Your assignment is to start with a basic wumpus agent that does not have very good behavior (poor average score) and modify the agent to improve its average performance measure. Unlike your agent for Lab #3, this agent will be allowed to use as much internal state information as you desire—in fact, your grade will depend in part on how much modeling you do. Grading criteria are specified in Section 5.

1 Playing Wumpus World

You agent will be playing the same game that it played for Lab #3, except that you will have to design an agent to play the complete wumpus game. As mentioned in Lab #3, there are really two stages to the complete game: (1) finding the gold (or determining that this is impossible), and (2) getting back to the start square to climb out. These two stages require different methods to achieve good solutions. This means that a wumpus agent needs to be “goal directed” at least to the extent that it recognizes which stage of the game it is in.

The scoring/performance criteria will be exactly the same as they were in Lab #3—except that you will not get credit for having grabbed the gold unless you make it back to the start square and do a climb action. You can still just elect to quit at any time. An important part of your strategy now must be determining that it is impossible (or at least unsafe) to get the gold.

2 Basic Wumpus Agent

The agent to start with is BASIC-WUMPUS-AGENT, defined in the file basic-wumpus-agent.lisp. The function BASIC-WUMPUS-AGENT returns an AGENT (object) with a certain PROGRAM (lambda body) that implements the behavior of the agent. The PROGRAM takes PERCEPTS as an input and returns the action to take next. As a side-effect of selecting an action it can update an internal model and infer facts about the environment from the percepts and its existing model. This internal model is called the knowledge base (KB).

The design of BASIC-WUMPUS-AGENT has a main body that calls three functions in sequence:

1. UPDATE-KB-FROM-PERCEPTS –
   From the current percept and the current state of the KB update the KB to model the current state of the world. It is in this function that the agent infers unseen aspects of the world—e.g., where the wumpus or pits may be.

2. SELECT-ACTION –
   Based on the current state of the world as represented in the KB, pick the “best” action to take next and return it.

3. UPDATE-KB-FROM-ACTION –
   Use the selected action to update the KB to reflect the state of the world after the action is taken. It is in this function that the agent infers things like its position.
BASIC-WUMPUS-AGENT does a very limited amount of modeling and has a very simple
strategy for taking actions. Basically, it keeps moving forward unless it detects glitter or it
has the gold. If it bumps into a wall then it quits. It gets a very negative score on average
because it does not use its stench and breeze percepts to reason about where the wumpus/pits
are and avoid moving to these squares.

3 Building a Better Agent

You should start with BASIC-WUMPUS-AGENT and improve its performance by extending
each of its three main functions. The key extensions that you need to make are:

1. Develop more complete models of the world state by using the percepts to infer possible
   wumpus/pit locations, etc.
2. Develop a better strategy for selecting actions to take.
3. Extend the modeling of action effects to handle the full range of actions.

The KB structure already has slots for many of aspects of a world model that you will
want to develop. In particular, it has a MAP slot that is a 6 × 6 array. You will want to keep
a map of the wumpus/pits and status (an iconic model). A good way to do this is to have
each array element correspond to a grid location and be a list of symbols representing the
information you want. E.g., if you have the list (pit? no-wumpus), this could represent that
that location might contain a pit but cannot contain the wumpus. You are free to extend
KB in any way you want, if there is additional information that you want to store.

A key element that you may want to make use of in KB is the PLAN slot. This can be
used if you determine that a sequence of actions should be taken and you need to retain this
sequence between agent calls.

4 Utilities Provided

A number of functions have been provided in the file wumpus-agent-utilities.lisp to assist
you in writing your agent. There are four sets of helper functions:

1. Position and Orientation Functions –
   Can be used to check and change the agent’s position and orientation as a result of
   movement actions.
2. Map Element Get/Change Functions –
   Can be used to get the contents of the map array elements and to set them.
3. Map Check/Change Functions –
   Apply some function to every element in the map array. Can be useful for initializing
   the map in some way or for looking to see if something is true—e.g., there is some OK
   square.
4. Search Functions –
   Can be used to discover minimum cost sequences of forward and turn actions that will
   get the agent from its current position/orientation to a position/orientation that meets
   a goal test.
5 Grading Criteria

The following criteria will be used in grading your project submission:

1. Does the code compile and run? Code that does not compile and run (at least most of the time) will not be examined further. No points will be awarded for such code.

2. The (long term) average score that your agent attains.

3. How much of the possible modeling knowledge you encode. I will consider whether you model:
   - agent position and orientation;
   - possible wumpus and pit locations vs. safe locations;
   - sets of alternative wumpus/pit locations (additional credit).

4. How sophisticated your action selection strategy is. I will consider whether your agent:
   - Makes appropriate choices?
   - Is effective in searching?
   - Is efficient in moving to desired locations?
   - Makes use of plans?
   - Shoots the arrow (appropriately)?
   - Stops when appropriate.

5. Do you handle every possible action in updating the KB?

6. The general quality of your code. Have you learned anything about Lisp? E.g., are you reimplementing Lisp functions or using inappropriate (C-like) approaches to accomplish something?