stove, the tree waits until the temperature of the food reaches the right level. This wait period may be randomized somewhat based on the cooking skill of the person, and his patience level. The main loop of the food may check the temperature and the current graphic to make animated steam, or some other cooking effect.

TBD: it would be nice if the tree could run cooking animations during this time, but since the cooking object could be a microwave or a barbecue, reaching is the only practical thing to do, which probably wouldn’t look that good since stirring is above the slot. Is this a problem we need to solve?

TBD: can the person be taken over by the user during this time? This could be easily accomplished by a high level state variable that the main interaction tree gets and sets and calls the appropriate subtree.

When the food is finished cooking, another reach is invoked to get the food off the stove.

This time, the reach tree of the stove will turn the stove off, then invoke the reach primitive to grab the food. If the reach fails, the reach tree fails, and the original reach primitive fails.

If reaching was successful, the prepare at stove part is done.

Place on table

The first thing this tree does is to find a table. It uses the find slot primitive with the open slot set and extremely generous height parameters, since the table can be basically any height. If no slot is found, this tree fails.

TBD: what about eating off the ground if no table is found? Or trying to find a chair? Perhaps the find slot primitive could create a special temporary invisible flat object near the person and return that. The object would not allow any interactions and its reach tree would signal that it was being used, and would kill itself after a time limit. Perhaps the Place On Table tree could look for a chair.

Once the slot is found, it is routed to using a goto primitive with any orientation, a 1 tile proximity, and preferences for chair sitting, ground sitting, then standing, and the found slot. If the goto primitive fails, the tree fails.

If the table's slot was routed to successfully, the graphic of the food is changed to the ready to eat graphic, a plate or bowl. Then a reach primitive is invoked to place the food on the table. If reaching fails, the tree fails.

Eat

Since the food is on the table, ready to be eaten, this tree is almost exactly the same as in Example 1 above.

TBD: One additional whiz-bang feature may be that the food actually imbeds several food chunk objects that it can create and place in slots on top of its plate graphic. Then, for eating, the reaching process can reach for the chunks instead of the main food object, then delete the food chunks after a hand-to-mouth animation.

TBD:

- How does the fridge get open if someone is trying to "goto" the food inside it?
- Is the multi-level route planning going to be efficient enough?
- Will the cascaded preferences produce unexpected results?