CS 175: Project in AI (in Minecraft): Fall 2019

Assignment 1: Finding the Shortest Path

Sameer Singh
https://canvas.eee.uci.edu/courses/20194

1 Task Description

In this assignment, you have to guide the player agent in Minecraft through a maze to reach the goal block. For this exercise, we will assume that the agent can see the whole maze, i.e. it is completely observable. Given this information, the agent has to reach the goal in the minimum number of moves.

This assignment is based heavily on `tutorial_7.py` in the Malmo Python tutorials, so please follow the tutorials 1 through 5 to familiarize yourself with the API and the Malmo platform (tutorial 6 is not relevant yet).

1.1 Provided Source Code

We have provided two Python files. The most important one is `assignment1.py` which contains the complete code to setup the Malmo environment with different mazes, and have the agent run through it. The provided implementation is incomplete, and the agent needs your help to solve the maze most efficiently. We have also provided `priorit_dict.py`, a simple implementation of a heap-based priority queue. See the implementation for details, and discuss on Campuswire if you have any doubts. You can ignore this implementation if you prefer.

1.2 Setup and Running the Code

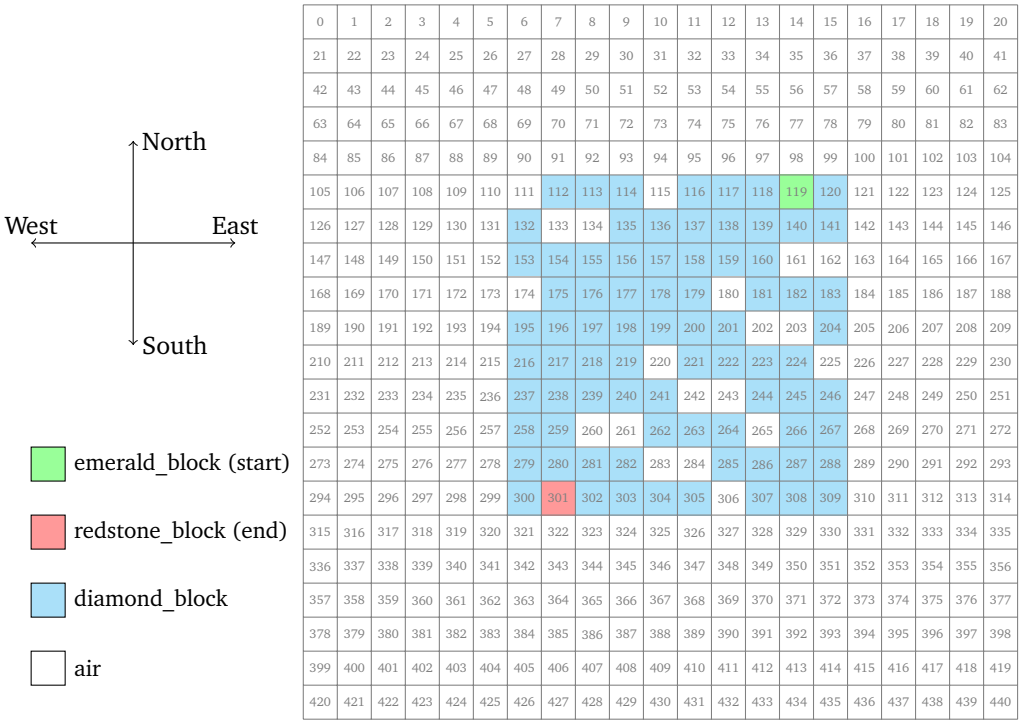
Assuming you have installed Malmo and started working your way through the tutorials, all you need to do to run this assignment is to copy the two files above to the `Python_Examples` folder, and after launching Minecraft, run `python assignment1.py` (Python 3 or higher). If everything run successfully, the agent should do nothing while the timer counts down for each of the ten missions. The output in the terminal should look like the following:

```
Size of maze:  6
Waiting for the mission 1 to start
Mission 1 running.
Output (start,end) 1 :  (None, None)
Output (path length) 1 :  0
Output (actions) 1 :  []
Error:  out of actions, but mission has not ended!
Error:  out of actions, but mission has not ended!

Mission 1 ended
Size of maze:  6
Waiting for the mission 2 to start
Mission 2 running.
Output (start,end) 2 :  (None, None)
Output (path length) 2 :  0
Output (actions) 2 :  []
Error:  out of actions, but mission has not ended!
Error:  out of actions, but mission has not ended!
```

1.3 Overview of the Code

As mentioned before, much of this assignment is based on `tutorial_7.py`, so please refer to the Malmo Tutorials in the `Python_Examples` directory. The source code creates a series of increasing difficult mazes made of *diamond_block* blocks as the floor, each with a start (*emerald_block*) and an end (*redstone_block*) block. The mission starts with the player at the start block, and ends when the player reaches the end block. Make sure you are at least somewhat familiar with the whole source code, however the main control code that is relevant for your implementation are in the following lines:



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83
84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125
126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146
147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167
168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188
189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209
210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230
231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251
252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272
273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293
294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314
315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335
336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356
357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377
378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398
399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419
420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440

Figure 1: **Grid Layout:** Layout of the two-dimensional grid laid out as a one-dimensional array. The location of the maze, especially the start and end blocks, may be different in the assignment. The array contain string-values representing the four types of possible blocks.

```
223  grid = load_grid(world_state)
224  start, end = find_start_end(grid) # implement this
225  path = dijkstra_shortest_path(grid, start, end) # implement this
226  action_list = extract_action_list_from_path(path)
```

We first get the array of blocks that consists of the 2-dimensional maze from Malmo in `load_grid`. Then, we want to identify the start and the end blocks (each represented by an integer index into the `grid` array). Given the start and the end blocks, and the whole maze, we want to compute the shortest path of blocks from the start to the end, computed using Djikstra's algorithm. Finally, we convert these list of blocks into Malmo *actions* that are carried out by the agent. We provide the correct implementation of the first and the last functions, but leave the correct implementation of the second and the third functions to you (more on this later).

1.4 Grid Layout

It is important for you to understand how the grid is laid out. We have requested the 21×21 grid of blocks from the world that lie at the floor of the player, with the player at the center of it ($(10, 10)$, if zero-indexed). This two-dimensional is laid out as a single-dimensional array of strings (representing the type of the block) in a row-major way (where row is the east-west direction), i.e. the (i, j) coordinate is represented by the index $j \times 21 + i$, where j is the east-west coordinate, and i is the north-south coordinate. For the precise indexing, see the illustration in Figure 1.

Since the grid is represented by a single-dimensional array, you need to get the neighboring blocks of the current block, represented by an integer index i. Looking at `extract_action_list_from_path` should give you a hint; the blocks to the immediate left (west) and right (east) are of course represented by $i-1$ and $i+1$, while the blocks in the north and south require jumping one whole row, thus $i-21$ and $i+21$, respectively. The function `extract_action_list_from_path` uses this to go the other way: if block i and j are next to each other on the grid, the action to get from i to j is represented by $j-i$, i.e. if $j-i$ is $+1$, the move is `moveeast`, if -1, the move is `movewest`, if -21, the move is `movenorth`, and if $+21$, the move is `movesouth`.

> ***Note:*** The player is facing south, so do not get confused if `movesouth` is *forward*.

2 What Do I Submit?

Here we'll describe what exactly you need to submit to the assignment on Canvas.

1. **Code: Finding Start and End Blocks (5 points):** As a simple exercise, implement the `find_start_end` function, that takes the grid as an array of string describing the block types, and returns the indices of the start and the end block. This should only require, at maximum, a few lines of code. Submit this snippet as your submission.
2. **Output: Start and End Block Indices (5 points):** Run the code with the above implemented, and look at the output lines that start with `Output (start,end)` and paste them as your submission. There should be 10 lines, one for each mission.
3. **Code: Shortest Path Implementation (50 points):** Implement Dijkstra's algorithm in order to find the shortest path from the source to the destination. The set of possible actions from each block is one step in north, south, east, or west directions, i.e. taking multiple or diagnol steps is not allowed. The path you compute should be shortest in terms of number of moves (all of them cost the same), should be a list of block indices (integers), should include both the start and the end blocks, and of course, should not contain any air blocks. Submit the complete implementation of your function.
4. **Output: Length of the Shortest Paths (30 points):** Run the code with the above implemented, and look at the output lines that start with `Output (path length)` and with `Output (actions)` and paste them as your submission. There should be 20 lines, two for each mission.
5. **Comments:** Any comments about your submission that you want to bring to our attention as we are grading it. This is completely optional, I expect most of you to leave this empty.
6. **Statement of Collaboration (10 points):** It is **mandatory** to include a *Statement of Collaboration* with respect to the guidelines below. Include the names of everyone involved in the discussions (especially in-person ones), and what was discussed. You should also include the links to all online resources you used for the assignment in this section.

All students are required to follow the academic honesty guidelines posted on the course website. For programming assignments, in particular, I encourage the students to organize (perhaps using Campuswire) to discuss the task description, assignment requirements, bugs in our/Malmo code, and the relevant technical content *before* they start working on it. However, you should *not* discuss the specific solutions, and, as a guiding principle, you are not allowed to take anything written or drawn away from these discussions (i.e. no photographs of the blackboard, written notes, etc.). The same holds for online resources: you are allowed to read the description of algorithms, but your code should be your own. Especially *after* you have started working on the assignment, try to restrict the discussion to Campuswire as much as possible, so that there is no doubt as to the extent of your collaboration.

Acknowledgements

This homework was originally created with help from Moshe Lichman.