Senior Thesis in Mathematics

Creating Intelligence: A Survey of Artificial Intelligence Approaches to Solving Puzzles

Author:
Neel Adhiraj Kumar

Advisor:
Dr. Vin de Silva

Submitted to Pomona College in Partial Fulfillment
of the Degree of Bachelor of Arts

December 6, 2017

Abstract

In this paper we investigate various approaches to creating Artificial Intelligence by desiging algorithms to play the game 2048, and consider what it can teach us about the human mind.

Contents

Chapter 1

Introduction

Artificial Intelligence has the twin goals of understanding and simulating intelligence. In recent years, due to a combination of new techniques, profilerate data, and faster computation, there has been major progress towards this goal (cite some recent major advances?). However, we are only just beginning to understand the human mind, and certainly do not yet have a clear notion of the intelligence we are trying to reverse-engineer. Pulitzer-prize winning cognitive scientist Douglas Hofstadter, in his book 'Fluid Concepts and Creative Analogies', claims pattern finding is at the core of intelligence [**?**]. He tries to create AI that, instead of relying on computational power and a vast library of expert level domain specific knowledge, has abstract, analogy making skills. These skills rely on ideas such as symmetry, elegance, consistency, and simplicity, ideas often associated with mathematics. In his view intelligence is intimately linked to the ability to connect ideas and detect patterns in a flexible, creative manner. This is something humans do all the time, with remarkable effectiveness and flexibility.

We are a long way from creating or understanding anything like human intelligence, which appropriately responds to an enormous variety of circumstances, and effectively adapts existing ideas to new, unencountered situations. However, one domain where human intelligence is certainly demonstrable, and perhaps more easily understandable, is games. Games provide a limited and clearly defined environment where we can observe, emulate, and evaluate (aspects of) intelligence. There tends to be a natural, in-built measure of performance - usually a game score, or a winner - and depending on the game, success may require what we might ordinarily call intelligence. Developing AI for games involves the classic AI topics of knowledge representation, searching, and learning and hence, games provide a useful framework to test, build, and advance AI.

This thesis investigates different AI approaches to the single-player sliding-block puzzle game, 2048. Success in the game seems to require the strategic detection and manipulation of patterns on the board towards a set of goals. The number of possible board states is too high for brute-force searches, making the solution non-trivial, and human-players certainly do not play by calculating all possibilites. Hence, developing AI for the game not only involves modelling and simulating abstract pattern finding skill and to an extent, mathematical ability (itself a component of intelligence), but also allows us to try out a variety of AI techniques.

The first chapter describes the mechanics of the 2048 game, and discusses the game's value as a framework for investigating AI approaches. The next chapter provides an overview of

various AI approaches, discussing their goals, advantages, and disadvantages. The following chapters analyse these approaches in detail. First we consider rule-based approaches, where the AI is an attempt to model a successful human player using heuristics and programmed "intuitions". Next, we consider machine learning approaches, where the AI learns how to play the game without previous knowledge. These include reinforcement learning and evolutionary algorithms, which also provide insight into human intelligence by demonstrating how it may have arisen.

Chapter 2

2048

2.1 Rules

2048 is a single-player sliding block puzzle. The player swipes the board up, left, down, or right to slide the numbered tiles on the board around. Tiles combine if one tile is slid towards another with an equal value, in which case they result in a single tile with double the value. The objective is to reach the 2048 tile, though the game continues after it is attained, and skilled players usually try to reach bigger and bigger tiles. There is also a game score that increases every time tiles are combined (the increase is equal to the value of the tiles being combined).

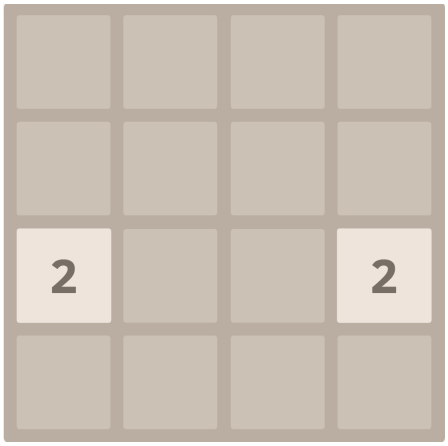

Figure 2.1: Example of a starting game board

Each valid move by the player leads to some movement on the board. At the end of the move (when all tiles have been moved and combined), a 2 appears on a randomly chosen empty tile on the board, though there is a 0.1 probability of it being a 4.

The game ends when there are no more possible moves.

Talk about subtleties of merge function

Talk about theoretical maximum tile possible on the board (https://stackoverflow.com/questions/22803
the-game-2048-what-is-the-biggest-theoretical-tile)

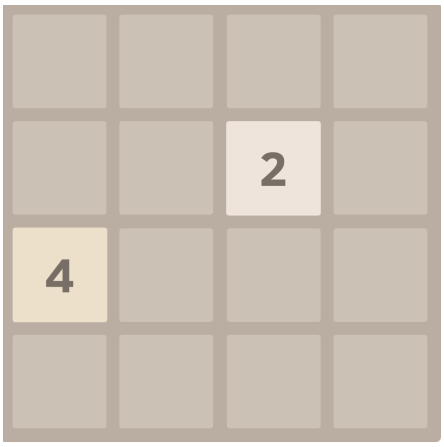

Figure 2.2: The game board from Figure 1.1 after a left swipe

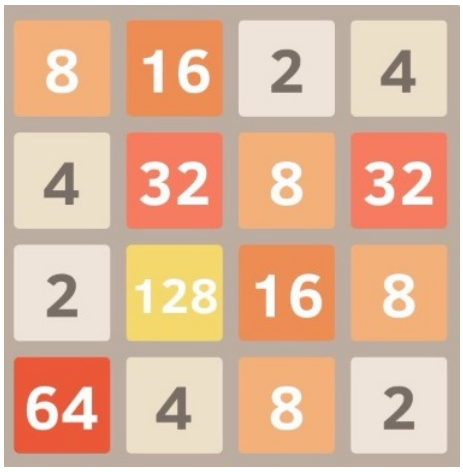

Figure 2.3: Example of a 'game over' board with no moves left

2.2 Why are we interested in this?

A human playing the 2048 game is finding and manipulating patterns on the board to his advantage. Since there are potentially many possible locations for a tile spawn at the end of a move, it is hard for human players to think many moves ahead, and most successful players usually try to mantain some sort of order or pattern on the board to ease tile combination throughout the game, and deal with "unlucky" tile spawns as swiftly as possible, without ruining the set up of the rest of the board.

Though this is certainly not the only strategy for playing 2048, it is somewhat prototypical, and any approach requires a strategic detection and manipulation of patterns towards a set goal. This ability is ultimately a (very) simple instance of mathematical ability more generally. Modelling and simulating it is a small step towards both understanding human intelligence and building AI that compares to it.

Moreover, this problem involves classic AI topics such as knowledge representation, searching, and learning. We have an obvious way to evaluate game performance - we simply look at the value of the biggest tile on the board at the end of the game (or alternatively,

the game score). Hence, 2048 provides an effective framework for building and testing many different techniques in AI.

2.3 Implementation

This section describes the implementation of the game in Python. I used starter code from [**?**], and borrowed ideas from [**?**].

Chapter 3

Past Work

mainly from stackoverflow thread (https://stackoverflow.com/questions/22342854/what-is-the-optimal-algorithm-for-the-game-2048):

nneonneo's expectimax (using aspects of is implementation) –¿ powerful, state of the art; brute-computation, little insight about intelligence benchmark

minimax search with alphabeta pruning using heuristics: monotonicity and smoothness

no hard coded intelligence: Monte Carlo search (examined in detail in this thesis)

GOFAI (top-down decision rules)

expectimax search learned using reinforcement learning (from scratch, without human expertise)

we explore: monte carlo; cognitive model; reinforcement learning

Chapter 4

Algorithms for 2048

In this chapter we describe algorithms of increasing complexity to play the 2048 game. Later algorithms may refer to previous ones for specific cases. For simplicity, I will focus on the case where we have a 4×4 board of tiles (the standard game board).

The first section describes simple,"unintelligent" algorithms that may be used in more complex algorithms and will provide a performance benchmark for our AIs.

Then I give a detailed description of the AI algorithms. In each case, the game ends when no more moves are possible. If there is only one move possible, we must make that move. Hence, the algorithms below only come into action if there is more than one possible move.

4.1 Random

At each turn:

Make a list of possible moves.

Chooses a move (uniformly) randomly from the list.

4.2 Sequential

Chooses moves in the following sequence:

Right, Down, Left, Up.

If any move is impossible, the sequence simply continues in the above order.

4.3 Randomly Alternate

Alternates between randomly choosing between up and down, and randomly choosing between left and right. The program only chooses from possible moves on each axis.

In other words, if we number each move, on odd numbered moves, we randomly choose between up and down, and on even numbered moves, we randomly choose between left and

right. If the chosen move is impossible, the program simply makes the opposite direction move and then alternates to the other axis of movement.

4.4 Always Up

Always tries to move up.

If not possible, it moves left.

If that is not possible too, it moves right.

If that is not possible either, it moves down.

4.5 AI5

This is our first program that has some logic to its moves, even if only for limited cases. The overall strategy is to try and keep the biggest tile in the corner, a strategy most successful players of the game seem to employ.

First, make a list of all the possible moves.

If only one move is possible, make that move.

Otherwise, look at all the highest valued tiles on the board.

If we do not have a unique maximum, choose randomly between possible moves.

If we have a unique maximum:

Case 1: The maximum is in a corner.
Each corner tile has 2 moves that it "prefers" since it keeps the tile in the corner. For instance, if the maximum tile is on the top left corner $(0, 0)$, we prefer to move up and left.

[add an example with a game board here]

We make a list of the preferred and possible moves.

If the list is non-empty, we pick randomly from it.

If it is empty, we choose randomly from all possible moves.

Case 2: The maximum is on an edge (but not on a corner).
Similar to the case above, we build a preference vector that we randomly choose from. However, in this case our preferences are not equally strong, and hence are weighted differently.

If the maximum is on an edge, and we can move in the same direction, we add a weight of 1 to that move (for instance, if the maximum tile is on the left edge and we can move left, moving left has weight 1).

Then we look at the other axis of movement. We move towards the nearest corner with weight 4, and move in the opposite direction, to the second nearest corner, with weight 1. Hence, if our maximum is in the $(0, 1)$ position, we move up with weight 4 and down with weight 1.

[add an example with a game board here]

We randomly sample our move from the list of all moves with the above weights (0 if unspecified).

Case 3: The maximum is one of the 4 center tiles.
Like the above case, we build a weighted preference vector. We move towards the nearest

corner with weight 4 and away from it with weight 1. Hence, if the maximum is on the top left center tile, we move up and left with weight 4 each, and down and right with weight 1 each.

4.6 AI6

This is an extension of AI5 to the case of multiple maxima. It employs the same strategy of trying to keep the maxima in corners, but now accounts for the preferences of each maximum.

Again, we first, make a list of all the possible moves. If only one move is possible, make that move.

Otherwise,

We construct a list of all the maxima on the board, along with their location.

Then we build 2 lists.

The first list is a list of preferred moves, and has higher priority over the second, which is a weighted list of (less-preferred) moves.

We construct this list by looking at each maximum, similar to AI5. For each maximum:

Case 1: The maximum is in a corner.
Each corner tile has 2 moves that it "prefers" since it keeps the tile in the corner. For instance, if the maximum tile is on the top left corner $(0, 0)$, we prefer to move up and left.

Add the tile's preferred moves to the list of preferred moves.

Case 2: The maximum is on an edge (but not on a corner) and our preferred moves list is empty.
Here, we focus on the second list of weighted preferences (with lower priority to the list in the above case). If the maximum is on an edge, and we can move in the same direction, we add a weight of 1 to that move (i.e. if on the left edge and we can move left, moving left has weight 1)

Then we look at the other axis of movement. We move towards the nearest corner with weight 4, and move in the opposite direction, to the second nearest corner, with weight 1. Hence, if our maximum is in the $(0, 1)$ position, we move up with weight 4 and down with weight 1.

Case 3: The maximum is one of the 4 center tiles and our preferred moves list is empty
In this case too we continue building the second list of weighted preferences.

We add weight 2 to the moves towards the nearest corner and weight 1 to the moves away from it. Hence, if the maximum is on the top left center tile, we add weight 2 each to the moves up and left, and 1 each to the moves down and right.

Finally, if our list of preferred moves is non-empty, we pick randomly from it. Note that If a given move is preferred by 2 corner tiles, it has twice the weight in this random selection.

If the preferred moves list is empty, we look at our second list, and randomly sample our move with the weights as calculated above (0 if unspecified).

If both lists are empty, we randomly choose a possible move.

4.7 AI7 (Greedy)

A greedy algorithm that tries to maximise the score increase from each move.

If we have more than one move possible:

We save the current game board and score.

Then we try out each possible move (on the stored game board), and record the increase in score from the move.

We revert back to our stored game board and make the move that most increased the score.

4.8 AI8 (Greedy AI6)

A greedy modification of AI6.

We use the same algorithm as AI6 to construct a list of preferred moves and a lower-priority weighted list of moves.

Then, instead of randomly choosing from our preferred moves, we greedily choose the preferred move that most increases our score.

If there is no preferred move, we randomly sample from the lower-priority weighted list of moves, and make that move.

If we have nothing in this list either, we simply make a random possible move.

4.9 AI9 (Greedier AI6)

A greedy modification of AI6. It uses greediness in more cases than AI8.

We use the same algorithm as AI6 to construct a list of preferred moves for corner maxima (we do not construct a lower-priority weighted list of moves for other maxima).

Then, we greedily choose the preferred move that most increases our score.

If there is no preferred move, we greedily choose the possible move that most increases our score.

4.10 AI10 (Greedy for empty tiles)

A greedy algorithm that tries to maximise the number of empty tiles on the board. The logic here is that (all other factors equal) the more empty tiles there are on the board the further away we are from getting stuck and losing the game.

If we have more than one move possible:

We save the current game board.

Then we try out each possible move (on the stored game board), and record the number of empty tiles on the board at the end of it.

We revert back to our stored game board and make the move that resulted in the most empty tiles on the board.

4.11 AI11 (Monte Carlo)

An algorithm that chooses moves by using Monte Carlo to explore the game space resulting from each move.

For each move:

Store a copy of the game board and score (we can call this the original board).

Make a move in one of the possible directions.

After each such move:

Store a copy of the current game board and score. Randomly play the game till we lose. Record the score.

We then revert back to the stored current game board, and randomly play again till we lose. We do this n times.

Once we have done this for each possible starting move, we go back to our original game board and choose the move whose average final score was the highest (at the end of random play).

Essentially, we are using "random play to the end n times" as a way to evaluate the strength of a given board position.

We do this evaluation for each possible move, and make the move that leads to the highest score on average.

Chapter 5

Future Work

The point of this
    Extensions to this
    Where to go from here