Assignment 4: Guess the Word

AI-2 Systems Project (Summer Semester 2023)

Jan Frederik Schaefer

Friedrich-Alexander-Universität Erlangen-Nürnberg, Department Informatik

Topic:	Decision trees/information gain
Due on:	September 29, 2023
Version from:	October 26, 2023
Author:	Jan Frederik Schaefer

1 Task summary

Implement an agent for a word guessing game. You can test your agent by competing on the server.

Didactic objectives

1. Gain hands-on experience with decision-making based on information gain,
2. solve various challenges involving probabilistic reasoning.

Prerequisites and useful methods

1. Basics of working with probabilities,
2. decision trees/information gain as discussed in the lecture.

2 Rules

The rules are inspired by the popular word-guessing game hangman [WH]. The player has to guess a randomly chosen word by repeatedly making letter guesses or word guesses. After each guess, the player gets feedback (e.g. about the positions of the guessed letter). The goal is to use as few guesses as possible to find the right word. In this assignment we will use two different sets of rules: *standard rules* and *advanced rules*.

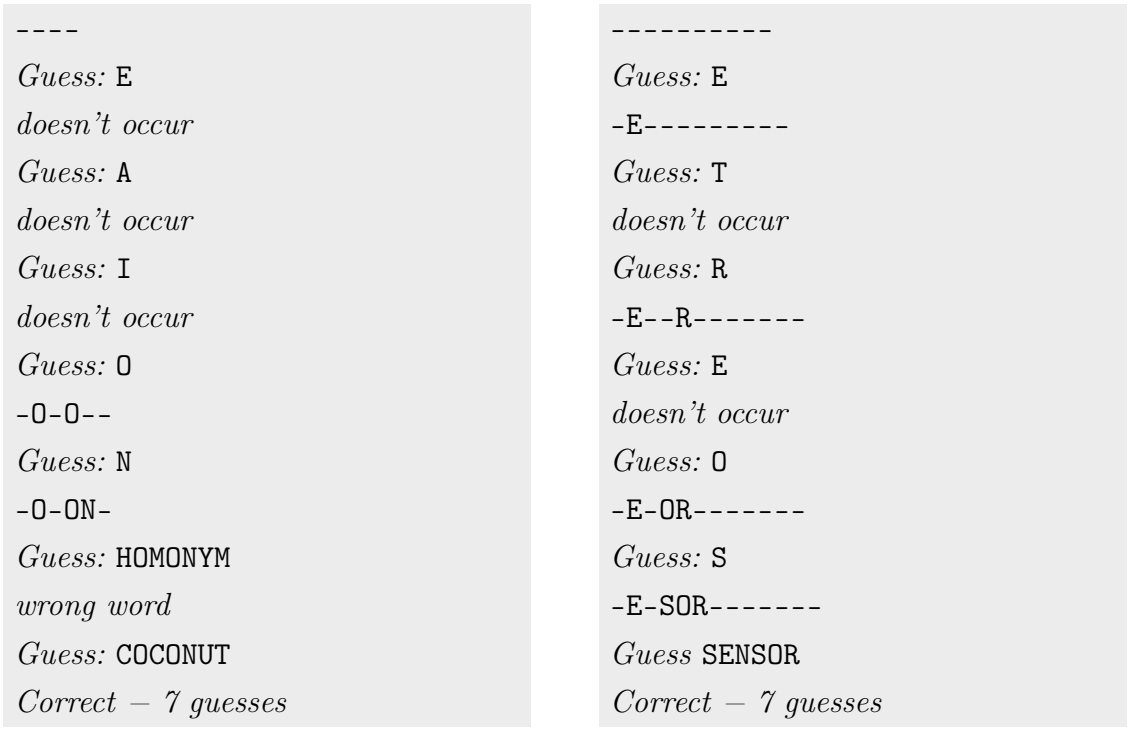

Figure 1: Example games with the standard rules (left) and the advanced rules (right).

2.1 Standard rules

The word is initially represented by dashes, where each dash replaces a letter of the word. If the player makes a letter guess, all occurrences of that letter in the word are revealed. The game ends if the player correctly guesses the word. See Figure 1 (left) for an example game.

2.2 Advanced rules

There are two changes in the advanced rule:

- Only a single letter location is revealed at a time (it is picked uniformly random among all unrevealed positions of that letter).
- The length of the word is disguised by appending more dashes in the feedback.

Figure 1 (right) shows an example game with the advanced rules.

2.3 Word list

The words are picked uniformly random from a list of nouns from [GNL]. The assignment repository [AR] contains a copy of that list (you should use the copy because the original might get modified). Words that contain diacritics or spaces are ignored, but hyphens are allowed. Note that hyphens are not valid letter guesses.

3 Guessing words on the server

You should test and evaluate your agent by competing on the server. The rating of your agent is the lowest average number of guesses in 1000 consecutive games – the lower your rating, the better. Your agent can communicate with the server via HTTP requests. A Python implementation of the protocol is provided in the assignment repository. The details of the protocol are described in Appendix A, but you will only need them if you want to create your own implementation (e.g. in a different programming language).

3.1 Action requests

The server will send you action requests, which contain an identifier for the request and a JSON object describing the current state of your word-guessing endeavour. Here is an example state description:

```
{
    "feedback": "-O-ON--",
    "guesses": ["E", "A", "I", "O", "N", "HOMONYM"]
}
```

3.2 Sending Actions

Your agent should respond to an action request by sending an action as a string:

1. For word guesses, the agent should send the word (in capital letters) as a string.
2. For letter guesses, the agent should send the letter as a string (again capitalized). As there are no single-letter words in this game, the server can distinguish letter guesses from word guesses.

4 What to submit

Your solution should be submitted to your team's repository. It should contain:

1. all your code,
2. a README.md file explaining how to run your code to compete on the server (including how to install dependencies),
3. a brief evaluation of your solution (what approach worked well, what did not work well) either as a PDF file ($\approx$ 1 page) or as part of your README.md.

5 A few tips

1. Make sure you understand the idea of using the expected information gain for decision making in decision trees.
2. The standard rules are noticably simpler than the advanced rules, which have hidden complexities. For example, consider a game with only two (equally likely) words: `DOG` and `DAD`. After the letter guess `D`, the feedback is `D-`. Given this feedback, the word is now more likely to be `DOG` than `DAD`.
3. You can improve the performance by caching results (e.g. the best initial guess for each word length).

6 Points

You can get up to 80 points for the rating (average number of guesses in 1000 games) of your agent according to the server (assuming it is reproducible). Concretely, you will get the following points for the standard rules:

- 20 points if the rating is ≤ 8.
- 30 points if the rating is ≤ 6.
- 40 points if the rating is ≤ 5.5.

For the advanced rules, you will get additionally

- 20 points if the rating is ≤ 10.
- 30 points if the rating is ≤ 8.
- 40 points if the rating is ≤ 7.5.

Assuming you have at least a partial solution, you can additionally get up to 20 points for the quality of the submission (README, evaluation, ...). The maximum number of points

is therefore 100. If the grading scheme doesn't seem to work well, we might adjust it later on (likely in your favor).

References

[AR] *Repository for Assignment 4: Guess the Word.* URL: `https://gitlab.rrze.fau.de/wrv/AISysProj/ss23/a2.4-guess-the-word/assignment`.

[GNL] *The Great Noun List.* URL: `http://www.desiquintans.com/nounlist` (visited on 07/06/2022).

[WH] *Hangman (game).* URL: `https://en.wikipedia.org/wiki/Hangman_(game)` (visited on 07/27/2022).

```
// First request:
{"agent": "MyAgent", "pwd": "r7iUM8o1NLbFdkI2WmBDldsYHD3wLwUQAKoG_2_xBcE", "actions": []}

// First response
{"errors": [], "messages": [], "action-requests": [
    {"run": "40#1", "percept": ...},
    {"run": "7#3", "percept": ...}]}

// Second request
{"agent": "MyAgent", "pwd": "r7iUM8o1NLbFdkI2WmBDldsYHD3wLwUQAKoG_2_xBcE",
    "actions": [{"run": "40#1", "action": ...},
                {"run": "7#3", "action": ...}]}
```

Figure 2: Example interaction with the server.

A The AISysProj Server Protocol

For some assignments, agents are tested by interacting with the AISysProj server. This appendix describes the protocol and is **only relevant if you want to implement the server protocol yourself**.

The protocol is rather minimal: You send an HTTP request to the server with your credentials and actions, and the server responds either with an error message or with a list of action requests that you should respond to in the next request. For the first request, you will have to send an empty list of actions to get action requests from the server (see Figure 2 for an example).

Configuration files The server details and your credentials are stored in configuration files. Usually, we will generate configuration files for you and commit them to your repository. They are JSON files that contain the following fields:

- url: the server URL,
- env: the environment,
- agent: the agent's name, and
- pwd: the agent's password.

The request You should send a PUT request to [url]/act/[env], where [url] and [env] are provided by the configuration file. The request body should contain a JSON object with the fields:

- agent: the agent's name (from the configuration file).
- pwd: the agent's password (from the configuration file).
- single_request: true or false, indicating if a only a single action request should be sent (can be helpful for debugging).
- actions: the actions the agent wants to do as a list. Each action is represented as an object with two fields: run is an identifier of the action request (provided in the server responses) and action is the action the agent wants to do – the value format depends on the assignment.

The response If the request was accepted, you will receive a JSON response with the fields:

- action-requests: a list of action requests that you should send actions for in the next request. Each action request is an object with two fields: run is an identifier so that your action can be linked to the request and percepts describes what you know about the current state (e.g. the position in a game).
- errors: a list of error messages (e.g. if your move was invalid).
- messages: a list of other messages.

Error response In case of an error (e.g. invalid credentials), you get a JSON response with the fields:

- errorcode: The HTTP error code.
- errorname: The name of the error.
- description: A more detailed description of the error.