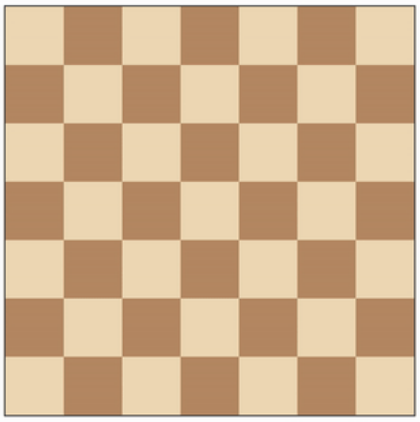
No description, website, or topics provided.

.udacity-pa	Added error message when project name command line argument omitted	7 months ago
isolation	fix isolation and sample players to match updated interface	7 months ago
isoviz	FIX isoviz compatibility with multiple browsers and instructions for …	7 months ago
.gitignore	Create .gitignore	9 months ago
README.md	Added couple of import statements in the sample code in README.md, to…	6 months ago
agent_test.py	refactor project unit tests for Udacity project assistant	7 months ago
competition_agent.py	refactor agent classes to simplify and clarify project requirements	7 months ago
game_agent.py	Remove instruction for custom_score_2 to be best heuristic	7 months ago
sample_players.py	FIX center score calculation (Issue #19)	7 months ago
tournament.py	Fix misleading warning message for forfeits	2 months ago
viz.gif	Add vizualization tool for isolation	8 months ago

README.md

Build a Game-playing Agent



Synopsis

In this project, students will develop an adversarial search agent to play the game "Isolation". Isolation is a deterministic, two-player game of perfect information in which the players alternate turns moving a single piece from one cell to another on a board. Whenever either player occupies a cell, that cell becomes blocked for the remainder of the game. The first player with no remaining legal moves loses, and the opponent is declared the winner. These rules are implemented in the `isolation.Board` class provided in the repository.

This project uses a version of Isolation where each agent is restricted to L-shaped movements (like a knight in chess) on a rectangular grid (like a chess or checkerboard). The agents can move to any open cell on the board that is 2-rows and 1-column or 2-columns and 1-row away from their current position on the board. Movements are blocked at the edges of the board (the board does not wrap around), however, the player can "jump" blocked or occupied spaces (just like a knight in chess).

Additionally, agents will have a fixed time limit each turn to search for the best move and respond. If the time limit expires during a player's turn, that player forfeits the match, and the opponent wins.

Students only need to modify code in the `game_agent.py` file to complete the project. Additional files include example Player and evaluation functions, the game board class, and a template to develop local unit tests.

Instructions

In order to complete the Isolation project, students must submit code that passes all test cases for the required functions in `game_agent.py` and complete a report as specified in the rubric. Students can submit using the Udacity Project Assistant command line utility. Students will receive feedback on test case success/failure after each submission.

Students must implement the following functions:

- `MinimaxPlayer.minimax()` : implement minimax search
- `AlphaBetaPlayer.alphabeta()` : implement minimax search with alpha-beta pruning
- `AlphaBetaPlayer.get_move()` : implement iterative deepening search
- `custom_score()` : implement your own best position evaluation heuristic
- `custom_score_2()` : implement your own alternate position evaluation heuristic
- `custom_score_3()` : implement your own alternate position evaluation heuristic

You may write or modify code within each file (but you must maintain compatibility with the function signatures provided). You may add other classes, functions, etc., as needed, but it is not required.

The Project Assistant sandbox for this project places some restrictions on the modules available and blocks calls to some of the standard library functions. In general, standard library functions that introspect code running in the sandbox are blocked, and the PA only allows the following modules `random` , `numpy` , `scipy` , `sklearn` , `itertools` , `math` , `heapq` , `collections` , `array` , `copy` , and `operator` . (Modules within these packages are also allowed, e.g., `numpy.random` .)

Quickstart Guide

The following example creates a game and illustrates the basic API. You can run this example by activating your aind anaconda environment and executing the command `python sample_players.py`

```
from isolation import Board
from sample_players import RandomPlayer
from sample_players import GreedyPlayer

# create an isolation board (by default 7x7)
player1 = RandomPlayer()
player2 = GreedyPlayer()
game = Board(player1, player2)

# place player 1 on the board at row 2, column 3, then place player 2 on
# the board at row 0, column 5; display the resulting board state.  Note
# that the .apply_move() method changes the calling object in-place.
game.apply_move((2, 3))
game.apply_move((0, 5))
print(game.to_string())

# players take turns moving on the board, so player1 should be next to move
assert(player1 == game.active_player)

# get a list of the legal moves available to the active player
print(game.get_legal_moves())

# get a successor of the current state by making a copy of the board and
# applying a move. Notice that this does NOT change the calling object
# (unlike .apply_move()).
new_game = game.forecast_move((1, 1))
assert(new_game.to_string() != game.to_string())
print("\nOld state:\n{}".format(game.to_string()))
print("\nNew state:\n{}".format(new_game.to_string()))

# play the remainder of the game automatically -- outcome can be "illegal
# move", "timeout", or "forfeit"
winner, history, outcome = game.play()
print("\nWinner: {}\nOutcome: {}".format(winner, outcome))
print(game.to_string())
print("Move history:\n{!s}".format(history))
```

Coding

The steps below outline a suggested process for completing the project -- however, this is just a suggestion to help you get started. A stub for writing unit tests is provided in the `agent_test.py` file (no local test cases are provided). (See the unittest module for information on getting started.)

The primary mechanism for testing your code will be the Udacity Project Assistant command line utility. You can install the Udacity-PA tool by activating your aind anaconda environment, then running `pip install udacity-pa`. You can submit your code for scoring by running `udacity submit isolation`. The project assistant server has a collection of unit tests that it will execute on your code, and it will provide feedback on any successes or failures. You must pass all test cases in the project assistant before you can complete the project by submitting your report for review.

0. Verify that the Udacity-PA tool is installed properly by submitting the project. Run `udacity submit isolation`. (You should see a list of test cases that failed -- that's expected because you haven't implemented any code yet.)
1. Modify the `MinimaxPlayer.minimax()` method to return any legal move for the active player. Resubmit your code to the project assistant and the minimax interface test should pass.
2. Further modify the `MinimaxPlayer.minimax()` method to implement the full recursive search procedure described in lecture (ref. AIMA Minimax Decision). Resubmit your code to the project assistant and both the minimax interface and functional test cases will pass.
3. Start on the alpha beta test cases. Modify the `AlphaBetaPlayer.alphabeta()` method to return any legal move for the active player. Resubmit your code to the project assistant and the alphabeta interface test should pass.
4. Further modify the `AlphaBetaPlayer.alphabeta()` method to implement the full recursive search procedure described in lecture (ref. AIMA Alpha-Beta Search). Resubmit your code to the project assistant and both the alphabeta interface and functional test cases will pass.
5. You can pass the interface test for the `AlphaBetaPlayer.get_move()` function by copying the code from `MinimaxPlayer.get_move()`. Resubmit your code to the project assistant to see that the `get_move()` interface test case passes.
6. Pass the test_get_move test by modifying `AlphaBetaPlayer.get_move()` to implement Iterative Deepening. See Also AIMA Iterative Deepening Search
7. Finally, pass the heuristic tests by implementing any heuristic in `custom_score()`, `custom_score_2()`, and `custom_score_3()`. (These test cases only validate the return value type -- it does not check for "correctness" of your heuristic.) You can see example heuristics in the `sample_players.py` file.

Tournament

The `tournament.py` script is used to evaluate the effectiveness of your custom heuristics. The script measures relative performance of your agent (named "Student" in the tournament) in a round-robin tournament against several other pre-defined agents. The Student agent uses time-limited Iterative Deepening along with your custom heuristics.

The performance of time-limited iterative deepening search is hardware dependent (faster hardware is expected to search deeper than slower hardware in the same amount of time). The script controls for these effects by also measuring the baseline performance of an agent called "ID_Improved" that uses Iterative Deepening and the improved_score heuristic defined in `sample_players.py`. Your goal is to develop a heuristic such that Student outperforms ID_Improved. (NOTE: This can be *very* challenging!)

The tournament opponents are listed below. (See also: sample heuristics and players defined in sample_players.py)

- Random: An agent that randomly chooses a move each turn.
- MM_Open: MinimaxPlayer agent using the open_move_score heuristic with search depth 3
- MM_Center: MinimaxPlayer agent using the center_score heuristic with search depth 3
- MM_Improved: MinimaxPlayer agent using the improved_score heuristic with search depth 3
- AB_Open: AlphaBetaPlayer using iterative deepening alpha-beta search and the open_move_score heuristic
- AB_Center: AlphaBetaPlayer using iterative deepening alpha-beta search and the center_score heuristic
- AB_Improved: AlphaBetaPlayer using iterative deepening alpha-beta search and the improved_score heuristic

Submission

Before submitting your solution to a reviewer, you are required to submit your project to Udacity's Project Assistant, which will provide some initial feedback.

Please see the instructions in the AIND-Sudoku project repository for installation and setup instructions.

To submit your code to the project assistant, run `udacity submit isolation` from within the top-level directory of this project. You will be prompted for a username and password. If you login using google or facebook, follow the instructions for using a jwt.

This process will create a zipfile in your top-level directory named `isolation-<id>.zip` . This is the file that you should submit to the Udacity reviews system.

Game Visualization

The `isoviz` folder contains a modified version of chessboard.js that can animate games played on a 7x7 board. In order to use the board, you must run a local webserver by running `python -m http.server 8000` from your project directory (you can replace 8000 with another port number if that one is unavailable), then open your browser to `http://localhost:8000` and navigate to the `/isoviz/display.html` page. Enter the move history of an isolation match (i.e., the array returned by the Board.play() method) into the text area and run the match. Refresh the page to run a different game. (Feel free to submit pull requests with improvements to isoviz.)

PvP Competition

Once your project has been reviewed and accepted by meeting all requirements of the rubric, you are invited to complete the `competition_agent.py` file using any combination of techniques and improvements from lectures or online, and then submit it to compete in a tournament against other students from your cohort and past cohort champions. Additional details (official rules, submission deadline, etc.) will be provided separately.

The competition agent can be submitted using the Udacity project assistant:

```
udacity submit isolation-pvp
```