

## Computational Many-Body Physics – Sheet 1

To solve this sheet we have wrote the attached code that is divided into five main sections (as we can see in figure 1). The first one is only there to import the important packages that we need. Section 2&3 are there to solve Rule 30. And finally, section 4&5 are there to solve the game of life.

```

2  # Importing Packages
3  from subprocess import list2cmdline
4  from tkinter import N, OFF
5  from tkinter.messagebox import YES
6  import matplotlib.pyplot as plt
7  from matplotlib import colors, figure
8  import numpy as np
9
10
11
12  # RULE 30 CODE
13
14  > def Newgeneration(numberofcells, generations):...
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  > def plotting(list, gridsize=[10,10]):...
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68  > def countcells(numberofcells, generations):...
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87  # Example Code - Rule 30
88
89  # example for getting the final result of rule 30 as a list:
90  Newgeneration(120,50)
91
92
93
94  # example for plotting the final rule 30 grid:
95  plotting(Newgeneration(120,50), [100,100])
96
97  # example for counting the alive cell in each generation:
98  countcells(120, 50)
99
100
101
102  # GAME OF LIFE CODE
103
104  # the rule:
105  > def Gol_Rule(oldgenmat, index):...
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122  > def Gol_application(configuration = [(1,3),(2,3)],size = (20,20),generations = 5, plot = True):...
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  > def countingcells(configuration = [(1,3),(2,3)],size = (20,20),generations = 5):...
209
210
211
212
213
214
215
216
217
218
219
220
221
222  # Example Code - Game of Life
223
224  # application of o
225  Gol_application(configuration=[(1,2),(2,3),(3,1),(3,2),(3,3)],size = (10,10), generations= 20)
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
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

Figure 1: Showing the different sections of our code

### Exercise 1: Rule N:

#### a) Writing a code that produces rule 30 for 120 generations:

To do so, we have defined in section 2 of our code two functions: “New\_generation()” and “plotting()”. The first function will create a list of lists, each represent a generation. And we do so by the following steps:

- Define the rule 30 condition as a list.
- Creating the starting point (all zero except the middle cell)
- Create a loop that loop through all cells, and check if the condition is satisfied. And we wrap this rule with a loop for the amount of generations that we should create.
- And saving the end result in a list of lists called “endplot”.

And then the second function is just for representing the list of lists as grid with blue cells representing the alive cells and each row representing a new generation.

### Running the code:

you can run the code by simply running the function “Plotting()” with two inputs: the first being our “new\_generations(cell numbers, #generations)” function, which gives us the list of lists that we want. And the second input is the size of our figure. When we ran it for our specific example, we got the results that are shown

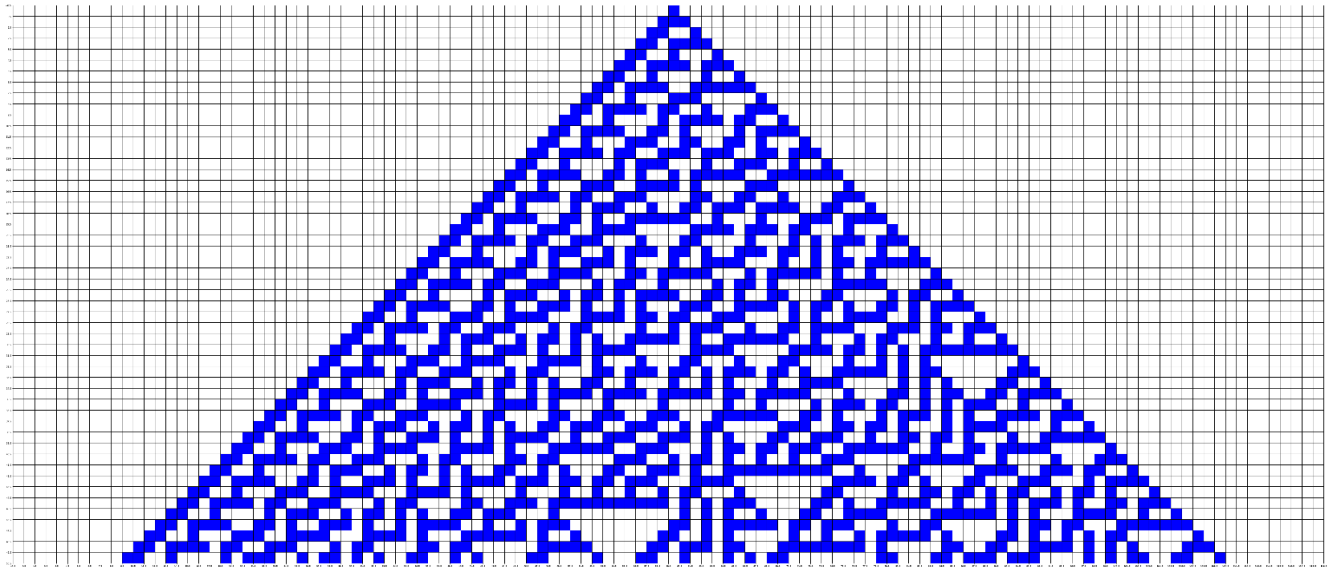


Figure 2: Showing Rule 30 applied for 50 generations.

### **b) Counting the alive cells in each generation:**

To do this, we have created a function called “countcells()” which takes two inputs: number of cells and the number of generations. At the end of this function we will have a list of the alive cells at each generation. and to represent that, we thought it would be best to do a simple scatter plot of that list (figure 3).

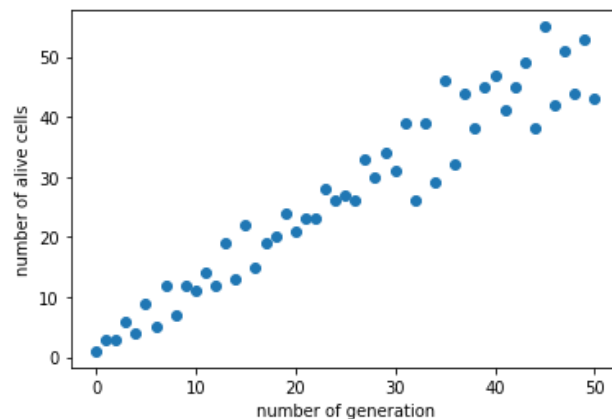


Figure 3: Showing the number of the alive cells in each generation.

### **c) The rule that reproduces the exact result for different configurations:**

After discussing with our classmates, we didn't really understand the question. But if by configuration is meant the initial configuration where the state begins, then we believe it is the 256 rule, where all the cells will turn alive on the first generation.

---

## Exercise 2: Game of life:

This exercise is solved using section 4&5 of our code.

### a) Creating the game of life code for different initial configurations:

To do so, we have defined two functions: "GoL\_Rule()" and "GoL\_application()". Let's start with the former.

"GoL\_Rule()": This function returns a Boolean value depending if a certain cell will be dead or alive in the next generation and we did so in the following way:

- Create an array (under the name "effectmat") that represents the effective cells that would affect a certain cells (its immediate 8 neighbors).
- To count for boundary conditions in this part, we were able to do a modular addition to the indices so that it would wrap around the matrix.
- Summing over this effectmat, and checking if the value follows the game of life rules. Depending, of course, on the state of the cell beforehand (if it was dead or alive).

Now, this function doesn't do anything but check if the input cell is going to be dead or alive in the next generation.

To do the actual loop over all cells and over all generations, we have created the second function

"GoL\_application()". We did so in the following manner:

- Creating a for loop that goes through all the cells and checking if "GoL\_Rule()" will be True or False.
- After checking that for each cell. We have created a list for the alive cells (aliveindices) , and a list for the dead cells (deadindices) .
- Each of those cells contain only the indices of the cell in question.
- After having this list. We create a new array out of these two lists and then plot it (using our old plotting function).
- And then we loop over all the required generations.

And as a result of this function, we were able to apply it to the following initial configuration and get the following results (row 1: initial configuration. Row 2: first generation):

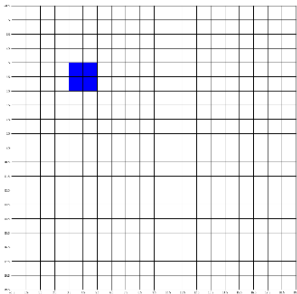


Figure 4: after running this for five generations nothing changed in the configuration

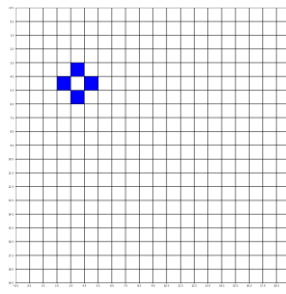


Figure 5: after running this for five generations nothing changed in the configuration

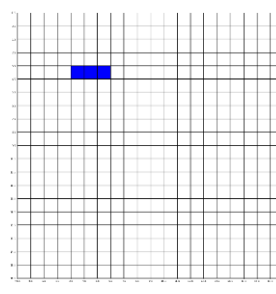


Figure 6: after running this for five generations, we noticed the periodic flipping between those two configs.

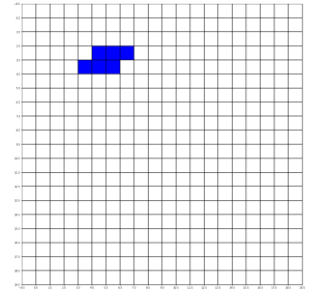


Figure 7: after running this for five generations, we noticed the periodic flipping between those two configs.

b) **The glider:**

For this question, we had everything ready. We just needed to run our function, and see what happens. As we can see from the picture, the initial shape is repeated every 4<sup>th</sup> generation with a sliding toward the right bottom corner.

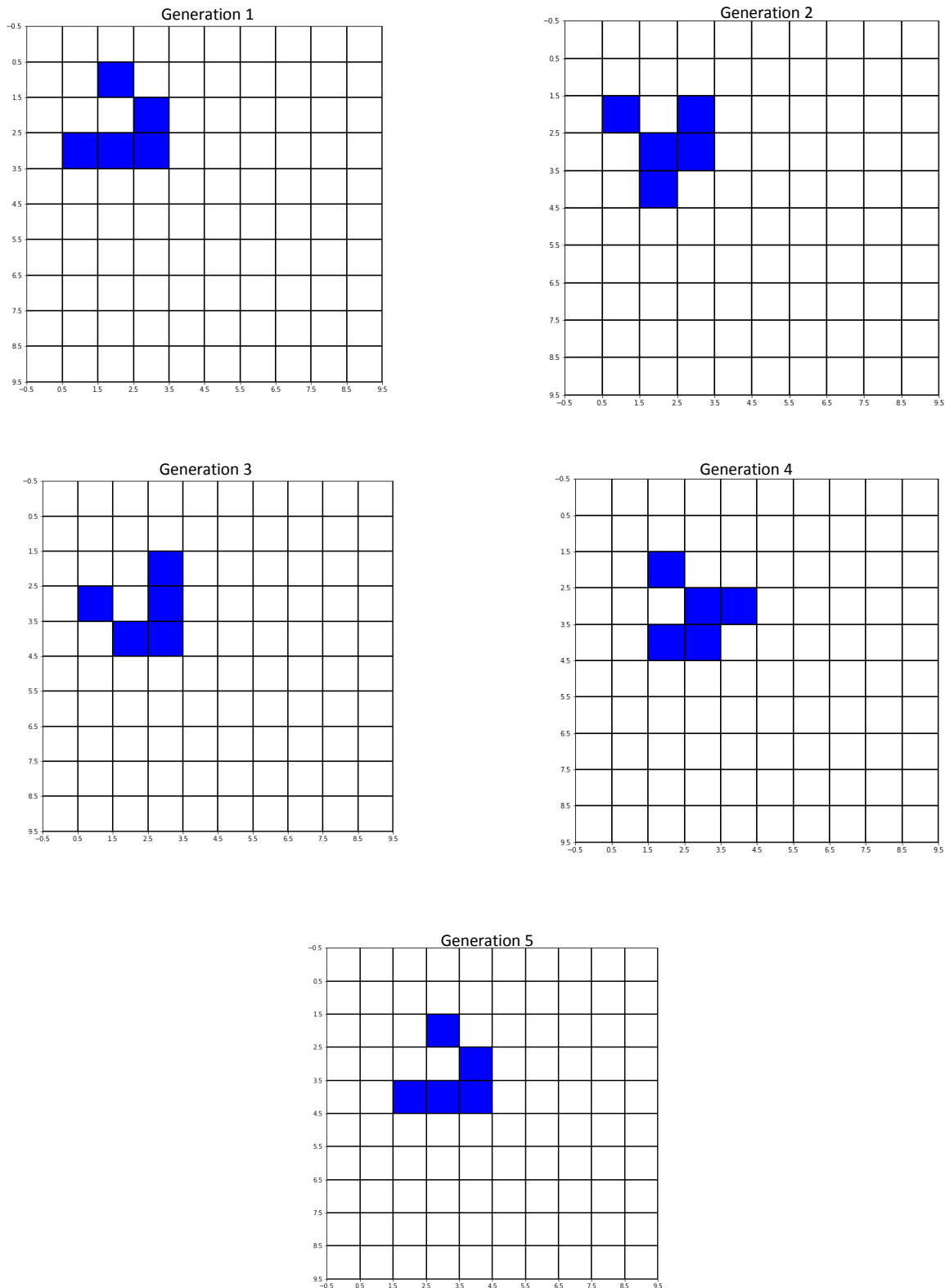


Figure 7: Showing the first 5 generations of the glider.

c) **Calculating the number of alive cells in a “diehard”:**

Just like part b in exercise 1, we will represent this using a scatter plot where we have used our game of life functions to check the alive cells at each generation. and we got the following result:

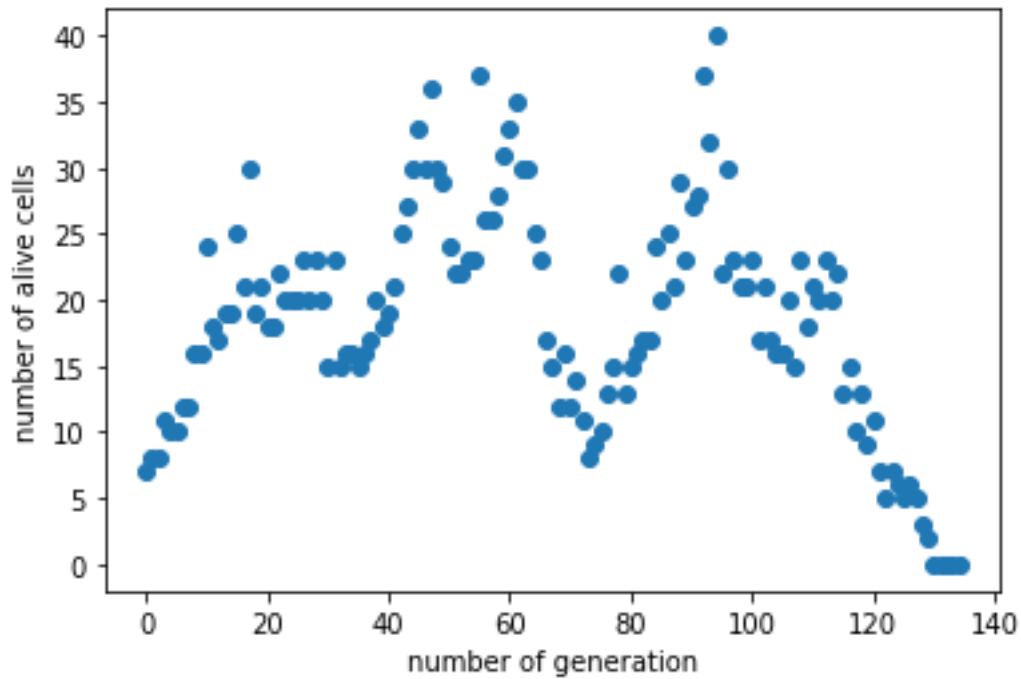


Figure 8: showing the number of alive cells at each generation for a diehard initial configuration.

As we can see, the number of alive cells goes to zero at the 130<sup>th</sup> generation.

---