Matrix puzzle solver project

Introduction

This project is to build a system that provides a GUI for solving matrix puzzles. It is flexible in what puzzles can be: they all use a matrix but they have different constraints or rules that specify what values are in the matrix and what constitutes a solution.

The main parts of the project are to make an object oriented design, to implement a GUI that allows a user to choose what puzzle to try and to use the GUI to solve it. And then to write a solver for magic square and a solver for sudoku. You have to give a performance analysis with empirical results for your solvers.

Detailed Requirements

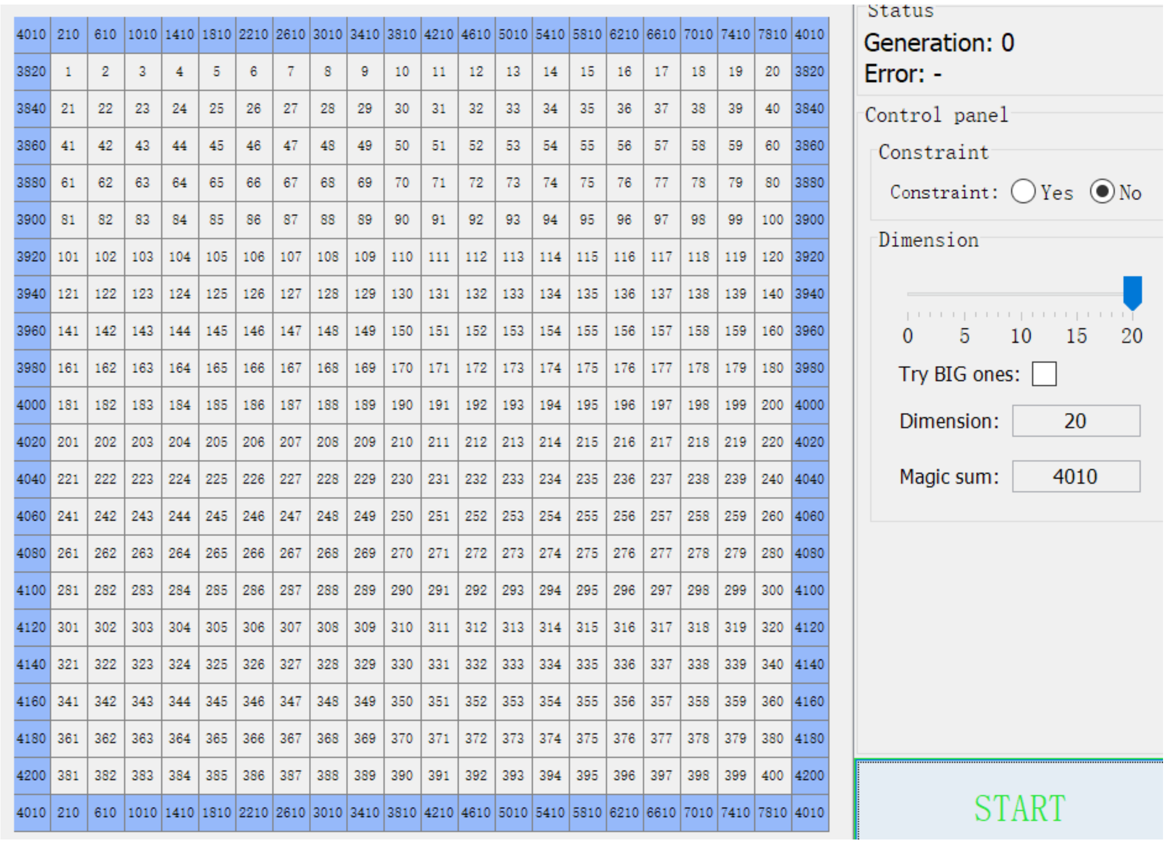
This project requires you to do the following:

1. Produce an object-oriented **design** for matrix puzzle representation, the design should allow for the following requirements.
    - Represent a matrix puzzle type as an **Abstract Data Type** with an independent internal data representation (e.g. as a vector, a single dimensional array, a 2d array).
    - Different **values** can be included in the puzzle (see below for the kinds of puzzle we consider to represent – magic square, soduku, etc)
    - Different **constraints** should be able to be set, these define the problem that has to be solved "The Puzzle". In magic square there are constraints on the value of the sum of diagonals, rows, columns, in sudoku there are constraints on the arrangement of the values 1-9 throughout the matrix (see description below).
    - Certain sets of allowable **operations** can be made to update the puzzle to try to find a solution (e.g. swap values). Other operations include to get a string representation, and to compare a solution is better or worse than another one.
    - You need to provide a documented class hierarchy that allows for representing different kinds of matrix puzzle.

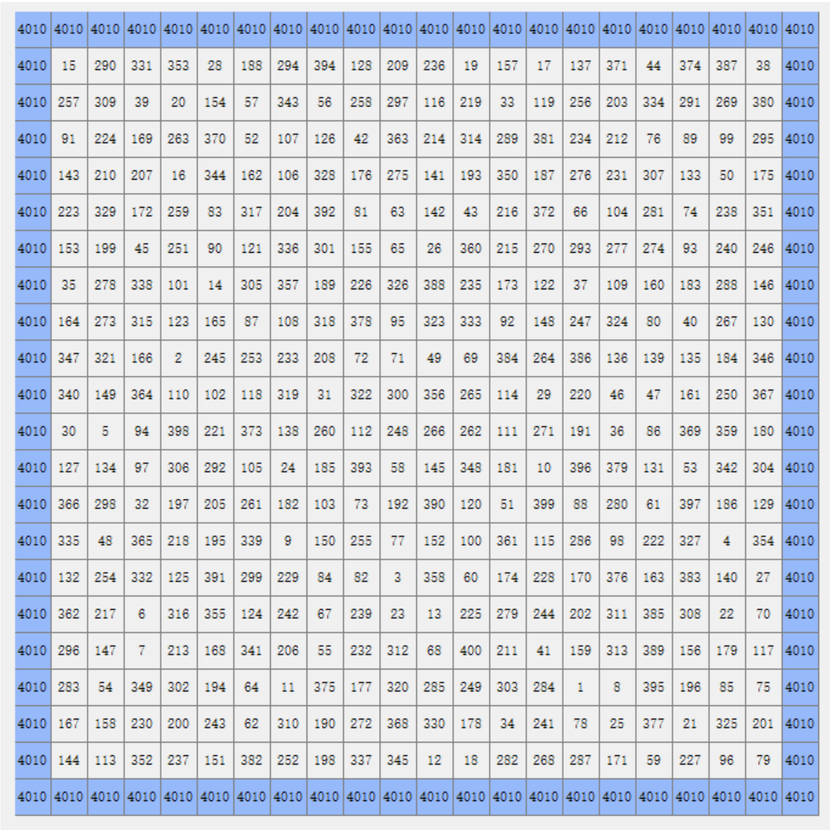
2. **Implement GUI:** your matrix puzzle representation and provide a graphical user interface. Requirements:
    - Allow a user to choose different previously stored puzzle types to use (e.g. to play magic square, or to play sudoku).
    - Display a puzzle matrix to a user.
    - Allow a user to manipulate the puzzle and try to find a solution.
    - Storage to save completed or in progress solutions to different puzzles to a file and reloaded.
    - Storage to save puzzle files – that is preconfigured
    - You can either use a web based GUI app

3. **Implement solver**: You will implement a solver for Magic Square and Sudoku. Requirements:
    - There can be two types of solver – one for magic square and on for sudoku.
    - It is suggested to use evolutionary computation (e.g. genetic algorithms) to implement your solver. Alternatives are with integer programming but this is not recommended because will likely be too slow.
    - The solver should generate a solution in a **short amount of time**.
        - The magic square solver should solve a 20x20 magic square in less than 5 minutes. **Full marks will only be given if it can solve a 20 x 20 magic square in less than 1 second** on average in 30 runs on a standard laptop and 10 seconds for a 200 x 200 square.
        - The Sudoku solver should be faster.
        - Provide a table of results which show the results of a doubling experiment in which you double the size of the problem and measure the average time taken by the solver to find a solution for (completed) runs of up to 1 hour each. Provide also a model to estimate the time needed to solve much larger problems based on these statistics.
    - While the solver is running the GUI should not just be frozen, it should update to show some progress (e.g. a partial solution)
    - It should be possible for the user to stop the solver at any time during its running and see the current best solution if the solver was taking too long.
    - The user should also be able to start the solver to finish their current in progress solution.
    - Allow a user to set constraints on the values of certain elements in the matrix. EG to add another constraint on the location of the value 1 to be at index 1,1 in the matrix.

Runtime requirement: your solver should go from a square like this to the one below in less than 1s.



4010	210	610	1010	1410	1810	2210	2610	3010	3410	3810	4210	4610	5010	5410	5810	6210	6610	7010	7410	7810	4010
3820	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	3820
3840	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	3840
3860	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	3860
3880	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	3880
3900	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	3900
3920	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	3920
3940	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	3940
3960	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	3960
3980	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	3980
4000	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	4000
4020	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	4020
4040	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	4040
4060	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	4060
4080	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	4080
4100	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	4100
4120	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	4120
4140	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	4140
4160	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	4160
4180	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	4180
4200	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	4200
4010	210	610	1010	1410	1810	2210	2610	3010	3410	3810	4210	4610	5010	5410	5810	6210	6610	7010	7410	7810	4010



4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010
4010	15	290	331	353	28	188	294	394	128	209	236	19	157	17	137	371	44	374	387	38	4010
4010	257	309	39	20	154	57	343	56	258	297	116	219	33	119	256	203	334	291	269	380	4010
4010	91	224	169	263	370	52	107	126	42	363	214	314	289	381	234	212	76	89	99	295	4010
4010	143	210	207	16	344	162	106	328	176	275	141	193	350	187	276	231	307	133	50	175	4010
4010	223	329	172	259	83	317	204	392	81	63	142	43	216	372	66	104	281	74	238	351	4010
4010	153	199	45	251	90	121	336	301	155	65	26	360	215	270	293	277	274	93	240	246	4010
4010	35	278	338	101	14	305	357	189	226	326	388	235	173	122	37	109	160	183	288	146	4010
4010	164	273	315	123	165	87	108	318	378	95	323	333	92	148	247	324	80	40	267	130	4010
4010	347	321	166	2	245	253	233	208	72	71	49	69	384	264	386	136	139	135	184	346	4010
4010	340	149	364	110	102	118	319	31	322	300	356	265	114	29	220	46	47	161	250	367	4010
4010	30	5	94	398	221	373	138	260	112	248	266	262	111	271	191	36	86	369	359	180	4010
4010	127	134	97	306	292	105	24	185	393	58	145	348	181	10	396	379	131	53	342	304	4010
4010	366	298	32	197	205	261	182	103	73	192	390	120	51	399	88	280	61	397	186	129	4010
4010	335	48	365	218	195	339	9	150	255	77	152	100	361	115	286	98	222	327	4	354	4010
4010	132	254	332	125	391	299	229	84	82	3	358	60	174	228	170	376	163	383	140	27	4010
4010	362	217	6	316	355	124	242	67	239	23	13	225	279	244	202	311	385	308	22	70	4010
4010	296	147	7	213	168	341	206	55	232	312	68	400	211	41	159	313	389	156	179	117	4010
4010	283	54	349	302	194	64	11	375	177	320	285	249	303	284	1	8	395	196	85	75	4010
4010	167	158	230	200	243	62	310	190	272	368	330	178	34	241	78	25	377	21	325	201	4010
4010	144	113	352	237	151	382	252	198	337	345	12	18	282	268	287	171	59	227	96	79	4010
4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010

Constraint requirement: The user should be able to fix some portions of the matrix such as the 1-9 here, the final solution will respect the locations that are set (nb if the user constraints result in an impossible to find solution it will still be possible for the user to stop the program and see the current best result as is also required).

4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010
4010	351	377	61	332	53	170	321	63	202	119	379	10	101	278	185	88	341	14	382	183	4010
4010	28	51	100	307	369	44	47	318	68	388	198	288	111	135	82	371	234	291	252	328	4010
4010	157	112	214	151	253	221	228	72	364	1	2	3	393	71	165	389	340	323	339	212	4010
4010	325	355	261	76	75	293	142	335	338	4	5	6	315	125	230	244	192	36	308	345	4010
4010	376	146	245	370	305	69	400	224	258	7	8	9	374	147	271	153	116	162	150	220	4010
4010	295	194	336	254	181	158	106	218	215	161	380	15	275	303	309	85	316	23	27	159	4010
4010	42	306	331	349	56	270	208	207	274	81	184	237	216	164	128	41	43	200	320	353	4010
4010	24	144	363	175	203	50	395	241	375	204	179	31	20	201	330	301	49	93	399	233	4010
4010	80	209	195	394	250	113	52	263	78	385	57	350	327	108	264	104	280	94	105	302	4010
4010	240	143	109	107	34	356	285	259	296	299	117	312	186	59	97	262	26	373	140	310	4010
4010	38	131	329	16	66	114	384	99	77	222	197	392	206	358	22	282	359	372	174	172	4010
4010	90	133	126	383	333	25	40	155	236	348	190	366	17	37	98	357	21	260	397	398	4010
4010	229	91	168	255	173	269	360	287	39	346	396	281	86	176	166	102	289	156	130	11	4010
4010	265	122	313	210	180	367	73	378	137	152	149	129	46	361	272	132	317	64	256	87	4010
4010	273	381	120	65	96	311	268	32	227	199	193	238	171	167	386	211	138	297	219	18	4010
4010	342	154	189	12	226	286	60	279	248	337	314	67	177	292	284	58	29	391	83	182	4010
4010	187	70	243	141	298	136	118	354	239	169	232	223	387	235	231	79	242	257	139	30	4010
4010	322	334	89	148	160	368	33	115	55	362	163	390	110	246	145	326	213	103	123	205	4010
4010	251	267	134	191	347	266	196	19	249	48	343	344	365	247	62	121	277	225	13	45	4010
4010	95	290	84	74	352	124	294	92	35	178	324	319	127	300	283	304	188	276	54	217	4010
4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010	4010

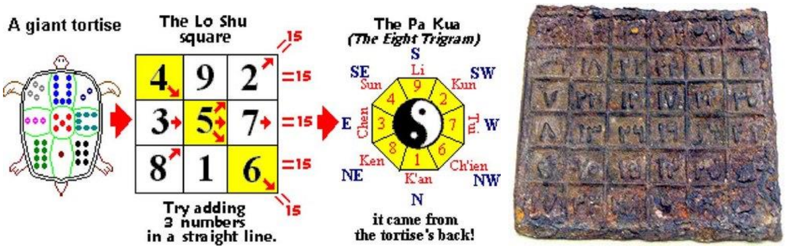
Format of the solver result table requirement:

N (Magic square size)	Runtime (average of 30 runs)
5	0.9 +/- 0.05
10	2.1 +/- 0.01
20	3 +/- 0.01
40	4.01 +/- 0.5

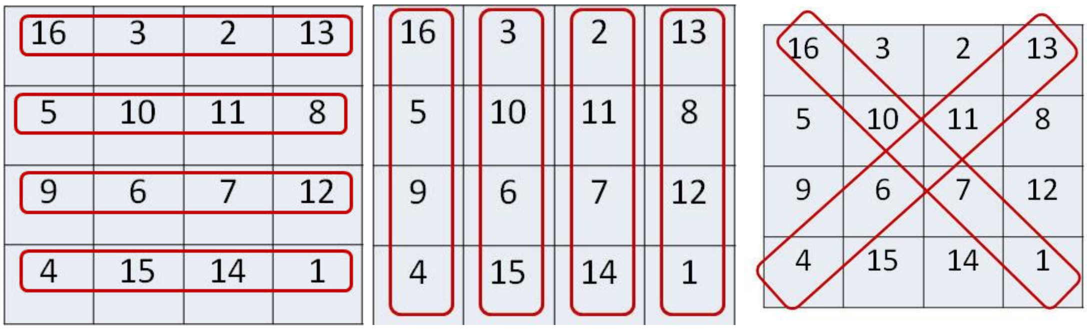
This table shows an example of the result table that should be provided for each solver. This example shows an example of the results that would be obtained if the algorithm ran in O(lg n + 1). Your algorithm will most likely not do this, but see if you can provide an estimate (see the textbook for further description of “doubling experiments”. The 95% confidence intervals shown should be found from 30 test runs.

Magic Squares:

Magic squares are a square matrix arrangement of n x n integers from 1 to n squared. They have an ancient heritage and here are some magic squares from ancient Chinese civilizations:



The rules are constraints on the values that require that all rows, columns and diagonals add up to the same value as shown here:



The size of the square can be any value.

Sudoku

Sudoku is another, related, type of matrix puzzle with different rules (constraints). The objective is to fill a 9x9 grid with the numbers 1 – 9 so that each column, row and diagonal contains all the digits 1 to 9. The square with 9 digits is placed in a grid of 6 3x3 grids (see figure below). Each row, column, and diagonal in the larger grid also contains the values 1 – 9 as well (see figure below).

In Soduku, a partially filled board is provided by a puzzle setter, the solver has to place the remaining values: (see picture below).

	7	1		9		8		
			3		6			
4	9					7		5
	1		9					
9		2				6		3
					8		2	
8		5					7	6
			6		7			
		7		4		3	5	

3	7	1	5	9	4	8	6	2
5	2	8	3	7	6	1	9	4
4	9	6	2	8	1	7	3	5
6	1	4	9	2	3	5	8	7
9	8	2	7	1	5	6	4	3
7	5	3	4	6	8	9	2	1
8	4	5	1	3	9	2	7	6
2	3	9	6	5	7	4	1	8
1	6	7	8	4	2	3	5	9