Assignment 1 — Extensive-Form Games

Tomáš Votroubek (*votroto1@fel.cvut.cz*)
Assignment by Michal Šustr

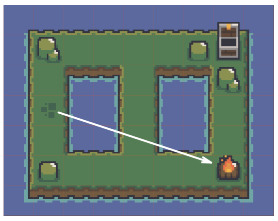
November 2, 2021

Consider the following problem: You find yourself on an island in possession of a grid-based map. Your goal is to reach the destination while collecting as much gold as possible. Two things make this difficult; first, you must step on each grid square at most once; second, bandits want to ambush you and steal the gold you have collected. The bandits may hide only on specific squares (termed "dangerous places," marked "E" on the map), and there can be at most one bandit at each dangerous place. You know how many bandits there are, but not where they are going to hide.

Each successful ambush results in the loss of all your gold and the end of the game. Otherwise, you dispatch of the bandit and may continue on your way, safe in the knowledge that one less bandit is waiting for you[1]. Bandits decide their position not just at the beginning of the game but can also use sensors. If the first dangerous place you enter is unoccupied, an alarm is triggered, and one bandit can relocate to a different empty dangerous place.

You receive 10 in-game points for reaching the destination and one additional point for each collected gold. If a bandit gets you, or you get stuck at a dead-end, you get nothing.

[1]Use this knowledge to your advantage!



```
7
9        ⟵——— A 7 by 9 map
#########
#E----EG#
#-##-##E#
#S##-##-#
#-##-##-#
#E-----D#
#########
2        ⟵——— with two bandits
0.5      ⟵— and 50% ambush success
```

Figure 1: Get from the starting position S to the destination D with as much gold G as possible, while avoiding bandits hiding in the dangerous places E.

1 Assignment

The assignment is split into two parts:

The Game Tree Part (6 pts.) formalize this problem as a two-player zero-sum extensive-form game. We prepared a template that you should implement. The template contains code to export the EFG into a format that can be parsed and loaded by Gambit. You can use the visual viewer to check if the game has been modeled as you intended. Automatic evaluation of the constructed trees is non-trivial, and you may receive points for invalid solutions. Therefore, the score in the upload system serves as only an upper bound of the final score. We will revise your submissions manually and deduct points if we find errors.

The Sequence-Form Part (8 pts.) specify the sequence-form linear program for this game and solve it using an external LP solver. We prepared a template that you should implement. You can use the following LP solvers: cvxopt, gurobipy[2]. You should consult the manual of your LP solver to figure out how to input your sequence-form LP and solve for the value of the game and player strategies.

2 Input and Output

The squares of the maze are defined as follows:

$\#$ obstacle

$-$ empty square

S start of your agent

D destination of your agent

G gold treasure

E dangerous place

Input for the first task the program reads input in following format:

```
number: number of rows of the maze (M)
number: number of columns of the maze (N)
M rows, N symbols representing squares of the maze
number: the overall number of the bandits
float number: the probability of a successful attack
```

Input for the second task the input starts the same as for the first task, and there is one additional row that specifies the index of the player (0 or 1).

[2]To setup gurobi locally, there is a great tutorial in the Combinatorial Optimization course.

Output for the first task The output must be a valid gambit file, as defined by the export_gambit function in the python template.

Output for the second task The output of your program is the expected utility in the root node for the given player.

3 Conventions and FAQ

- Your in-game ”avatar” is called the ”agent.”
- Your agent moves in the maze in 4 directions (up, down, left, right).
- As soon as the agent reaches the destination square, the game ends.
- Your solution must be within 1e−5 of the true solution.
- Use Python $\geq$ 3.6.
- Use the name game_tree.py for the first task, and game_lp.py for the second. You should import the tree file in the LP solving file.
- the alarm is triggered only when the first dangerous place the agent visits is unoccupied
- the bandits know which dangerous place caused the alarm
- Only one bandit can move to another unoccupied dangerous place but does not have to.

3.1 Notes

Actions are ordered in the sense that for two histories h, g in the same infoset it holds that h.actions() == g.actions(). You can use action’s index in the list for its identification. Do not use str(action) or some other way of encoding the action based on your previous implementation.

To identify sequences it is not enough to use the list of action indices! Consider simple poker: first player can fold or bet. Folding in one $Infoset_1$ (player has card J) is however not the same thing as folding in $Infoset_2$ (player has card Q)!

In class, we identified folding as f1 and f2, based on the infoset they came from.

Also, do not rely on Action being able to be used as a Dict key as it does not have the hash() function implemented.

4 Examples

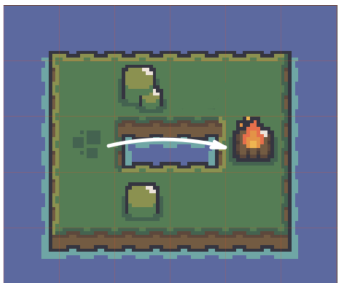


Figure 2: Consider playing this game against a single 100%-successful bandit. The agent's value is clearly 5 as the best the bandit can do is to hide in either of the bushes and hope for the best.

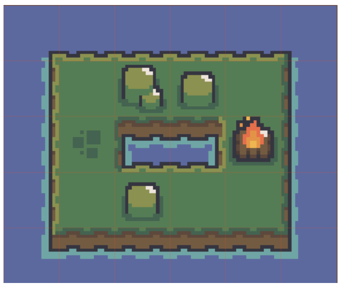


Figure 3: Contrast the example above with the case when one path has two dangerous places. The task is now impossible. By picking the bottom bush, the first bush on the top stays unoccupied and triggers an alarm when stepped on, leaving the agent with only one option — to step on the second bush in the full knowledge a bandit is waiting there.

You are encouraged to write your own simple test-cases (start simpler than this).