Comparative study of field of view algorithms for 2D grid based worlds

v1.2

J.C.Wilk, February 2009

Contents

1. Introduction

Computing field of view is a frequent problem in video games. It consists of the determination of which part of the world is visible from a certain position. This paper focus on 2D grid based worlds, in other words, worlds represented by a 2D grid of square cells. This type of worlds can be found in a range of video games, including but not limited to 2D isometric games and text console based games like roguelikes.

While there are lots of different algorithms to solve this problem, not much has been written about the difference between those algorithms, in term of gameplay or speed.

This is not a comprehensive study of all available algorithms. Instead, I will focus on following popular or innovative algorithms :

- basic raycasting [BASIC]
- diamond raycasting [DIAMOND]
- recursive shadowcasting [SHADOW]
- precise permissive fov [PERMISSIVE]
- digital fov [DIGITAL]

Note that basic raycasting alone generates too many artifacts to be really usable. The algorithm tested here use a post-processing step to remove most wall lighting artifacts.

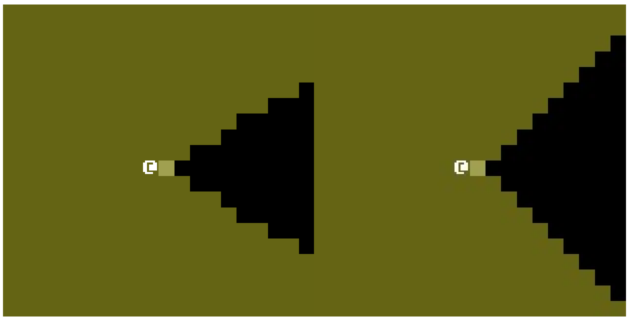
In this version of the study, permissive has been replaced by an enhanced version of the algorithm, from the same author, with a variable permissiveness parameter. This parameter can take values in the range 0-8, 0 being the less permissive and 8 being equivalent to the standard precise permissive algorithm.

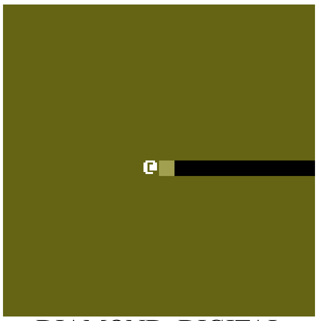
All tests below are done using the C++ wrapper for the Doryen library [LIBTCOD] and the fov algorithms implementations it contains. Note that digital fov is no more in the library, but its source code is still on libtcod's svn repository.

2. Gameplay study

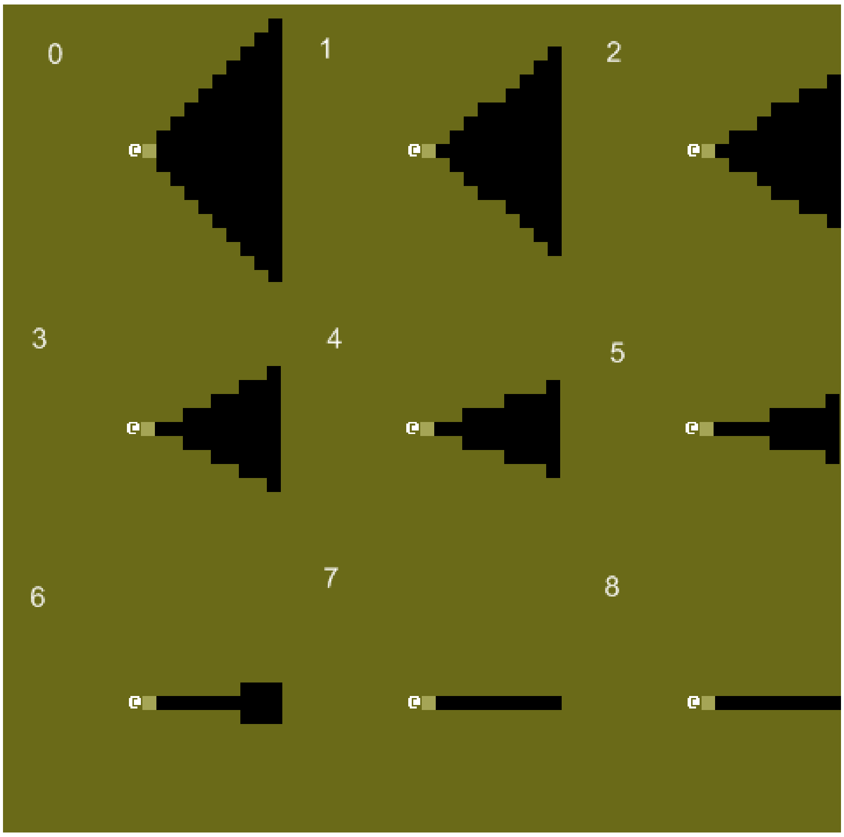
2.1. Pillar behavior

We call a wall cell surrounded with empty cells a pillar. First, let's see how the fov looks when the viewer's cell is adjacent to the pillar :





BASIC SHADOW DIAMOND, DIGITAL

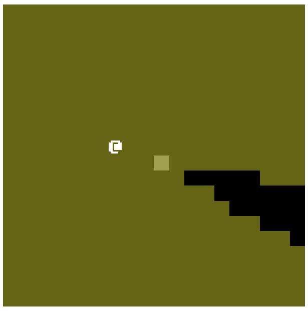


PERMISSIVE

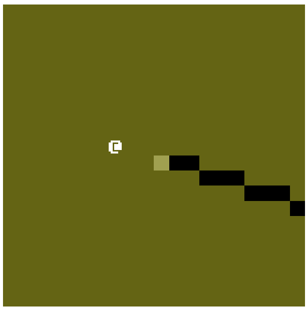
DIAMOND and DIGITAL result in a shadow limited to a single line, which makes it almost impossible for a creature to sneak from the east side of the map to the @ position without being noticed.

PERMISSIVE offers any shadow angle. In particular, PERMISSIVE0 has the same shadow angle as SHADOW. PERMISSIVE2 the same as BASIC.

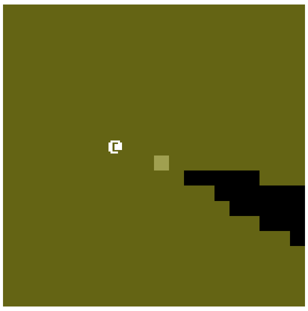
Now let's see the behavior when the viewer is a few cells away from the pillar :



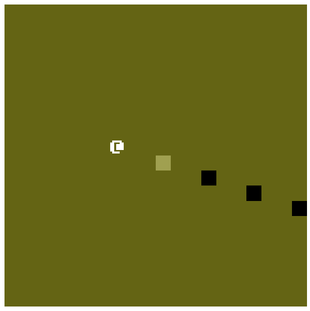
BASIC



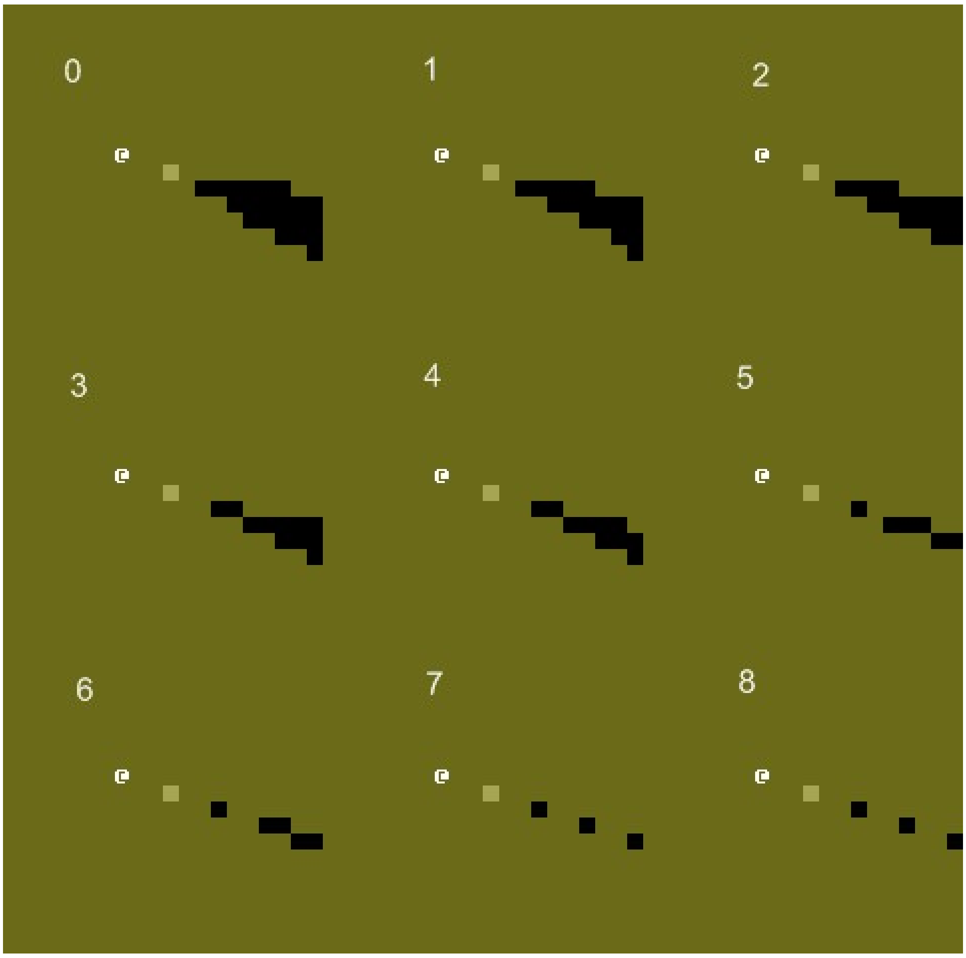
DIAMOND



SHADOW



DIGITAL



PERMISSIVE

BASIC and SHADOW still have a triangular shadow, but the cells near the pillar are now completely in the field of view.

DIAMOND still has a shadow limited to a single line.

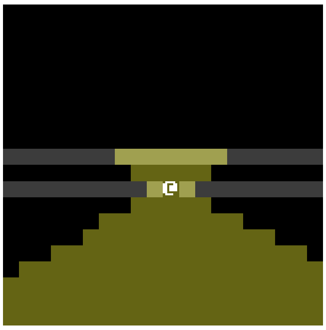
DIGITAL don't even have a line shadow.

Once gain PERMISSIVE is very similar to SHADOW/BASIC with low permissiveness and is equivalent to DIGITAL with maximum permissiveness.

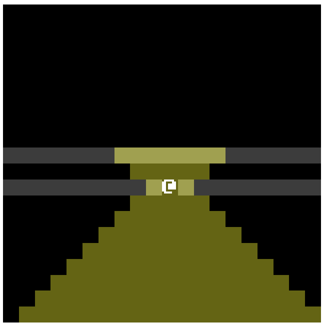
The conclusion is that if you want to use pillars for stealth gameplay, you have to choose either BASIC or SHADOW or PERMISSIVE with low permissiveness. You can still use the other algorithms provided you use 2x2 pillars instead of single cell pillars. Not that no algorithm provides a good looking shadow.

2.2. *Corner peeking*

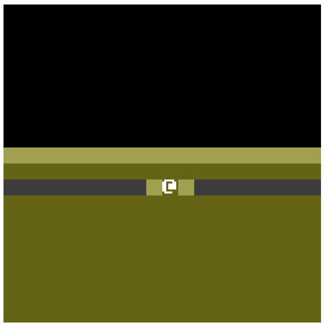
Corner peeking involves seeing a corridor when you're standing at a T junction.



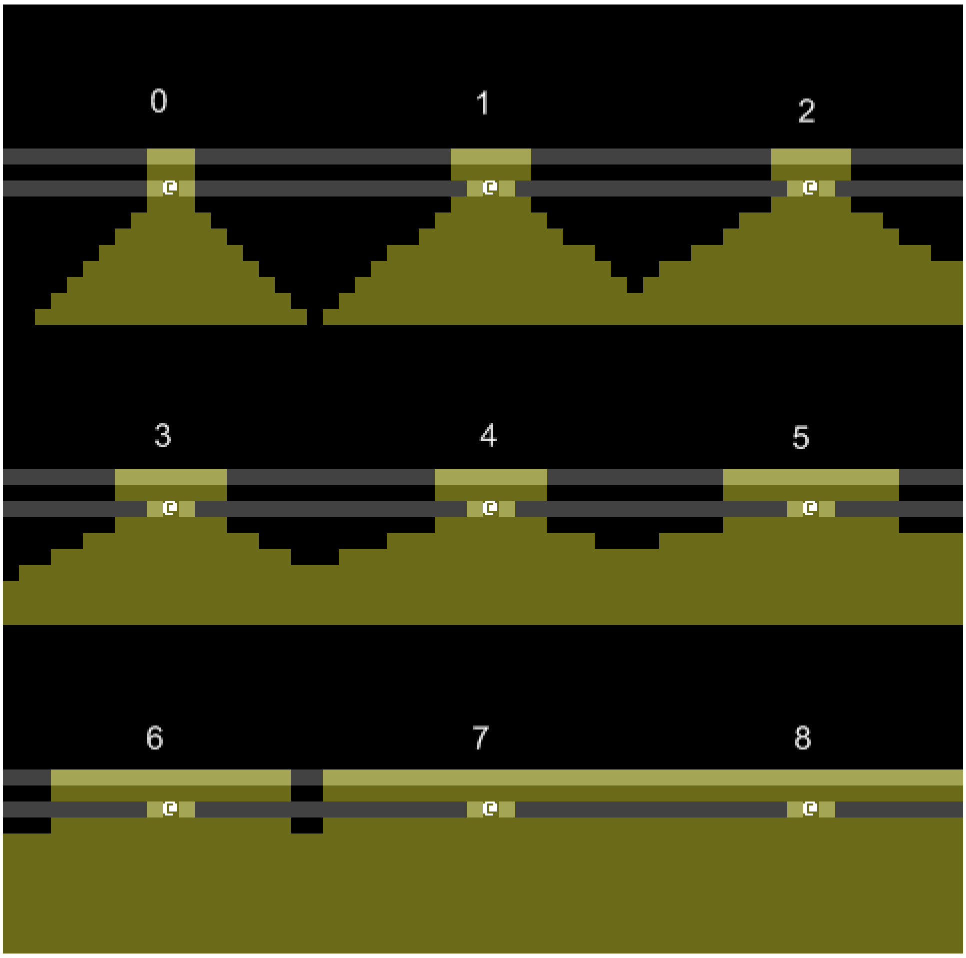
BASIC



SHADOW



DIAMOND, DIGITAL

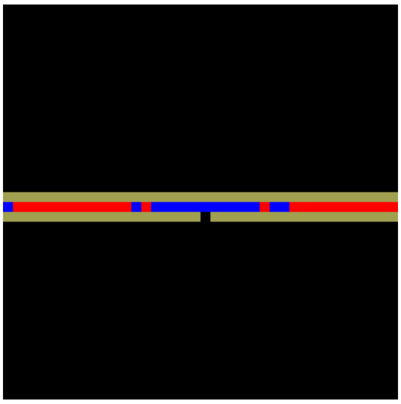


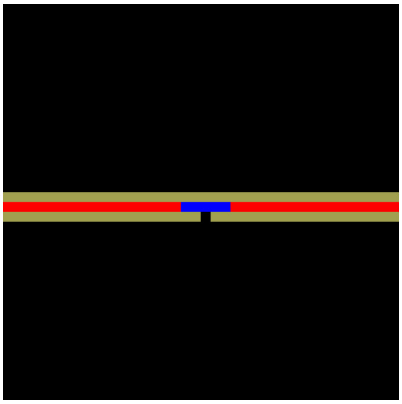
PERMISSIVE

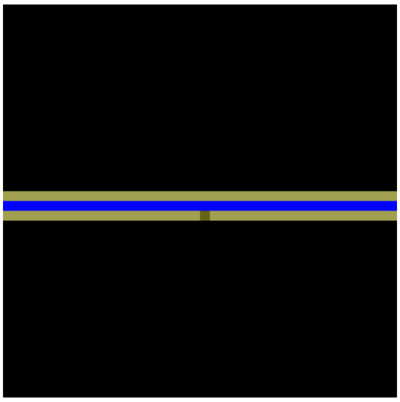
BASIC and SHADOW don't allow corner peeking. You have to step into the corridor to be able to see it. The other algorithms allow corner peeking.

Now let's see the opposite case. If you're in the corridor, will you be able to see someone hiding in the T junction ?

When on a red cell, you don't see him. When on a blue cell, you see him.


BASIC


DIAMOND


SHADOW, PERMISSIVE, DIGITAL

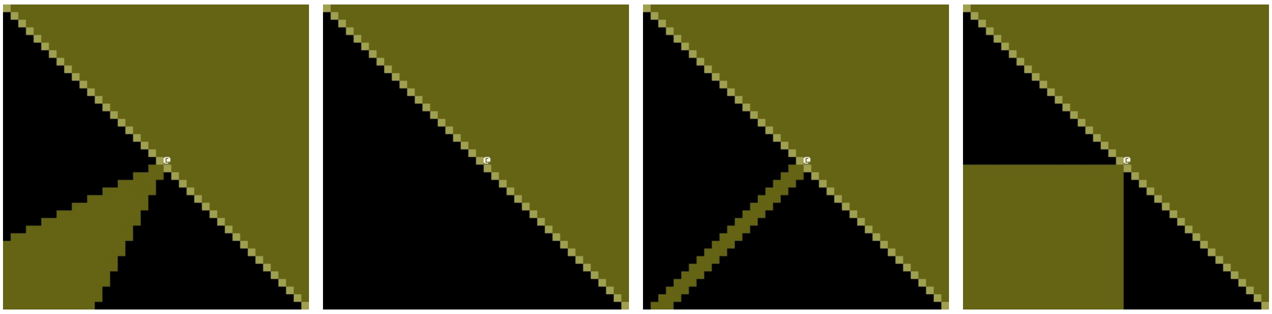
For BASIC, the cell might or might not be visible, depending on the viewer position in the corridor. Most of the time, it's not in fov, but at certain position where a ray pass exactly through the T junction cell, it is in fov. This is clearly not an acceptable behavior.

DIAMOND has the better behavior here : the cell is not visible until the viewer is really close to the junction.

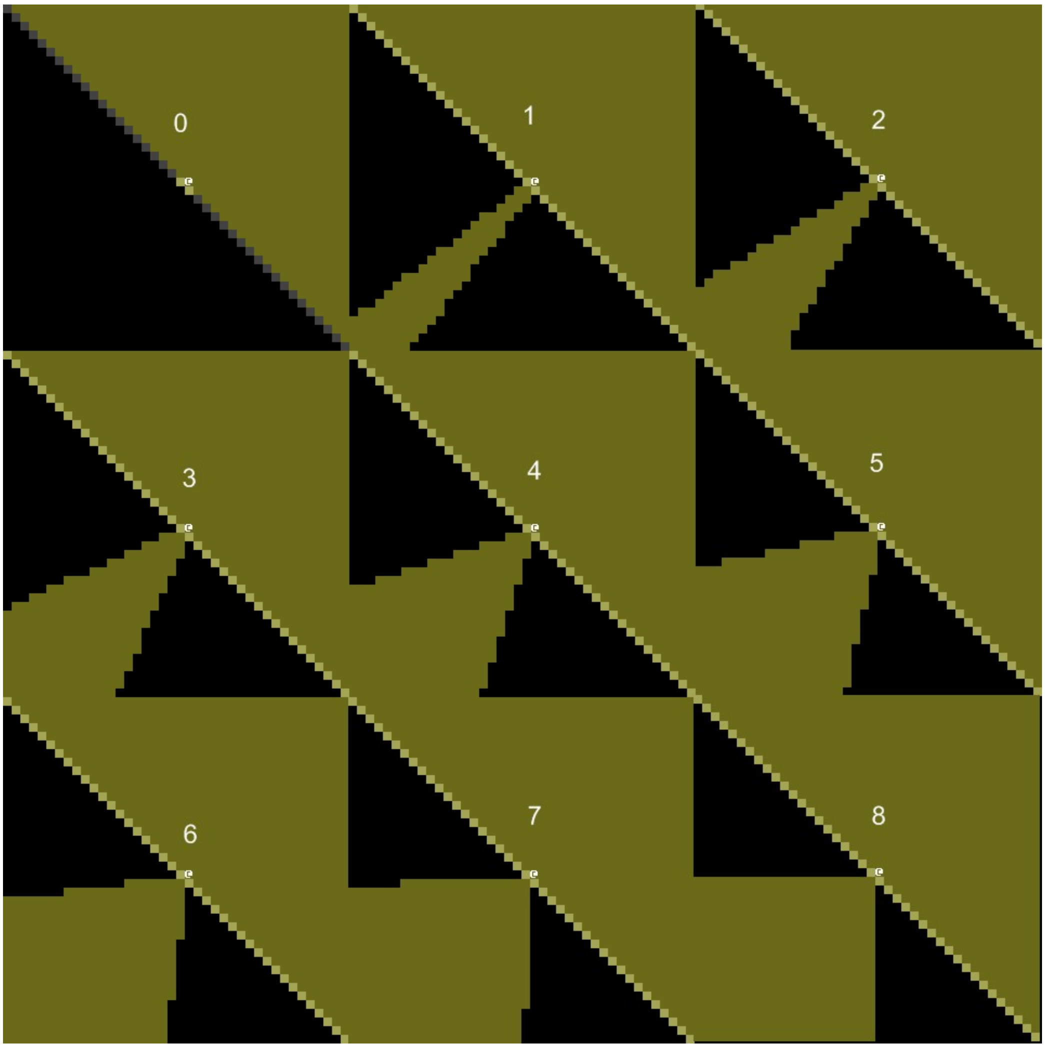
For other algorithms, the cell is always visible, which is counter-intuitive, but acceptable.

2.3. Diagonal walls

Most roguelikes allow diagonal movements. We can then expect the field of view to go through diagonal walls too.



BASIC DIAMOND SHADOW DIGITAL



PERMISSIVE

BASIC and some of the PERMISSIVE have good result.

DIAMOND completely blocks the field of view, which might be a real issue if your game allows diagonal movement. Also note that if you *want* the field of view to be blocked by diagonal walls, you'll have to tweak any algorithm except DIAMOND and PERMISSIVE0.

SHADOW through diagonal walls is limited to a thick line, which is not very natural.

PERMISSIVE8 and DIGITAL have a 90° field of view through the wall, which is fine.

2.4. *Symmetry*

The measures are done on following maps :

- “Outdoor” maps (empty maps with random 1x1 obstacles) : 100x100
- “Indoor” maps (random cave levels using algorithm [BSPDUNGEON]) : 40x40

Symmetry is measured by calculating a field of view F0 on a random map from a random position P0. Then, for each cell Pi in F0, we calculate the field of view Fi from the position Pi and check that P0 is inside Fi. If not, we increase an error counter. We do this for several random maps and get the average number of error per map cell.

Green cells have no symmetry error.

Orange cells have less than 1% symmetry errors.

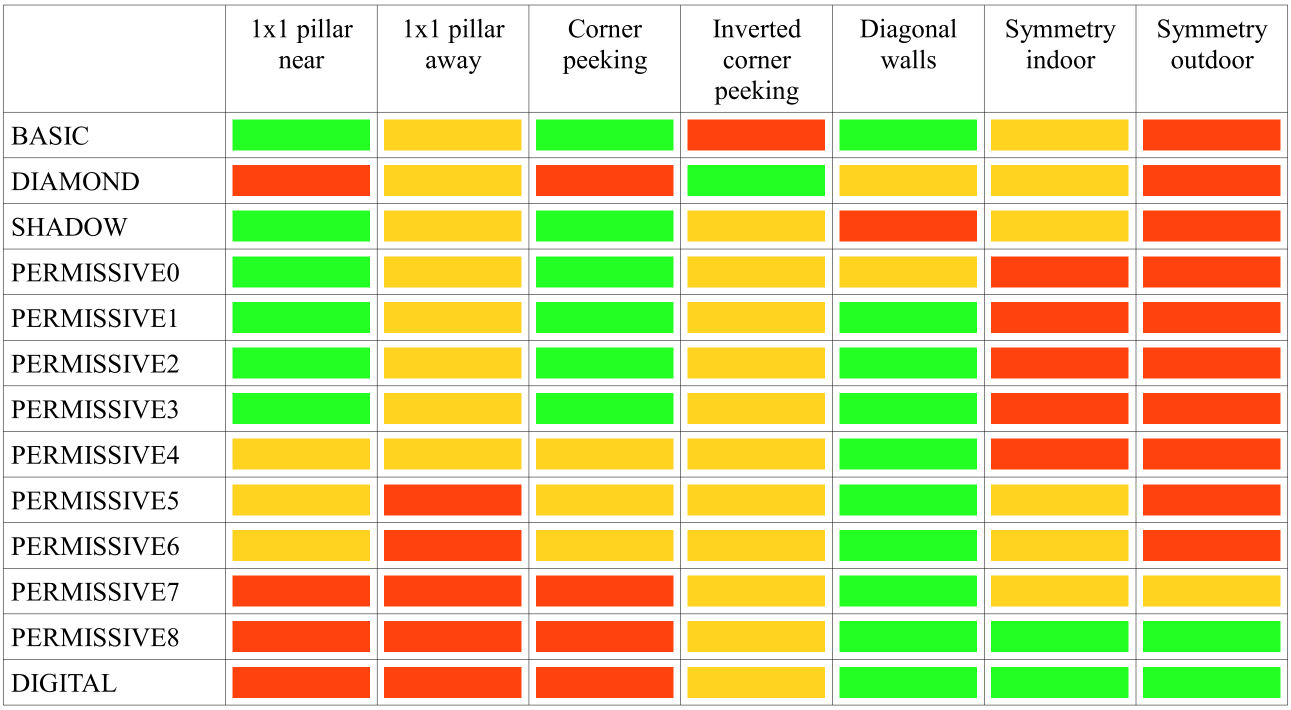
Red cells have more than 1% symmetry errors.

Algorithm	Error / cell – indoor (%)	Error / cell – outdoor (%)
BASIC	0.75	4.4
SHADOW	0.93	6.9
DIAMOND	0.82	6.5
PERMISSIVE 0	2.14	10.95
PERMISSIVE 1	2.06	10.5
PERMISSIVE 2	1.9	9.65
PERMISSIVE 3	1.39	8.1
PERMISSIVE 4	1.12	7
PERMISSIVE 5	0.84	4.1
PERMISSIVE 6	0.61	2.25
PERMISSIVE 7	0.16	0.3
PERMISSIVE 8	0	0
DIGITAL	0	0

Conclusion :

- Obviously, symmetric algorithms have no errors.
- On permissive, symmetry is inversely proportional to permissiveness. The symmetry is really bad with permissiveness <= 4.
- All other algorithms have equivalent error rates, acceptable for indoor, but not for outdoor. The outdoor error rate is high enough to be a gameplay issue.
- Error rate is higher in outdoor maps

2.5. *Gameplay summary*



Gameplay ranking :

BASIC	3	2	2
PERMISSIVE3	3	2	2
PERMISSIVE2	3	2	2
PERMISSIVE1	3	2	2
PERMISSIVE8	3	1	3
DIGITAL	3	1	3
SHADOW	2	3	2
PERMISSIVE0	2	3	2
DIAMOND	1	3	3
PERMISSIVE7	1	3	3
PERMISSIVE6	1	4	2
PERMISSIVE5	1	4	2
PERMISSIVE4	1	4	2

Conclusions :

- there is no perfect algorithm amongst the ones observed
- each algorithm has its own weakness
- the resulting ranking is rather arbitrary. You should carefully check every algorithm features to see if it can fit your game.

3. Performance study

The measures are done on following maps :

- Empty maps (worst case) : 600x600, 100x100, 20x20
- Maps full of wall (best case) : 600x600, 100x100, 20x20
- “Outdoor” maps (empty maps with random 1x1 obstacles) : 600x600, 100x100, 20x20
- “Indoor” maps (random cave levels using algorithm [BSPDUNGEON]) : 80x80, 40x40, 20x20

For each map type, we run 50 tests on 50 different random maps (for the worst/best cases, there's only one map for all 50 tests). For each random map, we run a number (between 10 and 2000 depending on the current test's average speed) of fov computations from different positions in the map. Each algorithm runs through the exact same set of maps/viewer positions. The cumulative time is calculated for each algorithm and the average time per computation is deduced.

The absolute speed values are not really significant. More important is the difference of speed between two algorithms on the same map and the same computer.

The color code uses following convention :

- algorithms with total time lower than 2x the fastest are green
- algorithms with total time higher than 5x the fastest are red
- the others are orange

Empty map	600x600 (µs)	100x100 (µs)	20x20 (µs)
BASIC	29000	589	35
SHADOW	16000	383	30
DIAMOND	91000	925	58
PERMISSIVE 0	27439	606	37
PERMISSIVE 1	27039	604	36
PERMISSIVE 2	27399	607	37
PERMISSIVE 3	27160	600	37
PERMISSIVE 4	27140	618	37
PERMISSIVE 5	26900	607	36
PERMISSIVE 6	26879	585	36
PERMISSIVE 7	27059	606	37
PERMISSIVE 8	26980	606	37
DIGITAL	148000	3958	166

Full map	600x600 (µs)	100x100 (µs)	20x20 (µs)
BASIC	2507	101	13
SHADOW	485	21	3
DIAMOND	6086	155	11
PERMISSIVE 0	893	111	10
PERMISSIVE 1	736	113	10
PERMISSIVE 2	731	110	10
PERMISSIVE 3	721	111	10
PERMISSIVE 4	750	111	10
PERMISSIVE 5	731	112	10
PERMISSIVE 6	726	111	10
PERMISSIVE 7	736	111	10
PERMISSIVE 8	747	111	11
DIGITAL	639	106	22

Outdoor map	600x600 (µs)	100x100 (µs)	20x20 (µs)
BASIC	9000	242	32
SHADOW	48000	309	35
DIAMOND	19000	318	48
PERMISSIVE 0	9989	280	34
PERMISSIVE 1	10777	303	36
PERMISSIVE 2	11594	320	36
PERMISSIVE 3	12093	336	38
PERMISSIVE 4	12564	338	38
PERMISSIVE 5	13043	354	39
PERMISSIVE 6	13621	358	40
PERMISSIVE 7	14133	367	40
PERMISSIVE 8	14563	375	41
DIGITAL	206000	4255	272

Indoor map	80x80 (µs)	40x40 (µs)	20x20 (µs)
BASIC	93	51	26
SHADOW	38	32	21
DIAMOND	130	67	40
PERMISSIVE 0	99	53	31
PERMISSIVE 1	102	55	32
PERMISSIVE 2	104	56	32
PERMISSIVE 3	104	56	33
PERMISSIVE 4	103	58	33
PERMISSIVE 5	105	58	35
PERMISSIVE 6	106	59	33
PERMISSIVE 7	107	59	36
PERMISSIVE 8	107	60	36
DIGITAL	440	277	135

Speed ranking :

SHADOW	11	0	1
PERMISSIVE	8	3	1
BASIC	8	3	1
DIAMOND	3	6	3
DIGITAL	1	1	10

Conclusions :

- with usual visible map size in games (between 20x20 and 40x40), any algorithm but DIGITAL is fast enough for most usages.
- SHADOW is the fastest on indoor maps and the best overall choice for performances.
- BASIC is the fastest on outdoor maps
- DIGITAL is way slower than the others.

4. Summary

	Gameplay			Speed			Complexity	
							To understand	To implement
BASIC	3	2	2	8	3	1	*	*
PERMISSIVE3	3	2	2	8	3	1	**	***
PERMISSIVE2	3	2	2	8	3	1	**	***
PERMISSIVE1	3	2	2	8	3	1	**	***
PERMISSIVE8	3	1	3	8	3	1	**	***
DIGITAL	3	1	3	1	1	10	***	***
SHADOW	2	3	2	11	0	1	*	*
PERMISSIVE0	2	3	2	8	3	1	**	***
DIAMOND	1	3	3	3	6	3	**	**
PERMISSIVE7	1	3	3	8	3	1	**	***
PERMISSIVE6	1	4	2	8	3	1	**	***
PERMISSIVE5	1	4	2	8	3	1	**	***
PERMISSIVE4	1	4	2	8	3	1	**	***

Conclusions :

- There is no big winner, but SHADOW and BASIC are particularly adapted to most FOV usages except if symmetry is mandatory. They also happen to be the simplest to understand and implement.
- The new permissive fov is pretty handy and can adapt to any usage, but no permissiveness level gives perfect results and the symmetry gets really bad for low permissiveness.
- While having a reputation for being the slowest, BASIC is indeed one of the fastest.
- While it does not rank very well in this study, DIAMOND has some very interesting and unique features and it's definitely worth digging more into it to see if it can be improved.
- The final conclusion is that there is still lot of room for improvement in FOV algorithms, especially on the gameplay side...

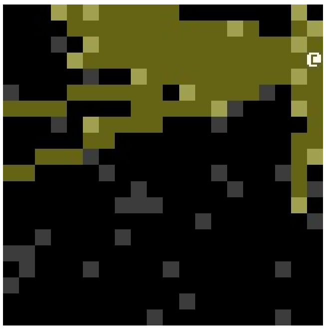
5. References

1. [BASIC] J.C.Wilk, Sep 2007. Piece of cake visibility determination algorithm : http://jice.nospam.googlepages.com/visibilitydetermination
2. [DIAMOND] Modeling Rays for Line of Sight in an Object-Rich World : http://www.geocities.com/temerra/los_rays.html
3. [SHADOW] Björn Bergström, 2001. FOV using recursive shadowcasting : http://roguebasin.roguelikedevelopment.org/index.php?title=FOV_using_recursive_shadowcasting
4. [PERMISSIVE] Jonathon Duerig. Precise Permissive Field of View : http://roguebasin.roguelikedevelopment.org/index.php?title=Precise_Permissive_Field_of_View
   Enhanced version :
   http://groups.google.com/group/rec.games.roguelike.development/msg/b77fe6999651d023
5. [DIGITAL] Digital field of view : http://roguebasin.roguelikedevelopment.org/index.php?title=Digital_field_of_view
6. [LIBTCOD] J.C.Wilk, 2007-2009. The Doryen library : http://thedoryenlibrary.appspot.com
7. [BSPDUNGEON] J.C.Wilk, Sep 2007. Basic dungeon generation : http://jice.nospam.googlepages.com/basicdungeongeneration

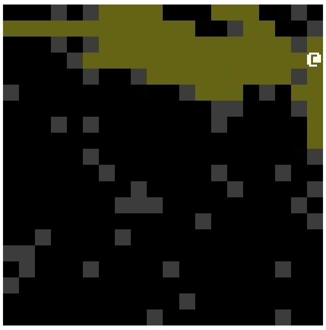
6. Appendix

Some nifty screenshots.

6.1. Outdoor 20x20

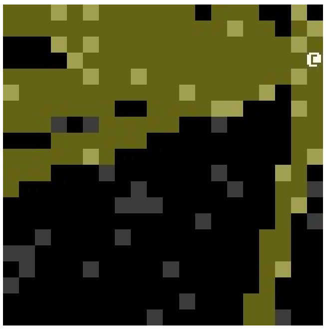


BASIC

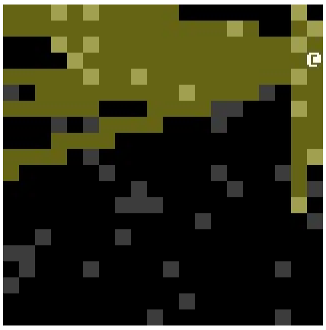


DIAMOND

PERMISSIVE

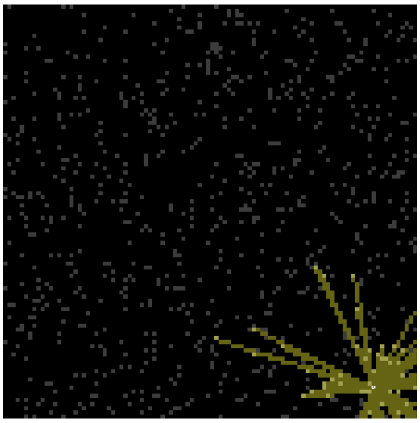


DIGITAL

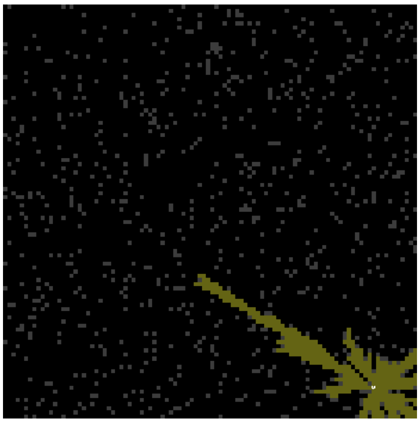


SHADOW

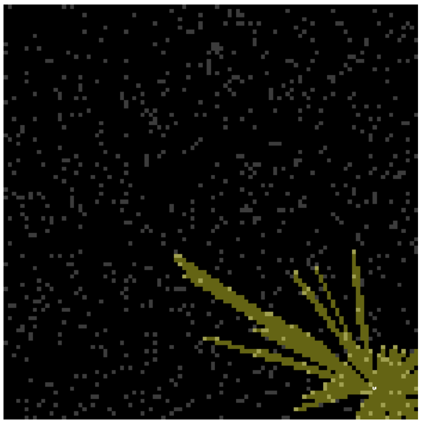
6.2. Outdoor 100x100



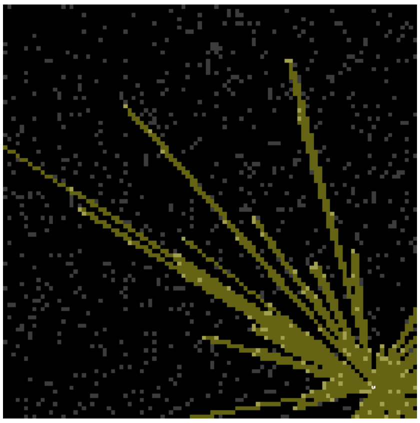
BASIC



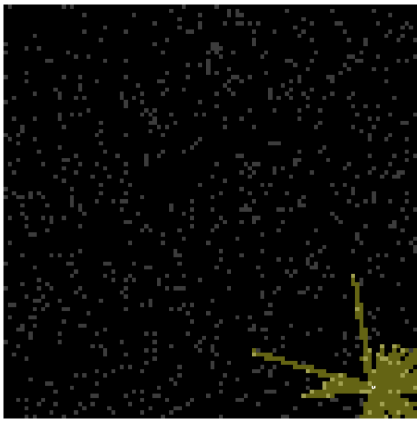
DIAMOND



PERMISSIVE

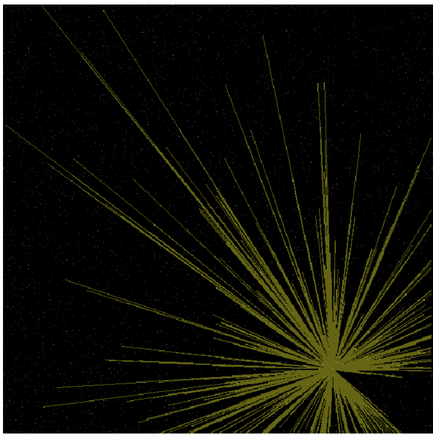


DIGITAL

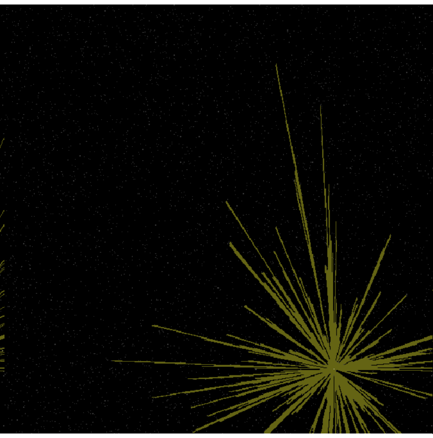


SHADOW

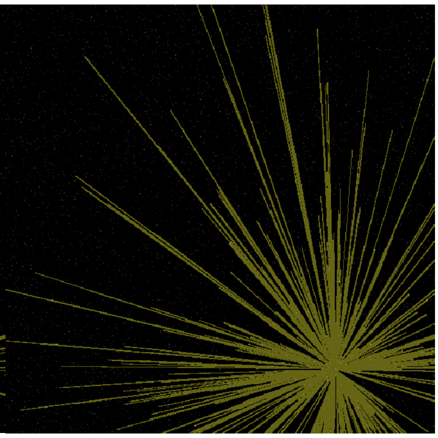
6.3. Outdoor 600x600



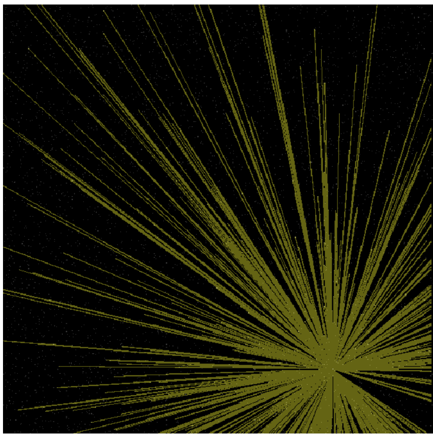
BASIC



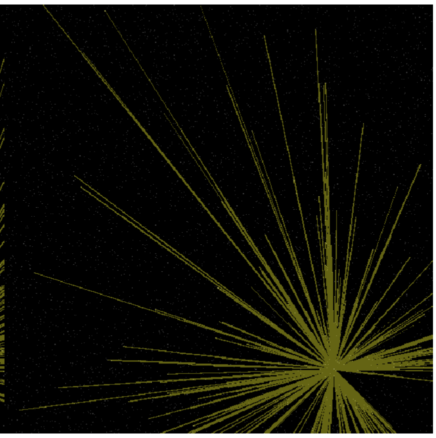
DIAMOND



PERMISSIVE

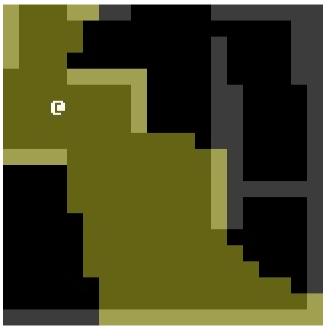


DIGITAL

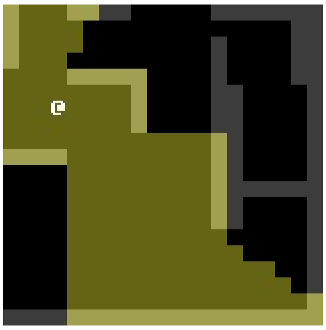


SHADOW

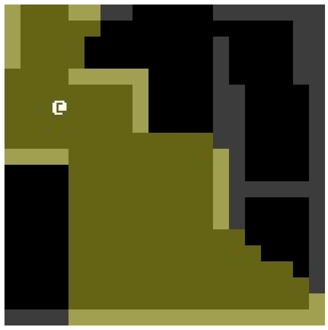
6.4. Indoor 20x20



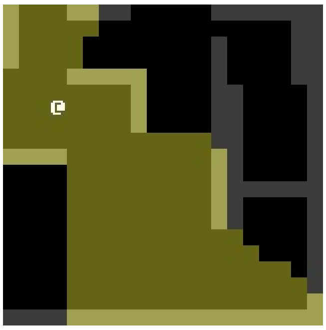
BASIC



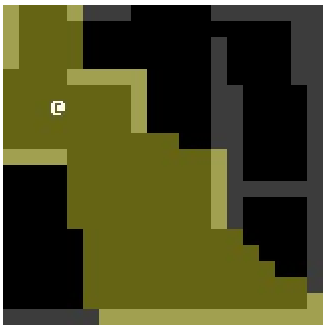
DIAMOND



PERMISSIVE

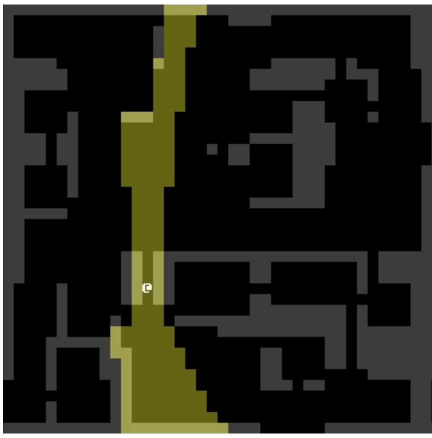


DIGITAL

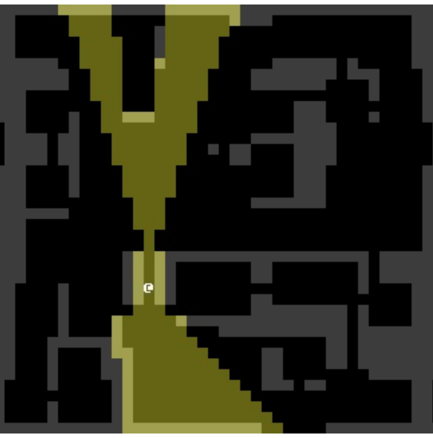


SHADOW

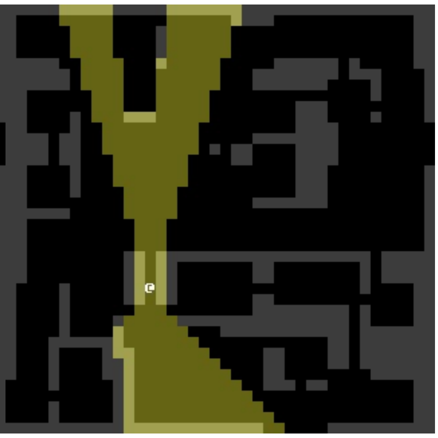
6.5. Indoor 40x40



BASIC

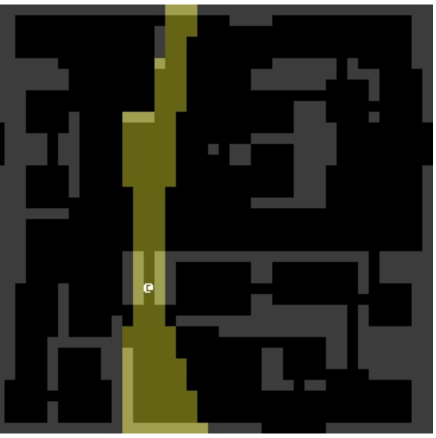


DIAMOND



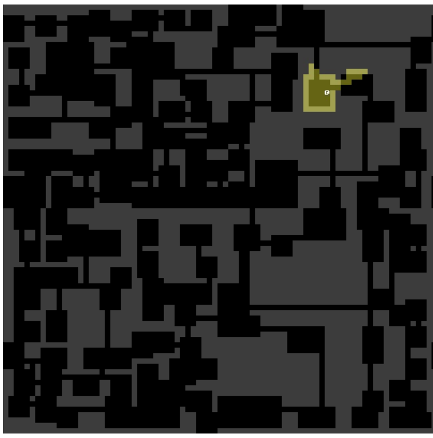
PERMISSIVE

DIGITAL

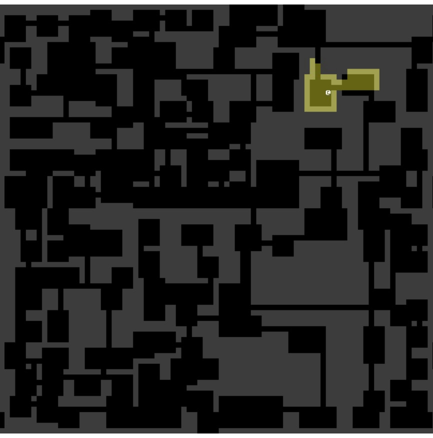


SHADOW

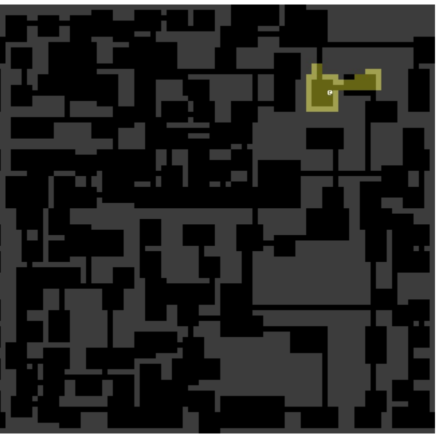
6.6. *Indoor 80x80*



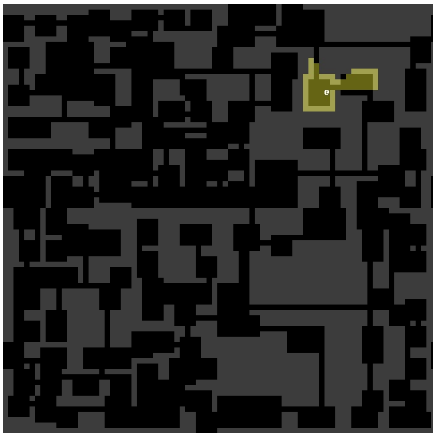
BASIC



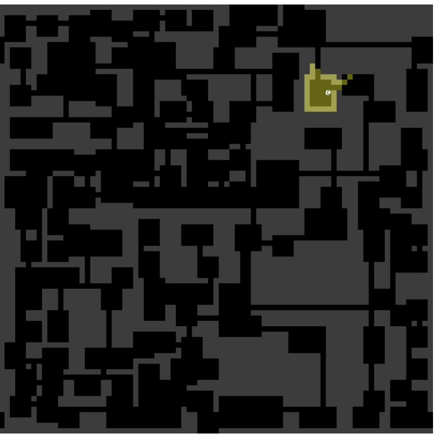
DIAMOND



PERMISSIVE



DIGITAL

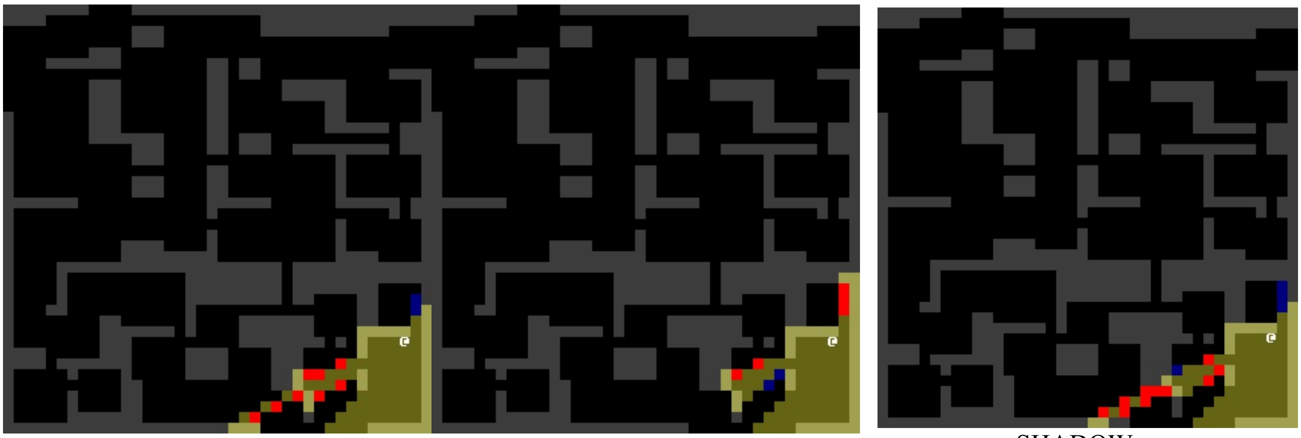


SHADOW

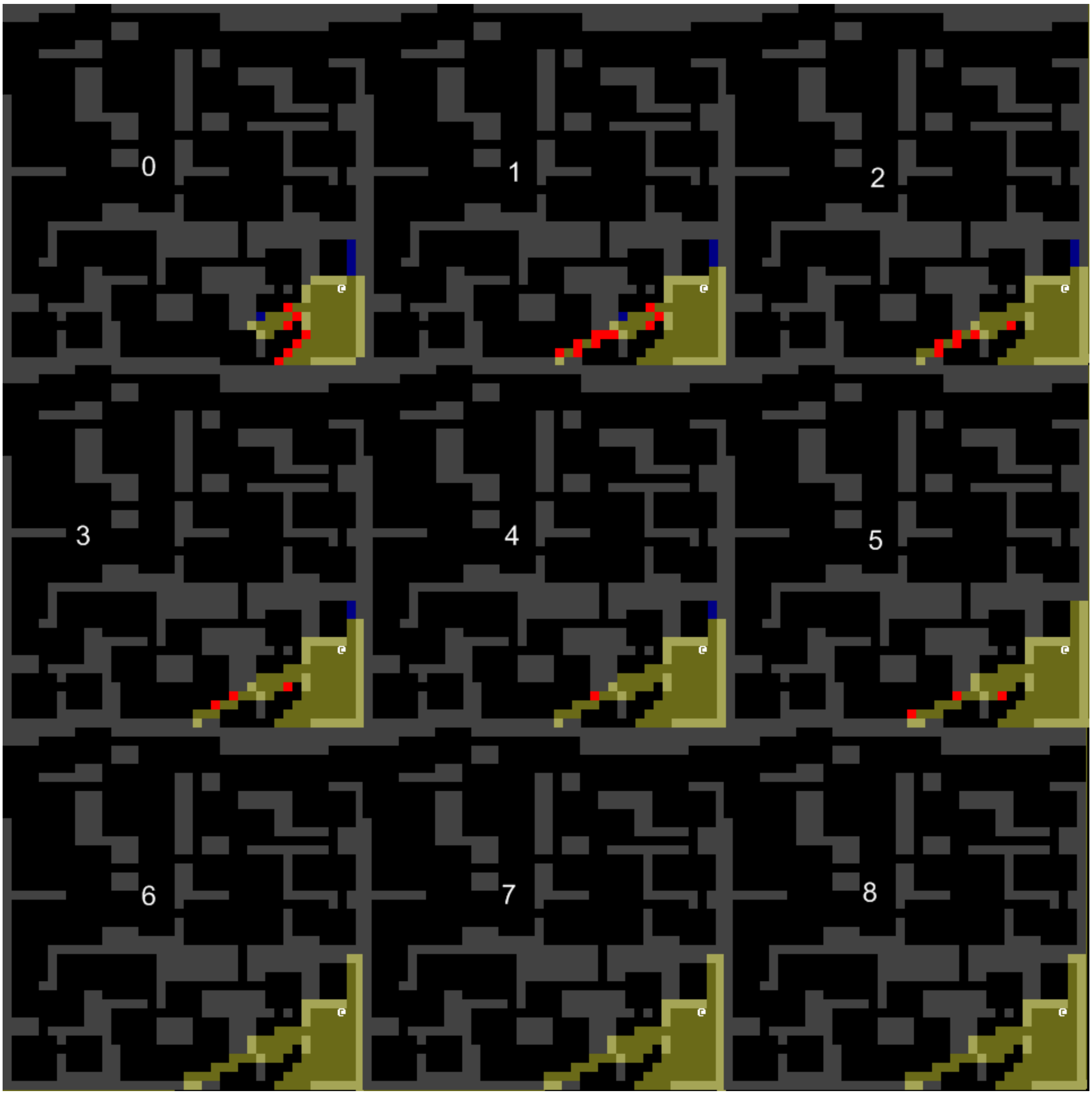
6.7. Indoor symmetry

Red cells indicate cells visible by the player but that cannot see the player.

Blue cells indicate cells not visible by the player but that can see the player.



BASIC DIAMOND SHADOW



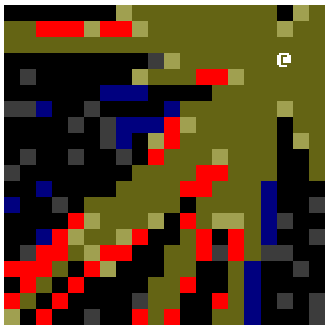
PERMISSIVE

6.8. *Outdoor symmetry*

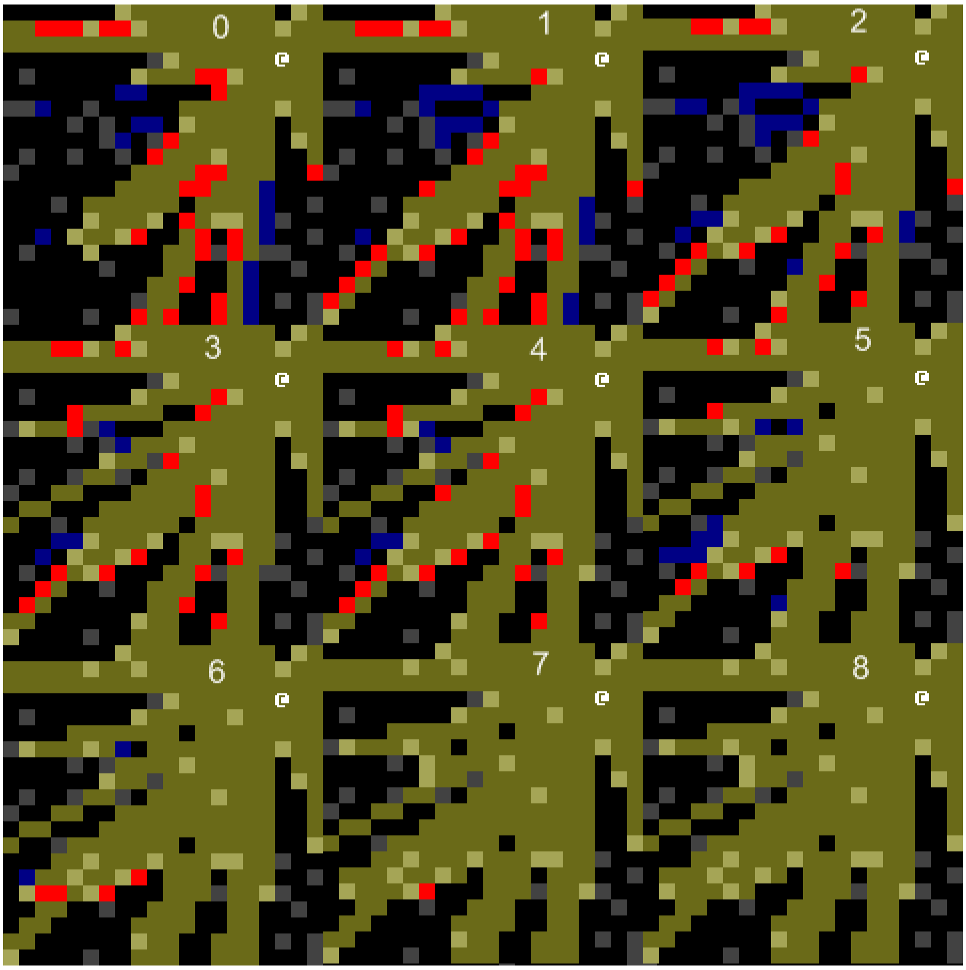
Red cells indicate cells visible by the player but that cannot see the player.

BASIC

DIAMOND



SHADOW



PERMISSIVE