To Do List

- parsing of attributes in scenario files. e.g. `k.width(10)` in scenario files. IsisObject generators specify/randomly select the default values during `__init__()`, but then the defaults can be changed sometime after `__init__` and before `setup()`.
- Loading and running IsisScenarios files:
  - buttons for starting a task, running a training and test scenario
  - recording statistics about the task: how many steps since it started, state of task (failed/completed/ongoing)
  - displaying state of task in the menu
  - ~~checking for whether the goal state is met~~
  - ~~migrating the `kitchen.isis` into an "scene initialization" section of the "scenario/" files.~~
  - ~~a DirectGUI for loading tasks, which is the default screen when the simulator loads.~~
  - ~~recording state of task/scenario in logging file~~
- specifying scale ranges for some of the common models sizes, the same way the size of the kitchen is chosen from a random range.
- documenting a skeleton generator file with all possible superclass attributes, so that other people can work on the project by adding / describing models.
- working out the kinks in packaging binaries (try by running `make package` on mac)
- separating actions from the `main.py` as a different data structure in a different file.
- ~~exporting screen shots~~
- ~~RoomLayout layout manager that puts objects around the walls~~

IsisScenarios

Isis Scenarios are Python files found in the `scenarios/` directory.

Settings:

- Sequential or unordered: does the simulator reset IsisWorld between each task, or are they staged incrementally?
- Metadata: a string describing the scenario

Environment function

This specifies how to build an IsisWorld.

Tasks

Each isisScenario has one or more **tasks**, defined using the python `def task__name_of_task`. Each task can contain a **training** phase and one or more **test** functions. Allows you to specify different versions of Actions to use during training/test.

Training

For example, putting ralph in front of various objects.

Test

Any function that returns *true* or *false*. Has access to the entirety of IsisWorld.

About the IsisWorld Simulator

The IsisWorld simulator is available to researchers for building and evaluating problem solving and language learning systems with everyday problems that take place in a kitchen. We aim to use IsisWorld to simulate everyday commonsense reasoning problems that span many realms, such as the social, visual, kinesthetic, physical, spatial and mental.

What is a problem "realm"? Consider the problem of *hailing a taxi*. You could represent and reason about this problem in several different ways.

1. **Temporally**: Wait for a taxi. Maybe if you stay put, a taxi will drive by
2. **Spatially**: Find a taxi. You must eliminate the distance between yourself and a taxi.
3. **Socially**: Call a taxi. Communicate your position to a dispatching agent and an available taxi will come to you.

It is this resourcefulness—having many ways to solve a problem—that allows human problem solvers to flexibly adapt to many problem solving situations. A system that lacks these abilities is *brittle*.

Further, we are looking for test bed to study the problems of meta-reasoning: where a super-level planning system reasons about the world of a sub-planning system. Returning to a taxi example, we could consider the failure mode which causes a meta-level reasoner to step in and change the state of the planner. For example, it could ask the system to *elevate* the problem description to pursue the parent goal: *instead of "searching for a taxi" reconsider the problem as "traveling to your destination" and pursue other options: e.g., walking, train, asking a friend etc.*

More detailed arguments about using a simulator for studying AI and the choice to use kitchen problem domain is explain in this paper:

- An open source commonsense simulator for AI researchers. Dustin Smith and Bo Morgan. *Proceedings of AAAI–10. Workshop on Metacognition.*
- IsisWorld Presentation *Presented at the AAAI–10 Workshop on Metacognition.*

Use cases / Problem Scenarios

The development of the simulator is focused on the following test scenarios ### 1. Toast Making: studying first-level planning

Ralph is in the kitchen and has the goals of making and eating toast. Ralph has to "use" the knife to cut the bread, and then put the bread in the toaster. Problems addressed [#1]:

- Bodily, what actions does Ralph have available?
- Functional, how objects states can change: the effects of actions upon them and how they affect each other.
- Spatially, navigating through space without bumping into objects
- Motor routines: what macro actions can be represented to coordinate common sequences of primitive actions?
- Self models: how does the situation of Ralph's model (e.g., location of limbs in space, objects in hand, eyes opened or closed) influence the functions of actions he can perform.
- Visual: what objects are in the environment, how far are they, what shape and texture do they have?
- Mental debugging: if all toast-making problem solvers reach impasses, Ralph could reflect on his problem and pop up to his higher goal.

2. Knife sharing: studying social interactions

Ralph and his mother Sue are in the kitchen. Sue is currently using the only knife. Ralph has to ask Sue to use the knife. If he grabs the knife from her hand, he will be cut.

- Mental: what is Sue's active goal ("intention"), does she have the same intention? how will she react to my actions?
- Social representations, from [#1]
  - Social networks: who knows who? who has interacted with whom?
  - Dominance: who sets the goals of this group? who to listen to?
  - Goal interactions: do my actions help or hinder others? Who might interfere with my goals?
  - Impriming: can they do something I can't? what can I learn from them?
  - Groups: what are the roles in this group? What are the functions of these roles?

3. Learning by observation

Sue is communicating a new sequence of actions. Ralph must identify Sue's plan and then recognize her goal as different, and then learn the new plan as some deviation of the existing plan (e.g., making toast with jelly)

- Plan and goal recognition
- Planning and debugging in simulated mental worlds

4. Imprimer learning

Sue is teaching Ralph how *not* to use a kitchen. He must learn that the faucet must be turned off after being used, doors closed after they are opened, not to leave the refrigerator open for more time than necessary, etc. He must learn these how to represent and pursue these imagined goals and antigoals of his imprimer. This must cover the problem of **shared attention**, where the teacher deliberately acts a certain way to encourage the learner to focus on a relevant aspect of the shared situation.

- Shared attention and shared models of inference: "What will Sue think if she sees the mess on the floor?"

5. Language learning and instruction execution

Learning the labels of objects from examples. Learning to label events/actions with verb phrases at different temporal resolutions. Learning how to use and resolve pronouns, the linguistic equivalent of pointing. Learning the proper sequence to carry out a sequence of actions from a linguistic description

- learning and representing labels for things (nouns) in the world ("toast")
- learning and representing adjectives ("black", "hot", "large") — re-representing items in a perceptual measure space (concept domain) and using adjectives evoke discriminative boundaries with respect to a shared world model.
- learning and representing composition of linguistic concepts (e.g., "red" in "red wine" is different than "red hair") or the linking constraints between verbs and nouns ("run" + "dishwasher" versus "run" + "marathon")
- semantic parsing of a sentence into possible "interpretations"
- interpreting verbs as states or actions (e.g., modeling the taxonomic organization of verbs and how they map to between sets of events, related by generality relationships)
- interpreting nouns as constituents of states or actions (e.g., modeling the taxonomic organization of nouns and how they map to generality relationships between sets of items)
- interpreting prepositions by representing items in a visual geometry / using spatial relationships
- metaphoric mappings between concepts and spatial relationships w.r.t. some decision making process.
- articulating an event that Ralph has completed with respect to a set of planning decisions, modulated by Ralph's model of the listener's knowledge
- using hierarchical structure of linguistic phrases and aspectual connectives to model relationships between events and control structure of planning (do X "during", "after", "while" doing Y)
- language parsing using a cognitively plausible model (e.g., Shift-Reduce parsing)

6. Communicating instructions

Ralph describes his actions or Sue's actions as an English verb phrase.

IsisWorld Sub-Projects

Over the summer of 2010, we plan to make many significant improvements to IsisWorld. These projects are:

Implementation of Physics

Target Scope: one week

Angular and linear forces will be modeled, densities of objects will be represented by a Physical simulator. Because ODE integration in Panda3D is still preliminary [#1, #2], and commitment to open source principles has eliminated NVidia's PhysX platform as a viable option, our only current option is to use Panda3D's built-in physics support and collision detection. Some decent tutorials exist [#1 #2], and for ODE, when Panda3D's support becomes more robust [#1].

Design ideas

Changing physics engines should be as easy as switching the import statement in `simulator/physics.py`.

- Make the physics handling modular, accessible through the `IsisWorld.worldManager` variable
- Agents, Objects, and Environment items are initialized separately:
  - **Agents**: Are sub-classed instances, defined in `simulator/ralph.py` of `PhysicsCharacterController` class, which has an `updatePhysics` method. Geometrically, these are represented as capped cylinders, as TriMeshes are too computationally expensive.
  - **Objects**: In ODE, there are two types of objects: kinematic (non-self propelled) and dynamic (capable of self-motion). Most objects can be represented as a boundedBox, although some are best represented as cylinder or sphere.
    - Objects can be added through `worldManager.addObjectToPhysics`
  - **Environment**: The ground is represented as an infinite plane. Should represent walls of house as blocks, not TriMesh.
- Gravity is represented as a linear force downward

Initialization scripts for designing and loading environments

Target Scope: two weeks

As mentioned in the position paper, we want environments to be *generated* from a space of possible dimensions. Most of the variable properties of the items (size, location, plurality, color, state) could be left to future work. Default locations can be specified in a configuration file corresponding to prepositions: - X on Y in Z: "on" and "in" are less descriptive than 3D (relative) coordinates

This will require a general module for describing and loading components, a large range of visual, physical, linguistic, spatial (default locations), and functions (properties, methods to modify properties) for each object.

- **Physical**: shape (for Physical collision mask), dynamic or kinesthetic
- **Visual**: visibility, color mask, transparency level, scale size (of model)
- **Spatial**: default orientation of model, default locations (as represented in abstract descriptions: "in kitchen on table")
- **Functional**: use a commonsense **type system** and inheritance structure for **attributes** and **values** and their **default value** (e.g., `{'is_on': {'domain': [True,False], 'defa` and **event listeners** to change the properties of these actions, some of which can affect the other kinds of item properties (physical), etc.

Design ideas

- Read `kitchen.isis` world in, creating a labeled graph structure: "X on Y" becomes: *graph.addEdge(x,y,label='on')*
- With root node "kitchen", topologically sort all items. For each item:
  - Visually, add models to their parent renderer `attachNodeWrtParent()`
  - Set default properties of the item
  - Physically, register item within physics handler

Extending corpus of models

Lots that can possibly be added. Stay focused on the kitchen and the use cases (make toast)

- Can add models from Google's 3D warehouse, export them to **egg** files. Instructions: #1
- Blender models can be exported using Chicken.

- Alice I saw a list of these somewhere that were already in the egg file format. Some of these already have animation methods!
- List of game models
- (not? immediately useful?) list of resources

Event Handling in simulated world

In addition to items being able to re-act to actions of the character, events can be initiated by states of the world. For example, when a toaster "is_on" and "contains_item", then after 5 minutes, it changes the state of the item inside to "burned", darkens the color, and turns itself off.

Design Ideas

Use the FSM of Panda3D to do this. Possible approach: using Honda's OMICS corpus

Kitchen use-cases

Several tasks for evaluating intelligent agents in IsisWorld.

Creating Toast in the Kitchen

Resources for Developers

Here is a list of resources for developers that are getting started working with IsisWorld.

Panda3d.org has a really good manual, though it doesn't have 100% coverage of all the features in the API. Also the Panda3D forum is very valuable resource.

To learn about Git, Dustin recommends GitHub's videos.