Three focused Artificial Intelligence assignments based on children's games

John Maraist

Computer Science Department
University of Wisconsin - La Crosse
1725 State Street, La Crosse, Wisconsin 54601
jmaraist@uwlax.edu

Abstract

We present assignments based on three children's games for an entry-level artificial intelligence class, each focused on a specific topic of the standard introductory algorithmic AI curriculum. *Rushhour* asks students to focus on the design and comparison of heuristics for an A* search problem. *Chocolate Fix* is a puzzle game well-suited to constraint search, both in a simple backtracking implementation and with consistency propagation. Finally, *Tsuro* is a multi-player strategy game suitable for both adversarial and Monte Carlo search-based agents. The scope of effort required of students for all three assignments can be adjusted by providing a greater or lesser amount of scaffolding code.

1 Introduction

Game agents are appealing assignments in the artificial intelligence class: Students see them as fun, and good solutions require a solid understanding of curriculum topics. Various games have consistently motivated advanced research over the last seventy years, notably among them chess, Go, and poker [1, 2]. In this paper we describe three children's games well-suited for automated play by agents using algorithms typically studied in the first-semester artificial intelligence class. Each assignment allows a focus on specific common topics: *Rushhour* for the design and comparison of heuristics for A* search; *Chocolate Fix* for constraint satisfaction problems (CSPs), and *Tsuro* for both adversarial and Monte Carlo search. The scope of effort required of students for all three assignments can be adjusted by providing a greater or lesser amount of scaffolding code. For each of these games we report our approach to constructing an assignment around that game, discuss some observations of the students' experiences with the assignments, and sketch some possible future variations. All three games we present meet the bar of requiring thoughtful student work, but are also sufficiently simple as to be quickly understandable by students who less avid board game players. The multiplayer game in particular can admit tournament play among submitted agents, a friendly competition which can further motivate students.

We assigned these games in an elective class focusing on search and probabilistic algorithms in artificial intelligence. The class combined undergraduate and graduate sections, both covering the same major topics but with some additional material and higher assessment standards in the latter. Our department offers separate electives on Monte Carlo and other simulation-based techniques and on machine learning, so these topics are not addressed in the artificial intelligence elective. Our core coursework currently uses Java, and most assignments in our AI class did the same. Materials from these assignments, including all code distributed to students, is available for download [4].

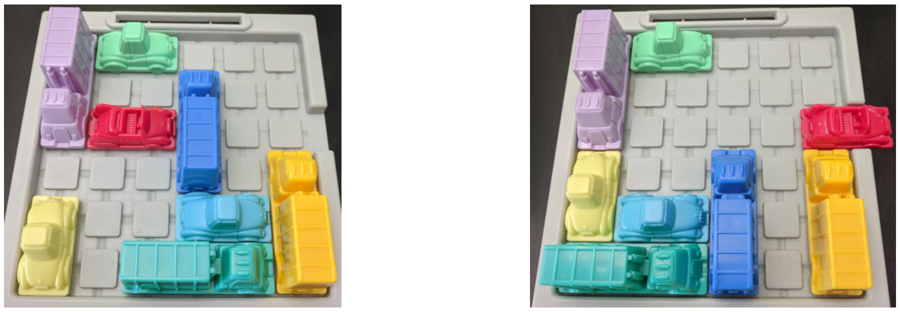
2 Design of heuristics for A* search: *Rushhour*

Rushhour is a children's puzzle based on sliding blocks back-and-forth on a grid. One of the blocks represents the family car, which is stuck in traffic: one or more blocks sit in between the family car and an exit at the end of the family car's row of the grid. Solving a puzzle configuration requires moving cars within each one's row or column so that the family car is moved to the exit. The blocks must remain flat on the board, and may not be placed on top of each other, nor to "jump" over each other. Only the family car is able to exit; the other vehicles must remain in the frame. Figure 1 shows an opening puzzle state, and its solution.

Rushhour as an assignment

We used *Rushhour* to explore the design of heuristics for A* search, and in particular to:

- Construct different heuristics for a single problem;
- Describe the techniques used to derive their heuristics from a full solution;
- Discuss the properties of their heuristics, including whether each is admissible and/or consistent, and the computational complexity of each; and



(a) Stuck in traffic. (b) Driving free!

Figure 1: *Rushhour* boards. Image (a) shows an opening position, with the red family car stuck in traffic. In the solution position shown in Image (b), the family car can drive through the exit.

- Experimentally compare their heuristics to each other, and to simple breadth-first search (BFS).

This assignment contained both programming and written components. To focus the students' efforts on the heuristics themselves, we gave the students a substantial amount of code, both for a generic A* implementation, and for a model of the *Rushhour* game and several boards. We expected the students to program the heuristics themselves, but since they had already completed an assignment on implementing the core of A*, we provided an A* implementation parameterized over the heuristic. We asked them to consider at least three distinct heuristics. As an example we provided a trivial heuristic estimating zero additional cost for all moves (so essentially equivalent to BFS). We further asked the students to implement the composite heuristic described in their text which uses the point-wise maximum over their individual heuristics.

We asked the students to consider the two techniques discussed in the text for deriving admissible heuristics [3, Sec. 3.6], solving relaxed problems and solving subproblems, and describe how each of their solutions fits into that taxonomy. Admissibility and consistency being essential properties of heuristics for effective A* deployments, the students gave informal "proofs" of how each of their heuristics satisfied these properties. Although we did ask the students to discuss their implementations' computational complexity, it was not a primary focus since not all had completed our algorithms elective. Finally, we asked the students to use the data collected by the provided A* implementations to calculate the effective branching factor (EBF) of their algorithms on each of the several sample configurations, examine the stability of the EBF across different initial boards, and to use these results to discuss the quality of their heuristics.

This assignment was generally well-received by the students, who mostly submitted well-completed implementations. We did notice more weakness than we expected in their written use of technical definitions of heuristic properties. This weakness manifested in particular when justifying the admissibility and consistency of their heuristics; in some cases the answer was a simple unjustified declaration that the properties hold. In the future we would better scaffold the awareness of the technical details of these definitions with low-stakes quizzes before, or soon after, releasing the assignment. We would also more explicitly point to examples in the text, or to provided answers on past/similar projects, to show the form of an appropriate response. One specific issue with EBF calculations is the simplified formula which, although not discussed in our adopted text [3], is available online. In the future we would either give them this formula up front, or else explicitly mention it as an unacceptable oversimplification.

Another possible *Rushhour*-based assignment which we did not explore this semester would be for local search algorithms, to see if a process like simulated annealing would be able to find puzzle solutions.

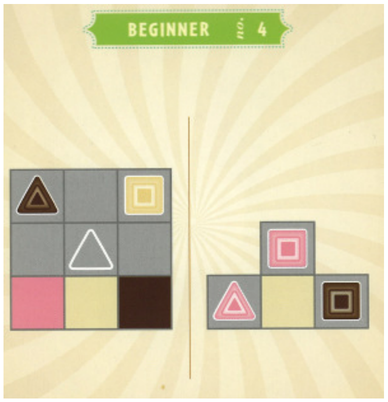
3 A domain for constraint satisfaction search: *Chocolate Fix*

Chocolate Fix is a puzzle game in which the player finds the correct arrangement of pieces on a square grid. The pieces represent candies, each with a unique combination of a *shape* and a *color*. The conceit of the game is that the player is assembling candy sampler boxes, where the particular arrangement of the candies for each box must be inferred from hints specific to that "order." For a *Chocolate Fix* puzzle of size n:

1. There are n different shapes and n different colors.
2. There are n^2 pieces, each with a different shape/color combination.

Each puzzle has a set of clues showing how the grid, or some part of the grid, must be arranged. The physical game (and the examples which we present here) are on a 3×3 grid, but we expected students' solutions to address the general case of $n \times n$ puzzles.

Consider the following problem:

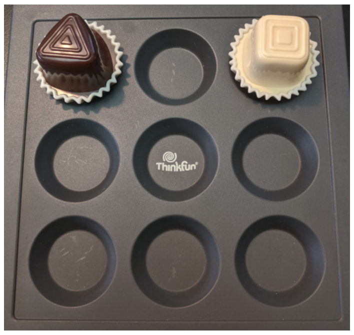


This problem has two clues. The first (left-hand) clue spans the entire 3×3 grid. There are several requirements which this clue requires of a solution:

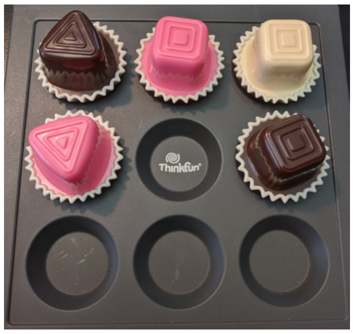
- The upper-left corner must have the brown triangle, and the upper-right corner must have the white square.
- It gives a partial hint for the center square; a triangular piece will go there but the color is not disclosed.
- In the bottom row, the clue requires specific colors but does not insist on a particular shape. The clue requires red in the bottom left, white in the bottom middle, and brown in the bottom right.

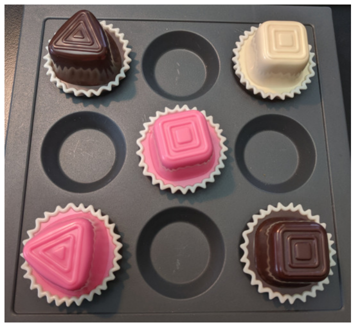
- It says nothing about three squares: the middle of the top row, the middle of the left column, and the middle of the right column.

So after considering the first clue, we can place two pieces with certainty:

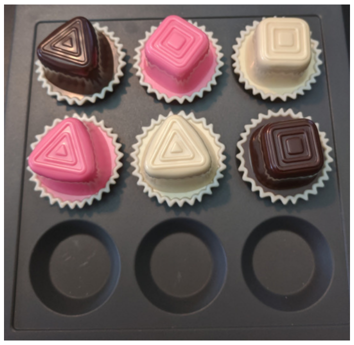


However the first clue alone does not give us enough information to place other pieces with certainty. The second (right-hand) clue fits within a 2×3 rectangle, which means it could apply to *either* the top two rows, *or* to the bottom two rows:





But the latter case, when we apply the clue to the two bottom rows, contradicts the first clue's requirement that a triangular piece should fit in the middle. Therefore we can rule out that placement for the second clue. Proceeding with five pieces in place, we can see there remains only one triangular piece for the center square,



We have one piece of each color remaining, and the bottom row of the first clue gives us their final placements.

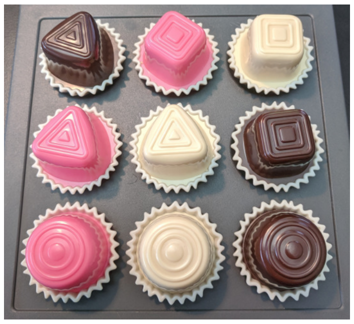


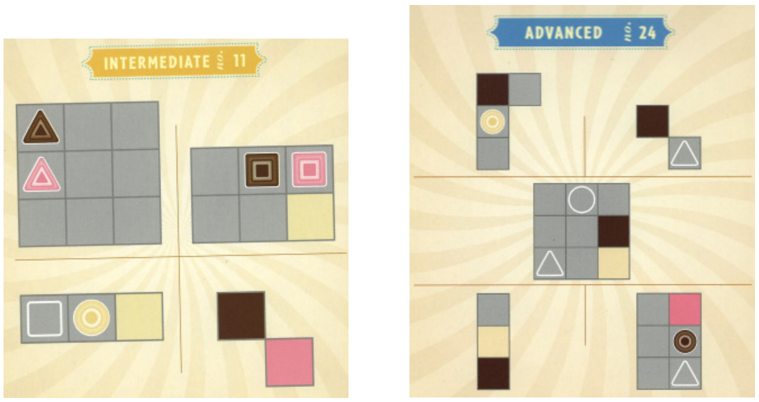
Figure 2 shows two more difficult problems, Candy Boxes (a) and (b). Box (a) has more clues than the simple problem, but as with the initial problem's right-hand clue, some of them might be applicable to many places. For example, the fourth clue (bottom-right) says that one of the brown pieces will have a red piece below and to its right, but there are four possible positions where we could apply this requirement. Note also that, like the first clue of the simple problem, many of the clues for Box (b) are silent as to some of the squares in their grids. The upper-right clue calls for a slot containing a triangle (of unspecified color) above and to the right of some other slot, but places no restrictions on the content of that other slot. Finally, the most advanced example Candy Box (b) contains more clues than the others, but most governing only some fragment of the overall grid, and most giving only a partial requirement for the squares of its fragment.

Chocolate Fix as an assignment

We assigned *Chocolate Fix* as one question of an open-book take-home final exam asking students to design (but not implement) an agent for *Chocolate Fix* puzzles using the techniques we studied over the semester, and to outline the steps the agent would take for a small number of specific examples, at least one of which would have been labeled "advanced." We asked the students to analyze their solutions in terms of the metrics we studied for the techniques they chose, and to give some discussion of their agent's computational complexity.

It is fairly easy to see that *Chocolate Fix* puzzles are best modeled as CSPs. Moreover a correct solution must address a number of subtle issues which make this game well-suited to a long-form exam question. *Chocolate Fix* puzzles translate most naturally to CSPs with two distinct variable types, one to assign pieces to slots, another to anchor partial-grid clues in the overall result grid. The interactions among these constraints give a meaningful role to consistency checking for pruning unassigned variables' domains. The problems are also rich enough to justify experimenting with the impact of variable and value orderings on search efficiency. So while a *Chocolate Fix* agent does not necessarily require more than a very simple CSP encoding, the assignment will allow the student with greater mastery of more advanced techniques to demonstrate their progress.

In future semesters we would consider presenting this game as an implementation rather than a design project. While a full implementation of a CSP solver for *Chocolate Fix* would be a very reasonable assignment, as with *Rushhour* we could provide scaffolding code for a generic CSP algorithm to allow students to focus more closely on experimentation with optimizations to the search process.



(a) (b)

Figure 2: Two more advanced problems from *Chocolate Fix*.

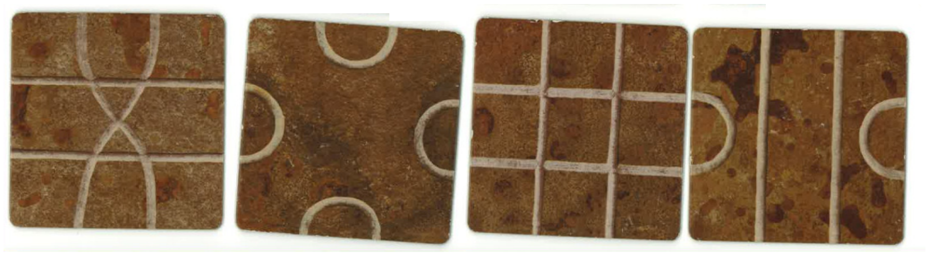


Figure 3: *Tsuro* tiles.

4 Adversarial search in a multi-player game: *Tsuro*

Tsuro is a multiplayer game in which players build paths and advance along them by placing tiles on a board. Paths are printed on each tile; Figure 3 shows some example tiles. The tiles are all the same size. Paths run to and from the same points on the edges of each tile, but different tiles connect these points in different ways. When we place a new tile next to tiles already on the grid, paths traced by the old tiles are extended with the new tile.

Each player is represented by a small stone, which will always be at the edge of one of the paths. At the beginning of the game, a dealer shuffles the deck of tiles, and deals three tiles to each player. Each player's turn consists of two steps: first placing one tile in the empty grid space next to their stone, and then drawing a new tile from the deck (if any remain). When the player places a tile, it extends the path they are following, and the player must immediately move their stone to the new endpoint of the path. For example, Figure 4 shows the possible opening moves for Player Green: The leftmost of the pictures shows Green in their starting position. The stone sits on a small white dash, Green's tiny initial path; similar possible starting paths are visible above and below Green. Each player starts on a different starting path. In Green's first turn, they will select one tile from their hand, place it on the empty grid space next to the Green stone, and move their stone through the new tile along the path. The middle picture shows the outcome of one possible way of

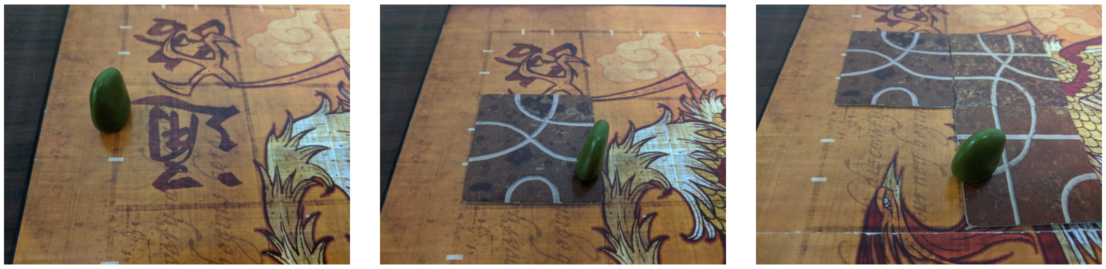

(a) (b) (c)

Figure 4: Opening moves for Player Green.

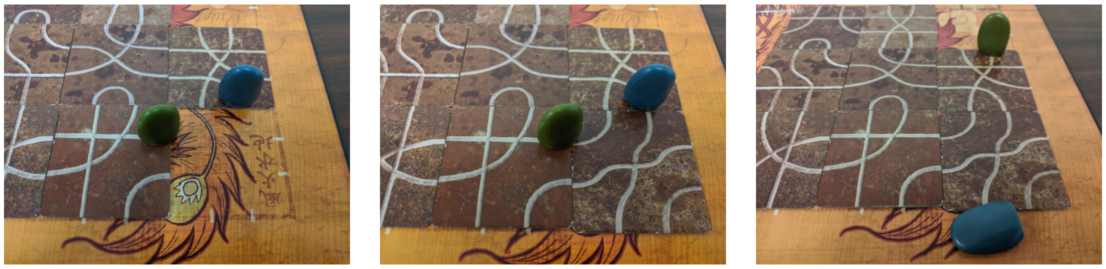

(a) (b) (c)

Figure 5: Eliminating another player when their path run off the edge of the board.

placing a tile by Green. Turn after turn, Green's stone advances along its path as it grows. The rightmost picture above shows Player Green's position after three turns.

A player must always place a tile in the empty space adjacent to where their stone sits at the end of its path. However, the player may choose to place any of the tiles in their hand, and may orient the tile as they place it. If a newly-placed tile leads a player's path to the edge of the grid, then that player is eliminated from the game. So it important to choose tiles and their rotation carefully. When all but one player has been eliminated, the last remaining player wins.

The moves in Figure 4 have only one player in view, but all players' stones sit at their own positions on the same board. So it is possible — especially later in the game — that one newly placed tile might extend the paths of *several* players, not just the player who places the tile. Figure 5.a shows a position where Players Green and Blue are in such a situation. When either player adds a tile to the empty corner space, both players' paths will grow, and each stone must move forward to the end of its path. Let us assume that Green plays first, and makes the very wise choice of play shown in Figure 5.b. Then we see in Figure 5.c how Green advances safely, but Blue's path leads to the edge of the grid, so Blue is eliminated.

When a player is eliminated, any tiles in their hand are returned to the draw pile. If the draw pile had been empty, and other players have been waiting to draw tiles, then these players draw from the refreshed pile in the order they had been waiting. When all players but one have been eliminated, the survivor wins the game. If all remaining players are eliminated by the same tile, then these players tie for the win. If more than one player is still in the game after all tiles have been played, then again these players tie for the win.

Tsuro as an assignment

We assigned *Tsuro* as a project on adversarial search, asking the students to incorporate some form of the Minimax algorithm to drive their agents' decisions. For an end-of-semester amusement we conducted a tournament of the students' agents, which many found motivating and enjoyable even though the tournament aspect was not graded. As for *Rushhour* we provided a common model of game elements. To facilitate multi-agent play both for the tournament and for testing of agent versions, we also provided a game-running module, plus sample agents whose behavior was largely random. The game runner imposed time limits on agent initialization, processing of newly dealt tiles, and tile play decisions, enforced with a forfeit for exceeding time limits. The size of the search tree required the students to implement some form of evaluator for non-ending positions, and we encouraged them to consider strategies such as varying search depth to reflect the significantly larger branching factor at the beginning of games, identifying subproblems excluding some players or board areas, and some form of probabilistic reasoning based on the known unplayed tiles (agents had access to the full deck of available tiles at the beginning of a game before shuffling, but not to the subsequent assignment of tiles to other agents). The former two techniques turned out to be generally fruitful approaches; the latter, less so. Their agents were also expected to cope with any size of game board. The physical game is a 6×6 square of tile slots; for some rounds of the end-of-semester tournament we used a larger board.

5 Conclusion and resources

A common aspect of all three of the games which we consider here is their minimalism. Simple game mechanics allow the relevant AI technique to take a prominent role in an exercise, and the limited role for random elements makes the link between an algorithm and the game play clearer. As such the games are well-suited both for implementation and for discussion on exams or in presentations linking game elements to AI algorithms and techniques.

We finish with two further resources for instructors seeking ideas for games to be the basis of class assignments. First, the *Spiel des Jahres* [5] is an annual series of three awards by a group of German games critics and publishers recognizing excellence in game design for over forty years. Although many of the games will not be suitable for an elementary AI class (such as games based on word play, or requiring physical actions), there is nonetheless a broad selection of good assignment candidates among the past winners, nominees, and recommended games. Second, until recently we were not aware that many public libraries keep an inventory of family- and child-friendly games available for checkout. We would have tried many fewer games without this community asset!

Acknowledgments. It can be challenging to write clear instructions to a familiar game for readers who may not have immediate access to the game and its pieces: my colleagues Elliott Forbes, Thomas B. Gendreau and Samantha Foley provided detailed feedback on the readability of the original assignments described in this paper. Jason Sauppe gave feedback on a draft of this paper. And most of all, I am grateful to Jill Maraist and Marcel Maraist for happy hours of participatory research on real-time strategies and tactics for these and many other games.

References

[1] Noam Brown and Tuomas Sandholm. Superhuman AI for multiplayer poker. *Science*, 365(6456):885–890, 2019.

[2] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[3] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2010.

[4] Github repository https://github.com/jphmrst/AIGameAssignments, 2020. This repository is private, but instructors are welcome to request access.

[5] *Spiel des Jahres* web site. https://www.spiel-des-jahres.de/spiele/.