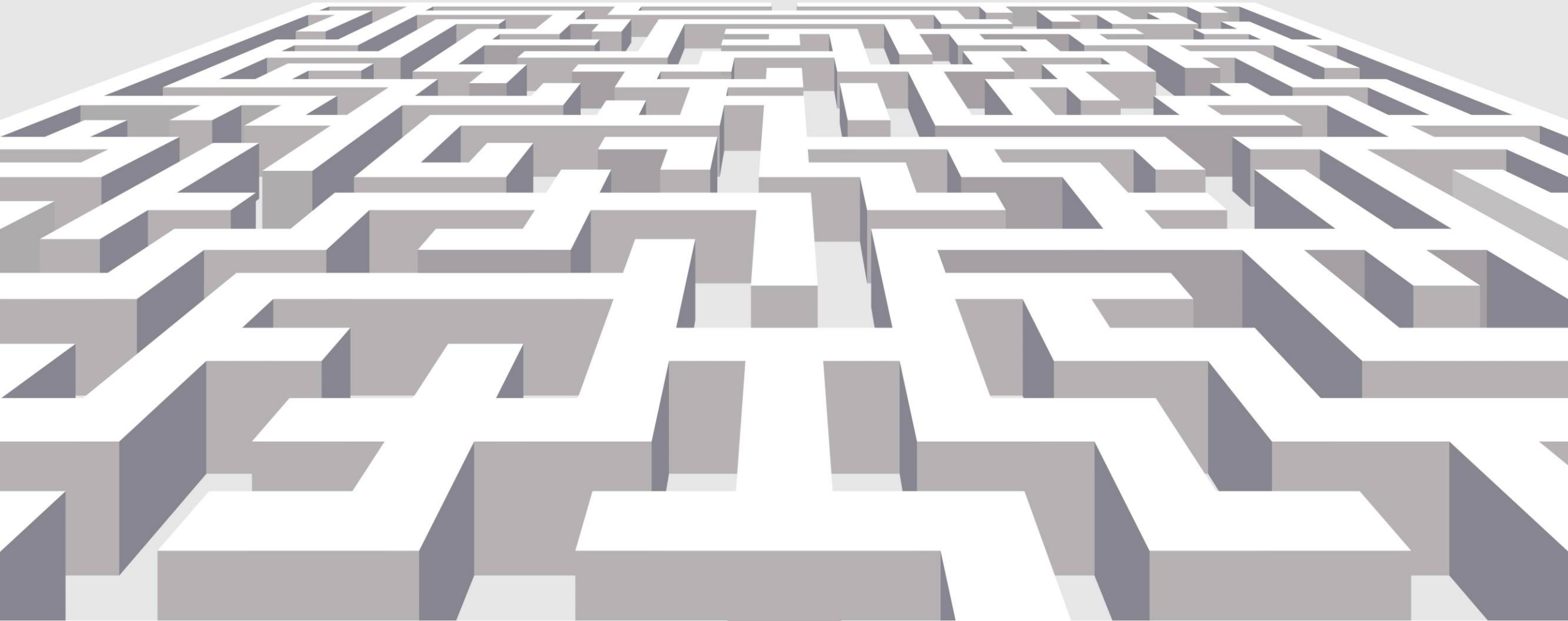
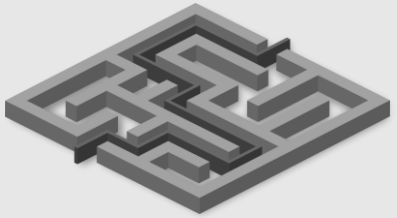
Labyrinth-Algorithmen im Vergleich

von Christian Graff



Einleitung
Hunt and Kill
Sidewinder
Vergleich
Demo
Fazit

Was sind Labyrinthe?



Definition:

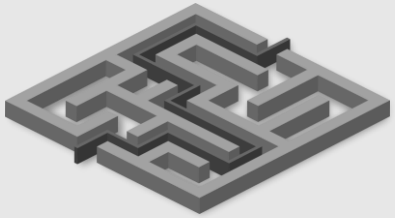
„Ein Labyrinth ist ein System von Linien oder Wegen, das durch zahlreiche Richtungsänderungen ein Verfolgen oder Abschreiten des Musters zu einem Rätsel macht.“

In der Informatik betrachten wir Labyrinthe als Graphen oder Bäume, wobei die Knoten Kreuzungen und die Kanten Gänge zwischen den Kreuzungen darstellen

Quelle: https://de.wikipedia.org/wiki/Labyrinth

Einleitung
Hunt and Kill
Sidewinder
Vergleich
Demo
Fazit

Bedeutung in der Informatik?



Pfadfindung & Algorithmenentwicklung:

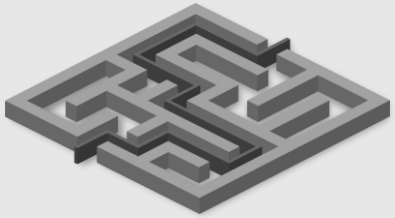
- Labyrinthe dienen als Modelle für komplexe Netzwerke, Schaltkreise usw.
- Algorithmen wie Tiefen-/ Breitensuche werden verwendet, um kürzesten Weg zwischen zwei Punkten zu finden
- Relevanz in verschiedensten Anwendungsgebieten

Anwendungsgebiete →

- **Videospiele**
- **Netzwerkoptimierung**
- **Simulationen**
- **Testen von Algorithmen**
- **Kryptografie**

Einleitung
Hunt and Kill
Sidewinder
Vergleich
Demo
Fazit

Welche Algorithmen gibt es?



- Recursive Backtracking
- Eller‘s Algorithm
- Kruskal‘s Algorithm
- Prim‘s Algorithm
- Recursive Division
- Aldous-Broder Algorithm
- Wilson‘s Algorithm
- Houston‘s Algorithm
- **„Hunt and Kill“ Algorithm**
- Growing Tree Algorithm
- Binary Tree Algorithm
- **Sidewinder Algorithm**

Quelle: https://www.jamisbuck.org/mazes/

Einleitung

Hunt and Kill

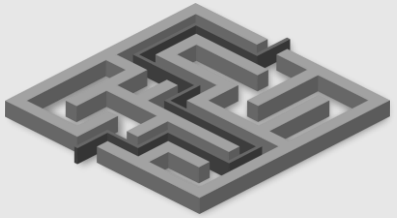
Sidewinder

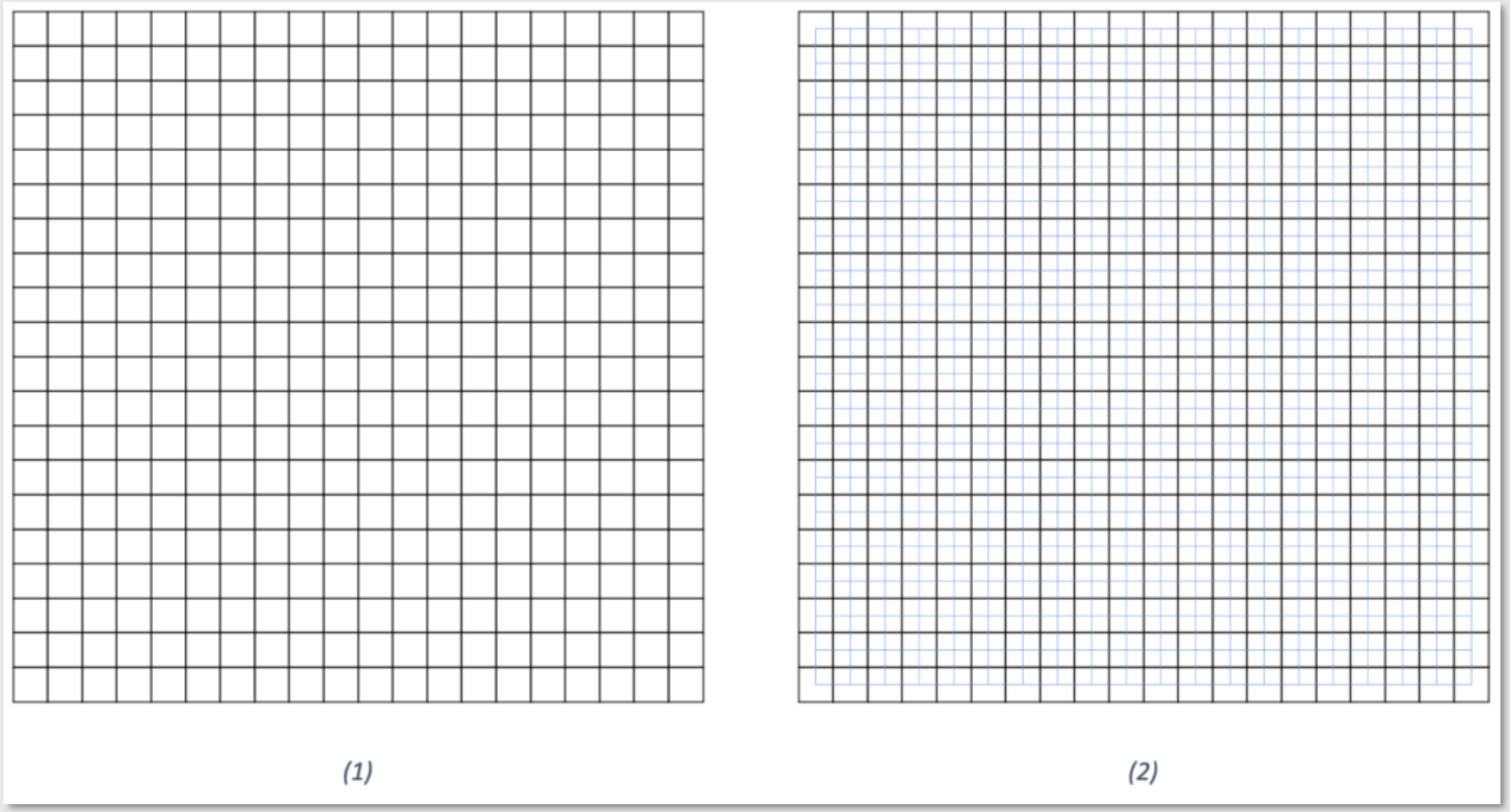
Vergleich

Demo

Fazit

Vorraussetzung





Quelle: https://www.baeldung.com/cs/maze-generation

Einleitung

Hunt and Kill

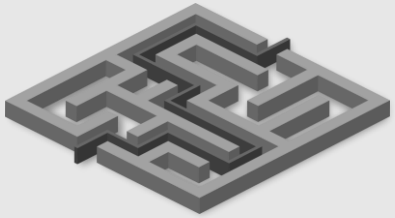
Sidewinder

Vergleich

Demo

Fazit

„Hunt and Kill“ Algorithmus



Funktionsweise:

1. Wähle im Grid eine beliebige Zelle als Startzelle.
2. Führe einen zufälligen Spaziergang durch die Zellen durch, indem du Verbindungen zu unbesuchten Nachbarzellen erschaffst.
3. Wenn die aktuelle Zelle keine unbesuchten Nachbarn mehr hat:
   - Wechsle in den „Huntmodus".
   - Suche nach einer unbesuchten Zelle, die an eine besuchte Zelle angrenzt.
   - Erstelle eine Verbindung zwischen den beiden Zellen und setze die unbesuchte Zelle als neue Startzelle.
4. Wiederhole Schritte 2 und 3, bis der "Jagdmodus" die gesamte Matrix durchsucht hat und keine unbesuchten Zellen mehr findet.

Quelle: https://weblog.jamisbuck.org/2011/1/24/maze-generation-hunt-and-kill-algorithm

Einleitung

Hunt and Kill

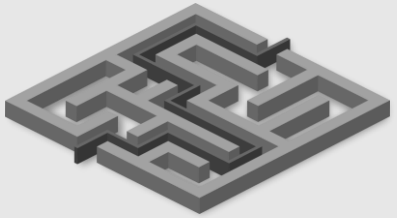
Sidewinder

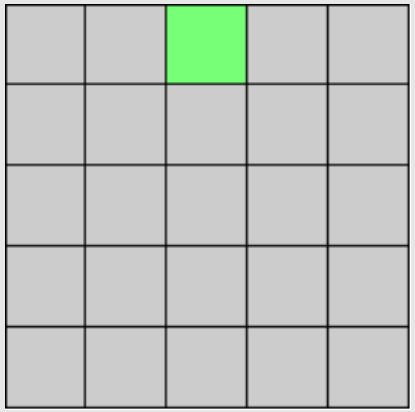
Vergleich

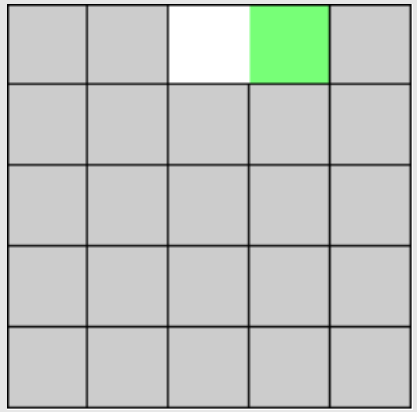
Demo

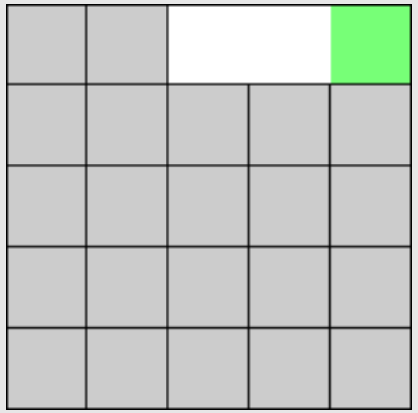
Fazit

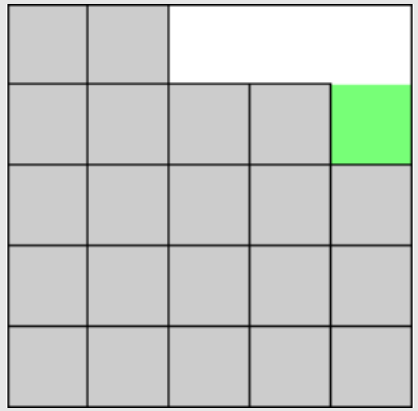
„Hunt and Kill“ Algorithmus

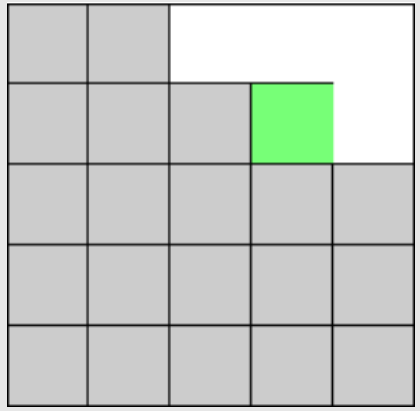


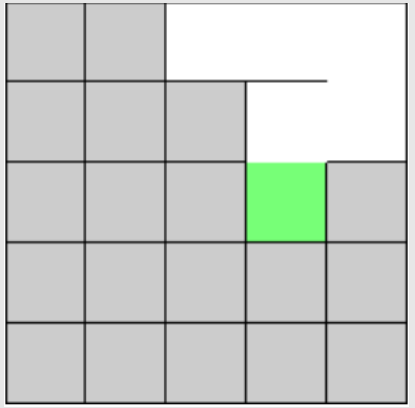








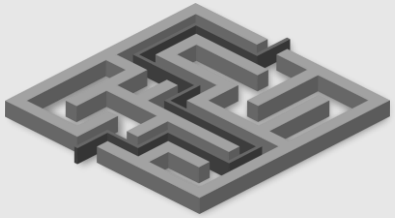


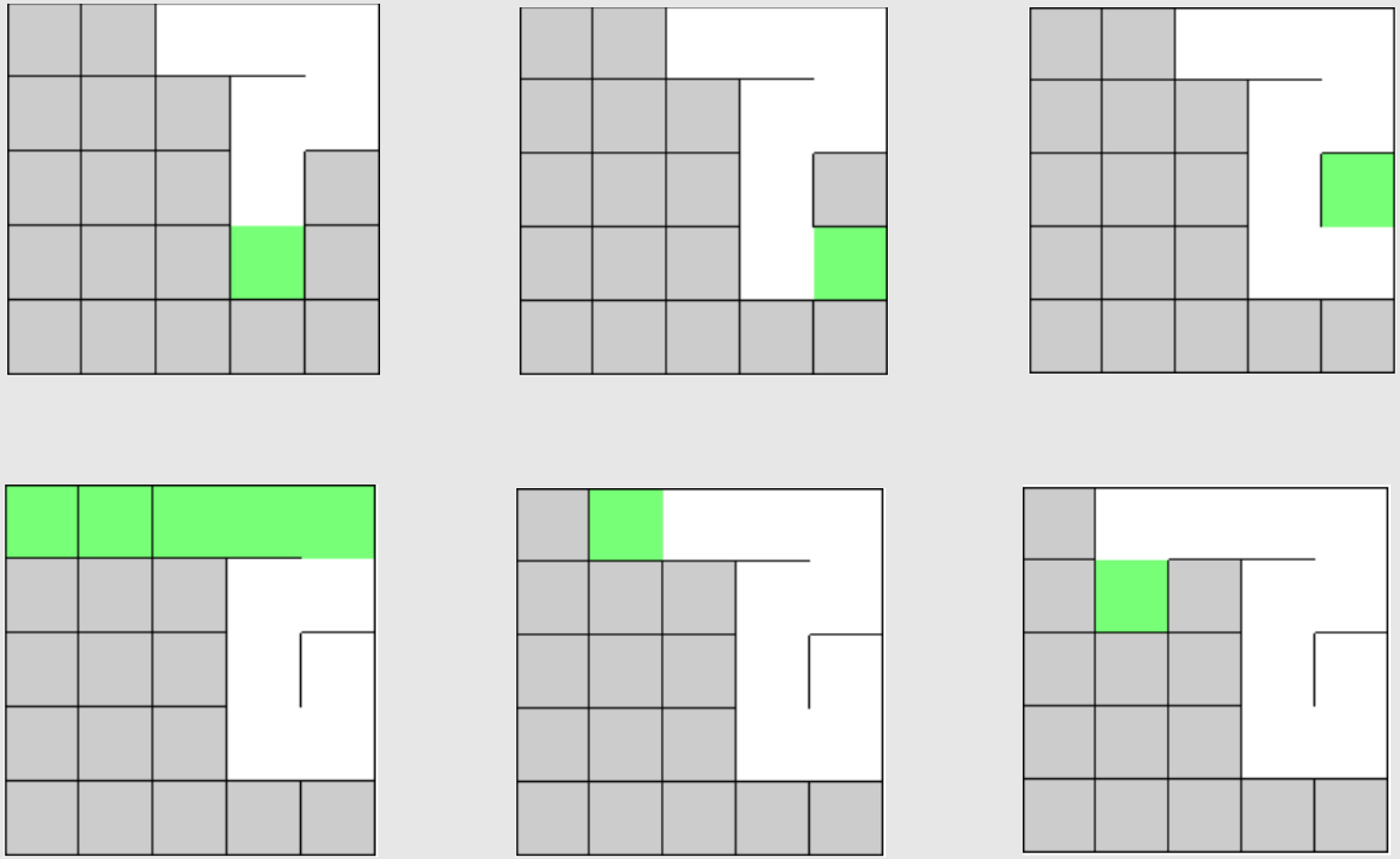


Quelle: https://weblog.jamisbuck.org/2011/1/24/maze-generation-hunt-and-kill-algorithm

Einleitung
Hunt and Kill
Sidewinder
Vergleich
Demo
Fazit

„Hunt and Kill“ Algorithmus

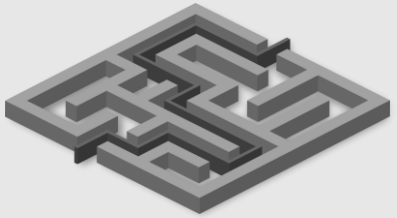


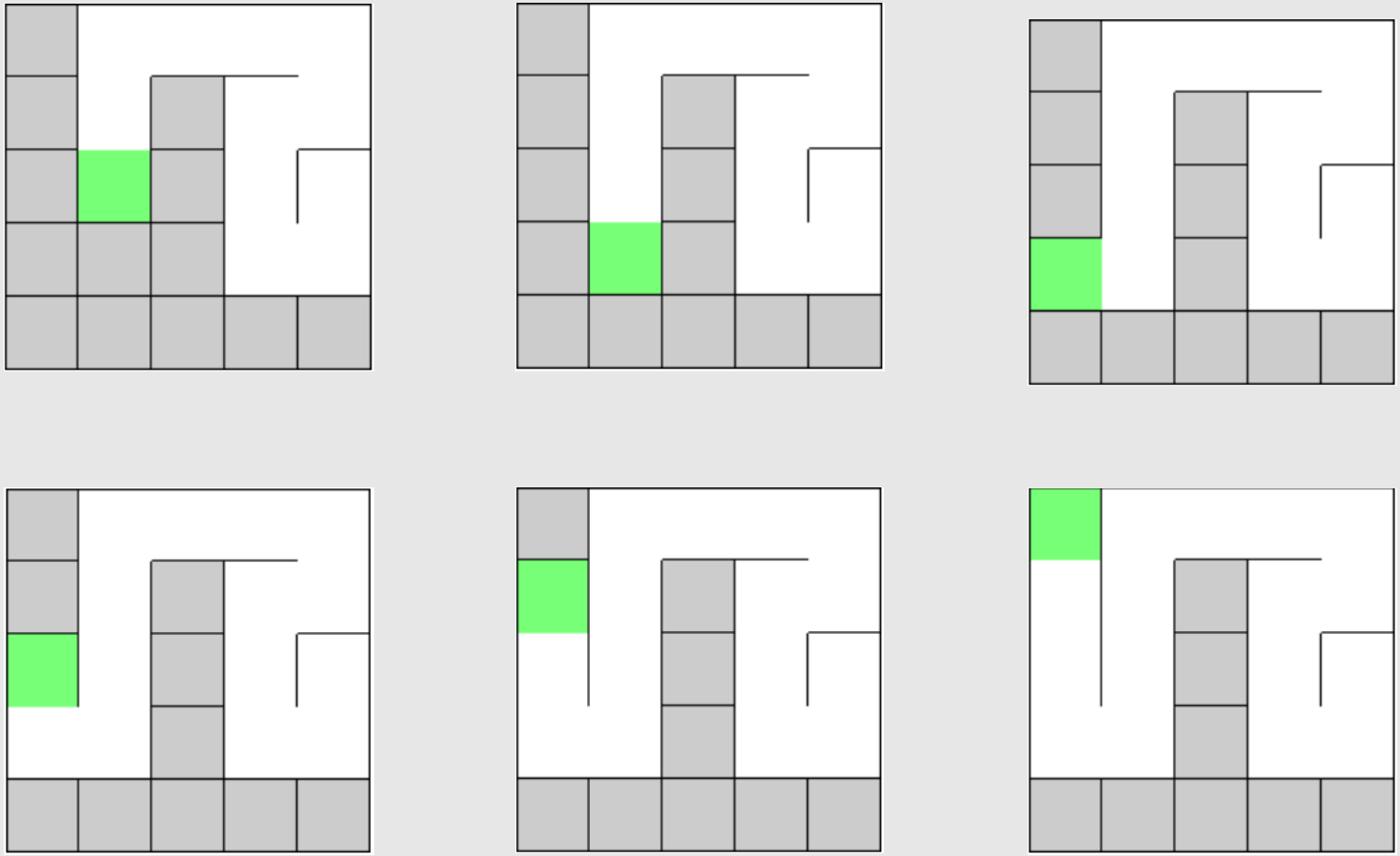


Quelle: https://weblog.jamisbuck.org/2011/1/24/maze-generation-hunt-and-kill-algorithm

Einleitung
Hunt and Kill
Sidewinder
Vergleich
Demo
Fazit

„Hunt and Kill“ Algorithmus

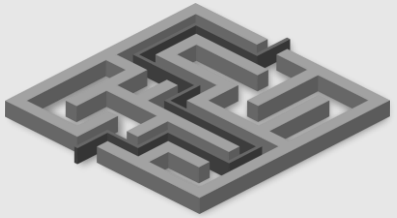


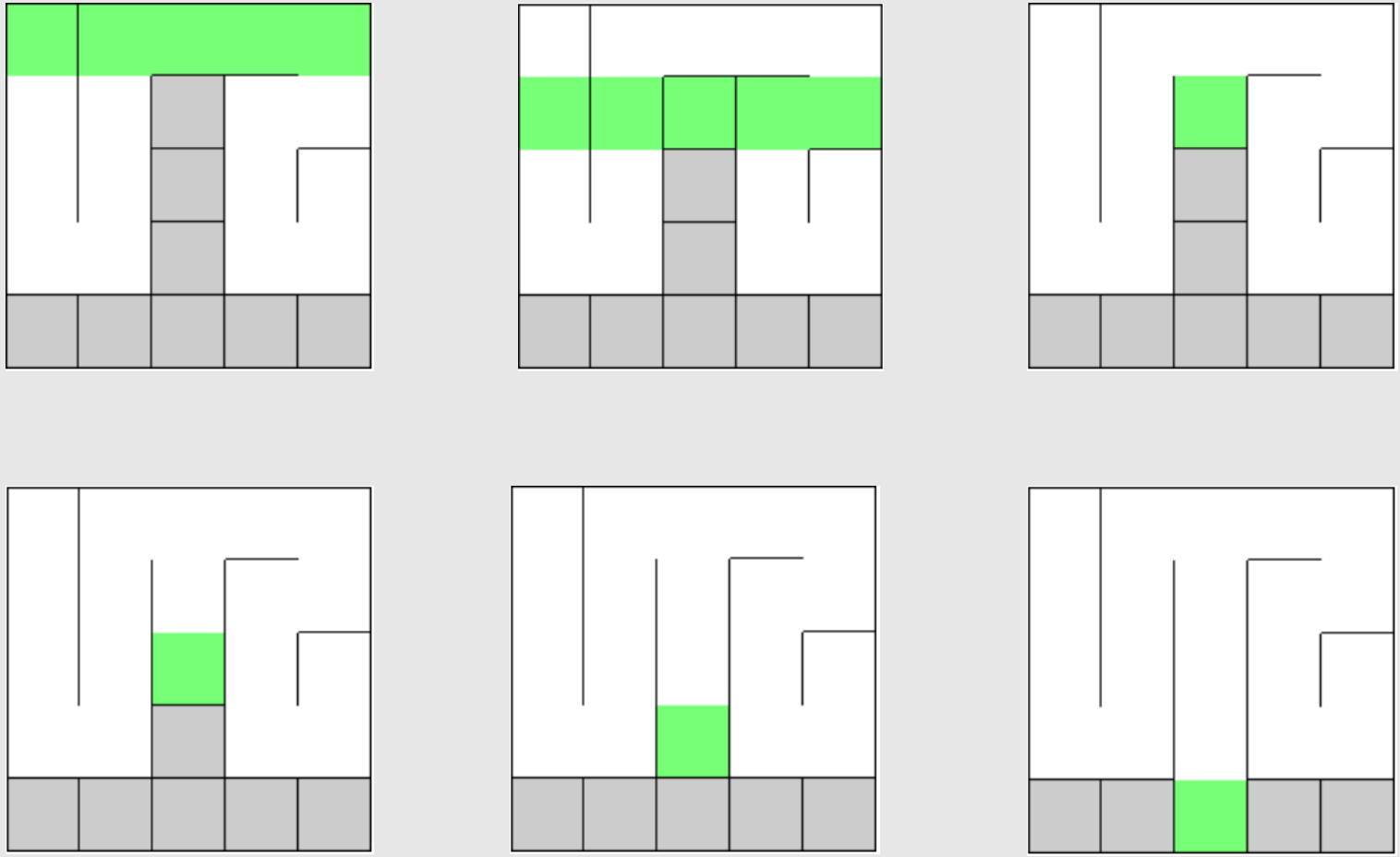


Quelle: https://weblog.jamisbuck.org/2011/1/24/maze-generation-hunt-and-kill-algorithm

Einleitung
Hunt and Kill
Sidewinder
Vergleich
Demo
Fazit

„Hunt and Kill“ Algorithmus

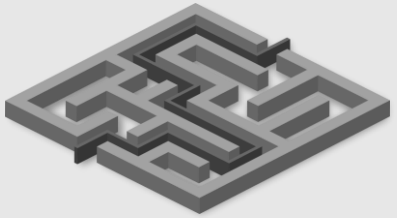


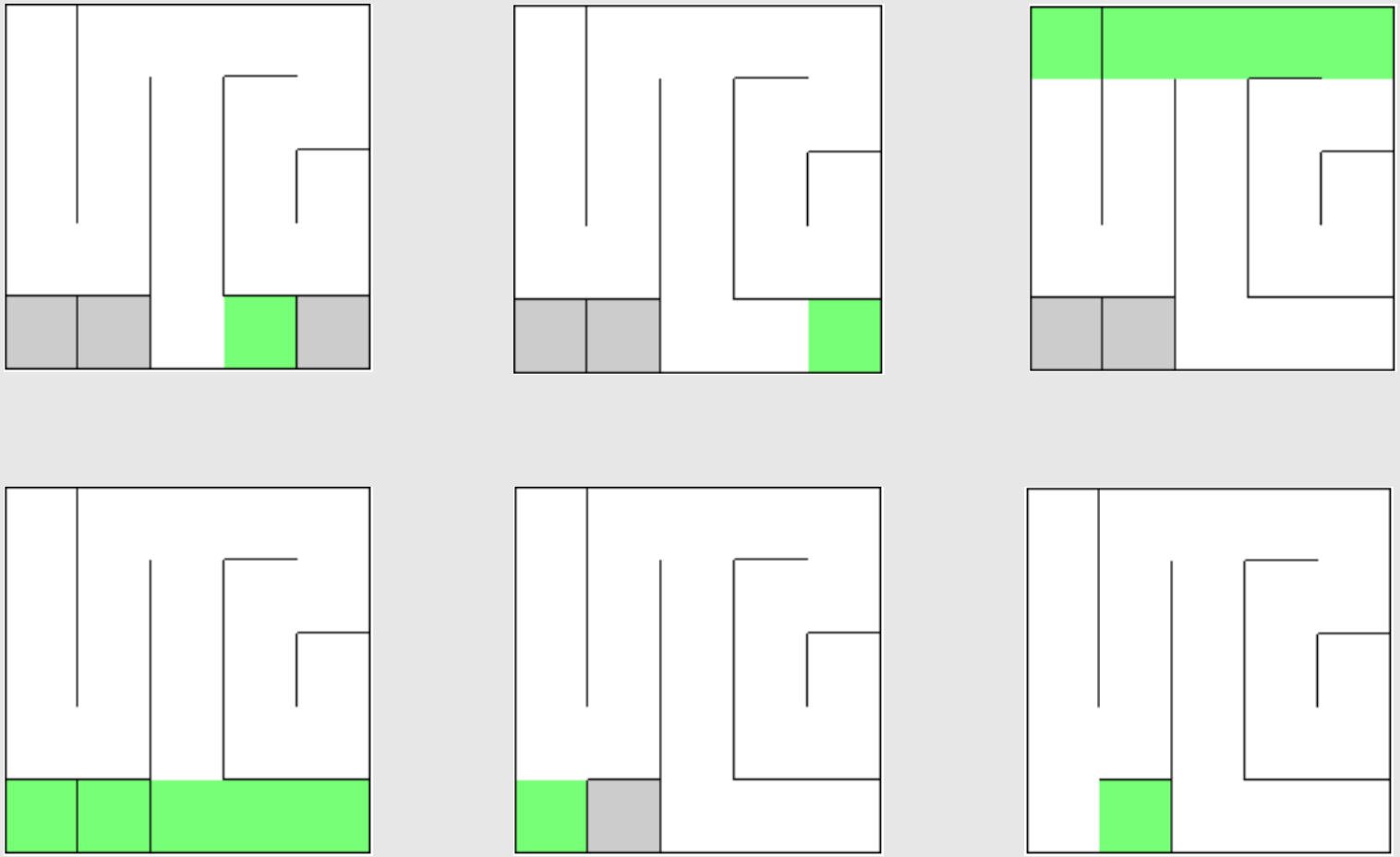


Quelle: https://weblog.jamisbuck.org/2011/1/24/maze-generation-hunt-and-kill-algorithm

Einleitung
Hunt and Kill
Sidewinder
Vergleich
Demo
Fazit

„Hunt and Kill“ Algorithmus





Quelle: https://weblog.jamisbuck.org/2011/1/24/maze-generation-hunt-and-kill-algorithm

Einleitung

Hunt and Kill

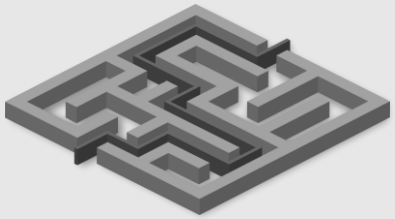
Sidewinder

Vergleich

Demo

Fazit

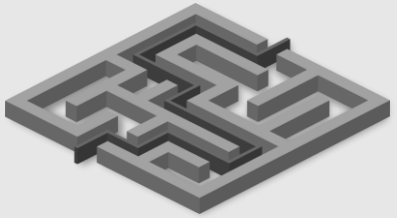
„Hunt and Kill“ Algorithmus



```
def HuntAndKill(maze):
            randomX, randomY = random.randint(0, maze.cols-1), random.randint(0, maze.rows-1)
            current = maze.grid.cells[randomX][randomY]

            while current:
                unvisited_neighbours = [cell for cell in current.neighbours if len(cell.connections) == 0]
                if len(unvisited_neighbours) > 0:
                    neighbour = random.choice(unvisited_neighbours)
                    Grid.JoinAndDestroyWalls(current, neighbour)
                    current = neighbour

                else:
                    current = None
                    for y in range(maze.cols):
                        for x in range(maze.rows):

                            this = maze.grid.cells[x][y]
                            visited_neighbours = [cell for cell in this.neighbours if len(cell.connections) > 0]

                            if len(this.connections) == 0 and len(visited_neighbours) > 0:
                                current = this
                                neighbour = random.choice(visited_neighbours)
                                Grid.JoinAndDestroyWalls(current, neighbour)
                                break
```

Einleitung
Hunt and Kill
Sidewinder
Vergleich
Demo
Fazit

„Sidewinder“ Algorithmus



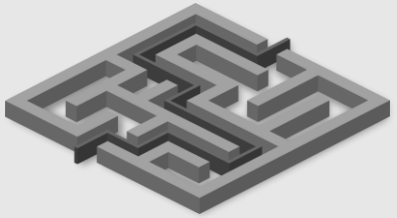
Funktionsweise:

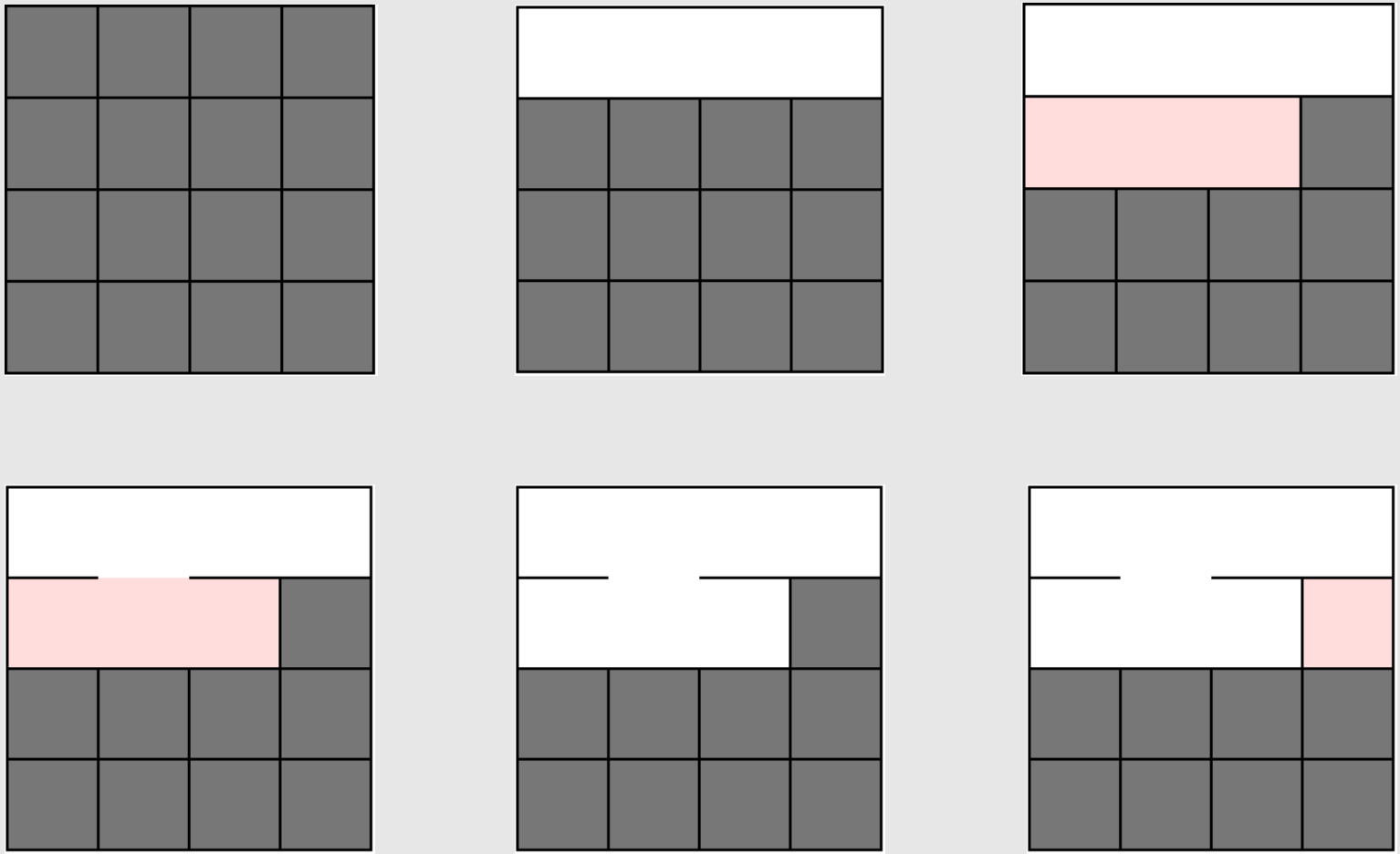
1. Wir arbeiten uns Zeile für Zeile durch das Grid. Die Startzelle befindet sich bei 0,0. Es wird ein leeres „run“-Set initialisiert.
2. Füge die aktuelle Zelle dem „run“-Set hinzu.
3. Entscheide zufällig für die aktuelle Zelle, ob ein Durchgang nach rechts erstellt wird.
4. Wurde ein Durchgang erstellt, mache die neue Zelle zur aktuellen und wiederhole die Schritte 2 bis 4.
5. Wurde kein Durchganz erstellt, wähle irgendeine Zelle aus dem „run“-Set und erschaffe einen Durchgang nach oben. Leere nun das „run“-Set und lege die nächste Zelle in der Reihe als aktuelle Zelle fest. Wiederhole dann die Schritte 2 bis 5.
6. Mache das bis alle Zeilen bearbeitet wurden.

Quelle: https://weblog.jamisbuck.org/2011/2/3/maze-generation-sidewinder-algorithm

Einleitung
Hunt and Kill
Sidewinder
Vergleich
Demo
Fazit

„Sidewinder“ Algorithmus

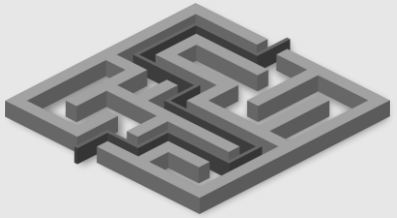


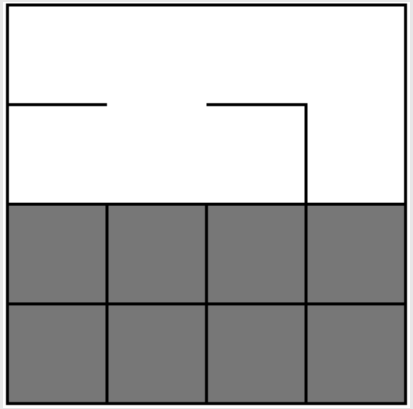
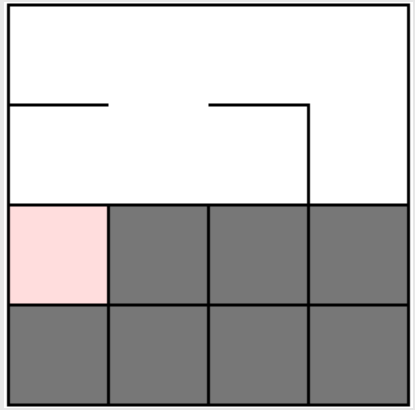
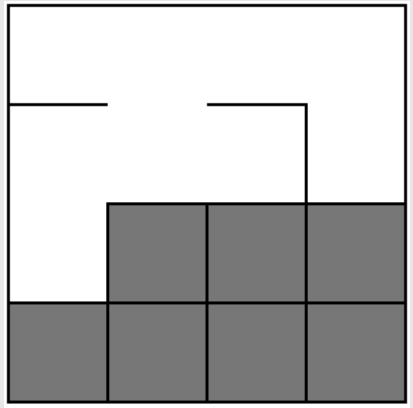
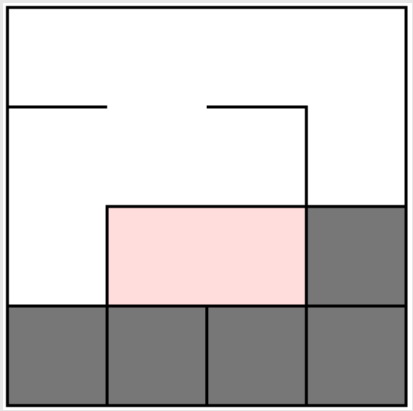
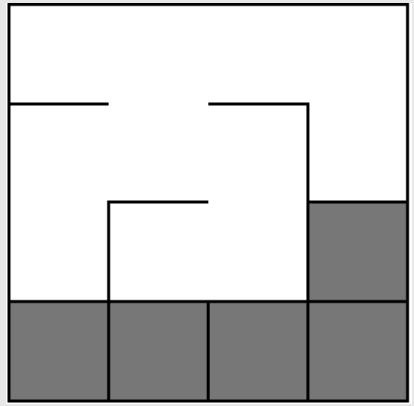
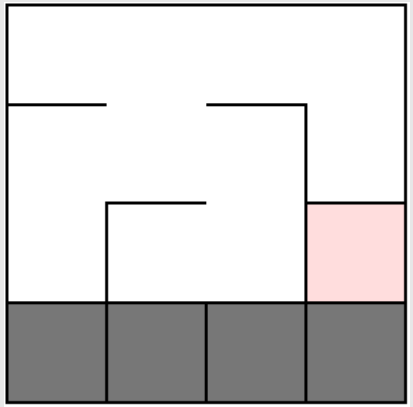


Quelle: https://weblog.jamisbuck.org/2011/2/3/maze-generation-sidewinder-algorithm

Einleitung
Hunt and Kill
Sidewinder
Vergleich
Demo
Fazit

„Sidewinder“ Algorithmus

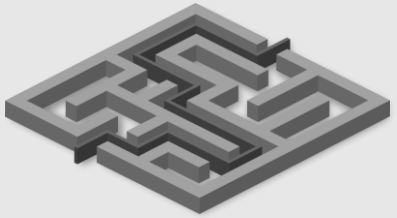


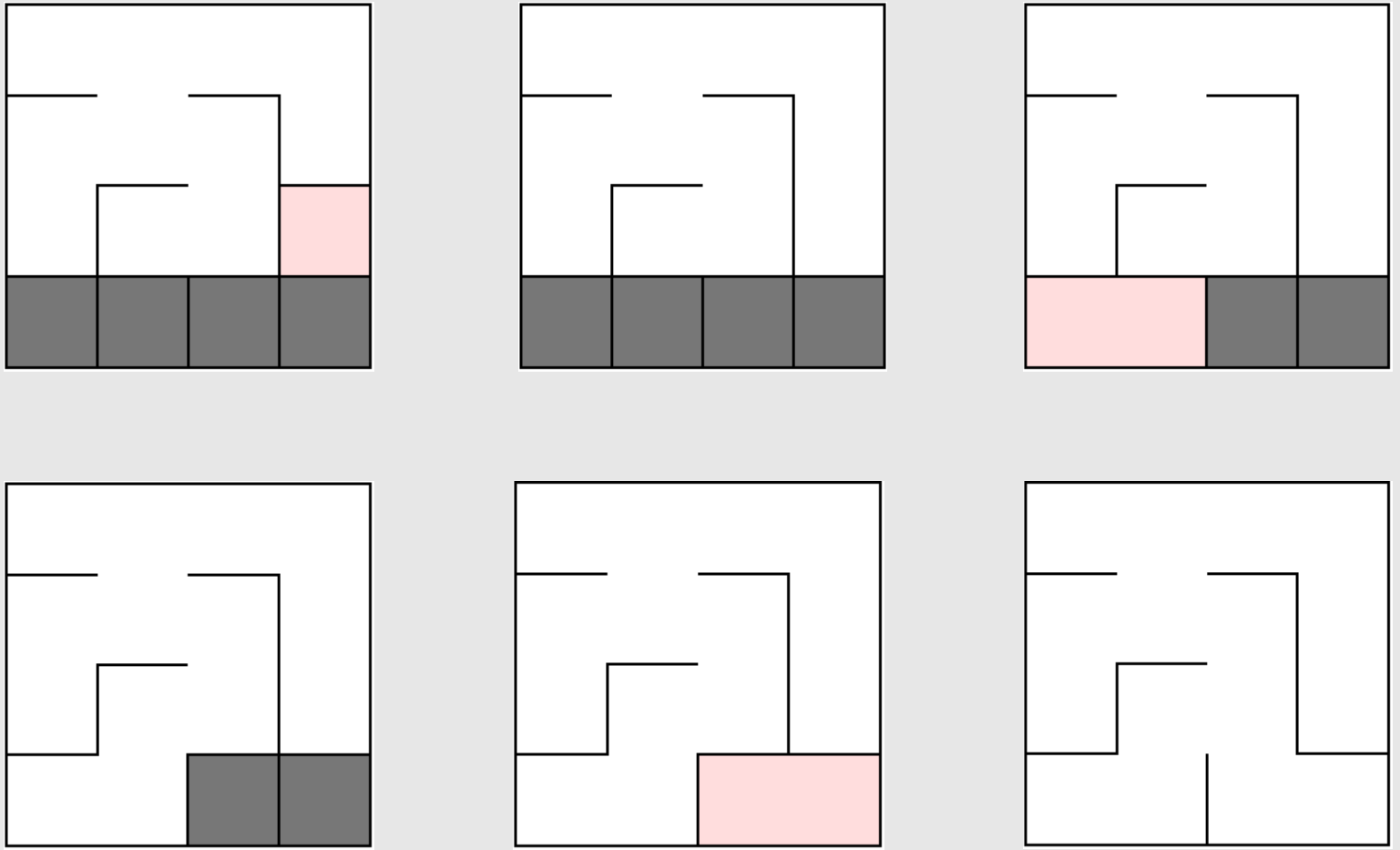







Quelle: https://weblog.jamisbuck.org/2011/2/3/maze-generation-sidewinder-algorithm

Einleitung
Hunt and Kill
Sidewinder
Vergleich
Demo
Fazit

„Sidewinder“ Algorithmus

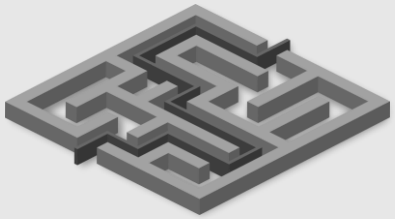




Quelle: https://weblog.jamisbuck.org/2011/2/3/maze-generation-sidewinder-algorithm

Einleitung
Hunt and Kill
Sidewinder
Vergleich
Demo
Fazit

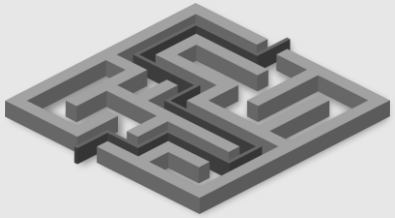
„Sidewinder“ Algorithmus



```
def Generate(maze):
            for y in range(maze.grid.rows):
                history = []

                for x in range(maze.grid.cols):

                    current = maze.grid.cells[x][y]
                    history.append(current)

                    at_eastern_edge = False
                    at_northern_edge = False

                    if current.East == None:
                        at_eastern_edge = True
                    if current.North == None:
                        at_northern_edge = True

                    if at_eastern_edge or (at_northern_edge == False and random.randint(0, 1)==1):
                        random_cell = random.choice(history)
                        if random_cell.North:
                            Grid.JoinAndDestroyWalls(random_cell, random_cell.North)
                        history.clear()
                    else:
                        Grid.JoinAndDestroyWalls(current, current.East)
```

Einleitung
Hunt and Kill
Sidewinder
Vergleich
Demo
Fazit

Vergleich



Hunt and Kill:

- on average 10% dead ends
- Erzeugt lange, verschlungene Wege („river“)
- Geringe Speicheranforderung
- Potentiell lange Generierungsdauer, da viele Zellen mehrfach gescannt werden müssen

Sidewinder:

- on average 27% dead ends
- Erzeugt eine starke vertikale Struktur
- Generiert immer einen durchgehenden Korridor in der obersten Zeile
- Kann extrem große Labyrinthe erstellen, da sehr schnell
- Keine Sackgassen von Süden nach Norden

Quelle: Mazes for Programmers – Jamis Buck (S. 73, 252, 254)

Einleitung

Hunt and Kill

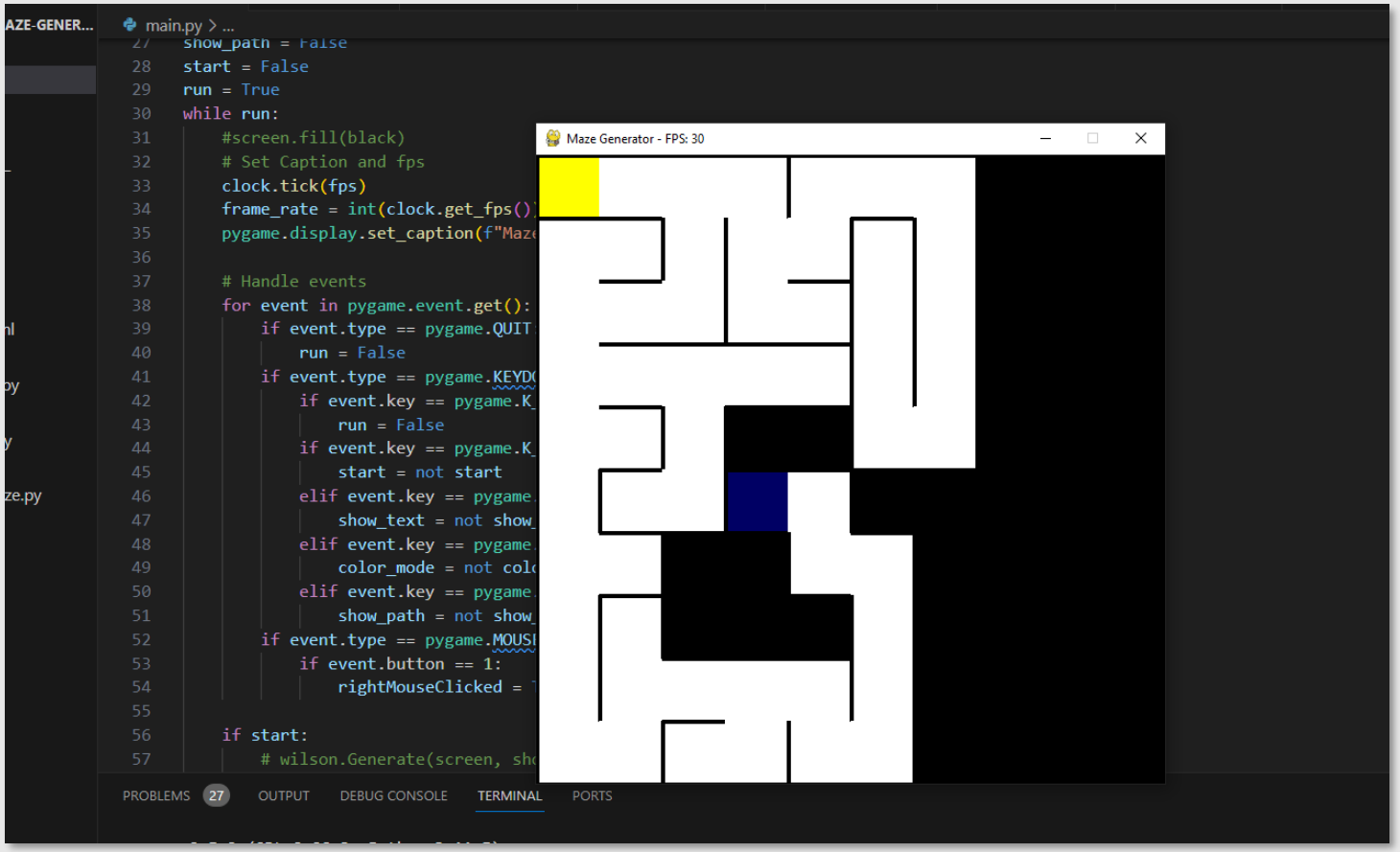
Sidewinder

Vergleich

Demo

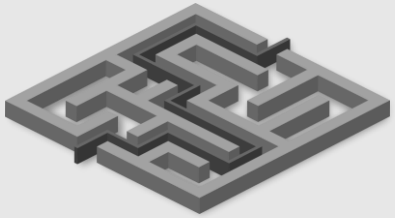
Fazit

Demonstration



Einleitung
Hunt and Kill
Sidewinder
Vergleich
Demo
Fazit

Fazit



- Beide Algorithmen sind leicht zu verstehen
- Mit entsprechenden Python-Bibliotheken sind diese auch relativ einfach implementierbar
- Die Ästhetik der Hunt and Kill Labyrinthe ist deutlich ansprechender als beim Sidewinder Algorithmus
- Für komplexe Labyrinthe würde ich den Hunt and Kill Algorithmus bevorzugen
- Für einfache und schnell zu lösende Labyrinthe würde ich den Sidewinder Algorithmus bevorzugen

Danke für ihre Aufmerksamkeit ☺