Artificial Intelligence Project 2: Design Document

Group 11: Lisa Peters, Janette Rounds & Monica Thornton

1. Description of the Problem

The Wumpus world is an environment in which a knowledge-based agent explores a cave fraught with perilous obstacles in an attempt to find the hidden gold. This agent, or explorer, starts with very little knowledge about their surroundings and they overcome their lack of knowledge about the environment with logical reasoning. Our explorer's actions have associated performance measures, with a payoff of 1000 for finding the gold in the cave, and a bonus of 10 for killing the wumpus. The explorer's performance can also be penalized, with a 1000 point hit for dying, a cost of 10 for each arrow fired, and a penalty of 1 for every move or direction change made by the explorer. The version of the Wumpus world we are implementing is based on the specifications provided by Russell and Norvig, with a few key changes from their formulation, mostly with respect to the environment (Russell & Norvig, 2003).

As in the classic Wumpus world, the environment is a square grid of cells within a cave where some of the cells contain certain death (in the form of pits and wumpi), and where one of the cells holds a glittering pile of gold. Unlike the classic Wumpus world, our implementation takes as input the probability of a cell being a pit (P_{pit}), containing a wumpus (P_{wumpus}), or being an obstacle (P_{obs}). The gold is the goal of the exploration, and in our implementation, the explorer teleports out of the cave once they find it. Within this environment, the explorer has a variety of actions. The explorer can move by going forward one square (*Forward*), turning left or right 90° (*TurnLeft*, *TurnRight*), picking up the gold in the current square (*Grab*), and shooting an arrow (*Shoot*). When an arrow is shot, it travels in a straight line until it hits a wall or a Wumpus. Any Wumpus hit by an arrow is immediately killed and released a perceptible *Scream*. The explorer gains information about the environment through five Boolean percepts: *Stench* which indicates that a wumpus is in an adjacent cell, *Breeze* which informs the explorer that they are in a cell adjacent to a pit, *Glitter* which tells the explorer they are in the same cell as the gold, *Bump* which means the explorer has encountered an obstacle or a wall, and *Scream* which tells the explorer that the wumpus has been slain.

The software we are developing for Project 2 has two major functions. The first task our software will perform focuses on the generation of several Wumpus worlds varying in size from 5×5 to 25×25. The second task involves using two explorers (each with different strategies) to navigate these environments in search of gold. For each explorer, we will keep track of a number of performance statistics, in order to compare the performance of each agent. Additional details regarding the implementation of these explorers can be found in Section 3 of this document, and our plan to evaluate their performance will be outlined in Section 4.

2. Software Architecture

In the last project, we had five algorithms and many of those algorithms overlapped in the steps they completed. In order to be able to overwrite some steps but not the shared steps, we used the Template Method software design pattern. In this project, we only have two algorithms (i.e. explorers or "Dudes"), and although they are making the same types of moves, they are building a knowledge base in fundamentally different ways. The steps do not necessarily overlap. In this instance, a Strategy software design pattern is more applicable (Freeman, Robson, Bates, & Sierra, 2004). We still have an abstract class that contains a few concrete methods, but all of the explorer methods will be instantiated by the concrete classes as a separate strategy.

A model of our software is provided as Figure 1. This diagram illustrates the major classes implemented in our software, as well a selection of some of the methods in each class, and the connections between the classes. The `RunModels` class provides the user with a method to run the `WorldGenerator`, as well as each of the explorers (`InformedDude`, `ReactiveDude`). The world generator takes the desired world size, P_{pit}, P_{wumpus} and P_{obs} as input, and outputs a 2D array corresponding to the world to be explored. These explorers get basic information about the world (world size, number of wumpi) as input, and each of the explorers implemented in this work inherit from `AbstractDude` which provides a general template of the relevant methods for the explorers, which are fleshed out in the explorer classes.

Because both the explorers have the same available actions and world to explore, the `Move` class provides both of the explorers with the ability to interact with the world through actions such as *TurnLeft*, *Grab* or *Shoot*. The `Move` class also tracks all of the relevant statistics about the explorer's performance, and provides console output as to how the move impacted the explorer and world. Additionally, we wanted to keep the explorers from having access to the complete map of the world. We foresee that storing the complete map with the explorers could lead to bugs where the explorers are not making inferences or building their knowledge base about the world, but accessing the information about the world directly. To prevent this from happening, we will keep the complete map of the world in the `Move` class. The explorers will call the `Move` class when they make a move, but the `Move` class will track the results of these moves, change the explorer's position in the world and communicate percepts to the explorers.

Both explorers will gain knowledge about the world as they explore it, and they store this information in the `KnowledgeBase`. As previously mentioned, the two explorers employ different strategies, and only one of the explorers (Informed Dude) uses first-order logic to navigate the world, therefore the `InformedDude` class employs the `InferenceEngine` to make decisions as to which actions to take.

3. Design Decisions

In the following subsections, we outline a number of the key design decisions we have made when implementing our solution. The decisions more directly related to our experimental design are presented in Section 4 of this document. Additionally, although not depicted in Figure 1, our software will also include classes for running experiments and printing results.

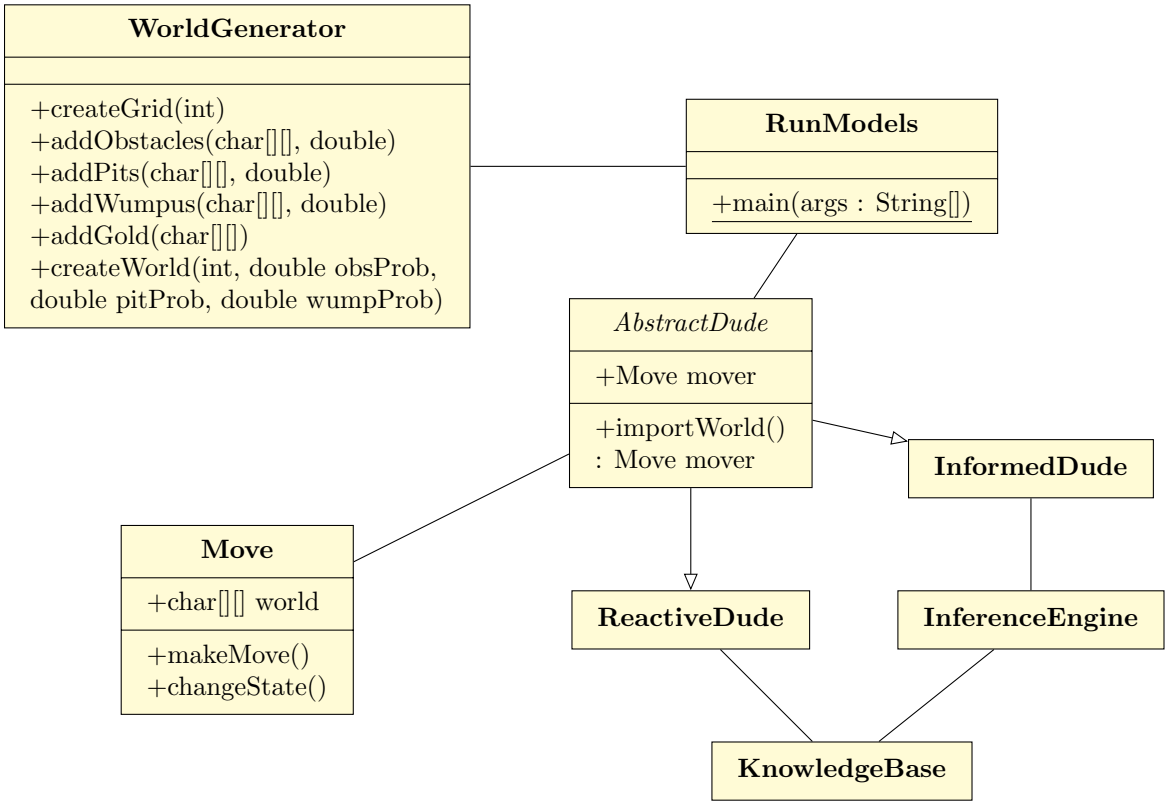

Figure 1: UML diagram that outlines the architecture of the described Wumpus World software.

Time permitting we will also include unit tests for all of the major functions, to ensure that the software continues to perform as expected.

3.1 Wumpus World Generator

We represent the world as a two dimensional character array of size $n \times n$ where $n \in \{5, 10, 15, 20, 25\}$. In this array, a "_" indicates a safe cell, a "p" represents a pit, a "w" represents a wumpus, an "o" represents an obstacle and "g" represents the gold. Pits, obstacles, and wumpi each have an associated probability that determines how many of each of these will exist in the world. We always reserve a space for the explorer to start in and the space for the gold. A world with no pits, obstacles, or wumpi would be excruciatingly boring, whereas a world with only pits, obstacles, and wumpi (excluding the explorer's start cell and the gold cell) would likely be impossible. Therefore, we set maximum and minimum probabilities. For the minimum probabilities, at least one probability must be greater than or equal to $\frac{1}{n*n}$, and the sum of all probabilities must be greater than or equal to $\frac{1}{n*n}$. For the maximum probabilities, in order to prevent overlap of the gold, wumpii, obstacles, and pits, the sum of all probabilities must be less than or equal to $\frac{(n*n)-2}{n*n}$. Subtracting two cells allows us to leave one space for the explorer and one for the gold. This would be an

extreme upper bound. In practice, we would set our maximum sum of all probabilities to be less than or equal to $0.75 * \frac{(n*n)-2}{n*n}$, which would allow only 75% of the cells in the world to be covered by pits, wumpi, and obstacles. See Section 4 to see how we will determine these probabilities.

3.2 Explorers

For our creation of the Wumpus world, we implement two version of our agent, the more careful and thorough Informed Dude who uses the Inference Engine to explore and reason about features of the map, and the Reactive Dude, who as his name implies, does not do any premeditation of actions. Accordingly, we made the design decision to create an `AbstractDude` class. The skeleton of the `AbstractDude` class will contain abstract functions for calling allowable move functions, enumerated in the description of the `Move` class. Additionally, the Dudes will utilize the `Move` class to update statistics and keep track of the various performance measures.

The first explorer, Informed Dude, uses first-order logic to explore each Wumpus world. Our plan for the explorer is to model it after the hybrid agent in the text, where they simultaneously maintain and update their knowledge base, along with a plan for future action (Russell & Norvig, 2003). In the knowledge base, the explorer will keep a list of known safe cells as an array. Additionally, they will keep a list of cells to be explored as an array in their knowledge base, this will serve as their plan of action - and the priority of those cells will be based on the priority of the goals. For example, the explorer gets the highest payoff from finding the gold, so if the explorer senses *Glitter* and a decision needs to be made between visiting safe spaces that might contain gold, or finding and killing the wumpus, the agent will prioritize those moves that aid them in safely finding the gold. In the absence of information from the percepts, the explorer will go to the closest unexplored square that it believes to be safe. Within the knowledge base, the percept information, the safe cells, and the plan will all be indexed so we do not lose the valuable time or location information. Before making any moves, the explorer will consult the inference engine to determine the best plan of action at that time. The hybrid agent is not the most computationally efficient approach, and if this becomes problematic with larger worlds, we will look into using a caching process, as described by Russell and Norvig (Russell & Norvig, 2003).

As a comparison to the Informed Dude, the Reactive Dude does not do any inference about the Wumpus world or the map. The Reactive Dude will also have a knowledge base, in the sense that he will update a map according to what he knows or suspects about his surroundings. However, rather than using further exploration to ascertain the exact whereabouts of obstacles and dangers, this simple updating will be the extent of his planning. Similar to the Informed Dude, the knowledge base will be an array of cells. The Reactive Dude will travel to a nearby safe cell randomly picked from his neighboring cells. When no more safe cells are available, this explorer will randomly pick to move to an unsafe cell. A design decision of ours was to not have the Reactive Dude backtrack across previously traveled cells to find another safe cell. For example, if the explorer starts in (1,1), knows (1,2) and (2,1) are safe, travels to (1,2) and is surrounded by unsafe cells, he will not travel back to (2,1).

3.3 Inference Engine

For our Informed Dude, we will use an inference engine to draw conclusions about the Wumpus world. Specifically, the inference engine can be told information, such as the presence of *stench* in a cell, $Stench(3, 4)$, or queried about information, such as $Wumpus?(4, 4)$. When the inference engine is told something by the explorer, it will update the knowledge base to reflect the explorer's discovery. The inference engine will also be informed about each move of the explorer. After an update of the knowledge base, the inference engine will use First Order Logic, hereafter abbreviated to FOL, to use the input to discover any new rules or predicates that can be drawn and put into the knowledge base. This is a recursive learning process, where any new rules may prompt the creation of still more. After no new rules can be created, the inference engine will return an action, in the form of a move, for the explorer to take, using the recently updated information to choose the wisest choice.

Because FOL allows the use of quantifiers, variables, and predicates, and due the nature of the knowledge base being comprised of Horn Clauses, FOL is an optimal choice to maintain or ascertain knowledge about the Wumpus world. This will allow us to use Resolution and Unification algorithms to continually update the knowledge base. Unification, as described by (Russell & Norvig, 2003) will allows us to use multiple rules to draw conclusions about the safety (or lack-there-of) of cells. We will also be using resolution, which uses a method of proof by contradiction to deduce facts about the Wumpus world (Russell & Norvig, 2003). If we find that inference speeds become intractable, especially for large Wumpus worlds, we will look at implementing some of the resolution strategies given in the text, namely unit preference, linear resolution, or subsumption.

3.4 Move

The `Move` class will store the map of the Wumpus world as a 2D array. As detailed earlier, we made the decision to not store the map with the explorers, to ensure that they were relying exclusively on their knowledge base. Additionally, `Move` will output the relevant performance statistics (enumerated in Section 4) as a flat file, which we will use in our experiments to gauge the performance of each explorer.

4. Experimental Design

We want to compare the efficacy of both explorers on five different sizes of worlds: 5×5, 10×10, 15×15, 20×20, and 25×25. In order to prevent a single world from flummoxing a particular explorer, we will run each explorer on the same set of worlds. Since the worlds are stochastically generated, in order to prevent a particularly hard or easy world from skewing the results, we will create five different worlds of each size.

We also want to find the probabilities of pits, obstacles, and wumpi that leads to the most worlds solved, the least worlds solved and some middle range solved. We will create five worlds of each set of probabilities from Table 1 and run both explorers on these probabilities. We also plan to run each explorer ten times. For example, we will run the Informed Dude on five different sizes of worlds, five different values for creation probabilities for each size, five individual worlds for each size-probability combination, and ten replications for each world. This gives us 1250 runs just for the Informed Dude. If this proves infeasible due

World Size	Total Cells	Min Prob	Max Prob	Prob Range to Test
5	25	0.04	0.92	(0.12, 0.23, 0.46, 0.69, 0.92)
10	100	0.01	0.98	(0.03, 0.245, 0.49, 0.735, 0.98)
15	225	0.0044	0.991	(0.012, 0.248, 0.496, 0.743, 0.991)
20	400	0.0025	0.995	(0.009, 0.249, 0.498, 0.746, 0.995)
25	625	0.0016	0.997	(0.006, 0.25, 0.5, 0.75, 0.997)

Table 1: Maximum and minimum probabilities for each world size as well as the range of values for the world probability creation (that approximately correspond to 1%, 25%, 50%, 75%, and 100% of cells covered. These values are the sum of all probabilities used in world creation. In order to get the actual probability, we will divide the total probability by three.

to time constraints, we reserve the right to remove probabilities, replications, and (in an extreme situation) sizes, in order to ensure that we finish all experiments. However, we will keep our removals constant between the Informed and Reactive Dudes.

Our second experiment will compare the overall performance of the Informed Dude and the Reactive Dude. In order to measure performance, we are tracking six statistics. These five statistics are: number of wumpi killed by explorers, number of explorers killed by wumpi, number of explorers killed by pits, number of explorers that find gold, the average moves made by each explorer, and the average return for each explorer. We will also identify and track unsolvable worlds. For the comparison between explorers, we will run each world twenty five times, with five worlds of each size and five sizes. In order to create these worlds, we will select the probability from our range of probabilities in Table 1 that produces the greatest difference between the performance of the Reactive Dude and the Informed Dude.

References

Freeman, E., Robson, E., Bates, B., & Sierra, K. (2004). *Head First Design Patterns.* O'Reilly Media.

Russell, S. J., & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach* (3rd edition). Pearson Education.