Tileworld Project

By James Villemarette for CISC 489: Programming Assignment 1.

Table of Contents:

Initial Findings

- The position of the wall and obstacles will always stay the same.
- Each agent starts in the top left corner.
- Position `[0, 0]` is the top-left corner.
- Coins have values, 1 through 9.
- For some reason, the TA made x and y backwards on the coin and wall positions lists. It's stored as `[y, x]`.

The data we have access to looks like:

```
>> print('Coin Data:', get_coin_data())
Coin Data: (
    [5, 2, 7, 2, 5, 8, 9, 9, 3, 2, 2],
    [
        [100, 200],
        [400, 400],
        [500, 50],
        [300, 350],
        [50, 250],
        [450, 400],
        [450, 200],
        [150, 0],
        [150, 250],
        [0, 100],
        [150, 400]
    ]
)
>> print('Wall Positions:', get_wall_data())
Wall Positions: [
    [300, 50],
    [200, 200],
    [400, 200],
```

```
    [300, 150],
    [250, 400],
    [350, 450],
    [450, 100],
    [350, 400],
    [400, 450],
    [100, 300],
    [450, 250]
]
```

Questions

- Q: If both agents start in the top-left corner, does that count as a collision, or is that ignored?
  - A: Not a collision
- Q: Does the game world tick on regardless of how long the agents take, or is it blocking? Meaning, how efficient does our computation have to be?
  - A: Ask the TA. Professor Decker **thinks** it blocks, but not sure.
- Q: Does "wall" mean both the boundaries of the game board, or the obstacles in the middle? Or both?
  - A: **Both**.
- Q: Why is the coin and wall positions in the **hundreds?**
  - A: The positions of the objects are **padded** by 50. So just `position/50` and you'll get the true "normalized" position.
- Q: When these assignments are being graded, then will the games be run longer? 12 seconds is a short amount of time, and I think the randomness will take over in that time.
  - A: Professor Decker is not sure, but it does not really matter since the randomness is the same.
- Q: I ran the `case = 1` (two random agents), and I got the exact same score every time, what's up?
  - A: **The seed is not changing**. It's "random", but it is the same random every time. Professor Decker says the TA will run with different random seeds to compare against. The seed is set in `env.py`.

Runs

I ran the random agents (`case = 1`) a couple of times to calculate what the baseline is.

#	Time	Size (`N`)	Seed	Agent1 Score	Agent2 Score	Total
1	12	11	0	23	19	42
2	30	11	0	-27	-24	-51
3	12	22	0	22	23	45
4	30	22	0	32	52	84

Theories

- **Opening idea:** Both agents will create a heat map of the board, where coins (multiplied by their value) increase the "hotness" of an area, and walls decrease the hotness of an area. Agent 1 goes for the center of the hottest, and Agent 2 goes for the center of the second hottest area. Their path to their designated hot areas can be augmented by "left" or "right" by one if their is a coin of value 2 or greater.

Tuning

Okay, now that I got Agent 1 working as intended, I'm going to run it multiple times to improve it.

- **Baseline:** Total score is `51`
- Change 1
  - **Cause:** Increase the default weight of a node in the Dijkstra graph from 10 to 40, and make the coin values on decrease the node weight more significantly (multiple of 4)
  - **Effect:** Same total score of `51`
- Change 2
  - **Cause:** I gave Agent 2 logic. It will now just search for the nearest coin and go to it.
  - **Effect:** Total score is now `-829`
- Change 3
  - **Cause:** I added an `exclude_pos` (exclude position) argument to my `path_find` function. The path planning will now exclude the provided position (the position of the other player) when generating the graph.
  - **Effect:** Total score is now `-14`
- Change 4
  - **Cause:** I don't really know, I just added a few print statements.
  - **Effect:** Total score is now `186`, sometimes `181`, and sometimes crashes. I'm so stumped. It should be consistent. I don't know why it is random.
- Change 5
  - **Cause:** Player A excludes Player B's position from its path finding, as well
  - **Effect:** Did not crash, score still `186`.
- Change 6
  - **Cause:** In my `generate_graph` function, I upgraded the exclude position argument, so it now excludes the provided position (the other player's position) and excludes every adjacent cell (right, left, up, down) from path finding.
  - **Effect:** No collisions, and a new top score of `198`, sometimes `199`.
- Change 7
  - **Cause:** Bump up weights of graph generation, where every node is weighted as 100 by default, and if there is a coin on that node, then the weight of that node is subtracted by the coin value times 10. This gives an even greater weight to nodes with coins on it.
  - **Effect:** New high score of `207`.
- Change 8
  - **Cause:** I tried giving Agent 1 a `look_around` counter so that it would stay in a hot area and gather coins.

    - **Effect:** Agent 1 got stuck in a position, and then eventually crashed, somehow. I removed this look around counter.
- Analysis 1
    - I'm finding this problem would have been way easier if I had
        a. Organized all the environmental variables into some holder/data class.
        b. Followed the `Beliefs(...); Reconsider(...); Plan(...); Execute(...)` main control paradigm described in the book.
        c. More thoroughly planned my agents.
- Analysis 2
    - I'm also finding that my Agent 1 probably is not getting a lot of obvious/easy coin grabs because high value coins pop up on its way to an objective.
- Change 9
    - **Cause:** Massive re-write of Agent 1's logic, now representing its beliefs and desires through `"modes"`, where it is either starting up (`START_MODE`), moving to a hot area (`HOT_AREA`), or hunting for a coin in that hot area (`FOCUS_COIN`).
    - **Effect:** Errors, errors, and more errors.
- Change 10
    - **Cause:** Just make Agent 1 the same as Agent 2, except Agent 1 will look at the weighted value of coins, where Agent 2 will just pursue the nearest coin regardless of value.
    - **Effect:** New high score of `220`.

Conclusion

Functions/classes I wrote:

- `correct_positions` : To fix the pixel value positions of the coins and walls
- `normalize_coin` : For generating a heat map, calculate the temp of a coin based on its value (higher value coins are hotter).
- `generate_heatmap` : Makes the heatmap based on coins (hot) and walls (cold).
- `pretty_print_2d` : For printing the heatmap
- `find_peak_index` : For finding the highest value in the heatmap.
- `Dijkstra` : Class for path finding.
- `convert_str_to_pos` : For converting a cell's string representaiton (`C1, 1`) to a list representation (`[1, 1]`)
- `convert_pos_to_str` : Opposite of above function
- `generate_graph` : Makes the graph that the `Dijkstra` class will search in
- `path_find` : Wrapper function that uses `generate_graph` and `Dijkstra` to find the path between two vertices
- `convert_path_to_actions` : Converts the path found by `Dijstra` to a list of actions
- `find_nearest_coin` : Finds the nearest coin by 2D distance
- `construct_initial_plan` : To space out the two agents at the beginning, have this calculate the path
- `reconsider_surroundings` : A function that would reconsider the current plan, but I thought it would best just to manually implement it in the Agent `update()` function

- `random_action` : Comes up with a random action that's safe. I used it for testing. Never worked well
- `action_spelled_out` : Converted a single letter action to its full word (`r = right`)
- `other_agent_adjacent_old` : A function that became too complicated than was necessary. Deprecated
- `other_agent_adjacent` : Determines if another agent is adjacent or diagonal by one tile away
- `opposite_corner` : Calculate the closest corner to a given position, then give the corner opposite that corner. Used for if the agents get too close, run away

In conclusion, all of my heatmap stuff did actually work in some way. It was just hard to maintain and understand as time went on. I mentioned in my analysis above that I should've represented the problem as an object holding all of the environmental data. It would have made handling the function parameters a little easier.

My final high score was `220` for with all the default settings. I do not have enough time here at night to get testing different run configurations. I do know that running it multiple times, that it does not crash. So, that's pretty cool.

I wish I had even more time to figure out the heatmap stuff and more advanced algorithms. But I do have to put this down and focus on other homeworks.

High Level Summary: Agent 1 goes to the right by a few paces, and then starts looking for the closest coin, weighted by value. Agent 2 goes down by a few paces, and then starts looking for the closest coin, regardless of value. The agents "communicate" by telling each other their positions, and then avoiding a large area around each other in the path planning functions. This is the best design I could come up with multiple iterations of the agent code. You can see the agent code commented out, as well as some unused functions. All in all, this was a fun project, and I hope to do it again in the future.