Tic-Tac-Toe

USING ALPHA-BETA PRUNING ALGORITHM

Alex Sadeghi
Cristian Lewczyk
Delan Diaz
Muzammel Choudhery
Languages

- Java
- Python
- C

- Runtimes Differ, there are differences in productivity:
  - Dynamically Typed
  - Concise
  - Compact
Language Differences

- Dynamic Typing

```java
int myCounter = 0;
String myString = String.valueOf(myCounter);
if (myString.equals("0")) ...
```

```python
myCounter = 0
myString = str(myCounter)
if myString == "0": ...
```

```python
def determine(board, player):
a = -2
choices = []
if len(board.available_moves()) == 9:
    return 4
for move in board.available_moves():
    board.make_move(move, player)
    val = board.alphabeta(board, get_enemy(player), -2, 2)
    board.make_move(move, None)
    print "move:, move + 1, "causes:"
    if val > a:
        a = val
        choices = [move]
    elif val == a:
        choices.append(move)
return random.choice(choices)
```
Language Differences

- Compact

```
JAVA
import java.io.*;
...
BufferedReader myFile =
    new BufferedReader(
        new FileReader(argFilename));

PYTHON
# open an input file
myFile = open(argFilename)
```
Combinatorial Search Problem

- Solve difficult instances of problems by finding optimal solution or return a best solution for a specific part that has been explored.

- Combinatorial search problem examples:
  - Eight Queens Puzzle
  - Reversi
  - Chess

- Common Combinatorial Search Algorithms:
  - A* Search Algorithm
  - Alpha-Beta Pruning
  - Minimax
Minimax Example

MAX's Best Move
Alpha-Beta Pruning

- Maintains two values (α and β) to represent maximum score and minimum score
- Alpha-Beta Pruning increases Minimax efficiency

- Worst: $O(b^d)$
  - $O(b*b*b*b...)$
- Average: $O(b^{3d/4})$
  - Nodes ordered at random
- Best: $O(b^{d/2})$

b represents branching factors (# of children at each node)
d represents plies (# of turns)
Alpha-Beta Pruning Example

```
if (turn == 1) {
    placeMove(point, 1);
    currentScore = alphaBetaMinimax(alpha, beta, depth+1, 2);
    maxValue = Math.max(maxValue, currentScore);
    // Set alpha
    alpha = Math.max(currentScore, alpha);
}
else if (turn == 2) {
    placeMove(point, 2);
    currentScore = alphaBetaMinimax(alpha, beta, depth+1, 1);
    minValue = Math.min(minValue, currentScore);
    // Set beta
    beta = Math.min(currentScore, beta);
}
// If a pruning has been done, don't evaluate the rest of the sibling states
if (currentScore == Integer.MAX_VALUE || currentScore == Integer.MIN_VALUE) break;
```
Runtime Results

- **Java:**
  - 1,000: 1.342 seconds
  - 10,000: 10.783 seconds
  - 25,000: 26.676 seconds
  - 50,000: 53.056 seconds

- **C (Minimax):**
  - 1,000: 6.208 seconds
  - 10,000: 61.302 seconds
  - 25,000: 153.074 seconds
  - 50,000: 306.933 seconds

- **Python:**
  - 1,000: 276.739 seconds
  - 10,000: ??
  - 25,000: ??
  - 50,000: ??
Evaluation

- Tried implementation in Haskell…Results were not good
- Code… tl;dr (Needs to be condensed, it’s quite ugly and not very organized)