Example Questions for Final

1. In Minstrel, storytelling cases and queries are represented as graph structures connecting together a number of different types of knowledge frames (objects), including State, Goal, Act, Actor and Object frames. Labels on the links in the graph describe the relationship (predicate) holding between two nodes. Cases are concrete actions, involving specific actors, resulting in specific states, that have happened in the past. Queries leave some of the fields in the knowledge frames blank (with constraints), corresponding to partially specified actions.

Here are the frame and link types:

<table>
<thead>
<tr>
<th>Goal Type</th>
<th>State Type</th>
<th>Act Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;type of goal – we won’t worry about goal types for this test&gt;</td>
<td>&lt;type of state&gt;</td>
<td>&lt;type of act&gt;</td>
</tr>
<tr>
<td>&lt;name of actor frame of actor holding the goal&gt;</td>
<td>&lt;name of actor frame involved in state&gt;</td>
<td>&lt;name of actor frame of actor performing the act&gt;</td>
</tr>
<tr>
<td>&lt;name of object frame if the goal involves an object – we won’t worry about this for the test&gt;</td>
<td>&lt;name of object frame the state holds for, if applicable&gt;</td>
<td>&lt;name of object frame of object involved in the act&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;type-specific value of the state&gt;</td>
<td>&lt;if act involves transfer, actor or object frame transfer is to&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;actor or object frame if state is directed state (e.g. emotion)&gt;</td>
<td>&lt;if act involves transfer, actor or object frame transfer is from&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>Object</td>
</tr>
<tr>
<td>&lt;type of actor&gt;</td>
<td>&lt;type of object&gt;</td>
</tr>
</tbody>
</table>

Types of links (predicates)
- **Intends**: Links an act to a state
- **Achieves**: Links a state to a goal
- **Motivates**: Links a state to a goal
- **Thwarts**: Links a state to a goal

Types of States
- **Health**: The health of an actor (values: healthy, injured, dead)
- **Posses**: The state of an actor possessing an object
- **Affect**: Emotion felt by actor towards (values: hate, neg, neutral, pos, love)

Types of States
- **PhysObj1**: Sword
- **PhysObj2**: Potion
- **Berries**: Food
- **Woods**: Location

For the following questions, assume these objects and actions exist.
Note about uniquely named instances: in this graphical representation, unique instance names, like “Francine,” “PhysObj2” and “Troll1” appear as a label above their corresponding frame, rather than as a name field within the frame. Think of these labels as handles to the unique instance; use these unique handles as frame names when filling out fields within a frame (like filling out the object field of a state frame – this is the way the chapters I handed out do the graphical representation).

1a. Draw the boxes and arrows diagram (with field values filled in) for the minimally specified query for “Somebody has the goal of doing something to kill a troll.”

1b. Draw the boxes and arrows diagram for the fully specified case for “Troll1, motivated by his hatred for John, had the goal of fighting John with a sword (PhysObj1), resulting in John being killed.”
2. Minstrel uses TRAMS to implement imaginative memory, the process of recursively changing queries into related queries, hopefully retrieving cases based on related queries, and, in TRAM-specific ways, adapting the retrieved case back to something that works for the original query. If the TRAMS are black boxed as part of memory, this has the effect of the problem solver “remembering” specific cases that weren’t directly memorized in the past (hence “imaginative memory”).

Consider the two TRAMS below.

**TRAM: Similar-Outcomes-Partial-Change**

**Transform Strategy**

If the problem specification has an act that results in a partial relative change of a state, create a new specification in which the relative change is extended.

**Adapt Strategy**

Replace the change of state in the recalled episode with the change of state copied from the original problem specification.

**TRAM: Generalize-Constraint**

**Transform Strategy**

Select and generalize a feature of the scene specification. Use this new scene specification as an index for imaginative recall.

**Adapt Strategy**

Adapt the recalled solution to the current problem by substituting the original feature value back into the recalled solution.

2a. Given the query and case below (query on left, case on right), show how one of the two TRAMS above (you have to decide which one to use) can be used to satisfy the query. Showing this involves drawing the diagram for the transformed problem (which will now recall the case), and drawing the diagram for the adapted case.
I italicized the state value that was extended by Similar Outcomes Partial Change during query transformation, and copied from the original query during adaptation.

2b. Given the query and case below (query on left, case on right), show how one of the two TRAMS above (you have to decide which one to use) can be used to satisfy the query. Showing this involves drawing the diagram for the transformed problem (which will now recall the case), and drawing the diagram for the adapted case.
I italicized the features that were generalized by Generalized Constraint during query transformation, and substituted from the original query during adaptation.

3. Here are the two plans Micro-Talespin knows for the GetObject goal (the goal to get an object – DCont in the original terminology). I’ve replaced the goal terminology of non-terminal goals (e.g. DKnow, DProx, etc.) with more human-readable names. Terminal actions still use the Yale School terminology (e.g. PTrans, ATrans, etc.).

GetObject-Plan1(actor, object, owner)
/* One way to get an object is to persuade the current owner to give it to you (the owner atranses the object to the actor from the actor). There has to be a current owner.*/

Steps:
Know(actor, whereIs(owner))
MoveTo(actor, actor, location(owner))
Persuade(actor, owner, atrans(owner, object, actor, owner))

GetObject-Plan2(actor, object, currentOwner)
/* Another way to get an object is for the actor to know where the object is, move him or herself to the location of the object, and atrans the object to him or herself (the actor atranses the object to the actor from we-don’t-know-who). */

Steps:
Know(actor, whereIs(object))
MoveTo(actor, actor, object)
Atrans(actor, object, actor, nil)

The Persuade goal has the general form:
Persuade(<the person doing the persuading>, <the person being persuaded>, <the goal or action you’re trying to persuade them to perform>).

The effect of successful persuasion is to add a new goal to the persuaded character, which they will start performing after the persuader’s current plan is complete. Thus, if the persuader’s current plan contains multiple persuade subgoals, the effect will be to queue up multiple goals on the persuaded character.

The relate relation has the general form:
relate(<person who believes relation>, <person relation is directed from>, <person relation is directed to>, <relationship>)

There’s one plan for finding out something (accomplishing the Know goal) if you don’t already know the information:

Know-plan(actor, info)
  Precondition:
  // actor knows where some other agent is
  knows-loc(actor, some-other-agent)
  // some-other-agent likes the actor or actor doesn’t believe that some-other-agent dominates the actor
  relate(actor, some-other-agent, actor, like) or not relate(actor, some-other-agent, actor, “dominate”)

Steps:
  MoveTo(actor, actor, loc(some-other-agent))
  // Pursue the goal of actor persuading some-other-agent to mtrans the info from some-other-agent to actor
  Persuade(actor, some-other-agent, mtrans(some-other-agent, info, actor, some-other-agent))

This initial storyworld state applies to 3a and 3b.
The honey is in the field.
Nobody owns the honey.
John bear is in the cave.
The cave is next to the tree.
The tree is next to the field.
John bear wants the honey.
Sally bee is in the tree.
Sally bee thinks the honey is in the field.
Sally bee dominates John bear.
John bear thinks Sally bee is in the tree.
John bear is not friends with Sally bee.
John bear thinks Sally bee dominates him.
Marvin muskrat is in the cave.
Marvin muskrat thinks he likes John bear.
Marvin muskrat thinks he likes Sally bee.
Marvin muskrat thinks Sally bee is in the tree.
John bear thinks he likes Marvin muskrat.
John bear thinks Marvin muskrat is in the cave.
John bear thinks Marvin muskrat likes him.
Sally bee thinks she likes Marvin muskrat.

3a.
Which of the GetObject plans will work for John bear to acquire the honey?
A. GetObject-Plan1
B. GetObject-Plan2
C. Both plans
D. Neither plan

Explanation: GetObject-Plan1 isn’t applicable because nobody currently owns the honey. For GetObject-Plan2, the first Know subgoal will fail because John Bear does not like
Sally Bee and Sally Bee dominates him (second step of Know precondition). John Bear can't use the Know plan with Marvin Muskrat because, though Marvin Muskrat satisfies the preconditions, Marvin doesn't know where the honey is (and our little example plan library doesn't have a plan for finding out information for someone if they've asked you for it and you don't know it).

3b. Write another GetObject plan (GetObject-Plan3) that will enable a new way for John bear to get the honey by talking to Marvin Muskrat. Using only the predicates (relations found in preconditions), goals and primitives (found in the steps of plans) described above, it is possible to write such a plan that has a two line precondition and four steps.

**GetObject-Plan3(object, actor, owner)**

Precondition:
knows_loc(actor, some-other-agent)
relate(actor, some-other-agent, actor, like) // actor believes some-other-agent likes him.

MoveTo(actor, actor, location(some-other-agent))
Persuade(actor, some-other-agent, GetObject(some-other-agent, honey, nil))
Persuade(actor, some-other-agent, MoveTo(some-other-agent, some-other-agent, location(John bear))
Persuade(actor, some-other-agent, atrans(some-other-agent, honey, actor, some-other-agent)

**Explanation:** The three Persuaded actions will be queued up on Marvin Muskrat. Since he likes John Bear, Marvin Muskrat will agree to get the honey, go back to John Bear, and give him the honey. Note that we need the middle step - once Marvin Muskrat has gotten the honey, John Bear doesn't know where he is (since he doesn't know where the honey is), so Marvin has to come back to him. As Marvin Muskrat pursues GetObject(Marvin, honey, nil), he will successfully use GetObject-Plan2; since he knows where Sally Bee is, and has a good relationship with Sally Bee, Sally Bee will tell him where the honey is.

4a. To determine how much agency Zork provides the player, we:
A. Analyze the depth of the listen-think-speak loop.
B. Determine whether there is discourse-level variation in replay.
C. **Determine whether the material and formal affordances of Zork are in balance.**
D. Determine whether there is story-level variation in replay.
E. All of the above.
F. None of the above.

4b. In the listen-think-speak loop for Zork, the listen portion consists of
A. The player reading the text response produced by Zork.
B. The system simulating the effect of the verb selected by the player on the story world.
C. The system parsing text input from the player.
D. The system producing text output based on the new state of the simulated world.
E. None of the above.

4c.
To determine the material affordances of Zork, one should analyze
A. What actions it makes sense for the player to be thinking about given the overall plot (story) conveyed by the story world.
B. The details of what’s going on in the story world (detailed descriptions of locations, actions of the thief, etc.).
C. The interface, and the player verbs accessible through the interface.
D. A & B
E. B & C
F. All of the above.

5. Consider the following abstract story grammar:

```
Story → Setting + Episode
Setting → State
Episode → Event + Emotion
Event → Event + Event
Event → Episode
Event → Action
```

6a. Consider the following story. The terminal type of each line of the story appears in all caps after each line.
Nataly is in her room. STATE
She built a Lego tower. ACTION
She tipped it over. ACTION
It broke. ACTION
She was pleased. EMOTION

This story is allowed by the grammar. (5 points)
A. True
B. False

6b. Now we will add some more specific rules to this very generic grammar. When Nataly is chastised for being naughty, she feels either sad or angry. The following story incorporates the new terminals NAUGHTY-ACTION, CHASTISE, SAD, ANGRY.
Nataly is at the table. STATE
She purposefully spills food on the table. NAUGHTY-ACTION
Nataly is interested. EMOTION
Nataly cries. SAD
Nataly watches a bug on the wall. ACTION
Nataly is interested. EMOTION
This story is not accepted by the original grammar. Modify the grammar by adding one new Episode rule, plus two new Neg-Emotion rules, that will allow the grammar to accept/generate this story. The Neg-Emotion rules will each have one terminal on the right hand side. (6 points)

Episode → Naughty-Action + Chastise + Neg-Emotion
Neg-Emotion → Angry
Neg-Emotion → Sad