Daisy Intelligence Hackathon

January 28-29, 2017

In this hackathon, your goal will be to build an AI program, ideally using the GPU resources made available to you, to play ultimate tic-tac-toe against the other hackathon participants. Good luck!

1 Ultimate tic-tac-toe

Ultimate tic-tac-toe is a variation of the classic tic-tac-toe game. The game is played on a 3x3 grid of "boards", where each board is a 3x3 grid of squares. This is shown in figure 1.

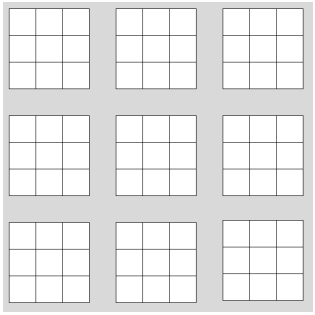

Figure 1: The basic ultimate tic-tac-toe gameboard - a 3x3 grid of boards.

The rules of the game are as follows:

1. Each turn, the active player plays either an 'X' (first player) or an 'O' (second player) in one of the empty squares. The active player alternates each turn.

2. A board is won according to standard tic-tac-toe rules. That is, when three Xs or three Os connect consecutively vertically, horizontally, or diagonally.

3. Once a board is won (or it is full), it cannot be played in again.

4. The previous player's move determines the next player's possible moves. For example, if the previous player plays in the top-left square of any board, the next player must play in the top left board. If the previous player plays in the centre square of any board, the next player must play in the central board. An illustrative example is shown in figure 2.

5. If a player would be forced to move in a board that is won or in a board that is full, or if it is the first move of the game, that player may play in *any* open square.

6. The game ends a win when three consecutive (horizontal, vertical or diagonal) boards are won according to standard tic-tac-toe rules, or ends in a tie when there are no more possible moves. An example is shown in figure 3.

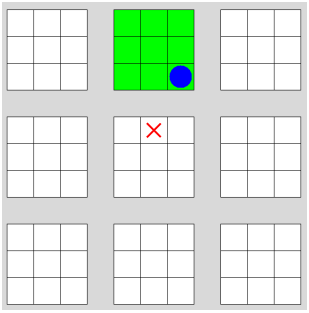

Figure 2: Showing the movement rules of ultimate tic-tac-toe. The First player (X) played in the top-middle square of a board, so the second player (O) must play in the top-middle board - any of the green moves would have been valid. X will then have to play in the bottom-right board.

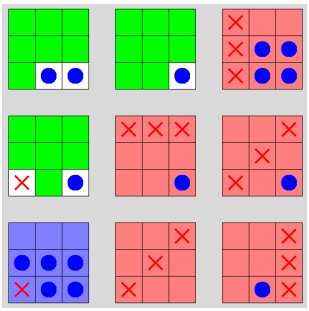

Figure 3: An example of a won game. X won three boards (top-right,middle-right and bottom-right), completing a vertical connection and thus winning the game.

2 Your Program

Your program must take in input reflecting the current game state, and produce output stating the next move you wish to move. The specifics of how to submit your program are described in section 2.1, the specifics of the input are described in section 2.3 and the specifics of the output are described in section 2.4.

2.1 Program Specifics

- You may write your program in whatever language you want, provided that we can easily access a compiler or interpreter for it. Please notify us ahead of time if you need any special programs or libaries installed to test or run your program, and we will be happy to accomodoate any reasonable requests.
- Our GPUs have Compute Capability 3.0, so anything you use on them must be compatible with that
- Your code **must** have a file name in the form of `program_X.Y`, where X is your team number and Y is the appropriate file extension. E.g., `program_1.py`, `program_2.c`, etc.
- Your code must be submitted by email by 10:30AM on Sunday.
- If your program is a python file, your code must have a method called `get_move(T, gamestate)`, which we will import and call to get your

move. In particular, this code must `return` an integer representing the move you wish to make, using the labelling system of section 2.2. The parameters T and `gamestate` are described in section 2.3.

- If your code is not a python file, please submit your source file in the email submission and we will compile them. Please include any specifical compilation instructions if necessary. Your code will take T and `gamestate` as command line inputs, and must write the move you want to make to `stdout`.

2.2 Square and Board Labelling

We use a square labelling system, from 0-80, shown in figure 4. Similarly, we label the boards, from 0-8, following the same order (e.g., squares 0-8 are in board 0, squares 9-17 are in board 1, etc.). This labelling system allows you to find the board of any square by `int(square/9)` and the location (0-8) of any square within its own board using `square mod 9`.

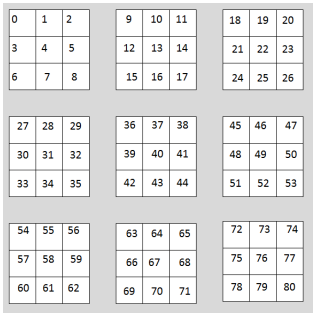

Figure 4: The square labelling system that we use.

2.3 Program Input

- The input to your program (function input for python, command line input otherwise) is T. T is the time, in seconds, your program has to return a move.

- The second input is an 83 character string reflecting the current state of the game. The first character tells you which move it is - if it is '1' then you are moving as 'X', and if it is '2' you are moving as 'O'. The

second character tells you which board you must play in, from '0' to '8'. If it is '9' then you may play in any valid square, as per rule 5. The next 81 characters reflect the current state of the board, according to the numbering system in figure 4. '0' means an empty square, '1' means a square filled by an 'X', and '2' means a square filled by an'O'.

- So for example, if you were the 'X' in figure 2 making the third move of the game, and you were allowed $T = 10.5$ sconds to make a move, your command line input would be (with no linebreaks)
`"10.5 180000000000000000020000000000000000`
`010000000000000000000000000000000000000000"`, or if you submitted python code we would call
`get_move(10.5,180000000000000000020000000000000000`
`010000000000000000000000000000000000000000).`

2.4 Program Output

- Your program must return (python) or output to `stdout`(otherwise), simply, the number of the square that you wish to move in.
- If your program does not produce output in the required time a random move will be chosen for you. You will also receive a penalty.
- If your program outputs a move that is not valid, a random move will be chosen for you. You will also receive a penalty.
- If your program gives unacceptable output (e.g., if it crashes with a segmentation fault or if it outputs a non-numeric string) a random move will be chosen for you. You will also receive a penalty.

3 Accessing Computational Resources

You will be given log-in credentials to one of our GPU boxes. You will also have access to run our main program `uttt.py`. It takes, on the command line, the name of a json file that it uses for input. You may copy and modify `default.json`. In particular, by changing the `CODE1` and/or `CODE2` parameters. You will also be given access to our `randombot.py` to test your code against.

4 Potentially Useful Pseudo-Code Snippets

In this section, you will be given some block of basic pseudo-code to help you get started.

4.1 Reading in the command line

```
timePerMove = float(argv[1])
gameString = argv[2]
whichPlayer = int(gameString[0]
nextBoard = int(gameString[1])
squares = empty array of length 81
for i from 0 to 80:
    squares[i] = int(gameString[i+2])
```

4.2 Checking who won a board

```
\\Return 0 if non-winner, 1 if X is winner, 2 if O is winner
function boardWinner(bigBoard):
    \\get the squares in the board and store in sq
    sq = subset of squares array from (BigBoard*9) to (BigBoard*9+8)
    \\check horizontal combinations
    if sq[0] equals sq[1] and sq[1] equals sq[2] and sq[0] > 0: return sq[0]
    if sq[3] equals sq[4] and sq[4] equals sq[5] and sq[3] > 0: return sq[3]
    if sq[6] equals sq[7] and sq[7] equals sq[8] and sq[6] > 0: return sq[6]
    \\check vertical combinations
    if sq[0] equals sq[3] and sq[3] equals sq[6] and sq[0] > 0: return sq[0]
    if sq[1] equals sq[4] and sq[4] equals sq[7] and sq[1] > 0: return sq[1]
    if sq[2] equals sq[5] and sq[5] equals sq[8] and sq[2] > 0: return sq[2]
    \\check diagonoal combinations
    if sq[0] equals sq[4] and sq[4] equals sq[8] and sq[0] > 0: return sq[0]
    if sq[2] equals sq[4] and sq[4] equals sq[6] and sq[2] > 0: return sq[2]
    \\found nothing
    return 0
```

4.3 Checking if a board is full

```
function boardIsFull(bigBoard):
    sq = subset of squares array from (BigBoard*9) to (BigBoard*9+8)
    for i from 0 to 8:
        if sq[i] == 0: return False
    return True
```

4.4 Finding valid moves

```
function findValidMoves:
    validMoves = empty list
    for i from 0 to 80:
        BigBoard=int(i/9)
        \\Make sure you're forced to play in that board or can play in any
        if BigBoard equals nextBigBoard or nextBigBoard equals 9:
           \\Make sure that the square is eligible to be played in
            if squares[i] equals 0 and boardWinner(i) equals 0
            and not boardIsFull(i):
                append i to validMoves
    return validMoves
```

4.5 Checking who won the game

```
BoardWinners = empty array of length 9
for i from 0 to 8:
    BoardWinners[i]=boardWinner(i)
\\Then use the same logic on BoardWinners[] that the BoardWinner function used on squares
```

4.6 Simulating a random game from an initial state

```
function PlayGame(squares,firstMove,mover):
    makingFirstMove=True
    do:
        if makingFirstMove:
            move=firstMove
            makingFirstMove=False
        else:
            move = random choice from validMoves
        squares[move]=mover
        nextBoard=squares[firstMove]%9
        if boardWinner(nextBoard)>0 or boardIsFull(nextBoard):
            nextBoard=9
        if mover=1:
            mover=2
        else:
            mover=1
        validMoves=findValidMoves()
        winner=checkForWinner(squares)
    while (winner equals 0 and len(validMoves)>0)
    return winner
```

5 Tournament Structure

The tournament will be played as a double elimination tournament played on Sunday, using standard brackets depending on the number of teams in attendance. Some teams may receive a bye in the first round due to the number of teams participating, and these byes will be determined randomly. Each match will be played in three games, with the winner being the team with the highest score after the three games. The score is determined as

- Winning a game: 100 points
- Each board won in a tie game: 1 point
- Submitting an invalid move: -1 point

The latter two essentially serving as tie-breakers. Teams alternate between Xs and Os with each game, after being determined randomly in the first game. If there is no winner after 3 games, games will continue until there is a winner. Players are given $T = 5$ seconds to make a move, although this is subject to change depending on how quickly the tournament is going. Time permitting, the third place match will be played with five games instead of three, and the finals will be played with 7 games.