

Artificial Intelligence Project: Maze Search

Authors:

Logan Bonney

Cody Stoner

Implementation:

This program was written in Python consisting of multiple parts. We decided to have a class mazes that just strictly dealt with reading the maze from the provided files and creating a node for each location on the maze. The next class we implemented was Search. This holds all of the code for doing BFS, DFS, A*, and Greedy algorithms. Within this class we used the node class to be able to traverse in given ways of each search algorithm. All the algorithms handle these nodes in similar ways, with difference mainly being the data structure used to organize them. Each search algorithm uses essentially the following pseudocode.

```
1  vistedNodes = []
2  while(queue.length > 0):
3      node = queue.next()
4      vistedNodes.append(node)

5      If(node.value == '*'):
6          print("Goal found")
7          Return true

8      for neighbor in node.neighbors:
9          If(vistedNode.contains(neighbor) == False and neighbor.value != '%'):
10             neighbor.previous = node
11             queue.add(neighbor)
```

There is also code to keep track of the number of nodes expanded small differences between algorithms. After the goal is found the path is found by following neighbor.previous back to the start.

Depth-First Search (DFS)

DFS adds neighbors to a stack whenever it moves to a new space. The nodes are then removed and added until the goal is found or there are no more nodes to search. DFS performed the best in terms of cost on the open maze but the final solution took many steps more than any others.

Breadth-First Search (BFS)

As seen BFS and DFS are very similar, but BFS uses a queue instead of a stack. So then we just had the top node dequeued from the que then all nodes adjacent nodes, that

haven't been visited. This was just repeated until the queue was empty. So once the finish node was found then the queue was empty.

Greedy Best-First Search (GREEDY)

Greedy search is different from BFS and DFS due to the knowledge of the goal state. When a node is expanded then it will expand to a node in the frontier. This is calculated using the manhattan distance from the given node in the frontier to the node at the goal state. We implemented this using a built in heap data structure with python (heapq). Using this allows a node to be added to frontier, then the node is added to the queue. As more nodes get added on then the first nodes in the queue are popped off. This is repeated until the goal state is found.

A* (ASTAR)

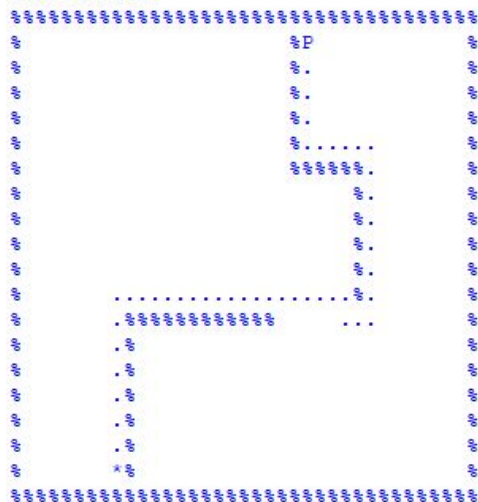
A* uses a min heap sorted by the distance to the goal added the distance traveled to get to the current position. This way it is always looking for the path with the least total expected cost. This algorithm performed fairly well in most cases but there is a bug that causes it to expand more nodes than should be possible in open spaces.

Contributions:

Logan was responsible for both the DFS and the A* search algorithms. Cody was responsible for both the BFS and the Greedy search algorithms. Both of us implemented various other methods such as node and read maze files so that we could further ease the implementations of the different searching methods.

Outputs:

```
Breadth First Search:  
Cost: 509, Steps 43  
Solved Maze:
```



Solved Maze:

Solved Maze:

The image displays a 10x10 grid of 100 small circular icons. Each icon contains a unique geometric design, often resembling a stylized letter or a complex pattern. The designs are composed of various shapes like circles, lines, and dots, arranged in a way that suggests a sequence or a set of variations. The overall effect is a dense, organized collection of abstract visual elements.

The figure shows a 2D spatial distribution of 1000 points. The points are arranged in a grid-like pattern, with a higher density in the center and tapering off towards the edges. The x and y axes are labeled from 0 to 1000. The points are concentrated in a central rectangular region, with a higher density in the center and tapering off towards the edges.

[illegible]

[illegible]

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	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----