STOR-609 Presentation

Reproducing Optimal Play of Pig(let)s

Group 3

Vlad Bercovici

Malcolm Connolly

Rebekah Fearnhead

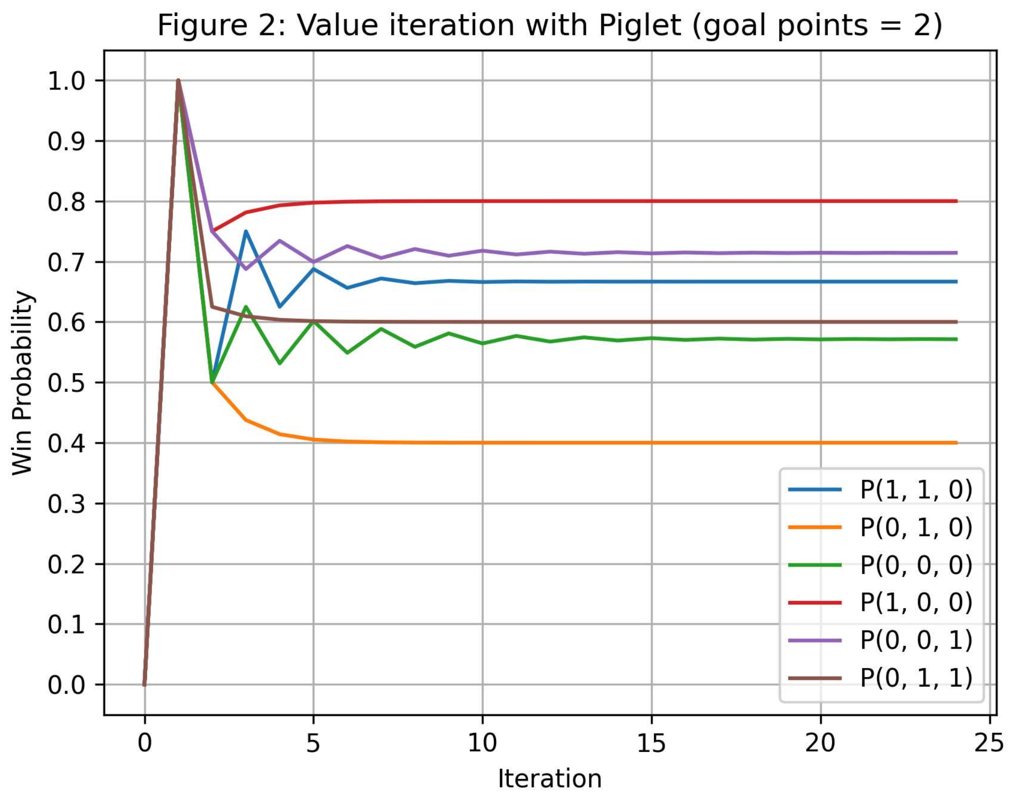
Niharika Peddinenikalva

Summary of our findings 🐷

- We believe we have managed to reproduce all figures from the original paper.
- We found that the figures of winning percentages were different from those in the paper.

Optimal Piglet play (Figure 2)

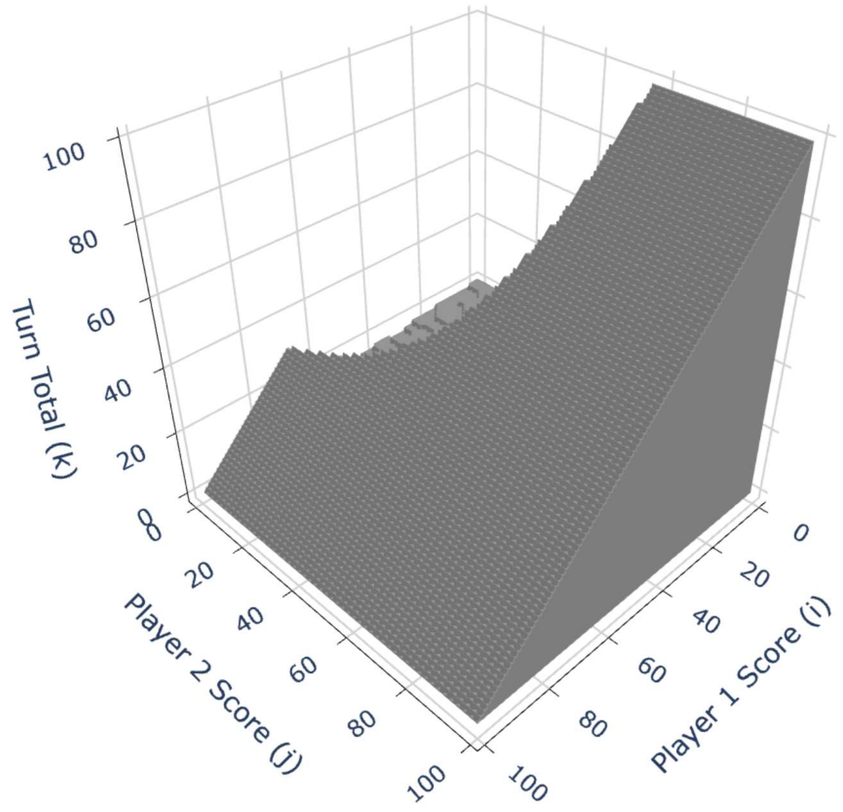
- Each of the lines represents how the estimates for each of the win probabilities change as iterations of the algorithm are performed.
- The win probability converges within the first 25 iterations.🐷
- Our figure differs in the first 5 iterations.



Optimal Pig play surface (Figure 3)

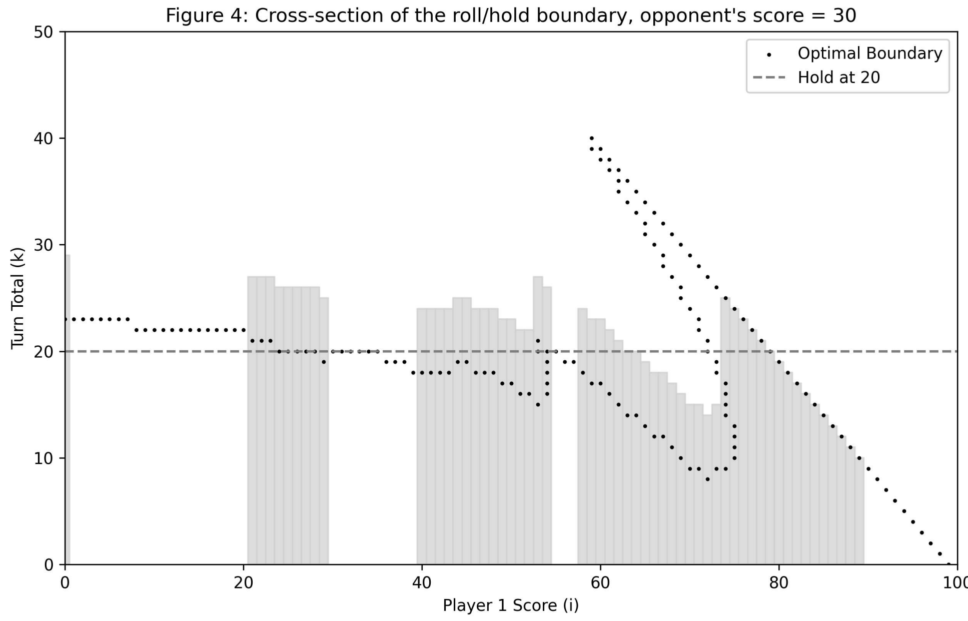
- In the graph the boundary surface is shown between the states where rolling again is the optimal policy (in grey) and those where the optimal policy is to hold (transparent)
- We use the optimal policy obtained from value iteration to plot all states where rolling is found to be the optimal policy.

Figure 3: 3D plot of roll/hold boundary for optimal Pig play policy



Roll/Hold Boundary plot (Figure 4)

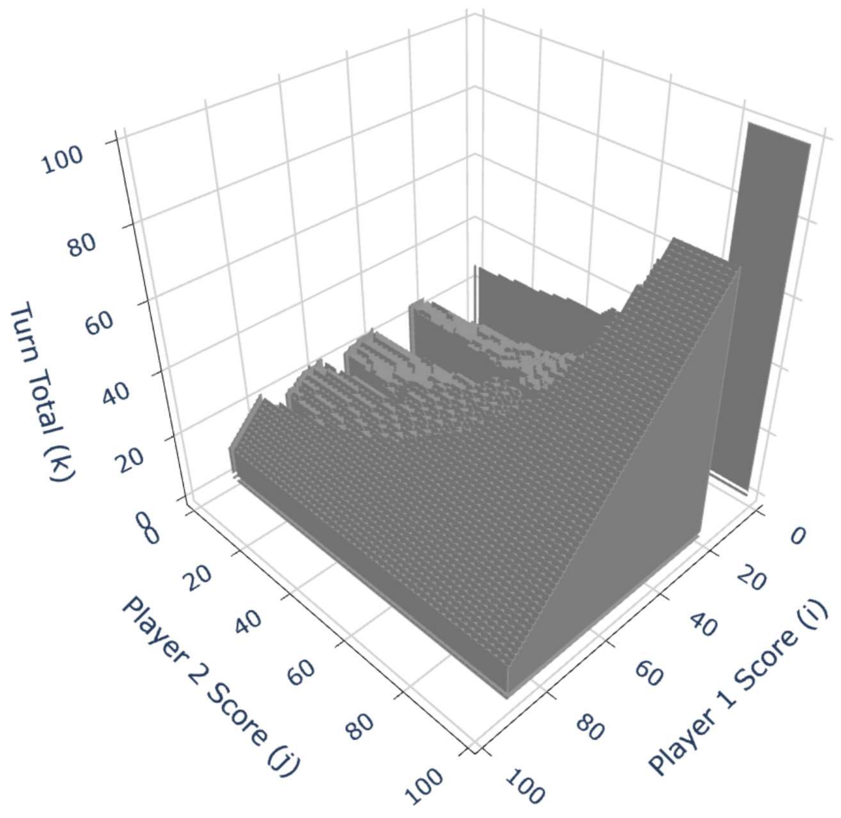
- Figure 4 is a cross-section of Figure 3, as well as Figure 5 (see next slide).
- Figure 4 is easier to compare to the graph created with our results, as it is only in two dimensions.🐷
- For lower values of player 1’s score they want to roll while they have a turn total of up to 23. This value decreases until a player score of around 72, at which point they wish to roll until they have won.



Reachable states (Figure 5)

- Figure 5 shows all the states that can be reached with a player following the optimal policy.
- There are 4 gaps in the reachable states, as in the paper.
- An optimal player with score of 0 will never hold before reaching a turn total less than 21 which can be seen in the graph.

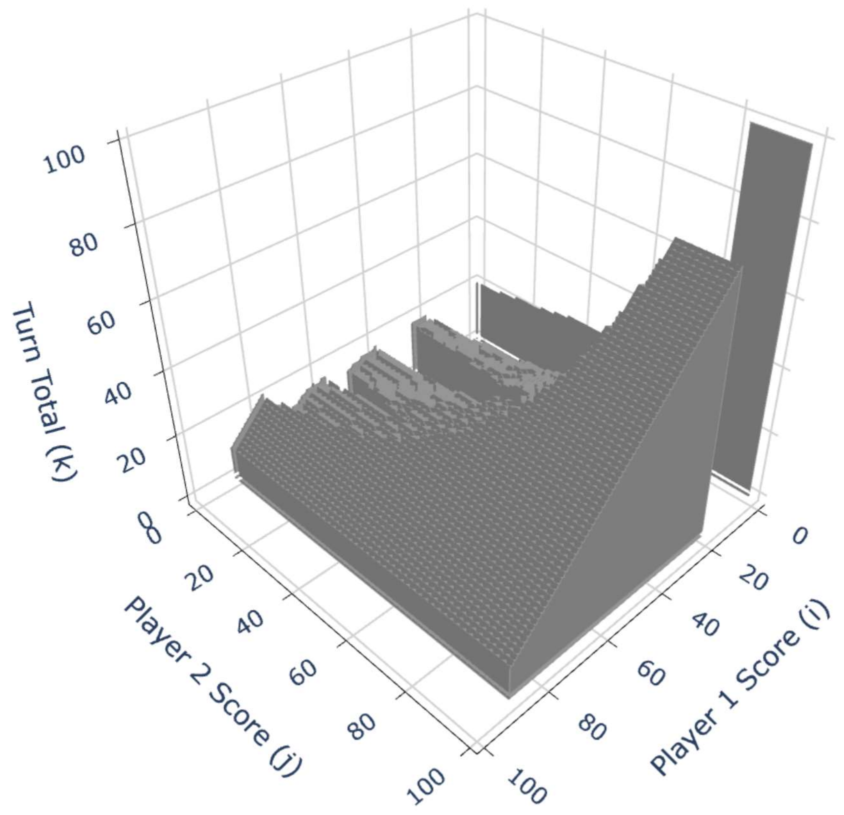
Figure 5: 3D plot of reachable states by an optimal Pig player



Reachable states with optimal rolling (Figure 6)

- Figure 6 is similar to Figure 3 but instead of showing the optimal policy for all states, it just shows the optimal policy for the states that are reachable as found in Figure 5.
- This is created by looking at the reachable states found for Figure 5, showing only those states at which the optimal policy is to roll.

Figure 6: 3D plot of reachable states where rolling is optimal



Comparing Figures 5 and 6

Figure 5: 3D plot of reachable states by an optimal P

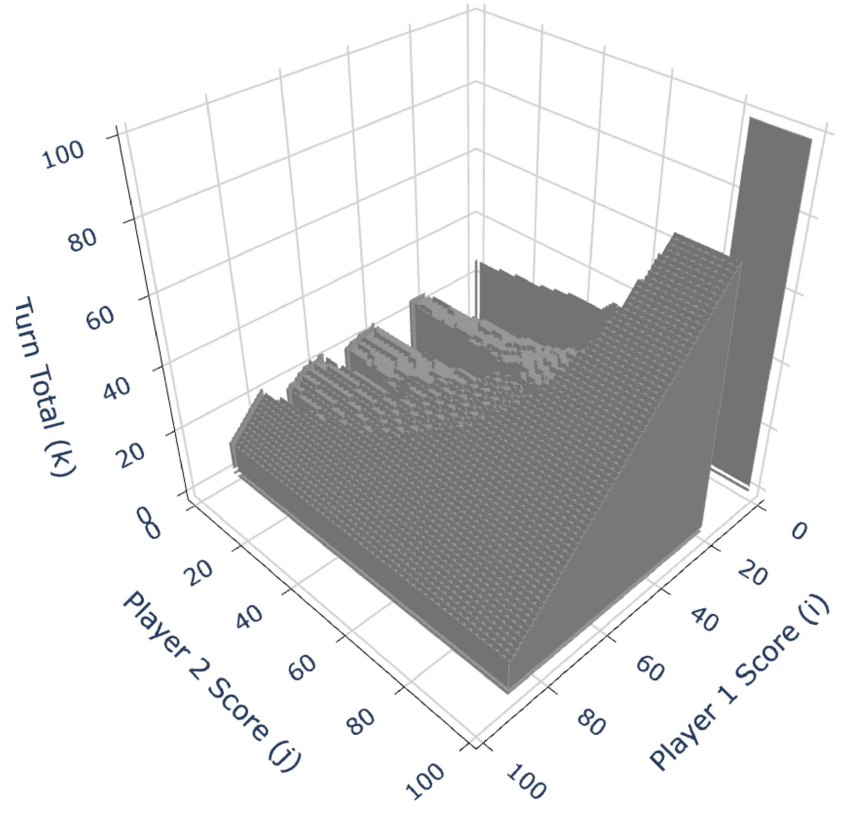
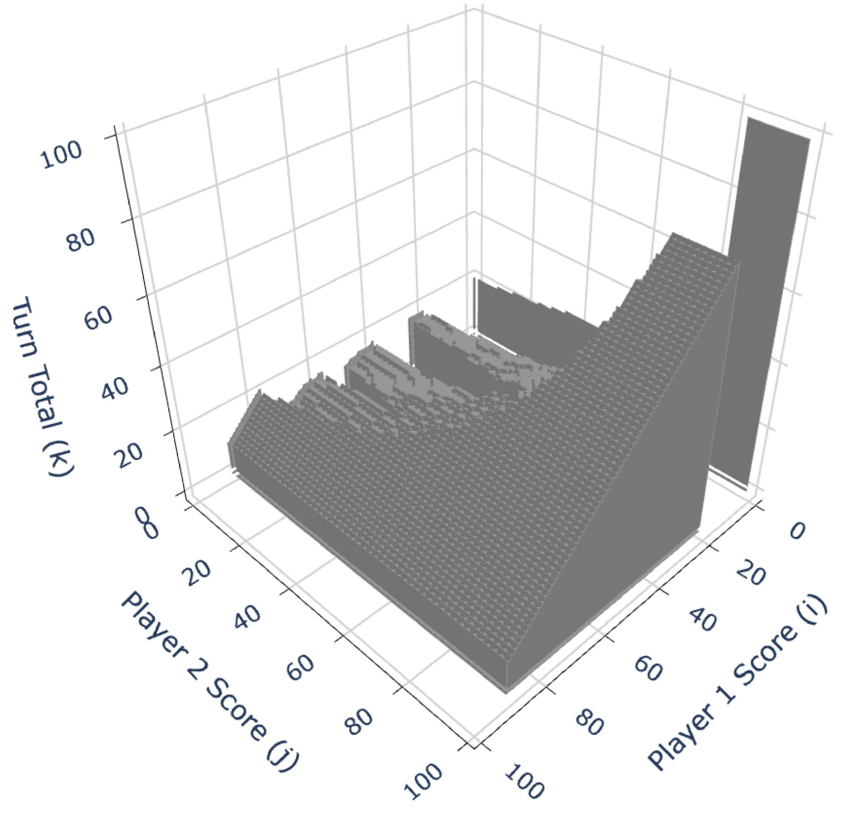


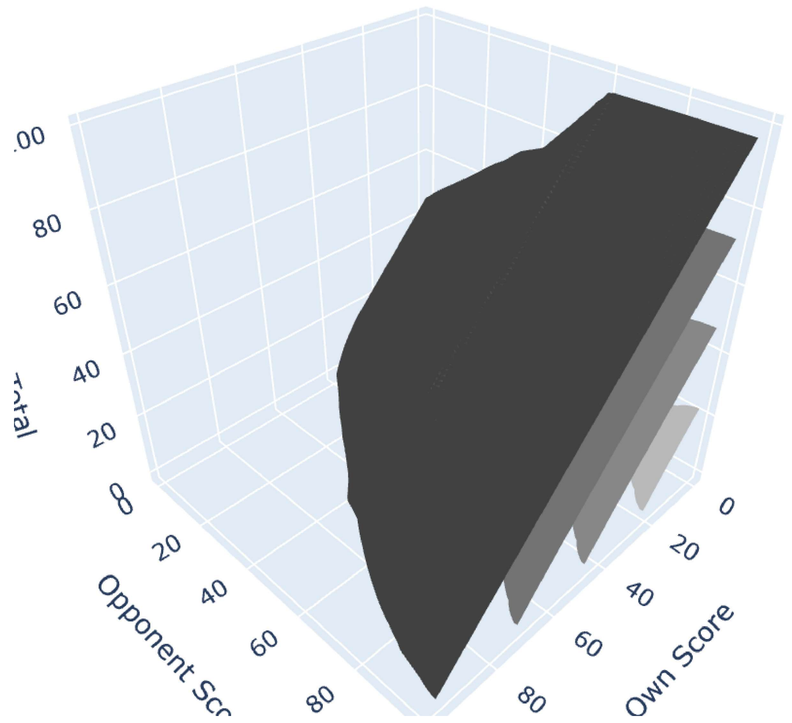
Figure 6: 3D plot of reachable states where rolling is



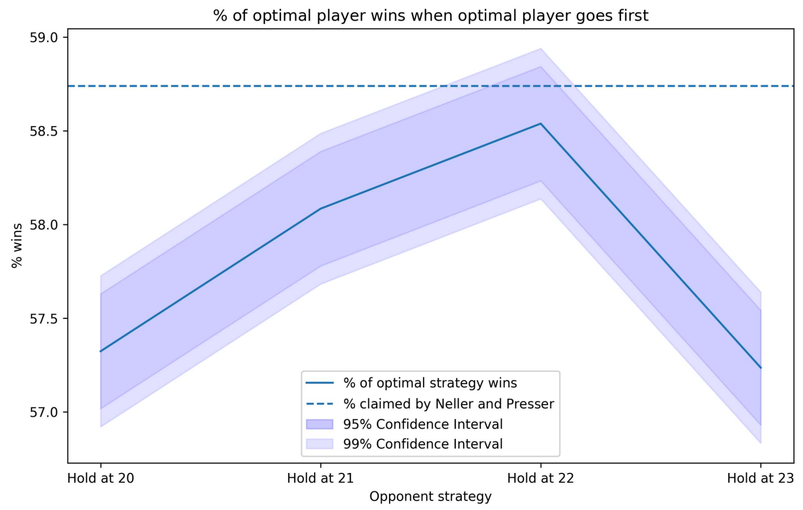
Winning probability contours (Figure 7)

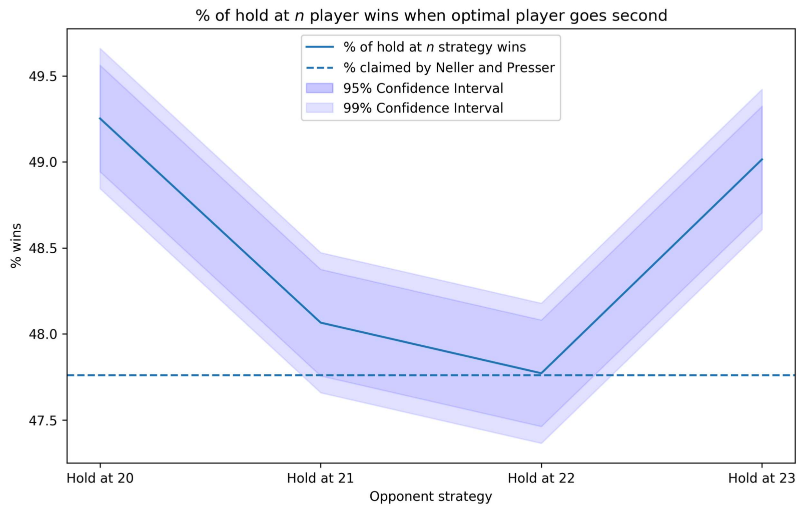
- Figure 7 shows contours of different winning probabilities computed from the optimal 🐖 policy.
- We draw a contour wherein the states have a win probability that is within a threshold t from the desired probability, e.g. in the interval $x\% \pm t$.
- The thresholds were manually selected for each of the 4 probabilities in order to yield smooth surfaces.

Figure 7: Win Probability Contours



Our simulated winning percentages differ





5 Rs of Neller and Presser (I)

Replicability refers to when an algorithm or solution can be recoded by someone else. 🐷

- The methodology as described in the paper is not replicable, in particular the description of the Value Iteration (V.I.) algorithm.
- We were able to replicate their figures and recoded their implementation from their description.
- Java code authors supplied for Piglet was re-runnable and repeatable.
- No code means not all Rs applicable.

5 Rs of Neller and Presser (II)

- V.I. algorithm in terms of tolerance Δ, though tolerance not specified in paper. We had to use maximum number of iterations instead, as in MDP literature [2].
- Though we reproduced the figures, the 3D views were not easily comparable. And there were no specific values to verify except $P_{(0,0,0)}$.
- The authors claimed that staged V.I. is more efficient but we found the opposite.🐷

Our work and the 5Rs (I)

- Our code is re-runnable by a third party, and we provide a GitHub package.
- Our results are repeatable, as we obtain the same optimal policy after convergence, and we specify iterations.
- Our solution is reproducible, robust to future versions of Python utilising sets, dictionaries and other basic data structures.
- Our results are replicable, we explain methodology in detail, and provide pseudocode.

Our work and the 5Rs (II)

- Our code is reusable for simple extensions of Pig such as goal-n-Pig, and routinely adaptable to different versions of Pig with equations (e.g. two 6-sided dice losing with snake eyes, changes in input equation).
- Further Piglet could be extended to have different goal numbers (twiglet, triglet, quiglet?). 🐷
- Did not explore other extensions mentioned in paper such as jeopardy approach games.

Conclusion

- We believe we have managed to reproduce all plots from the original paper.
- There were significant limitations to reproducibility of results in the original paper, such as ambiguous application of V.I., limited views of 3D plots and lack of methodology for quoted winning percentages.
- We believe we have adhered to the 5Rs in our work, providing a .pkl file of our optimal policy for external verification and open source repository for further work.

References

[1] Neller, Todd W. and Clifton G.M. Presser. (2004) “Optimal Play of the Dice Game Pig,” The UMAP Journal 25.1 , 25-47.

[2] Poole, D. L., & Mackworth, A. K. (2017). Artificial Intelligence: Foundations of Computational Agents (Second edition.). Cambridge University Press. https://doi.org/10.1017/9781108164085

Thank you!

Questions welcome. 🐷

How many pigs did you spot?