CS 1410 Final Project: TRON-41

Partner Matching Form: closes Thursday October 10
Warm-up Due: Monday October 28
Final Due: Monday December 9

1 Introduction

In this project, you will create a bot to play TRON-41, a modified version of the game TRON.

2 The Game

2.1 The Basics

TRON-41 is a two-player game played on a rectangular grid, in which players take turns moving in one of four directions (Up, Down, Left, and Right) and leave behind an impenetrable **barrier** in the position where they were.

A player loses by colliding with a barrier or one of the innate walls of the board. Below are two example game boards on the next page for your reference. The one on the left is the start of a game, and the one on the right is the same game board after each player has moved once.

```
###############
#1            #
#             #
#       *     #
#             #
#    @     ^  #
#             #
#        !    #
#            2#
###############
```

```
###############
#x            #
#1            #
#       *     #
#             #
#    @     ^  #
#             #
#        !   2#
#            x#
###############
```

The numbers 1 and 2 denote the current locations of players 1 and 2, respectively; the `#` symbols denote permanent walls; the `*`, `@`, `^`, and `!` symbols represent powerups; and the `x` symbols represent the barriers that either player has left behind.

2.2 Powerups

A player obtains a powerup automatically by stepping on it. There are four different types of powerups:

1. Trap: Represented by `*` on the map.
2. Armor: Represented by `@` on the map.
3. Speed: Represented by `^` on the map.
4. Bomb: Represented by `!` on the map.

2.2.1 Trap

Trap powerups place up to 3 (as many as can fit) barriers on the border of the 5x5 square surrounding the opposing player. The x's in the figure below denote the locations at which barriers can be placed if player 2 just stepped on a trap powerup. The locations of the barriers on this square are selected uniformly at random.

```
###############
#             #
#   xxxxx     #
#   x   x     #
#   x 1 x     #
#   x   x     #
#   xxxxx     #
#             #
#           2 #
###############
```

2.2.2 Armor

Armor powerups allow the player to travel through one barrier. When a player obtains an armor powerup, they will be allowed to use it any time afterward. It is used automatically once the player travels through a barrier. Note that the armor powerup only allows users to travel through *barriers* (represented on the map by `x`), not permanent walls (`#`) or other players (`1,2`).

2.2.3 Speed

Speed powerups allow the player to take 4 consecutive turns (as if they got a speed boost). This is mandatory and the player cannot choose to skip the extra turns.

2.2.4 Bomb

Bomb powerups destroy all the barriers (`x`) in the 9x9 square surrounding the bomb, replacing them with open space. They are activated immediately upon a player stepping on them. The `x`'s in the figure below denote the locations where barriers would be destroyed if the bomb in the center was activated.

```
##################
#                #
#  xxxxxxxxx     #
#  xxxxxxxxx     #
#  xxxxxxxxx     #
#  xxxxxxxxx     #
#  xxxx!xxxx     #
#  xxxxxxxxx     #
#  xxxxxxxxx     #
#  xxxxxxxxx     #
#  xxxxxxxxx     #
#                #
#                #
#                #
##################
```

2.3 Time Limit

Each player must make their decision **within 0.3 seconds.** If a player takes too long, they are forced to move Up. Furthermore, bots are not allowed to use multithreading.

2.4 Run an Example Game

A good way to learn how the game works is to run an example game. Running `gamerunner.py` (without any command-line arguments) will initiate a game between two bots who choose their actions randomly and print the stream of board positions in your terminal.

3 Code

3.1 Code to modify

Note: You ***may not*** use the `signal` library or catch `support.TimeoutException` as part of your solution.

- `bots.py` contains a stencil for the `StudentBot` class, where you should fill out the `decide()` and `cleanup()` functions. This module also contains code for bots against which you can test your `StudentBot`. You can also write new Bot classes in case you want to compare multiple strategies.

  We highly recommend you read through the code for the bots we have already implemented and try to understand it. This code will help you get used to the different variables and methods of the `TronProblem` and `TronState` classes that your bot can use.

- `support.py` contains a function `determine_bot_functions()` to which you can add clauses that correspond to new bots you write in `bots.py`. It is only necessary to do this if you create a bot besides `StudentBot` for the purpose of testing `StudentBot`.

3.2 Necessary Source Code

Note: Do not modify any of these stencil files

- `tronproblem.py` contains code that defines the `TronProblem` and `TronState` classes.

  The function defined in this module that we expect to be the most useful is the static method `get_safe_actions(board, loc)`, which returns the set of actions one can take from the position `loc` (a tuple) that do not result in a collision. It will also be useful to familiarize yourself with the different instance variables of the `TronState` class. You can see `bots.py` for some examples of bots accessing these variables.

- `gamerunner.py` contains the code that actually runs the game. You may want to read this code to figure out how the code will behave when different command line arguments are set.

- `trontypes.py` contains constants that are used to identify cells on the board and types of powerups. These are used throughout `tronproblem.py` and `gamerunner.py` and may be helpful to use when writing your bots.

- `boardprinter.py` contains the code that handles printing the board and game information to the terminal. It is unlikely that you will need to look through this code.

- `adversarialsearchproblem.py` is identical to the file we distributed for the Adversarial Search assignment. The `TronProblem` class inherits from the `AdversarialSearchProblem` class. You should not need to use this file at all.

3.3 Testing Your Solution

You can run your code using the `main()` function of `gamerunner.py`. This function uses a few command line arguments, the most important of which we'll describe here:

- `-bots` lets you specify which bots will play against each other. The syntax is `-bots <bot1> <bot2>`
- `-map` lets you change the map that the game is played on. The syntax is `-map <path to map>`
- `-multi_test` lets you run the same kind of game multiple times. You may want to use this with the `-no_image` flag so the games go more quickly. This would look like `-multi_test <number of games> -no_image`.
- `-no_color` runs the game without coloring the board printout. Use this option if the coloring causes display issues.

For example, you can test your `StudentBot` against `WallBot` on the joust map with

```
python gamerunner.py -bots student wall -map maps/joust.txt
```

You can test your `StudentBot` against `TABot1` 15 times with no visualizer on the empty_room map with

```
python gamerunner.py -bots student ta1 -map maps/empty_room.txt
-multi_test 15 -no_image
```

3.3.1 Map files

Maps are stored in `.txt` files. They store the maps using the same characters that appear in the board printout. The only exception is the `?` character that appears in the files, which represent random powerups. When `gamerunner.py` reads in the map files, one of the four powerups is randomly chosen to replace each `?` character.

4 Warm-up

For Tron, you will hand in a preliminary assignment that will be due **Monday, October 28th**. This assignment is designed to get you familiar with the code

provided and the problem at hand. By the end of the Warm-up assignment you should know who your partner is, have a grasp on how to implement a basic bot, and understand what sort of algorithms and tactics may be useful to employ to make a successful one.

4.1 Find a Teammate

Note: If you are taking AI as a Capstone, you must work individually.

Otherwise, for this assignment, you are encouraged to work in pairs. You are welcome to choose your own partner or use the following google form to find a partner at random: `https://forms.gle/YPsF4imYq46Do1Ey7`. If you would like to be assigned a partner through the form, make sure to fill it out by 11:59PM on Thursday, October 10th!

4.2 Describe a Bot

You and your partner should discuss the elements of a good bot and synthesize a description based on what you think will make a bot successful.

- Write a brief description for the implementation of a bot that you would like to submit. How will your bot make its decisions and what factors will it take into account?

This description will be handed in via Gradescope, only one submission per group. See the install and handin section for more details.

4.3 Code a Bot

Along with your description, you will hand in a basic implementation of a bot.

- Implement a functional bot. A bot that returns a random action is sufficient but when implementing it, be sure to familiarize yourself with all the code provided.
- **Note:** This bot does not have to be consistent with the description you provide nor does it have to be very successful

We will also begin running the Tron Tournament on 10/28 so you can see how your bot stacks up against other student bots. See the **Tournament** section for more details.

5 Writeup

You and your partner should collectively turn in a final writeup by **Monday, December 9** containing the following information in clearly labeled sections:

- A full description of how your bot works. Your description should enable its reader to replicate your bot.
- Brief descriptions of the motivations behind each of the important decisions you made about how your bot works.
- A description of any known shortcomings of your bot, and specifically how you would attempt to improve upon them if you had more time. Answering this question is not necessary but will reduce the number of points lost from shortcomings that we notice.

6 Tournament

We will be running a daily tournament so you can compete with other students in the course. This tournament will begin on **Monday, October 28th** and will run until the last day of handins. To submit your bot to the tournament, do the following:

1. Copy your code from ∼`/course/cs1410/Tron/` to ∼`/course/cs1410/TronTournament/`.
2. Add a custom bot name (be creative so you can tell yours apart from other bots) by adding the following to your `StudentBot.__init__` function:

   ```
   self.BOT_NAME = "My bot name"
   ```

3. Submit your tournament bot with `cs1410_handin TronTournament`.

Only one tournament submission per group, please! The tournament will take the latest submissions every day around midnight and make the results available online at `http://cs.brown.edu/courses/csci1410/tron_results.html`. Participation in this tournament is not required but strongly encouraged - the winner may even get a prize!

7 Evaluating Your Bot

Your bot will play many matches against 4 different opponents on 4 different maps. Your bot will move first in exactly half of the matches.

7.1 Opponents

1. RandBot - always chooses uniformly at random among the actions that do not immediately lead to a loss.
2. WallBot - hugs walls and barriers to use space efficiently
3. TABot1 and TABot2 - TA bots with secret implementations

You can find the code for RandBot and WallBot in `bots.py`. The implementation of the TA bots is in `ta_bots.so` as a compiled module so that their source code is not exposed. You can still test your bot against them by using `ta1` or `ta2` as options for the `-bots` flag when running `gamerunner.py`.

7.2 Maps

You can find 3 of the maps, `divider`, `hunger_games`, and `joust`, in the maps directory. The fourth map in that directory, `empty_room`, is available for your testing but will not be used when grading. The fourth grading map is withheld but will have the same dimensions as the other maps.

7.3 Expectations

To get a good score, your bot should be able to defeat RandBot virtually all of the time, WallBot virtually all of the time on most maps, and each TABot most of the time.

7.4 Additional Capstone Expectations

There are additional expectations for those taking this course as a capstone:

1. You must work independently to research and implement your bot.
2. Your bot or at least some component of your bot must be implemented with machine learning (e.g. SARSA, Deep Q-Learning, Neural Networks).
3. Your writeup must also include an additional section describing the motivations, research/implementation process, and performance gains of your machine learning application. You should also discuss its shortcomings and potential areas of improvement.

Overall, it is estimated that doing this project as a capstone will constitute an additional $50 - 100\%$ the amount of work.

8 Advice

Here are some ideas and things to consider to get you started:

- It will be difficult to have a single decision-making function that works in all situations and never takes too long. As such, you may consider having multiple decision-making functions and switching between them based on easily determinable characteristics of the game state. For example, you may have a bot that uses a completely different decision-making function if there are no more powerups on the grid.

- You may want to consider either learning or imposing an evaluation function that maps game states to real numbers that indicate how good a game state is for your bot.
- The value of a powerup depends on the game state. Your bot may want to take this into account.
- As you iteratively improve your bot, it will be useful to keep past versions of it to use for testing your most recent bot. Additionally, you are permitted to test your bot against the bots of other students in the course, as long as you do not copy each other's code.

9 A Note About TA Hours

The final project is open-ended; there are many good solutions, and it's not obvious what will work and what will not. As such, you should not come to TA hours expecting definitive, "Yes, this will work" or "No, maybe try this instead" kinds of answers. Instead, you should view TA hours for this project as an opportunity to talk about your ideas and get a second opinion and also to review past material in the course. Don't forget to use your partner and classmates as resources as well!

10 Install and Handin

To install: `cs1410_install Tron`

To hand in your Warm-up code:

Copy your code from ∼`/course/cs1410/Tron/` to ∼`/course/cs1410/TronWarmup/`.

```
cs1410_handin TronWarmup
```

To hand in your final code: `cs1410_handin Tron`
(from within your ∼`/course/cs1410/Tron` directory)

Handin your writeup through Gradescope. **Only one warm-up, code, and writeup submission per group, please!** When submitting your TRON writeup and Warm-up on Gradescope please specify all members of your group. You can do this after you upload your document and hit "Submit". On the top right of the page there is a "View or edit group" button.

Additionally, please note that since this is a final project, the normal resubmission policy does not apply and you may not use any late days. **December 9 is the hard deadline for all parts of the project.**