Assignment 2 Report

Brian Thompson

Ricky Bernstein

Vince Capodanno

Part 1: CSP: Word Search

We tried various options for what to choose as our variables and the domain of values to fill those variables. At first we chose our variables to be the coordinates and the domain to be the possible words that start at this position (the domain size would be twice the size of words due to the fact that they can be horizontal or vertical). The problem with this was that not every coordinate had to have a word starting in it. The next attempt was to still use the coordinates as variables but now have the values be words going through the coordinate. So here, we had to try all the possible ways that a word could go through a certain coordinate, and keep recursing if it’s consistent. This worked for input 1, and after a lot of thought and optimization we got input to run in under a second. However, input 2 never terminated. Our third attempt we switched up the variables and values to have the words as variables and the starting positions as the values in their domain. This led to a much simpler solution that ran very efficiently. We found that trying words from largest length to smallest was best as larger words were more constraining. We also reduced the domain in the beginning by only adding coordinates in the domain such that if the word starts there, it would fit in the puzzle (We didn’t add (8,8) to the domain of any word). We knew what the constraints would be right away as they were pretty obvious. They were that all words must be used exactly once, they can only be put horizontally or vertically, and that no letter may repeat within a given row, column, or 3x3 square.

Input 1 Solution:

L I G H T E N M P

C O N F U S E A Y

S U P W I N D R T

E T U N D R A V H

M R F I C K Y E O

I A O M S H P L N

N G L B A U O I E

A E K L V M U N C

R D S Y E P T G K

Assignment Order:

V, 0 , 7 : MARVELING

V, 1 , 1 : OUTRAGED

H, 1 , 0 : CONFUSE

V, 2 , 0 : SEMINAR

H, 0 , 0 : LIGHTEN

H, 2 , 1 : UPWIND

V, 3 , 3 : NIMBLY

V, 0 , 8 : PYTHON

H, 3 , 1 : TUNDRA

V, 4 , 2 : FOLKS

V, 5 , 5 : HUMP

V, 5 , 8 : NECK

V, 5 , 6 : POUT

H, 4 , 3 : ICKY

V, 5 , 4 : SAVE

V, 2 , 6 : DAY

H, 2 , 0 : SUP

H, 1 , 5 : SEA

V, 4 , 8 : ONE

Number of Nodes expanded: 283

Time: 2.80302596092 seconds

Input 2 Solution:

D R I V E L S U B

C L A M P D O W N

O B S T I N A C Y

Q O V E N B I R D

U A G S Y M B O L

E T L O C K J A W

T I O A R P U N K

R N B L U I D E A

Y G E B X S P I T

H, 1 , 0 : CLAMPDOWN

H, 2 , 0 : OBSTINACY

H, 3 , 1 : OVENBIRD

V, 1 , 0 : COQUETRY

H, 5 , 2 : LOCKJAW

V, 2 , 1 : BOATING

H, 0 , 0 : DRIVELS

H, 4 , 3 : SYMBOL

V, 4 , 2 : GLOBE

H, 6 , 5 : PUNK

V, 5 , 4 : CRUX

H, 7 , 5 : IDEA

V, 3 , 7 : ROAN

H, 8 , 5 : SPIT

H, 6 , 2 : OAR

V, 1 , 4 : PIN

H, 0 , 6 : SUB

V, 6 , 5 : PIS

V, 6 , 3 : ALB

Number of Nodes expanded: 114

Time: 0.828738927841 seconds

Part 2: Game of Breakthrough

We formulated our AI agents as extending from a base class called Player. That base class contains all common information across the two types of AI (Minimax and Alpha-Beta). Most importantly, this base class also contains our implementations of the offensive and defensive heuristic functions. We put those in the base class instead of each individual AI in order to guarantee that they would both run the exact same heuristic, since then the differences between the AIs would be completely clear.

The actual Minimax and Alpha-Beta functions were relatively similar. We start by overriding an abstract base method in Player called makeMove, which then calls Minimax or Alpha-Beta with the appropriate parameters. Each function takes in a current depth (initialized to 3), the current board state, and whether we’re currently looking at a min or a max node. It then checks the base case of a game-over or a depth of 0 left, in which case the assigned evaluation function’s value on the board state is returned. Then, for each piece, it creates a new board state for each legal move available. Each board state is created by modifying the original one then undoing any changes after evaluation to avoid creating dozens of new 2D arrays per node expansion. Each of these legal moves is treated as a child of the current node, and as such is searched recursively with one less starting depth. Alpha-Beta will then check if the returned value falls outside of the allotted range, in which case the node can be pruned. Finally, the result is returned along with a reference to the best move found within the search. The AI then makes the best move found at the highest level of search.

Our offensive and defensive heuristic functions are relatively similar as well in terms of implementation, with some key values flipped. They first each check if the given board state has the AI running the heuristic winning the game, in which case a very large value is immediately returned, or if the AI loses, in which case a very low value is returned. Next, the number of pieces each player has left is checked. The offensive heuristic checks how many pieces the opponent has left and lowers the return value for each, while the defensive heuristic raises the return value for each piece the AI itself has left. Finally, the positions of each player’s pieces are evaluated. The offensive function raises the return value for how close to the opponent’s side of the board each of their pieces are, and the defensive function lowers the return value for how close to their side each of the opponent’s pieces are at.

As expected, the use of alpha-beta vs. minimax agents did not affect the result of the games at all. This is because alpha-beta and minimax should return the same results, just with alpha beta being more efficient in its searching. As such, the minimax functions with a depth of 3 took about .4-.7 seconds longer per move on a depth of 3 than the alpha-beta functions with a depth of 3. The minimax functions took around 1.5-1.7 seconds in total on average, and the alpha-beta functions took around 1.0-1.1 seconds each time. Interestingly, the different heuristic functions took very similar average times to evaluate, although the defensive games tended to end earlier than the offensive games. In addition, in both cases of a defensive vs. an offensive agent, the defensive would win. This may be because our defensive agents will actively attempt to prevent the other player from winning, while the offensive agents won’t care about an enemy approaching victory.

Matchup results:

1. Minimax vs. minimax w/ offensive first

B B B B B B B

W B B B B B B

B

W

W W W W W

B W W W W W W

------------------

Player 1 average seconds: 1.66467446547

Player 1 total nodes expanded: 169371

Player 1 average nodes expanded: 13028

Number of pieces captured by Player 1: 0

Number of total moves made by Player 1: 13

Player 2 average seconds: 1.72236193143

Player 2 total nodes expanded: 178226

Player 2 average nodes expanded: 13709

Number of pieces captured by Player 2: 0

Number of total moves made by Player 2: 13

1. Alpha-beta vs. alpha-beta w/ defensive first

W B B B B B B

B B B B

B W W W W W W

W W W W W W W

------------------

Player 1 average seconds: 1.0675098002

Player 1 total nodes expanded: 152096

Player 1 average nodes expanded: 9506

Number of pieces captured by Player 1: 0

Number of total moves made by Player 1: 16

Player 2 average seconds: 1.0069829305

Player 2 total nodes expanded: 133520

Player 2 average nodes expanded: 8901

Number of pieces captured by Player 2: 0

Number of total moves made by Player 2: 15

1. Minimax vs. alpha-beta (minimax goes first), offensive vs. offensive

W B B B B B B

B B B B

B W W W W W W

W W W W W W W

------------------

Player 1 average seconds: 1.43539850414

Player 1 total nodes expanded: 204528

Player 1 average nodes expanded: 12783

Number of pieces captured by Player 1: 0

Number of total moves made by Player 1: 16

Player 2 average seconds: 1.00851866404

Player 2 total nodes expanded: 133520

Player 2 average nodes expanded: 8901

Number of pieces captured by Player 2: 0

Number of total moves made by Player 2: 15

1. Alpha-beta vs. minimax (alpha-beta goes first), defensive vs. defensive

B B B B B B

B B B B B B

B W

W

W W W W W W

B W W W W W W

------------------

Player 1 average seconds: 1.1326178142

Player 1 total nodes expanded: 138776

Player 1 average nodes expanded: 9912

Number of pieces captured by Player 1: 0

Number of total moves made by Player 1: 14

Player 2 average seconds: 1.5068081958

Player 2 total nodes expanded: 188068

Player 2 average nodes expanded: 13433

Number of pieces captured by Player 2: 0

Number of total moves made by Player 2: 14