Aufgabe 1: Laubmaschen

Teilnahme-ID: 69408

Bearbeiter dieser Aufgabe:  
Tim Krome

12. April 2024

[Lösungsidee 2](#_Toc164108403)

[Wirkung eines Blasvorgangs 3](#_Toc164108404)

[Modellierung der Wahrscheinlichkeiten 7](#_Toc164108405)

[Formulierung als Bernoulli-Experiment 7](#_Toc164108406)

[Berechnen von P(X=k) 9](#_Toc164108407)

[Möglichkeiten zur Modellierung der Regeln 10](#_Toc164108408)

[Berechnen des Resultats eines Blasvorgangs (Pseudocode) 12](#_Toc164108409)

[Trivialer Ansatz: Heuristisches Vorgehen (1. Ansatz) 13](#_Toc164108410)

[Kenngrößen zur Bewertung des Zustands eines Hofs 14](#_Toc164108411)

[Kombination der Heuristiken 19](#_Toc164108412)

[Behandeln des Rands 20](#_Toc164108413)

[Gesamt-Algorithmus 24](#_Toc164108414)

[Laufzeit 25](#_Toc164108415)

[Kritik am Algorithmus 25](#_Toc164108416)

[Besseres Vorgehen: Generalisierte Ablaufpläne (2. Ansatz) 26](#_Toc164108417)

[Ziel 26](#_Toc164108418)

[Auf welchen Höfen kann das Laub nicht auf Q geblasen werden? 27](#_Toc164108419)

[Zugrundeliegende Strategie 27](#_Toc164108420)

[Definitionen: Generalisierte Ablaufpläne und Muster 29](#_Toc164108421)

[Können die Rand- und Eckfelder vollständig geleert werden? 30](#_Toc164108422)

[Muster zum Leeren von Rand- und Eckfeldern 31](#_Toc164108423)

[Bauen des generalisierten Ablaufplans 32](#_Toc164108424)

[1. Phase: Unterste Reihe des Hofs befreien 33](#_Toc164108425)

[2. Phase: Laub auf oberste Reihe bringen 34](#_Toc164108426)

[3. Phase: Laub auf der obersten Reihe konzentrieren 35](#_Toc164108427)

[4. Phase: Gesamtes Laub auf Feld Q verschieben 36](#_Toc164108428)

[Optimierungen 41](#_Toc164108429)

[Gesamt-Algorithmus und Kritik 41](#_Toc164108430)

[Laufzeit 42](#_Toc164108431)

[Kombination der Verfahren (3. Ansatz) 43](#_Toc164108432)

[Parametrisierung 43](#_Toc164108433)

[Zusammenfassung 43](#_Toc164108434)

[Bereits in Solver 1 bis 3 enthaltene Erweiterungen 43](#_Toc164108435)

[Erweiterung 1: Keine Mauer-Umrandung des Hofs 44](#_Toc164108436)

[Laufzeit 47](#_Toc164108437)

[Erweiterung 2: Mehrere Laubbläser 47](#_Toc164108438)

[Synergien ausnutzen 49](#_Toc164108439)

[Laufzeit 50](#_Toc164108440)

[j Laubbläser 51](#_Toc164108441)

[Erweiterung 3: Verschiedene Laubtypen 51](#_Toc164108442)

[Weitere Erweiterungsideen (mit Lösungsansätzen) 51](#_Toc164108443)

[Andere Hofformen: Rechtwinklige Polygone 51](#_Toc164108444)

[Weitere Ideen 53](#_Toc164108445)

[Umsetzung 53](#_Toc164108446)

[Hilfsfunktionen zur Berechnung von Binomialverteilungen (binomial\_util.py) 54](#_Toc164108447)

[Definieren einer Rules-Klasse (hof.py) 55](#_Toc164108448)

[Definieren einer Hof-Klasse (hof.py) 56](#_Toc164108449)

[Grundaufgabe Ansatz 1 57](#_Toc164108450)

[Grundaufgabe Ansatz 2: 59](#_Toc164108451)

[Beispiele – Grundaufgabe 60](#_Toc164108452)

[Ansatz 1 (aufgabe1\_solver.py) 61](#_Toc164108453)

[Beispiel 1 61](#_Toc164108454)

[Beispiel 2 62](#_Toc164108455)

[Beispiel 3 63](#_Toc164108456)

[Beispiel 4 64](#_Toc164108457)

[Beispiel 5 65](#_Toc164108458)

[Beispiel 6 66](#_Toc164108459)

[Ansatz 2 (aufgabe2\_solver.py) 68](#_Toc164108460)

[Beispiel 1 68](#_Toc164108461)

[Beispiel 2 71](#_Toc164108462)

[Beispiel 3 72](#_Toc164108463)

[Beispiel 4 73](#_Toc164108464)

[Beispiel 5 75](#_Toc164108465)

[Beispiel 6 76](#_Toc164108466)

[Beispiel 7 78](#_Toc164108467)

[Ansatz 3 (aufgabe3\_solver.py) 80](#_Toc164108468)

[Beispiel 1 80](#_Toc164108469)

[Beispiel 2 81](#_Toc164108470)

[Beispiel 3 82](#_Toc164108471)

[Beispiele – Erweiterung 1: Keine Mauer-Umrandung 83](#_Toc164108472)

[Beispiel 1 84](#_Toc164108473)

[Beispiel 2 85](#_Toc164108474)

[Beispiele – Erweiterung 2: Zwei Laubbläser 86](#_Toc164108475)

[Beispiel 1 87](#_Toc164108476)

[Beispiel 2 88](#_Toc164108477)

[Beispiele – Erweiterung 3: Verschiedene Laubtypen 90](#_Toc164108478)

[Beispiel 1 90](#_Toc164108479)

[Beispiel 2 91](#_Toc164108480)

[Quellcode 92](#_Toc164108481)

[Auszüge aus binomial\_util.py 92](#_Toc164108482)

[Auszüge aus hof.py 94](#_Toc164108483)

[Grundaufgabe: Ansatz 1 97](#_Toc164108484)

[Grundaufgabe: Ansatz 2 102](#_Toc164108485)

[Grundaufgabe: Ansatz 3 110](#_Toc164108486)

[Erweiterung 1 (basierend auf Ansatz 2) 110](#_Toc164108487)

[Erweiterung 2 (basierend auf Ansatz 2) 115](#_Toc164108488)

# Lösungsidee

Der Schulhof besteht aus gleich großen Planquadraten, die in einem gleichmäßigen Raster angeordnet sind. Der Schulhof selbst ist dabei quadratisch und hat somit die Maße (*n* Planquadrate, *n* Planquadrate) mit *n* ∈ ℕ. Der Aufgabe nach ist das Laub anfangs gleichmäßig auf dem Hof verteilt, d.h. auf jedem Planquadrat befinden sich gleich viele Blätter.

Auf dem Hof führt der Hausmeister Blasoperationen durch. Eine Blasoperation kann als Funktion verstanden werden, die von zwei Parametern abhängig ist: Das Feld, auf dem der Hausmeister steht (im Folgenden als „Feld 0“ bezeichnet), und die Richtung, in die er bläst. Im Folgenden verwende ich diese Kurzschreibweise für eine Blasoperation, die vom Feld 0 *feld0* aus in Richtung *direction* bläst: *blase(feld, direction).*

Bei Anwendung der Funktion ändern sich die Blattanzahlen auf den einzelnen Feldern. Die Modellierung einer Blasoperation wird dadurch erschwert, dass diese **Änderungen vom Zufall abhängen**. Die in der Aufgabe definierten Blasregeln sind probabilistisch, sie geben für jedes Blatt auf Feld A und Feld B die Wahrscheinlichkeit an, dass dieses Blatt auf ein bestimmtes anderes Feld geweht wird. Feld A befindet sich hierbei unmittelbar vor der Luftaustrittsdüse des Laubbläsers, Feld B befindet sich hinter Feld A (siehe Abbildung 1).

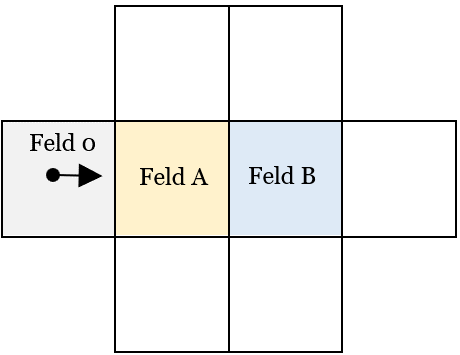


Abbildung 1: Feld A und B in Relation zu Hausmeister (Punkt) und Blasrichtung (Pfeil)

Der Hof kann als zweidimensionale Matrix modelliert werden. Jeder Eintrag der Matrix korrespondiert mit einem Feld des Hofs und speichert die Anzahl an Blättern, die sich auf dem Feld befinden. Die Modellierung erfolgt so, dass das Feld in der Zeile x und Spalte y durch den Matrix-Eintrag am Index *(x,y)* entspricht.

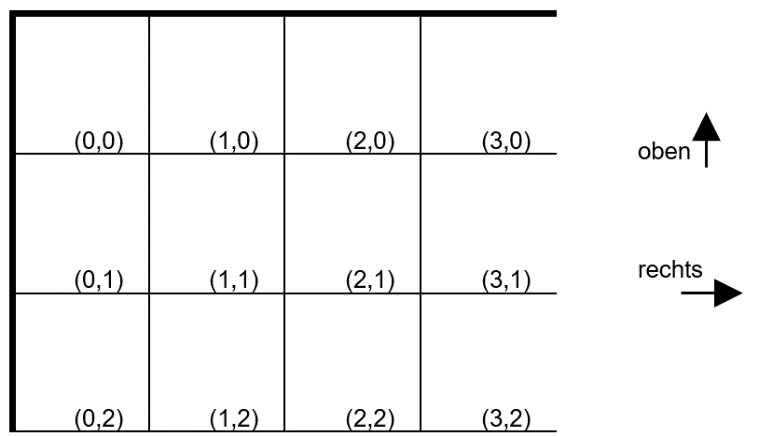


Abbildung 2: Veranschaulichung der Felder-Indizierung (jedes Feld ist mit seinem Index in der Matrix beschrieben)

Zur Darstellung einer Blasrichtung wird ein Einheitsvektor verwendet, der in die Richtung zeigt, in die der Laubbläser gehalten wird:

* Richtung (1,0) ≙ rechts
* Richtung (-1,0) ≙ links
* Richtung (0,1) ≙ oben
* Richtung (0,-1) ≙ unten

## Wirkung eines Blasvorgangs

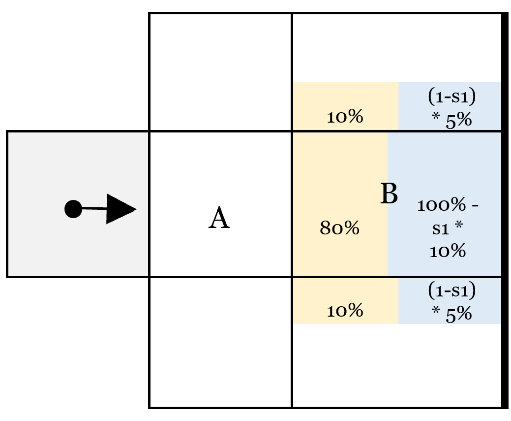
Die Aufgabe schreibt für die Wirkung eines Blasvorgangs folgende Regeln vor:

|  |  |
| --- | --- |
| **Vorher:** Ein Bild, das Text, Screenshot, Quadrat, Diagramm enthält.  Automatisch generierte Beschreibung | **Nachher:** Ein Bild, das Text, Screenshot, Diagramm, Quadrat enthält.  Automatisch generierte Beschreibung  PA\_to\_B = 0,8 PA\_to\_above\_B = 0,1 PA\_to\_below\_B = 0,1 PB\_stay\_on\_B = 0,9 PB\_to\_beyond\_B = 0,1 |

Diese Regeln sind nur anwendbar, wenn Feld A und B sowie die Nachbarfelder (horizontal und vertikal, nicht diagonal) von Feld B tatsächlich innerhalb des Hofs existieren. Dem ist nur der Fall, wenn Feld 0, Feld A und Feld B keine Rand- oder Eckfelder sind. Für alle anderen möglichen Fälle müssen zusätzliche Regeln definiert werden. Hierbei habe ich folgende Regeln berücksichtigt (die Reihenfolge gibt die Priorisierung der Prinzipien an, das erste Prinzip wird dabei am höchsten priorisiert):

1. In der Aufgabe werden keine Aussagen zur Begrenzung des Schulhofs getroffen, wodurch man einiges an Freiraum bei der Definition der zusätzlichen Blasregeln hat.  
   Ich lege zunächst fest, dass der Hof von einer hohen Mauer umgeben ist, über die keine Blätter fliegen können. Dies hat zur Folge, dass **die Gesamtlaubmenge auf dem Hof konstant bleiben muss (1. Prinzip)**.  
   *Anmerkung:* Zunächst erscheint es logisch, keine Operationen zuzulassen, die zu einer Verringerung der Gesamtlaubmenge auf dem Hof führen: Sonst könnte man ja einfach alle Blätter vom Hof runterblasen und auf die Art den Hof von Blättern befreien, was den Sinn der Aufgabe sicher nicht treffen würde.  
   Bei genauerer Betrachtung der Aufgabe stellt man jedoch fest, dass das Ziel des Hausmeisters klar definiert ist *als möglichst viele Blätter auf Feld Q bringen -* und nicht als *den Hof reinigen*. Vom Hof heruntergeblasene Blätter würden sich also negativ auf das Erreichen des Ziels auswirken, da diese das System verlassen haben und somit nicht mehr auf Q geblasen werden können. Würde man doch erlauben, dass Blätter den Hof durch Seitenabtriebseffekte verlassen können, dann wäre die Entwicklung einer Strategie tatsächlich schwieriger. (Mehr dazu in *Lösungsidee > Erweiterung 1)*
2. Die ergänzten Regeln sollen sich **an den von der Aufgabe vorgeschriebenen Regeln orientieren** und **nur dort, wo es notwendig ist** (weil beteiligte Felder außerhalb vom Hof lägen) **von ihnen abweichen (2. Prinzip).**
3. Dort, wo von den vorgeschriebenen Regeln abgewichen werden muss, sollen die neuen Regeln **realistisch** sein und sich **an tatsächlichen physikalischen Gesetzmäßigkeiten orientieren (3. Prinzip).**Es ist allerdings nicht möglich, allein aus den gegebenen Informationen mithilfe physikalischer Gesetze die exakten Wahrscheinlichkeiten für jeden Sonderfall zu ermitteln (es fehlen wesentliche Informationen wie die Oberflächenbeschaffenheit der Wand, die Reibung vorbeifliegender Blätter beeinflusst, es ist außerdem in den von der Aufgabe gegebenen Regeln nicht definiert wie die Blätter nach einer Blasoperation innerhalb der Planquadrate verteilt sind, erschwerend kommt die nichtlineare Flugbahn eines Blattes dazu). Bei den in der Aufgabe gegebenen Regeln handelt es sich ja auch nur um ein Modell.  
   Stattdessen werden plausible Abschätzungen betroffen, basierend auf denen modellhafte Regeln festgelegt werden. Dies erfolgt an vielen Stellen vereinfachend. Zur Generalisierung der Regeln werden außerdem verschiedene Parameter eingeführt. Über diesen Parameter kann die Wahrscheinlichkeit dort festgelegt werden, wo basierend auf den Informationen aus der Aufgabe eine eindeutige Festlegung der Wahrscheinlichkeit nicht möglich ist. Bei einer konkretenAnwendung des entwickelten Programms könnten diese Parameter kalibriert werden, um die Regeln an die tatsächlichen Gegebenheiten anzupassen.

Bei den nachfolgenden Regeldefinitionen ist der Anfangs- bzw. „Vorher“-Zustand immer, dass sich sowohl auf A als auch auf B (falls vorhanden) 100% der Blätter befinden. Dicke Kanten bedeuten, dass sich hier die den Hof eingrenzenden Mauern befinden.

**Sonderfall 1: Eine Mauer ist rechts von B**

PA\_to\_B = 0,8  
pA\_to\_above\_B = 0,1  
pA\_to\_below\_B = 0,1  
pB\_stay\_on\_B = 1 – 0,1 \* s1  
pB\_to\_above\_B = 0,05 \* (1-s1)  
pB\_to\_below\_B = 0,05 \* (1-s1)

Es fliegt kein Laub über die Mauer hinaus (Prinzip 1 erfüllt) und nur der Abtrieb von Feld B (= die 10% des Laubs von Feld B, die sonst auf das Feld rechts von B geflogen wären), fliegen den neuen Regeln nach anders (Prinzip 2 erfüllt).

Praktische Beobachtungen zeigen: Wenn Blätter gegen eine Wand geblasen werden, dann fliegen sie bei ausreichender Krafteinwirkung meist zur Seite weg. Mangels Informationen zu den Gegebenheiten lässt sich wie zuvor erläutert nicht eindeutig bestimmen, wie groß der Anteil der Blätter ist, der an der Wand hängen bleibt bzw. nicht auf ein seitliches Feld geweht wird.

Daher wird der *Parameter s1* eingeführt. *s1* gibt den Anteil am regulären Abtrieb von Feld B an, der (bei häufiger Experimentdurchführung durchschnittlich) in Feld B verbleibt und nicht seitlich wegfliegt. Im Folgenden wird mit s1 = 0,9 gearbeitet (d.h. ca. 90% des regulären Abtriebs von Feld B verbleibt in B und fliegt nicht zur Seite weg). Dies ist plausibel, da die vom Laubbläser ausgehende Krafteinwirkung am rechten Rand von Feld B nicht mehr besonders stark ist (das ist daran erkennbar, dass im Normalfall nur ca. 10% der Blätter von B abgetrieben werden).  
Prinzip 3 wurde also auch berücksichtigt.

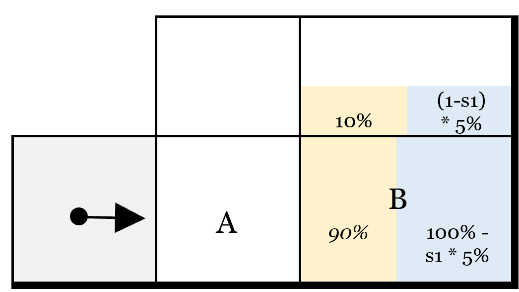
**Sonderfall 2: Eine Mauer ist unter B**

Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

Automatisch generierte BeschreibungPA\_to\_B = 0,9  
PA\_to\_above\_B = 0,1  
PB\_stay\_on\_B = 0,9  
PB\_to\_beyond\_B = 0,1

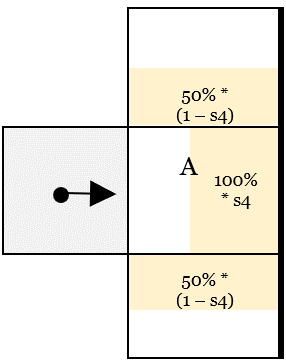
Es wird vereinfachend davon ausgegangen, dass der Seitenabtrieb von Feld A, der regulär auf das Feld unter B geflogen wäre, jetzt stattdessen vollständig auf Feld B geweht wird.  
Dieselben Regeln gelten analog, wenn sich die horizontale Mauer über Feld A, B und 0 befindet (in diesem Fall sind die Regeln an der Achse AB zu spiegeln).

**Sonderfall 3: B ist ein Eckfeld**

pA\_to\_B = 0,9  
pA\_to\_above\_B = 0,1  
pB\_stay\_on\_B = 1 – 0,05 \* (1-s1)  
pB\_to\_above\_B = 0,05 \* s1

Diese Regeln ergeben sich aus Kombination von Sonderfall 1 und 2, weshalb der Parameter *s1* von Sonderfall 1 übernommen wird. Das Laub, das in Sonderfall 1 von Feld B auf das Feld unter B geweht worden wäre, wird jetzt in die Mauerecke Ecke von Feld B geweht.  
Dieselben Regeln gelten analog, wenn sich die horizontale Mauer über Feld B befindet.

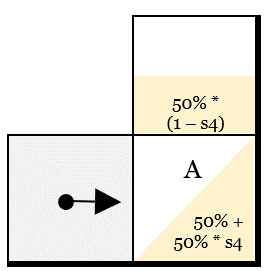
**Sonderfall 4: Eine Mauer ist rechts von A (➔ B existiert nicht)**

pA\_to\_above\_A = 0,5 \* (1 - s4)  
pA\_to\_below\_A = 0,5 \* (1 - s4)  
pA\_stay\_on\_A = 1 \* s4

Den regulären Regeln nach ist der Seitenabtrieb des Laubs von Feld A oben und unten gleich groß. Daher gehe ich davon aus, dass dem auch in diesem Sonderfall so ist.

Es wird der *Parameter s4* festgelegt, der festlegt, wie groß der Anteil des auf Feld A verbleibenden Laubs ist. Wegen der starken Krafteinwirkung des Laubbläsers auf Feld A (erkennbar daran, dass dieses Feld im Normalfall vollständig geleert wird) ist es plausibel, dass auch in diesem Sonderfall fast das ganze Laub von Feld A heruntergeweht wird. Es wird daher im Folgenden mit s4 = 0,05 gearbeitet.

**Sonderfall 5: A ist ein Eckfeld (➔ B existiert nicht)**

pA\_to\_above\_A = 0,5 \* (1 - s4)  
pA\_stay\_on\_A = 0,5 + 0,5 \* s4

Das Laub, das normalerweise auf das Feld unter A geweht worden wäre, wird jetzt in die Mauerecke geblasen.

**Sonderfall 6: Feld 0 ist ein Rand- oder Eckfeld**

Ein Bild, das Reihe, Screenshot, Rechteck, Design enthält.

Automatisch generierte BeschreibungEs wird davon ausgegangen, dass der Luftstrom an der Mauer nicht stark genug reflektiert wird, um die Blätter nennenswert zu bewegen. Daher wird festgelegt, dass in diesem Fall nichts passiert. Das direkte Blasen gegen die Mauer ist somit sinnlos. Im Übrigen wäre es auch nicht förderlich für den Zustand der Mauer. Es kann nicht im Sinne des Hausmeisters sein, den Schulhof zu demolieren. Daher wird in der Strategie auf ein direktes Blasen gegen die Mauer verzichtet.

## Modellierung der Wahrscheinlichkeiten

Angenommen auf Feld A befindet sich *genau ein Blatt*. Wird nun mit einem Laubbläser von Feld 0 aus in Richtung A geblasen, dann handelt es sich hierbei um ein Zufallsexperiment mit drei möglichen Ereignissen:

|  |  |  |
| --- | --- | --- |
| **Experiment:** Das Blatt auf Feld A wird geblasen | | |
| **Ereignis 1:** Blatt landet auf B (im folgenden E1) P(E1) = pA\_to\_B (den Blasregeln entnehmbar) | **Ereignis 2:** Blatt landet auf dem Feld über B (E2) P(E2) = pA\_to\_above\_B | **Ereignis 3:** Blatt landet auf dem Feld unter B (E3) P(E3) = pA\_to\_below\_B |

### Formulierung als Bernoulli-Experiment

Wie bei jedem Zufallsexperiment, das eine endliche Anzahl an Ereignissen hat, lässt sich auch dieses Zufallsexperiment in mehrere miteinander verknüpften Bernoulli-Experimenten umformulieren. Ein Bernoulli-Experiment hat nur ein Ereignis, das entweder eintritt oder nicht eintritt. Das obige Zufallsexperiment lässt sich in zwei miteinander verknüpfte Bernoulli-Experimente umformulieren[[1]](#footnote-2):

Das erste Bernoulli-Experiment überprüft, ob das Ereignis „Blatt landet auf Feld B“ (E1) zutrifft – die Wahrscheinlichkeit hierfür ist pA\_to\_B und in den Regeln für alle Fälle definiert.  
Trifft (E1) nicht zu, dann wird im zweiten Bernoulli-Experiment (E2) überprüft.  
Da (E2) nur dann angewandt wird, wenn (E1) nicht zutrifft, handelt es sich bei der Wahrscheinlichkeit dafür, dass das zweite Bernoulli-Experiment wahr ist, um die bedingte Wahrscheinlichkeit P¬E1(E2) =   
P(¬E1 ∩ E2) /­­­ P(¬E1) =[[2]](#footnote-3) P(E2) /­­­ P(¬E1) = (pA\_to\_above\_B ) / (1 - pA\_to\_B).

|  |  |
| --- | --- |
| **Ereignis von Bernoulli-Experiment 1:** Das Blatt von A landet auf B (E1) | |
| **Wahr** p = P(E1) = pA\_to\_B | **Falsch** p = 1 - P(E1) = P(¬E1) = 1 - pA\_to\_B   |  |  | | --- | --- | | **Ereignis von Bernoulli-Experiment 2:** Das Blatt von A landet auf dem Feld über B (E2) | | | **Wahr** p = P¬E1(E2) = P(E2) /­­­ P(¬E1) = (pA\_to\_above\_B )/ (1 - pA\_to\_B) | **Falsch** p = 1 - P¬E1(E2) = 1 - (pA\_to\_above\_B )/(1 - pA\_to\_B)  Der einzig übrig bleibende Fall ist (E3), der folglich auf das Blatt zutreffen muss. | |

Angenommen, es befinden sich nicht mehr nur 1 Blatt, sondern x Blätter auf Feld A (mit x ∈ ℕ), dann lässt sich *Bernoulli-Experiment 1* auf jedes dieser x Blätter einzeln anwenden. Hierdurch erhält man eine Bernoulli-Kette[[3]](#footnote-4) der Länge x. Die Voraussetzungen für das Vorhandensein einer Bernoulli-Kette sind:

* *Zwei mögliche Ergebnisse:*  
  Diese Voraussetzung ist durch die Formulierung als Bernoulli-Experiment erfüllt.
* *Unabhängigkeit der Versuche:*  
  Es wird vereinfachend davon ausgegangen, dass sich die Blätter beim Fliegen nicht gegenseitig beeinflussen. Dadurch ist diese Voraussetzung auch erfüllt.
* *Konstante Erfolgswahrscheinlichkeit:*Die Regeln schreiben Wahrscheinlichkeiten fest, die konstant sind bzw. während einem Blasvorgang konstant bleiben.
* *Diskrete Anzahl von Versuchen (n):*  
  Auf jedem Feld liegt eine diskrete, ganzzahlige Anzahl an Blättern, die die Anzahl an Versuchen bzw. Länge der Bernoulli-Kette darstellt (weil das Bernoulli-Experiment auf jedes Blatt einzeln angewandt wird und somit jedes Blatt als Versuch betrachtet werden kann).

Da die Voraussetzungen erfüllt sind, ist die Modellierung als Bernoulli-Kette legitim.

Um zu bestimmen, wie groß die Wahrscheinlichkeit ist, dass genau k Blätter von Feld A (aus den x Blättern, die auf Feld A liegen) auf Feld B landen, lässt sich folglich die Binomialverteilung erwenden: (hierin besteht der große Vorteil in der Darstellung als Bernoulli-Experiment)  
Die Zufallsgröße X zählt, wie viele Blätter von Feld A auf Feld B fliegen. Die Wahrscheinlichkeit, dass k Blätter von Feld A auf Feld B fliegen, beträgt somit P(X=k) mit n=x und p=P(E1).  
Auf die *x-k* Blätter, die nicht auf Feld B fliegen, lässt sich nun das zweite Bernoulli-Experiment anwenden: Die Zufallsgröße Y zählt, wie viele der Blätter, die nicht auf Feld B fliegen, stattdessen auf dem Feld über B landen. Die Wahrscheinlichkeit, dass j von diesen Blättern auf dem Feld über B landen, beträgt somit P(Y=j) mit n=*x-k* und p= P¬E1(E2), der Wahrscheinlichkeit des 2. Bernoulli-Experiments.

Ein Bild, das Text, Diagramm, Reihe, Screenshot enthält.

Automatisch generierte Beschreibung

Abbildung 3: Beispiel zur Veranschaulichung – Szenario: 10 Blätter liegen auf Feld A (links). Aus dem Histogramm (rechts) kann entnommen werden, wie groß die Wahrscheinlichkeit dafür ist, dass genau k aus diesen 10 Blättern auf Feld B fliegen.

Nach demselben Prinzip kann auch für ein Blatt, das sich auf Feld B befindet, ein Bernoulli-Experiment definiert werden:

|  |  |
| --- | --- |
| **Ereignis von Bernoulli-Experiment 3:** Das Blatt von B bleibt auf B | |
| **Wahr** p = pB\_stay\_on\_B | **Falsch** p = 1 - pB\_stay\_on\_B |

In Sonderfällen (siehe Vorkapitel) müssen die Bernoulli-Experimente entsprechend neu festgelegt werden. Die Festlegung erfolgt dabei nach demselben Vorgehen wie zuvor beschrieben und wird hier nicht im Einzelnen durchgegangen.

### Berechnen von P(X=k)

Die Wahrscheinlichkeitsfunktion der Binomialverteilung: P(X=k) = mit n = Länge der Bernoulli-Kette und k = Anzahl Versuche, bei denen das Ereignis des Bernoulli-Experiments eintritt. Eine Implementierung der Funktion in dieser Form zieht jedoch einige Probleme nach sich:

Der Binomialkoeffizient wird **bei großen n-Werten und k-Werten nahe an sehr groß**. So beträgt z.B. nur 252, während bereits 184756 ergibt und bei der Berechnung von eine 100-stellige Zahl herauskommt. Dies liegt daran, dass der Wert von exponentiell wächst, wenn n ansteigt. ist immer positiv und ganzzahlig und kann daher in einem Integer gespeichert werden. Die verwendete Programmiersprache (Python) verwendet für Integer-Werte eine dynamische Speicherverwaltung, was bedeutet, dass Integer-Werte so groß werden können, wie der verfügbare Speicherplatz auf dem System es zulässt – große -Werte sind daher grundsätzlich kein Problem. Problematisch ist allerdings die Multiplikation : Da als Float verliegt, muss für die Multiplikation auch in einen Float umgewandelt werden. In Python führt die **Konvertierung von langen Integers zu einem *OverflowError***. Wie groß der Wert ist, ab dem der *OverflowError* auftritt ist von der Computerarchitektur abhängig, bei 64-bit-Geräten beträgt er ca. 1.7976931348623157e+308. Für große n > 1000 kann diesen Wert überschreiten (so ergibt z.B. eine 329-stellige Zahl.

Außerdem ergibt für große n verschwinden kleine Dezimalzahlen nahe Null. In Python werden **Floats mit einer begrenzten Genauigkeit gespeichert** (15 bis 16 Dezimalstellen werden genau gespeichert), was bei großen n zu einem Genauigkeitsverlust führen würde.

Daher verwende ich zur Berechnung von P(X=k) die **logarithmische Darstellung** der Formel von Bernoulli: .

Die logarithmische Darstellung ist äquivalent zur üblichen Darstellung. Herleitung:

P(X=k) = (die übliche Darstellung der Formel von Bernoulli)

⇔ ln(P(X=k)) = | Beide Seiten logarithmieren

⇔ ln(P(X=k)) = | Angewandte Logarithmusregel:

⇔ ln(P(X=k)) = | Angewandte Logarithmusregel:

| Beide Seiten exponentieren

**Vorteile der logarithmischen Darstellung:**

Die sehr große Zahl wird nicht direkt mit einer Fließkommazahl verrechnet, sondern durch Anwenden des natürlichen Logarithmus zunächst in eine deutlich kleinere Zahl umgewandelt. Hierdurch werden *OverflowError*s vermieden. Außerdem ist das Berechnen der Summen von Logarithmen numerisch stabiler als das direkte Berechnen des Produkts. Die Multiplikation einer sehr großen Zahl mit einer sehr kleinen Zahl kann zu erheblichen Ungenauigkeiten führen, die in der logarithmischen Darstellung nicht auftreten.

Die logarithmische Darstellung hat zur Folge, dass die Sonderfälle p = 0 und p = 1 abgefangen werden müssen, um zu vermeiden (0 ist außerhalb der Definitionsmenge von ln()).

### Möglichkeiten zur Modellierung der Regeln

Bei der Simulation eines Laubblasvorgangs muss mit den probabilistischen Regeln angemessen umgegangen werden. Hierfür existieren unter anderem diese Möglichkeiten:

1. **Tatsächliche Simulation des Zufalls:**In den folgenden Schritten wird mit den zuvor definierten Bernoulli-Experimenten gearbeitet, die für den Regelfall (Feld 0, A und B sind vorhanden und keine Randfelder) gelten.

* *Schritt 1:* Bernoulli-Experiment 1 wird auf alle x Blätter, die sich auf Feld A befinden, angewendet. Hierfür wird die Zufallsgröße X eingeführt, die angibt, wie viele Blätter von A auf B landen. Anschließlich wird die Binomialverteilung P(X=k) für alle Fälle 0 <= k <= x mit n=x und p=P(Bernoulli-Experiment 1) berechnet, die hieraus resultiertenden Wahrscheinlichkeitswerte werden gespeichert.
* *Schritt 2:* Unter Verwendung einer Zufallsfunktion (wie z.B. *random.choice* in Python) wird einer der Fälle für k zufällig ausgewählt. Die Wahrscheinlichkeiten, zu denen die Zufallsfunktion die einzelnen Fälle auswählt, müssen hierbei den Wahrscheinlichkeiten entsprechen, die über die Binomialverteilung für den jeweiligen Fall ermittelt wurden. Auf die Art wird ermittelt, wie viele Blätter in der Simulation auf Feld B fliegen.
* *Schritt 3:* Auf die x-k Blätter, die nicht auf Feld B fliegen, wird nun analog zu Schritt 1 Bernoulli-Experiment 2 angewendet.
* *Schritt 4:* Analog zu Schritt 2 wird unter Verwendung einer Zufallsfunktion ermittelt, wie viele Blätter in der Simulation auf dem Feld über B landen. Die Blätter, die nicht auf dem Feld über B landen, landen folglich auf dem Feld unter B (einziges übrig bleibendes Szenario).
* *Schritt 5:* Auf die y Blätter, die sich auf Feld B befinden, wird analog zu Schritt 1 Bernoulli-Experiment 3 angewendet.
* *Schritt 6:* Analog zu Schritt 2 wird unter Verwendung einer Zufallsfunktion ermittelt, wie viele Blätter von Feld B auf Feld B liegen bleiben. Die Blätter, die nicht auf Feld B bleiben, müssen folglich auf das Feld rechts von Feld B abgetrieben werden (einziges übrig bleibendes Szenario).

Eine solche Simulation ist zwar am realistischsten, liefert aber bei mehrfachem Ausführen derselben Blasoperation auf identischen Höfen unterschiedliche Ergebnisse und ist daher nicht dafür geeignet, verschiedene Strategien in Hinblick auf ihre Qualität zu vergleichen. Wenn eine Strategie mit dieser Simulation arbeitet, dann muss sie dynamisch auf den aus den vorherigen Blasvorgängen resultierenden Zustand reagieren können, da nicht eindeutig vorhersagbar ist, wie viele Blätter auf welches Feld fliegen werden.

1. **Immer den wahrscheinlichsten Fall wählen:**Um deterministische Ergebnisse zu erhalten, kann man das zuvor vorgestellte Vorgehen so abändern, dass immer der wahrscheinlichste Fall ausgewählt wird. Hiermit gehen jedoch einige Probleme bzw. offene Fragen einher:

* Es ist möglich, dass zwei Fälle gleich wahrscheinlich sind. Welcher Fall sollte dann gewählt werden?
* Die Simulation ist nicht mehr realitätsbezogen, da in der Realität natürlich nicht immer der wahrscheinlichste Fall eintritt.

1. Es ist auch eine Simulation denkbar, die die Wahrscheinlichkeiten nicht auf die einzelnen Blätter, sondern auf „Blattmengen“ anwendet:

*Definitionen:*

A\_value := Blattanzahl auf A vor Durchführung der Blasoperation  
B\_value := Blattanzahl auf B vor Durchführung der Blasoperation  
*Neue Blattmengen auf den Feldern, die aus dem Blasvorgang resultieren:*  
new\_B\_value = B\_value \* (1 - pB\_to\_beyond\_B ) + A\_value \* pA\_to\_B

Seitenabtrieb auf Feld über B = A\_value \* pA\_to\_above\_BSeitenabtrieb auf Feld unter B = A\_value \* pA\_to\_below\_B

Seitenabtrieb auf Feld rechts von B = B\_value \* pB\_to\_beyond\_B

new\_A\_value = 0

Da der Erwartungswert in einer binomialverteilten Größe als n \* p definiert ist, lässt sich sagen, dass hier die **Erwartungswerte****verwendet** werden. Es wird also simuliert, wie die Blätter durchschnittlich flügen, wenn man die simulierte Laubblasoperation sehr oft wiederholen würde. *Vorteile dieser Herangehensweise sind:*

* Da simuliert wird, wie die Blätter durchschnittlich fliegen, ist diese Simulationsart besonders gut für das Entwerfen und Testen einer Strategie geeignet.
* Wird dieselbe Blasoperation mehrfach auf identischen Höfen ausgeführt, dann ist das Ergebnis immer gleich, was das Vergleichen verschiedener Strategien in Hinblick auf ihre Qualität erleichtert.

*Nachteile:*

* Die Blattanzahl pro Feld bleibt nicht ganzzahlig, sondern wird rational.
* Die Aufgabe legt über die Formulierung  
  „*und ungefähr 10% ist so zu verstehen, dass jedes Blatt mit Wahrscheinlichkeit 10% „beschließt“, das entsprechende Feld aufzusuchen“*nahe, dass die Wahrscheinlichkeiten auf jedes Blatt einzeln anzuwenden sind und nicht auf „Blatt-Mengen“ – eine Simulation dieser Art ist also eher nicht erwünscht.

*Zusammenfassung:* Zum Entwickeln und Testen einer Strategie eignet sich diese Simulationsart, sie ist aber keine realistische Simulation der tatsächlichen Anwendung.

### Berechnen des Resultats eines Blasvorgangs (Pseudocode)

Die in den Vorkapiteln vorgestellten Verfahren zum Berechnen der aus einem Blasvorgang resultierenden Blattverteilung werden hier als Pseudocode formalisiert. Zunächst ein Algorithmus zur Berechnung der Blasrichtung, die zu einer gegebenen Richtung direction (als Einheitsvektor vorliegend) orthogonal ist:

**Algorithmus 1: get\_orthogonal\_direction (direction)**

1. **return** (0,1) wenn direction[1] == 0, ansonsten (1,0)

**Algorithmus 2: blase(feld0, blow\_direction), entweder über Erwartungswerte oder über die Berechnung der Binomialverteilung und Simulation des Zufalls**

1. orthogonal\_direction = get\_orthogonal\_direction(direction)
2. feldA ermitteln, indem der Vektor blow\_direction zur Position feld0 addiert wird
3. feldB ermitteln, indem der Vektor blow\_direction zur Position feldA addiert wird
4. überprüfen, ob Feld B existiert. Wenn ja:
5. neue Blattanzahl auf Feld B mit entsprechendem Verfahren (Erwartungswerte oder   
    Binomialverteilung-Zufallssimulation) bestimmen
6. Felder über und unter Feld B ermitteln, indem orthogonal\_direction zur Position feldB einmal   
    addiert und einmal subtrahiert wird
7. Seitenabtriebsgröße, der auf je eines dieser Felder gerät, mit entspr. Verfahren bestimmen
8. Die Blattanzahlen auf den beiden Feldern über und unter B aktualisieren. Wenn eines der   
    beiden Felder nicht existiert, dann wird das Laub stattdessen zur Anzahl an sich auf B   
    befindenden Blättern addiert
9. Feld rechts von Feld B bzw. gegenüber von Feld A ermitteln, indem blow\_direction zur   
    Position feldB addiert wird
10. Seitenabtriebsgröße, der auf dieses Feld gerät, bestimmen
11. Die Blattanzahl auf dem Feld rechts von Feld B aktualisieren. Wenn dieses Feld nicht   
     existiert, dann wird basierend auf dem Parameter s1 entschieden, wie viele Blätter auf Feld B   
     und wie viele auf die Felder über und unter B geweht werden.
12. Ansonsten:
13. Seitenabtriebsgröße, der auf die Felder über und unter Feld A gerät, bestimmen.
14. Blattanzahlen auf Feld A und den Feldern über und unter A bestimmen. Wenn eines der  
     der Felder über oder unter A nicht existiert, dann verbleibt der entsprechende Seitenabtrieb   
     auf Feld A.

In den folgenden Kapiteln werden verschiedene Lösungsstrategien zum Sammeln des Laubs auf einem Feld Q vorgestellt, die in ihrer Konzeption grundlegende Unterschiede aufweisen. Es werden die Vor- und Nachteile der einzelnen Ansätze aufgezeigt, außerdem wird bewertet, welcher Ansatz am besten die Aufgabenstellung erfüllt.

## Trivialer Ansatz: Heuristisches Vorgehen (1. Ansatz)

*Hinweis: Es handelt sich bei diesem Ansatz nicht um meinen besten Ansatz. Im nächsten Kapitel auf Ebene 2 stelle ich einen besseren Ansatz vor.*

Dieser Ansatz ist konzeptionell am einfachsten: Es wird ein Greedy-Algorithmus durchgeführt, der während des Blasprozesses vor jedem Blasvorgang [[4]](#footnote-5) …

1. zunächst alle möglichen Blasoperationen bestimmt,
2. für jede dieser Blasoperationen den aus der Operation hervorgehenden Hof bestimmt,
3. basierend auf einer Heuristik jeden dieser Höfe „bewertet“ bzw. jedem Hof einen Score zuweist und
4. den Blasvorgang auswählt, aus dem der Hof mit dem höchsten Score resultiert.
5. Der ausgewählte Blasvorgang wird anschließend am „tatsächlichen“ Hof durchgeführt.

Das Ziel von Schritt 2 besteht darin, verschiedene Szenarien durchzurechnen, um schließlich die Blasoperation auszuwählen, die voraussichtlich den besten Gesamteffekt auf den Hof hat. Daher werden in Schritt 2 beim Simulieren der Blasvorgänge die Erwartungswerte verwendet, um ein deterministisches Ergebnis zu erhalten, das eine möglichst gute Aussage über die zu erwartende Wirkung des Blasvorgangs trifft.  
Man kann sich das so vorstellen, dass der Hausmeister in Schritt 2 verschiedene Szenarien durchrechnet, um sich anschließend für die Blasoperation entscheiden zu können, die am wahrscheinlichsten den besten Gesamteffekt auf den Hof hat.

Im fünften Schritt wird dann der tatsächliche Blasvorgang auf den realen Hof angewendet. Dabei wird der tatsächliche Zufall über die Binomialverteilung simuliert. In diesem Schritt geht es nicht mehr darum, Vorhersagen zu treffen, sondern darum, die Blasoperation tatsächlich zu simulieren. Hier greift der Hausmeister also tatsächlich zu seinem Laubbläser und führt die Operation durch.

Dieser Ansatz wird nur zum Leeren von Nicht-Randfeldern angewendet, für Randfelder eignet er sich nicht. Im folgenden Teil wird es daher zunächst um das Leeren von Nicht-Randfeldern gehen, sämtliche Blasoperationen, bei denen Feld 0 ein Randfeld ist oder die auf den Rand blasen, werden daher zunächst ausgeklammert. Später wird dann das Leeren des Rands thematisiert.  
 Formularbeginn

### Kenngrößen zur Bewertung des Zustands eines Hofs

Für die Heuristik in Schritt 3 (siehe Vorkapitel) werden Kenngrößen ermittelt, die eine Aussage darüber treffen, wie „gut“ im Sinne von „erstrebenswert“ ein Hof-Zustand ist.

Die Felder des Hofs sind rasterförmig angeordnet, daher ist eine Verwendung der euklidischen Distanz nicht sinnvoll. Stattdessen wird immer die Manhattan-Distanz verwendet, um den Abstand zwischen zwei Feldern zu bestimmen. Die Manhattan-Distanz zwischen den Punkten P und Q ist definiert als *|Q[0] – P[0]| + |Q[1] – P[1]|*.

**1. Kenngröße: Durchschnittliche Blattdistanz zu Feld Q**

Angenommen, auf einem Feld P ungleich Q befindet sich ein Blatt B, das auf Feld Q gebracht werden soll. Im Zielzustand befindet sich B auf Q und hat somit den Abstand 0 zu Q. Im Anfangszustand ist der Abstand zwischen B und Q folglich größer als 0 (da P ungleich Q). Eine Sequenz an Blasoperationen, die B von Feld P auf Q befördert, muss den Abstand zwischen B und Q also insgesamt reduzieren. Die Distanz zwischen Blatt B und Feld Q ist also eine Kenngröße, die gut für die Verwendung in einer Greedy-Heuristik geeignet ist, die in jedem Schritt die Blasoperation auswählt, die die Distanz am stärksten reduziert.

Ziel ist es nun natürlich nicht, ein einziges Blatt auf Q zu blasen, sondern alle Blätter des Hofs auf Q zu bringen. Als Kenngröße wird also die durschnittliche Blattdistanz zwischen allen Blättern auf dem Hof und Q verwendet, die sich mit folgendem Rechenausdruck berechnen lässt:

*blattverteilung((x,y))* repräsentiert dabei die Anzahl an Blättern auf dem Feld am Index (x,y), n ist die Höhe bzw. Breite des quadratischen Hofs und *sum(blattverteilung)* repräsentiert die Gesamtanzahl an Blättern auf dem Hof. Der obige Ausdruck berechnet die durchschnittliche Blattdistanz, indem für jedes Feld der Manhattan-Abstand berechnet und mit der Anzahl an Feldern auf dem Feld multipliziert wird. Die Ergebnisse werden über alle Felder gemittelt.

Im angestrebten Endzustand befinden sich alle Blätter auf Feld Q. In diesem Zustand ist die durchschnittliche Blattdistanz 0, faktisch ist er mit dieser Heuristik jedoch nicht erreichbar (mehr dazu später). Daher müssen *Abbruchkriterien* festgelegt werden. Wenn mind. eines der folgenden Abbruchkriterien erfüllt ist, dann wird der Greedy-Algorithmus auf jeden Fall abgebrochen:

* Es gibt keine Blasoperationen mehr, die eine Reduzierung der durchschnittlichen Blattdistanz zur Folge hätten.
* Der Parameter *satisfied\_constraint* ∈ [0;1] wird eingeführt. Sobald der Anteil des Laubs auf Q am Gesamtlaub, das sich auf dem Hof befindet, größer als *satisfied\_constraint* ist, wird der Greedy-Algorithmus abgebrochen.
* Der Parameter *max\_operations* ∈ ℕ wird eingeführt. Sobald mindestens *max\_operations* Blasoperationen durchgeführt wurden, wird der Greedy-Algorithmus abgebrochen.

Bei einer Anwendung des Greedy-Algorithmus mit dieser Heuristik-Kenngröße wird das Laub zwar auf Feld Q geblasen, dies geschieht aber sehr unorganisiert:  
Angenommen, man hat einen Hof der Größe (9,9) und als Feld Q das Feld in der Mitte bzw. Feld (4,4). Der Greedy-Algorithmus wird als erstes versuchen, sich auf ein von Feld Q relativ weit entferntes Feld zu stellen und Laub Richtung Q zu blasen (siehe Abb. 4, Mitte). Der größte daraus resultierende Laubhaufen in den folgenden Schritten auf direktem Weg Richtung Q geblasen. Durch das direkte Blasen des Laubhaufens Richtung Q kann der Hausmeister mit nur 5 Blasoperationen die Laubmenge auf Feld Q um fast 300 Blätter (Simulation der Blasvorgänge über die Erwartungswerte) erhöhen.

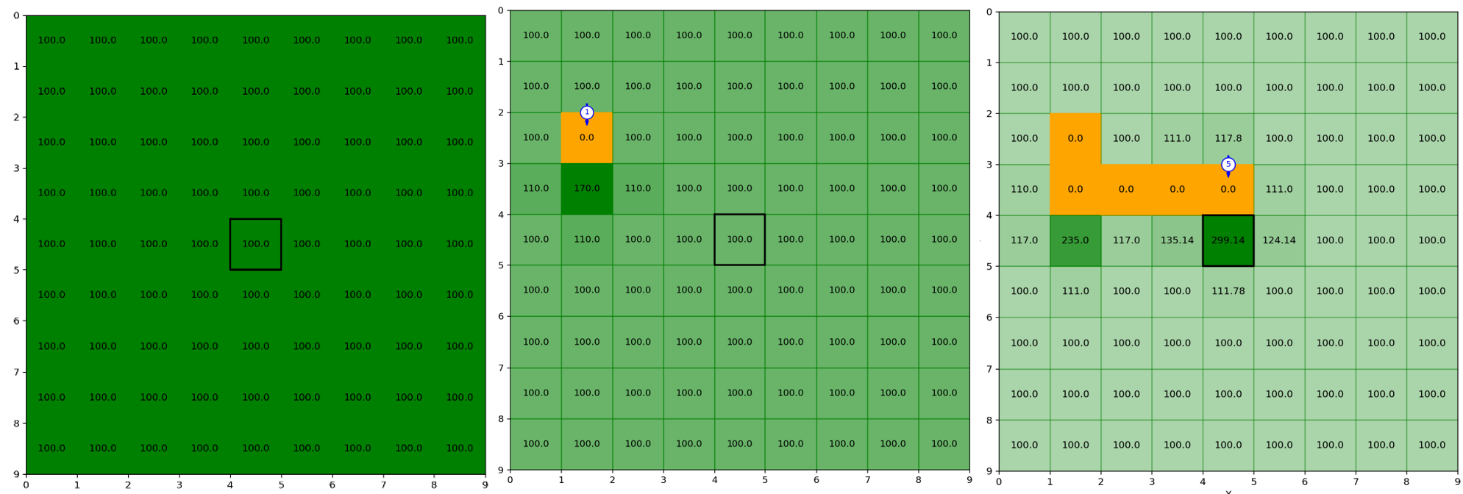


Abbildung 4: Blattverteilung des Hofs bei Anwendung des Greedy-Algorithmus mit durchschnittl. Blattdistanz als Heuristik unter Verwendung der Erwartungswerte nach 0, 1 und 5 Blasoperationen. Feld Q ist schwarz umkastet.

Diese Vorgehensweise ist der verwendeten Heuristik inhärent, da sie immer versucht, am meisten Laub Richtung Q zu blasen, und durch das Blasen des größten Laubhaufens nun mal am meisten Laub transportiert wird.  
Nach drei weiteren Blasoperationen wird allerdings sichtbar, was daran problematisch ist: Um Laub von anderen Feldern ebenfalls auf Q zu bringen, muss Laub über bereits geleerte Felder geblasen werden (siehe Abb. 5).

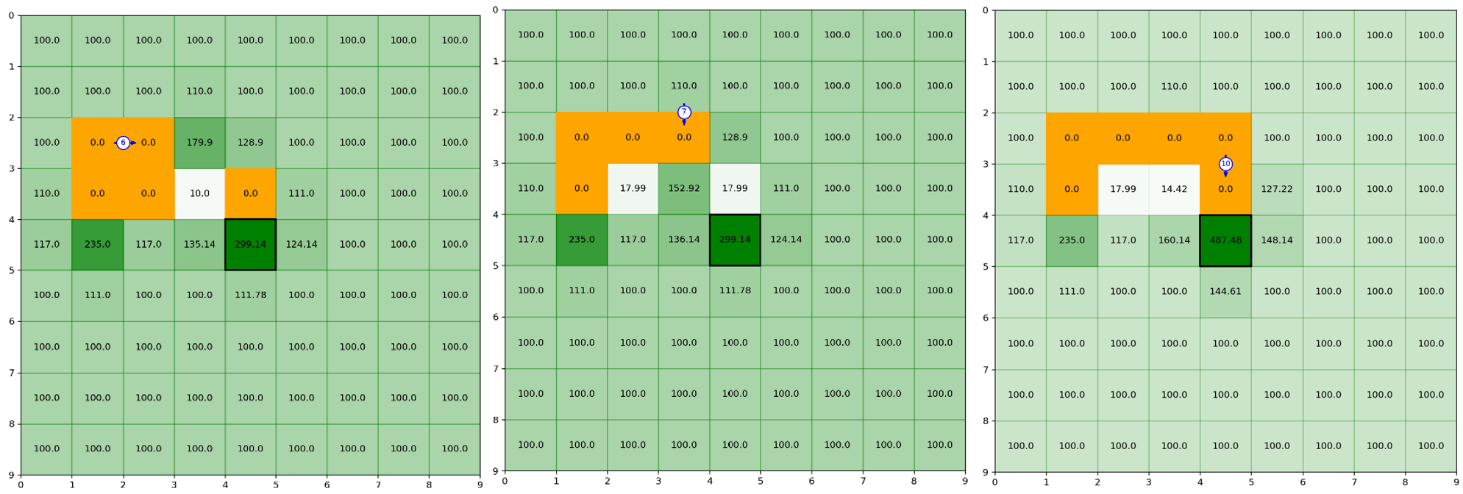


Abbildung 5: Blattverteilung nach 6, 7 und 10 Blasoperationen

Das ist äußerst ungünstig, da die dabei entstehenden Seitenabtriebe dafür sorgen, dass sich auf eigentlich bereits vollständig geleerten Feldern wieder große Mengen an Blättern ansammeln, wie in Abb. 5 sichtbar. Dies sorgt dafür, dass der Hausmeister das Laub dieser Felder erneut auf Q befördern muss – wenn hierbei erneut die durchschnittl. Blattdistanz als Heuristik verwendet wird, entstehen dabei aber wieder Seitenabtriebe auf bereits vollständig geleerte Felder. Wenn nur kleine Mengen an Laub auf Feld Q geblasen werden sollen, können diese Seitenabtriebe ignoriert werden. Soll aber möglichst viel Laub auf Q gelangen, müssen die durch Seitenabtriebe auf bereits geleerte Felder gelangenen Blätter ebenso auf Feld Q geblasen werden müssen, wobei wiederum neue Seitenabtriebe entstehen usw. Dies verlangsamt den Blasprozess deutlich.

Folglich kann der Hausmeister zwar sehr schnell kleinere Mengen an Laub (< 50% der Gesamtlaubmenge) auf Feld Q bringen. Um größere Mengen an Laub (> 50%) auf Feld Q zu konzentrieren, braucht er aber sehr lange. Generell lässt sich beim Blasen von Blättern Richtung Q zwischen einem „schnellen“ und einem „nachhaltigen“ Verfahren unterscheiden, was ich an einem eindimensionalen Beispiel verdeutlichen werden:

|  |  |
| --- | --- |
| **Schnelles Verfahren:**    Abbildung 6: Das schnelle Verfahren hat zwar nach nur einem Schritt bereits 90 Blätter auf Feld Q (schwarz umkastet) gebracht. Um >270 Blätter auf Q zu bringen, braucht es aber einen Schritt mehr als das nachhaltige Verfahren. | **Nachhaltiges Verfahren:**    Abbildung 7: Das nachhaltige Verfahren kann in einem Schritt weniger >270 Blätter auf Q bringen als das schnelle Verfahren. Dafür braucht es allerdings einen Schritt mehr als das schnelle Verfahren, um >150 Blätter auf Q anzuhäufen. |

Ein schnelles Verfahren zeichnet sich dadurch aus, dass innerhalb von möglichst wenig Schritten das Laub von den Nachbarfeldern von Q auf Q geblasen wird. Ein nachhaltiges Verfahren hingegen leert immer das Feld zuerst, das am weitesten von Q entfernt ist.

Würde man den auf der Blattdistanz-Heuristik basierenden Greedy-Algorithmus auf einen eindimensionalen Hof (einen Hof mit einer Höhe von 1) anwenden, auf dem Q ein Eckfeld ist, dann würde er sich wie ein nachhaltiges Verfahren verhalten. Wie zuvor gezeigt sorgt die Blattdistanz-Heuristik dafür, dass im ersten Schritt auf einem von Q weit entfernten Feld ein Laubhaufen gebildet wird, der anschließend auf direktem Weg Richtung Q geblasen wird (dies wird in Abb. 4 gut sichtbar). Dies entspricht genau der nachhaltigen Vorgehensweise, die in Abb. 7 dargestellt ist.

Die zu betrachtenden Höfe sind allerdings zweidimensional. Auf zweidimensionaler Ebene ist der verwendete Greedy-Algorithmus kein nachhaltiges Verfahren. Ein nachhaltiges Verfahren müsste einen zweidimensionalen Hof leeren, indem es zunächst alle Felder leert, die die Entfernung d von Q haben, anschließend alle Felder leert, die die Entfernung d-1 von Q haben usw. mit d := maximaler Manhattan-Abstand, den ein Feld des Hofs zu Q hat (Randfelder und Felder, die nur über Blasen von einem Randfeld aus geleert werden können, sind hier – wie zuvor erwähnt – nicht berücksichtigt).

Der beschriebene Greedy-Algorithmus geht anders vor: Er leert zunächst ein Feld, das die Entfernung d von Q hat, anschließend leert er ein weiteres Feld, das die Entfernung d-1 von Q hat usw. Dies reicht aus, um auf einem eindimensionalen Hof nachhaltig zu leeren, da es hier nur ein Feld gibt, das den Abstand d zu Q hat (wenn Q ein Eckfeld ist). Auf einem zweidimensionalen Hof ist ein nachhaltiges Leeren so aber nicht mehr möglich.

**2. Kenngröße: Varianz der Blattdistanzen zu Feld Q**

Um die mit der zuvor beschriebenen Greedy-Heuristik einhergehenden Nachteile, brauchen wir eine andere heuristische Kenngröße.

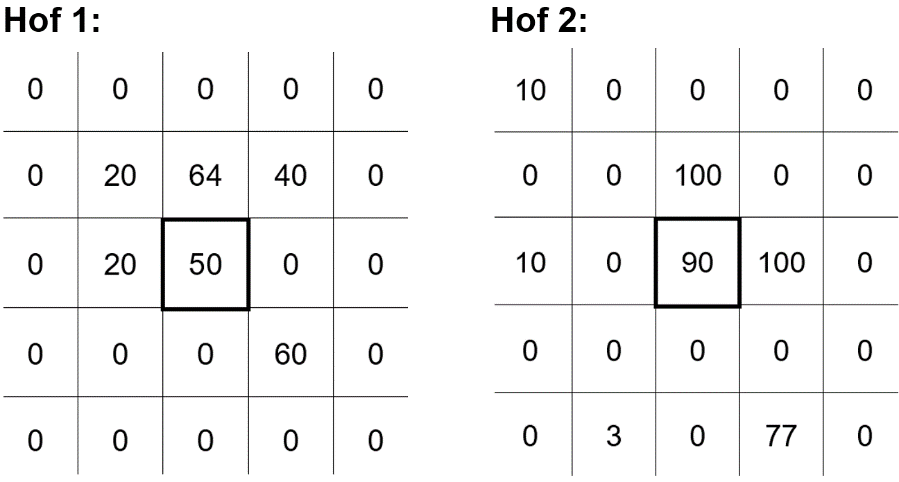


Abbildung 8: Ausschnitte aus zwei Höfen im Vergleich

Auf Hof 1 dürfte die durchschnittliche Blattdistanz einen ähnlichen Wert betragen wie auf Hof 2. Trotzdem ist es auf Hof 1 deutlich einfacher bzw. geht deutlich schneller, > 50% des Laubs auf Feld Q transportieren als auf Hof 2. Dies liegt daran, dass das Laub auf Hof 2 „verstreuter“ bzw. weniger geordnet als auf Hof 1 ist. Während es auf Hof 1 einen „Laubcluster“ gibt, dessen Mittelpunkt Feld Q ist, gibt es auf Hof 2 mehrere separate Laubcluster, zwischen denen leere Felder liegen, was (wie im Vorkapitel erläutert) für den weiteren Blasprozess ungünstig ist.

Der Zustand von Hof 1 ist folglich deutlich erstrebenswerter als der von Hof 2. Um eine Heuristik zu entwickeln, die dies berücksichtigt, müssen wir die „Unordnung“ / „Streuung“ der Blätter auf dem Hof quantifizieren. Als Maß hierfür bietet sich die Standardabweichung der Blattdistanzen zu Q an.

Die Standardabweichung der Elemente einer Menge m der Länge n lässt sich mit folgender Formel berechnen: Da es bei einem Greedy-Algorithmus aber nur um das Vergleichen der basierend auf der Kenngröße berechneten „Scores“ geht, kann auch genausogut die kumulierte Varianz statt der Standardabweichung verwendet werden, um Rechenleistung zu sparen (keine Wurzel): Die Transformation der Standardabweichung zur kumulierten Varianz ist monotonieerhaltend und beeinflusst das Ergebnis daher nicht. Im folgenden wird diese Heuristik-Kenngröße als „Blattvarianz“ bezeichnet.

**Analogie zur Entropie:** In der Thermodynamik existiert ein Maß namens Entropie, das die molekulare Unordnung in einem System bzw. die Nutzbarkeit der inneren Energie eines Systems quantifiziert. Auch wenn die zuvor vorgestellte Heuristik nichts mit Thermodynamik zu tun hat, stellt sie ebenfalls ein Maß für die Unordnung der Blätter auf dem Hof dar, das gleichzeitig ein Maß für die „Nutzbarkeit“ der Blattanordnung ist.

**Algorithmus 3: Berechnung der kumulierten Blattdistanz**

1. blattdistanzen = Leeres Array
2. **für jedes** x **im Bereich** [0 ; n-1]:
3. **für jedes** y **im Bereich** [0 ; n-1]:
4. Anzahl\_Blätter\_auf\_Feld = round(blattverteilung((x, y)))
5. **füge** Anzahl\_Blätter\_auf\_Feld \* manhattan\_distance((x, y)) **zu** blattdistanzen **hinzu**
6. **return** varianz(blattdistanzen)

Mit diesem Algorithmus lässt sich grundsätzlich auch die durchschnittliche Blattdistanz annähern, wenn statt der Varianz der Mittelwert bestimmt werden.

**Anwendung der Blattvarianz-Heuristik:**

Eine reine Anwendung der Blattvarianz-Heuristik, bei der immer die Blasoperation ausgeführt wird, die zur kleinsten Blattvarianz führt, sorgt zunächst nicht dafür, dass Laub auf Q geblasen wird. Stattdessen bewirkt sie, dass die Blattdistanzen der Blätter sich alle demselben Wert annähern (wenn alle Blattdistanzen nahe aneinander sind, dann ist die Abweichung zum Mittelwert folglich am geringsten). Die Blätter würden sich also „ringförmig“ auf Feldern um Q herum anordnen, die alle ähnliche Manhattan-Abstände zu Q haben, was aber natürlich nicht das Ziel des Hausmeisters ist.

Daher muss die Blattvarianz-Heuristik immer der Blattdistanz-Heuristik untergeordnet sein. **Das bedeutet konkret, dass nur die Blasoperationen als mögliche nächste Blasoperation infrage kommen dürfen, die zu einer Verringerung der durchschnittlichen Blattdistanz führen**. Hierdurch wird sichergestellt, dass es sich bei der als nächstes auszuführenden Blasoperation um keine kontraproduktive Operation handelt bzw. das Ziel des Sammelns von Laub auf Q verfolgt wird.

Aus diesen Blasoperationen kann dann die ausgewählt werden, die zur stärksten Verringerung der kumulierten Blattvarianz führt. Hierdurch wird die Unordnung der Blätter so niedrig wie möglich gehalten. Der Nebeneffekt ist hiervon ist aber, dass die Blattvarianz-Heuristik sehr, sehr lange brauchen kann, um nennenswerte Laubmengen auf Q zu versammeln. Wirklich gute Ergebnisse erhält man daher erst, wenn man die Blattvarianz- und Blattdistanz-Heuristik kombiniert.

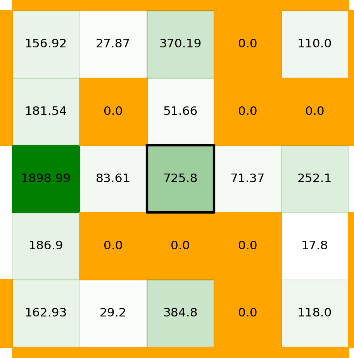


Abbildung 9: Hofausschnitt eines Hofs, auf den die Blattvarianz-Heuristik angewendet wurde

### Kombination der Heuristiken

Es werden die Parameter *weight\_avg* und *weight\_varianz* eingeführt, über die gesteuert werden kann, wie stark die Blattdistanzheuristik (*weight\_avg*) und Blattvarianzheuristik (*weight\_varianz*) an der Entscheidung, welche Blasoperation als nächstes durchzuführen ist, beteiligt sein sollen.

Da die Blattdistanzheuristik eher einem schnellen Verfahren ähnelt, sorgt ein hoher *weight\_avg* und ein niedriger *weight\_varianz* Wert dafür, dass Blasoperationen priorisiert werden, die innerhalb von kurzer Zeit viel Laub auf Feld Q bringen, danach aber deutlich weniger Laub auf Feld Q gelangt.

Die Blattvarianzheuristik hingegen ähnelt eher einem nachhaltigen Verfahren, daher sorgt ein niedriger *weight\_avg* und ein hoher *weight\_varianz* Wert dafür, dass Blasoperationen priorisiert werden, die zunächst eher wenig Laub auf Q bringen, später aber dafür deutlich mehr Laub auf Feld Q gelangt. Dank der Parametrisierung kann der Anwender über *weight\_avg* und *weight\_varianz* basierend auf seiner Zielsetzung das Verhalten des Algorithmus beeinflussen.

**Algorithmus 4: Greedy-Algorithmus mit Kombi-Heuristik  
zum Behandeln von Nicht-Randfeldern,**gibt die als nächstes auszuführende Blasoperation zurück

1. current\_mw\_bd = Durchschnittliche Blattdistanz am aktuellen Hof ermitteln
2. best\_score = -inf
3. **für jedes** x **im** Bereich [0, n-1]:
4. **für jedes** x **im** Bereich [0, n-1]:
5. **für jede** mögliche Blasrichtung blow\_direction:
6. **falls** (x,y) ein Randfeld **und** (x+blow\_direction[0], y+blow\_direction[1] ein Randfeld:
7. **nächste Schleifeniteration**
8. Blasoperation *blase((x,y,), blow\_direction)* an einer Kopie des aktuellen Hofs simulieren
9. new\_mw\_bd = die aus der Blasoperation hervorgehende durchschnittl. Blattdistanz
10. **wenn** new\_mw\_bd < current\_mw\_bd:
11. new\_bv = die aus der Blasoperation hervorgehende durchschnittl. Blattdistanz
12. score = new\_mw\_bd \* weight\_avg + new\_bv \* weight\_varianz
13. **wenn** score > best\_score:
14. best\_score = score
15. best\_op = In dieser Variable die Blasoperation *blase((x,y,), blow\_direction)* speichern
16. **return** best\_op

**Mögliche Optimierung:**

Die Blasoperationen, bei denen Feld B einen größeren Manhattan-Abstand zu Q hat als Feld A, führen sowieso nicht zu einer Verringerung des durchschnittlichen Blattabstands zu Q und brauchen daher gar nicht erst betrachtet zu werden bzw. es kann direkt in die nächste Schleifeniteration gesprungen werden, wenn auf eine solche Blasoperation getroffen wird.

### Behandeln des Rands

Da für Randfelder spezielle Blasregeln gelten, kann der Rand mit der zuvor vorgestellten Heuristik nicht vernünftig geleert werden.

Zunächst ist es sinnvoll, das gesamte sich auf dem Rand befindende Laub auf einem Target-Randfeld bzw. Rand-Zielfeld zu versammeln, von dem aus es dann auf ein Nicht-Randfeld befördert werden kann. (Warum dieses Vorgehen sinnvoll ist, wird später erklärt.) Der schnellste Weg, auf dem dies geschehen kann, besteht darin, das Laub am Rand entlangzublasen.

**Festlegung des Rand-Zielfelds:**Als Rand-Zielfeld sollte das Randfeld genommen werden, das den geringsten Abstand zu Q hat. Anschließend an das Sammeln des Randlaubs auf dem Rand-Zielfeld wird das Laub nämlich mit dem zuvor erläuterten Greedy-Verfahren auf Q transferiert, was schneller geht, wenn das ehemaligie Randlaub bereitsmöglichst nahe an Q dran ist.

Der Algorithmus zum Leeren des Rands benötigt eine Funktion zur Berechnung der Randdistanz. Hierbei handelt es sich um der Länge der minimalen Verbindungsstrecke zwischen zwei Randfeldern X und Y, die nur über Rand- und Eckfelder läuft. (Es handelt sich also um die Länge des kürzesten Wegs, den man am Rand entlang ablaufen müsste, um von Feld X auf Feld Y zu gelangen):

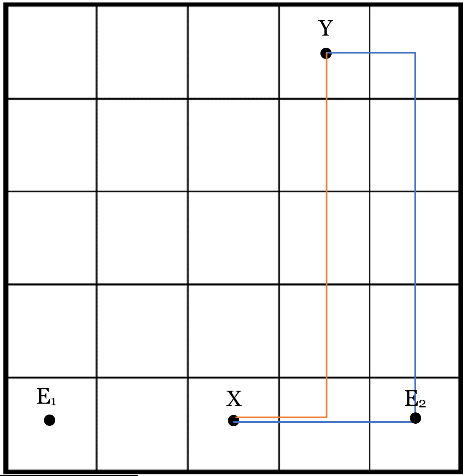


Abbildung 10: Veranschaulichung der Randdistanz: Die Länge der blauen Linie entspricht der Randdistanz. Als Kontrast dazu ist die Linie, deren Länge der Manhattan-Distanz entspricht, orange eingezeichnet. Wie man sieht, entspricht die Manhattan-Distanz nicht der Randdistanz.

Mit einem „Trick“ lässt sich die Randdistanz allerdings über zwei Manhattan-Distanzen berechnen: Wenn sich X und Y *nicht* auf demselben Randsegment befinden, dann kann man sich zunutze machen, dass die Verbindungslinie zwischen X und Y, die der Manhattan-Distanz entspricht, auf jeden Fall über eine der Ecken E1 oder E2 geht (in Abb. 11 eingezeichnt) – und der Abstand zwischen (E1 und Y) oder (E2 und Y) lässt sich wiederum mit der Manhattan-Distanz berechnen. Für diesen Fall gilt also diese Formel:

Ein Bild, das Reihe, Text, Screenshot, Quadrat enthält.

Automatisch generierte Beschreibung

Abbildung 11: Die durchgezogene Linie ist die Linie, deren Länge der tatäschlichen Randdistanz entspricht, die gestrichelte Linie ist die andere infrage kommende Linie

Wenn sich X und Y auf demselben Randsegment befinden, dann ist die Randdistanz zwischen Y und X ganz einfach die Manhattan-Distanz zwischen Y und X.

Um das Randlaub auf dem Rand-Zielfeld zu versammeln wird ein nachhaltiges Verfahren verwendet, das auf einem Greedy-Algorithmus aufbaut. Dieser Algorithmus speichert alle Randfelder, die er bereits geleert hat, in einer Liste. Er wählt in jedem Schritt das noch nicht geleerte Randfeld aus, das am weitesten vom Rand-Zielfeld entfernt ist, und bläst das sich auf dem gewählten Feld befindende Laub den Rand entlang auf ein anderes Randfeld, das näher am Rand-Zielfeld ist:

Ein Bild, das Reihe, Quadrat enthält.

Automatisch generierte Beschreibung

Abbildung 12: Entlangblasen des Laubs am Rand, um es auf einem Rand-Zielfeld zu versammeln

**Algorithmus 5: (grobe Skizzierung des) Greedy-Algorithmus zum Sammeln des Laubs auf dem Rand-Zielfeld,**gibt die als nächstes auszuführende Blasoperation zurück

Parameter: edge\_fields\_to\_clear = Eine Liste mit den noch zu leerenden Randfeldern (vor dem ersten Blasvorgang enthält diese Liste alle Randfelder)  
Parameter: Rand-Zielfeld = Das Randfeld, auf dem das Randlaub versammelt werden soll, bevor es schließlich auf ein Nicht-Randfeld geblasen wird.

1. field\_to\_clear = Das Feld in edge\_fields\_to\_clear finden, das die größte Randdistanz zum Rand-Zielfeld hat
2. target\_field = das Nachbarfeld von field\_to\_clear, das ein Randfeld ist, in edge\_fields\_to\_clear ist und am nächsten am Rand-Zielfeld dran ist (niedrigste Randdistanz)
3. **return** die Blasoperation, die einen Großteil des Laubs vom Feld field\_to\_clear auf das Feld target\_field bläst

Um dabei das Laub eines Randfelds auf ein benachbartes Randfeld (=“Operationszielfeld“) zu transferieren, muss nur einmal von dem Randfeld aus, das sich neben dem zu leerenden Randfeld und gegenüber vom Operationszielfeld befindet, auf das zu leerende Randfeld blasen:

Ein Bild, das Reihe enthält.

Automatisch generierte Beschreibung

Die Eckfelder lassen sich jedoch nicht mit einem Blasen vollständig leeren, da bei dem Blasen auf ein Eckfeld immer nur ein Teil des Laubs auf das Operationszielfeld gelangt. Ein Eckfeld hat nur zwei Nachbarfelder, von denen eines das Operationszielfeld ist (bzw. das Feld, auf das der Hausmeister das Laub befördern will). Das andere Nachbarfeld des Eckfelds ist das Feld, auf das sich der Hausmeister stellen muss, um Laub vom Eckfeld auf das Operationszielfeld zu befördern. Die Blasoperationen, die Eckfelder leeren, sind in Abb. 10 rot eingezeichnet.

Diese Blasoperation muss der Hausmeister dann solange ausführen, bis die Blattanzahl auf dem Eckfeld keine wesentliche Laubmenge mehr liegt bzw. die Restlaubmenge einen bestimmten Toleranzwert unterschreitet. In der Implementierung wird dieser Toleranzwert automatisch auf startwert \* (1-satisfied-constraint) gesetzt (startwert ist die Blattanzahl, die sich zu Beginn auf jedem einzelnen Feld befindet, und satisfied\_constraint ist die Abbruchbedingung des Programms bzw. der Parameter legt fest, wie hoch der Anteil des Laubs auf Q am Gesamtlaub sein muss, damit das Programm auf jeden Fall abgebrochen wird). Dies ist sinnvoll, da die Anzahl des aus den Ecken zu holenden Laubs natürlich davon abhängt, wie viele Blätter man denn überhaupt auf Q bringen möchte bzw. muss.

All dies wirft die Frage auf, ob es möglich ist, Ecken und die Ränder vollständig zu leeren. Mit dieser Frage werde ich mich in Ansatz 2 beschäftigen.

**Transferieren des Laubs vom Rand-Zielfeld** **auf ein Nicht-Randfeld:**

Nach Vollendung der zuvor erläuterten Schritte ist das Laub vollständig auf dem Rand-Zielfeld versammelt und muss nun auf ein Nicht-Randfeld gebracht werden.  
Da sich der Hausmeister nur auf Felder des Hofs stellen darf, ist das „direkte“ Blasen von Laub von einem Rand- auf ein Nicht-Randfeld nicht möglich. Stattdessen können Randfelder geleert werden, imdem auf den Nachbar-Randfeldern des Randfels orthogonal zum Rand das Laub solange hin- und hergeblasen wird, bis die Laubemenge auf den Randfeldern einen bestimmten Betrag unterschreitet.

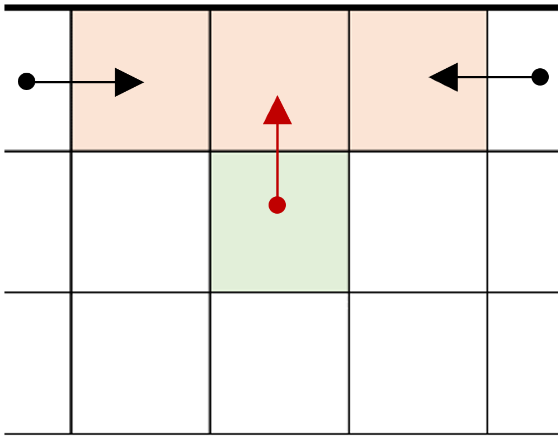


Abbildung 13: Blasoperationen zum Leeren des Rands

Bei jeder der beiden schwarz eingezeichneten Blasoperationen gelangt über den Seitenabtrieb Laub von den rot eingefärbten Feldern auf das grün unterlegte Feld. Um also das Laub von den drei roten Randfeldern auf das grüne Nicht-Randfeld zu befördern, müssen die schwarzen Blasoperationen mehrfach ausgeführt werden. Hierbei akkumuliert zunehmend Laub auf dem rot eingefärbten Feld über dem grün eingefärbten Feld, was den Gesamtprozess verlangsamt. Um dem entgegenzuwirken, kann zwischendurch die rot eingezeichnete Blasoperation eingeschoben werden, die zwar kein Laub auf das grüne Feld befördert, aber stattdessen dafür sorgt, dass bei den folgenden schwarzen Blasoperationen mehr Laub auf das grüne Feld gelangt.

Da bei jeder schwarzen Blasoperation ein bestimmter Prozentsatz des Laubs, das sich auf den roten Feldern, auf das grün eingefärbte Feld geblasen wird, ist das Vorgehen zum Leeren des Randfelds effektiver (d.h. der absolute Wert an pro Blasoperation auf das grüne Feld geblasenen Laubs ist größer), je mehr Laub sich auf den roten Feldern befindet. Hieraus lässt sich schlussfolgern, dass zunächst das gesamte Randlaub auf einem Randfeld versammeln werden sollte, bevor dieses Randfeld dann systematisch geleert wird.

Werden die zuvor erläuterten Vorgänge zu Algorithmus hinzugefügt, dann ergibt sich dieser Gesamt-Algorithmus zum Leeren des Rands. Die zuvor erläuterten Mechanismen sind dort entweder direkt oder indirekt umgesetzt, stellenweise ist der Pseudocode zur Verdeutlichung mit Kommentaren versehen:

**Algorithmus 6: Greedy-Algorithmus zum Leeren des Rands,**gibt die als nächstes auszuführende Blasoperation zurück

Globale Liste: edge\_fields\_to\_clear = Eine Liste mit den noch zu leerenden Randfeldern (vor dem ersten Blasvorgang enthält diese Liste alle Randfelder)  
Parameter: Rand-Zielfeld = Das Randfeld, auf dem das Randlaub versammelt werden soll, bevor es schließlich auf ein Nicht-Randfeld geblasen wird.

Globale Liste: edge\_target\_field\_neighbors = Liste, in der die Nachbarn des Rand-Zielfelds gespeichert werden

Globale Integer-Variable: clear\_edge\_last\_field\_index = Variable, die speichert, welcher Schritt zur Leerung des Rands als nächstes ausgeführt werden soll

1. field\_to\_clear = Wähle das Feld aus edge\_fields\_to\_clear aus, das die größte Randdistanz zum Rand-Zielfeld hat
2. **Falls** sich noch eine wesentliche Laubmenge auf Feld field\_to\_clear befindet:
3. **Wenn** field\_to\_clear ein direkter Nachbar des Rand-Zielfelds ist:
4. **Füge** field\_to\_clear **zu** edge\_target\_field\_neighbors **hinzu (falls noch nicht in Liste)**
5. **Falls** field\_to\_clear Nachbarfeld des Rand-Zielfelds ist:
6. Aktiviere das zyklische Blasverfahren, das das Laub vom Rand-Zielfeld hin und herbläst,  
    um es über die Seitenabtriebe nach und nach auf ein Nicht-Randfeld zu bringen:

Füge die Nachbarfelder des Rand-Zielfelds zur Liste der zu leerenden Felder hinzu

1. Speichere die in diesem Schritt auszuführende Operation des zyklischen   
    Randleerdungsverfahrens.
2. Rotiere durch die Operationen des zyklischen Verfahrens, indem festgelegt wird, das im   
    nächsten Schritt die nächste Operation des zyklischen Randleerungsverfahrens   
    durchgeführt wird (hierfür wird clear\_edge\_last\_field\_index um 1 erhöht, sobald es den   
    Wert 3 erreicht wird es auf 0 zurückgesetzt). Die Operationen des zyklischen   
    Randleerungsverfahrens sind die inAbb. 13 eingezeichneten Blasoperationen, das in  
    Abb. 13 rot eingefärbte und sich über dem   
   grünen Feld befindende Feld ist dabei das Rand-Zielfeld.
3. **return** die in Schritt 7 gespeicherte Blasoperation
4. target\_field = Das Nachbarfeld von field\_to\_clear finden, dass am nächsten am Rand-Zielfeld   
    ist (Rand-Distanz)
5. **Falls** field\_to\_clear **kein** Eckfeld:
6. **Entferne** field\_to\_clear **aus** edge\_fields\_to\_clear (Erläuterung: Nicht-Eckfelder können   
    durch einmaliges Blasen geleert werden)
7. **return** die Blasoperation, die am meisten Laub von field\_to\_clear auf target\_field bringt
8. **Andernfalls:**
9. **Entferne** field\_to\_clear **aus** edge\_fields\_to\_clear
10. **Führe Algorithmus 6 erneut aus**

### Gesamt-Algorithmus

Der Gesamt-Algorithmus setzt sich aus dem Greedy-Algorithmus zum Leeren des Rands (Algorithmus 6) und dem Greedy-Algorithmus zum Leeren der Nicht-Randfelder (Algorithmus 4) aus:

**Algorithmus 7: Gesamt-Algorithmus von Ansatz 1**

1. **while** keine Abbruchbedingung erfüllt
2. **solange, bis Algorithmus 6 keine Blasoperationen mehr findet** (dieser Fall tritt ein,   
    wenn sich auf keinem Randfeld mehr eine wesentliche Blattmenge befindet):
3. blasoperation = **führe Algorithmus 6 aus und speichere Rückgabe**
4. führe Blasoperation blasoperation aus
5. **falls** mind. eine Abbruchbedingung erfüllt:
6. **return**
7. **solange, bis Algorithmus 4 keine Blasoperationen mehr findet** (dieser Fall tritt ein,   
    wenn es keine Blasoperation mehr gibt, die die Laubmenge auf Feld Q erhöht):
8. blasoperation = **führe Algorithmus 4 aus und speichere Rückgabe**
9. führe Blasoperation blasoperation aus
10. **falls** mind. eine Abbruchbedingung erfüllt:
11. **return**

**Der Gesamt-Algorithmus nimmt folgende Parameter**:

* Q (Tupel): Ein Tupel, das den Index von Feld Q angibt
* hof\_size (Tupel): Ein Tupel, das die Größe des Hofs angibt
* startwert (ganze Zahl): Gibt die Anzahl an Blättern an, die sich zu Beginn auf den Feldern befindet
* use\_binomial (Wahrheitswert): Gibt an, ob bei der Simulation der Blasvorgänge am tatsächlichen Hof in Algorithmus 7 die Erwartungswerte verwendet werden sollen oder ob die tatsächlichen Wahrscheinlichkeiten zu simulieren sind
* satisfied\_constraint, max\_operations, weight\_avg, weight\_varianz: In den Vorkapiteln erläutert

### Laufzeit

Bei der Bestimmung der asymptotischen Laufzeit werden alle der oben genannten Parameter bis auf hof\_size als Konstanten behandelt. Besonders bei satisfied\_constraint und max\_operations ist dies sinnvoll, da es sich hierbei um obere Schranken handelt, die beliebig gesetzt werden können. Natürlich ist die tatsächliche Evaluierungszeit des Programms niedriger, wenn die Schranken niedrig gesetzt sind. Viel interessanter ist die asymptotische Laufzeit in Abhängigkeit zur Feldlänge bzw. -breite n.

Die Laufzeit in Abhängigkeit von n wird dabei eindeutig von Algorithmus 4 dominiert, der bei jeder Ausführung einmal über alle Felder iteriert und somit eine Laufzeit von O(n²). Mit Blick auf Algorithmus 7 lässt sich sagen, dass Algorithmus 4 im Worst-Case-Szenario max\_operations Male ausgeführt wird. Das Worst-Case-Szenario tritt dabei auf, wenn Algorithmus 7 ausschließlich Algorithmus 4 und nie Algorithmus 5 ausführt und Algorithmus 7 erst beim Erreichen von max\_operations durchgeführten Operationen abgebrochen wird (es darf keine andere Abbruchbedingung zuvor eintreten). Wenn max\_operations doch nicht als Konstante betrachtet wird, dann lässt sich also sagen, dass die Laufzeit des Gesamtverfahrens im Worst-Case-Szenario eine Laufzeit von O(n² \* max\_operations) hat. Diese Laufzeit ist leider nicht besonders gut, es handelt sich hierbei aber nicht um den einzigen Ansatz, den ich entwickelt habe (Ansatz 2 hat eine bessere Laufzeit).

### Kritik am Algorithmus

Der Vorteil des ersten Ansatzes besteht darin, dass über die Parameter satisfied\_constraint, max\_operations, weight\_avg und weight\_varianz flexibel gesteurt werden kann, wie viele Blätter auf Feld Q gebracht werden sollen, wie viele Operationen maximal ausgeführt werden, und wie schnell die Blätter auf Feld Q gelangen sollen.

Es ist jedoch fraglich, ob der Algorithmus den Anforderungen des Hausmeisters entspricht. In der Aufgabenstellung wünscht sich der Hausmeister *nicht, möglichst schnell Laub auf* Q zu bringen. Stattdessen wünscht er sich explizit, dass die Laubmenge auf Feld Q zu maximieren ist. Der Algorithmus des 1. Ansatz garantiert dabei nicht, dass die Gesamtlaubmenge auf allen Feldern außer von Q gegen Null konvergiert, wenn sehr viele Blasoperationen durchgeführt werden (und zur Modellierung der Blasoperationen durchweg die Erwartungswerte verwendet werden). Die Formulierung „Maximierung der Laubmenge auf Q“ legt aber nahe, dass ein solches Verfahren gesucht ist, das die Laubmenge auf Q maximiert und die kumulierte Laubmenge auf den anderen Feldern minimiert bzw. gegen Null konvergieren lässt. Im nächsten Kapitel werde ich meinen 2. Ansatz vorstellen, bei dem es sich um ein solches Verfahren handelt und der noch dazu eine deutlich bessere Laufzeit als mein 1. Ansatz hat.

Wegen der kaum systemathischen Herangehensweise gelingt es außerdem gerade bei kleinen Höfen nicht, große Mengen an Laub auf Feld Q zu versammeln.

## Besseres Vorgehen: Generalisierte Ablaufpläne (2. Ansatz)

### Ziel

In der Aufgabenstellung heißt es: „Das Ziel des Hausmeisters ist, das (…) Laub vom gesamten Schulhof auf ein einziges Planquadrat Q zu konzentrieren“. Hieraus lässt sich schlussfolgern, dass **das Versammeln von möglichst viel Laub auf Feld Q für den Hausmeister am wichtigsten ist** und alle anderen Aspekte wie z.B. Anzahl an durchzuführenden Blasoperationen (natürlich ebenfalls nicht unwichtig, aber) zweitrangig sind. Ich werde daher in den folgenden Kapiteln ein 2. Verfahren vorstellen, das sich in seiner Konzeption grundlegend vom 1. Ansatz unterscheidet.

**Konkret soll für das 2. Verfahren folgendes gelten (wenn zur Modellierung der Wahrscheinlichkeiten die Erwartungswerte verwendet werden):**

* In der Lösung sollen so viele Felder wie möglich (außer natürlich Q) vollständig geleert sein. Diese Aussage wird unter der Voraussetzung getroffen, dass zur Modellierung des Zufalls die Erwartungswerte verwendet werden. Das bedeutet: Es ist wirklich gemeint, dass so viele Felder wie möglich vollständig geleert werden. (Auf einem vollständig geleerten Feld ist die Laubmenge im Gegensatz zu einem *asymptotisch geleerten Feld* nicht nur asymptotisch Null bzw. geht gegen Null, sondern ist tatsächlich Null).
* Es können nicht alle Felder vollständig geleert werden (siehe hierzu nächstes Kapitel). Für die Felder, die nicht vollständig geleert werden können, wird ein Parameter tolerated\_amount eingeführt, der angibt, wie viele Blätter auf einem solchen Feld maximal „toleriert“ werden.
* Bei dem Algorithmus soll es sich folglich um einen Conquer-Algorithmus handeln, der (anders als das 1. Verfahren) nicht unorganisiert versucht, immer wieder auf Feld Q zu blasen, sondern stattdessen einen kontinuierlich größer werdenden Bereich des Hofs vollständig und endgültig leert. Es sind nämlich nur auf die Art die beiden zuvor genannten Ziele erreichbar.
* Auch wenn die Minimierung der Anzahl benötigter Blasoperationen kein primäres Ziel ist, soll das Verfahren den Hof trotzdem so leeren, dass nicht unnötig viel Aufwand für den Hausmeister entsteht. Unser Ziel ist es, den Hof zu leeren, ohne dass der Hausmeister den Eindruck hat, er müsse sich in einem Blasorchester wiederfinden!
* Anders als im 1. Ansatz wird in diesem Verfahren nicht mit einer volldynamischen Greedy-Heuristik gearbeitet. Die algorithmische Umsetzung des Verfahrens soll dennoch so erfolgen, dass – wie im 1. Ansatz – eine Funktion existiert, die entscheidet, welcher Blasvorgang als nächstes auszuführen ist (dies passt am besten zu der in der Aufgabe gegebenen Formulierung „Die Strategie entscheidet vor jedem Blasvorgang, auf welches Feld er sich stellen und in welche Richtung er von dort jeweils blasen soll“). Für das im Folgenden vorgestellte Verfahren benötigt man allerdings einen kognitiv deutlich fitteren Hausmeister, da er sich deutlich mehr Informationen „merken muss“.

### Auf welchen Höfen kann das Laub nicht auf Q geblasen werden?

Für Höfe mit n < 2 existiert offensichtlich keine Lösung, da solche Höfe nur aus Eckfeldern bestehen.

**In einem Hof der Maße (3,3) gibt es zwar ein mögliches Feld Q (das Feld in der Mitte bzw. mit den Koordinaten (1,1)). Es ist aber nicht möglich, Laub von einem Randfeld auf das Feld in der Mitte zu bringen**, da es keine auf diesen Hof anwendbare Regel gibt, mit der Laub von einem Randfeld auf ein Nicht-Randfeld gebracht werden kann. **Folglich ist es auch nicht möglich, Laub von einem Eckfeld auf das Feld in der Mitte zu bringen**, da Laub aus Ecken nur auf Randfelder gebracht werden kann (ein Eckfeld hat nur Randfelder als Nachbarn).  
In einem solchen Hof ist es daher am besten, gar keine Blasoperationen auszuführen, da der beste erreichbare Zustand bereits vorliegt.

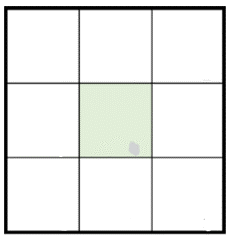


Abbildung 14: Hof mit den Maßen (3,3). Es gibt keine Möglichkeit, Laub auf das Feld #4 in der Mitte zu befördern - egal wie geblasen wird.

Auf allen anderen Höfen ist es möglich, die im Vorkapitel beschriebenen Ziele zu erreichen.

### Zugrundeliegende Strategie

Zunächst müssen wir uns von dem Gedanken lösen, dass es ein volldynamisches Verfahren braucht, um das Problem zu lösen. Der Umstand, dass die Blasoperation nichtdeterministisch sind, kann einen zu dieser Idee verleiten, dabei wird allerdings übersehen, dass nur auf manchen Feldern die Änderung der Blattmenge vom Zufall abhängt:

Ein Bild, das Text, Screenshot, Diagramm, Quadrat enthält.

Automatisch generierte Beschreibung

So ist die z.B. die Wahrscheinlichkeit dafür, dass ein Blatt, dass sich vor dem Blasvorgang auf Feld A befindet, auf Feld A verbleibt, 0% (siehe Abb. oben). **Feld A wird nach dem Blasvorgang also auf jeden Fall leer sein** (unter der Voraussetzung, dass Feld B im Hof existiert). Es ist außerdem zu 100% sicher, dass sich die Laubmenge auf den Feldern unter und über A nicht durch die Blasoperation verändern wird. Hieraus ergibt sich, dass **große Teile des Hofs vollständig (nicht nur asymptotisch) geleert werden können**, indem das Laub Reihe für Reihe bzw. Spalte für Spalte an eine Hofkante geblasen wird. Das folgende Beispiel zeigt einen Hof der Größe (5,5), der dieses Vorgehen veranschaulicht: Zunächst wird die 2. Spalte vollständig geleert, danach die 3. Spalte usw. bis sich nur noch in der 1. und der 5. Spalte Laub befindet.

Ein Bild, das Text, Reihe, Screenshot, Zahl enthält.

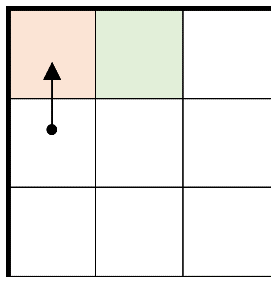
Automatisch generierte Beschreibung

Abbildung 15: Hof der Größe (5,5), auf dem das Verfahren zum vollständigen Leeren aller Felder (bis auf zwei Zeilen / Spalten mit Randfeldern) visualisiert ist. In jedem Hof sind die in der vorherigen Phase durchgeführten Blasprozesse eingezeichnet. Die grünen Felder sind nach den Blasprozessen vollständig geleert. Das von den grünen Feldern heruntergeblasene Laub befindet sich größtenteils auf den orangenen Feldern, teilweise auch auf den weißen Feldern. Die Blattanzahlen auf den gelb eingefärbten Feldern bleiben unverändert.

Dieses Verfahren leert die Felder, die sich zwischen der 1. und 5. Spalte befinden. Er ist ein wesentlicher Bestandteil des im 2. Ansatz verwendeten Conquer-Algorithmus. Die Leerung der grünen Felder erfolgt zu einer Wahrscheinlichkeit von 100%; es wäre daher unsinnig, hier ein dynamisches Verfahren zu verwenden.

Es gibt jedoch andere Fälle, in denen sich ein dynamisches Verfahren nicht vermeiden lässt.

**Beispiel hierfür: Leeren eines Eckfelds**

**

Ein Eckfeld lässt sich nur durch das kontinuierliche Blasen in die jeweilige Ecke leeren. Wenn dabei die tatsächlichen Zufälle simuliert werden, dann wandert beim Blasen jedes Blatt, dass sich auf dem Eckfeld befindet, zu einer bestimmten Wahrscheinlichkeit auf das Randfeld neben dem Eckfeld, auf dem nicht der Hausmeister steht. Die Anzahl an Blättern, die das Eckfeld verlassen, ist dabei vom Zufall abhängig, was ein dynamisches Verfahren erfordert.

### Definitionen: Generalisierte Ablaufpläne und Muster

Eine Sequenz an Blasoperationen, die den zuvor definierten Zielen gerecht wird, muss also auf jeden Fall aus nicht-dynamischen Phasen, während denen eine feste, „vorprogrammierte“ Abfolge an Blasoperationen durchgeführt wird, und aus dynamischen Phasen, während denen Blasoperationen aus einer begrenzten Operationsmenge basierend auf der tatsächliche Veränderung der Blattverteilung (die dann, wenn die tatsächlichen Zufälle über die Binomialverteilung simuliert werden, nicht eindeutig vorhersagbar ist) solange ausgeführt werden, bis ein bestimmtes Ziel erreicht ist.

Um diesem Dualismus gerecht zu werden, definiere ich im Folgenden zwei theoretische Konzepte, die in der Umsetzung beide implementiert werden:

|  |
| --- |
| **Definition „Muster“:** Als*Muster*wird im Folgenden eine Operation bezeichnet, das eine aus einer oder mehreren Blasoperationen bestehende Sequenz an Blasoperationen und anderen Mustern speichert. Ein Muster verfolgt das Ziel, Laub von einem oder mehreren *source*-Feldern auf ein *target-*Feld zu schaffen. „Das Muster wird ausgeführt“ bedeutet konkret, dass die vom Muster gespeicherten Blasoperationen ausgeführt werden.  Je öfter das Muster wiederholt wird, desto mehr Laub wird von den source-Feldern auf das target-Feld geschafft. Bei einer gegen unendlich gehenden Anzahl an Wiederholungen des Musters geht die Blattanzahl auf den *source-Feldern* daher gegen 0, die source-Felder werden also asymptotisch geleert.  Es wird ein Parameter *tolerated\_amount* ∈ ℚ > 0 eingeführt, der angibt, wie hoch die höchstens tolerierte Blattanzahl auf den zu leerenden source-Feldern ist ist. Bei Ausführung werden das Muster so lange wiederholt, bis die maximale Blattanzahl auf einem der source-Felder kleiner gleich *tolerated\_amount* ist. Ein Muster ist also dynamisch. Wird *tolerated\_amount* auf einen sehr niedrigen Wert gesetzt, dann wird muss das Muster folglich sehr oft wiederholt werden. Bei einem hohen Wert wird das Muster nicht so oft wiederholt, allerdings werden dann auch weniger Blätter von den *source-*Feldern entfernt.  Die Verwendung eines Musters ist immer dann sinnvoll, wenn ein Feld oder mehrere nebeneinanderliegende nicht alle gleichzeitig in einen vollständig geleerten Zustand gebracht werden können.  Ein Muster muss so definiert werden, dass beim Ausführen des Musters nur auf den source-Feldern und dem Feld *target* die Blattanzahlen verändert werden (alles andere würde dem Conquer-Prinzip widersprechen, das dem 2. Ansatz zugrundeliegt. |
| **Definition „Generalisierter Ablaufplan“:** Ein generalisierter Ablaufplan speichert eine Sequenz an Blasoperationen und Mustern. Die Verwendung eines generalisierten Ablaufplans ist dann sinnvoll, wenn zum Erreichen eines bestimmten Ziels sowohl Phasen, in denen der nächste Blasvorgang dynamisch auszuwählen ist, als auch nicht-dynamische Phasen, in denen die durchzuführenden Blasoperationen bereits vorher klar sind, durchgeführt werden müssen.  Die in nicht-dynamischen Phasen durchzuführenden Blasoperationen können direkt als solche im generalisierten Ablaufplan gespeichert werden. Um dynamische Phasen zu speichern, können Muster verwendet werden, die dynamisch arbeiten, wenn sie ausgeführt werden. |

Ein generalisierter Ablaufplan ist dabei etwas, was nicht während dem Blasprozess, sondern vor dem Durchführen der ersten Blasoperation erstellt wird. Während der eigentlichen Simulation des Blasprozesses werden die im generalisierten Ablaufplan enthaltenen Blasoperationen (nicht-dynamisch) und Muster (dynamisch) dann nur noch ausgeführt. Das im Folgenden vertiefte Lösungsverfahren funktioniert genau auf diese Art und Weise.

**Algorithmus 8: Grobe Skizzierung des Algorithmus von Ansatz 2**

1. Erstellen des generalisierten Programmablaufplans (der Hausmeister muss sich diesen gut einprägen)
2. Durchführen der Blasoperationen und Muster aus dem in Schritt 1 erstellten generalisierten Programmablaufplan

### Können die Rand- und Eckfelder vollständig geleert werden?

**Voraussetzung:** Zur Modellierung der Wahrscheinlichkeiten werden die Erwartungswerte verwendet (nur unter dieser Voraussetzung macht die Frage Sinn).

**Eckfelder können nicht vollständig geleert werden,** sondern nur asymptotisch geleert werden: Beim Blasen in eine Ecke verlässt immer nur ein bestimmter Anteil kleiner als 100% der sich auf dem Eckefeld befindenden Blätter das Eckfeld.

**Fast alle Randfelder können vollständig geleert werden,** es bleibt aber mind. 1 Randfeld übrig, das nicht vollständig geleert werden kann: Es ist möglich, das Laub (wie bei der Erläuterung des 1. Ansatz zu beschrieben) am Rand entlangzublasen. Hierbei werden alle Randfelder bis auf zwei vollständig geleert (Eckfelder nicht miteinbezogen): Das Randfeld, auf dem das Laub akkumuliert wird, und eines seiner Nachbarfelder können nur asymptotische geleert werden. (Wenn das Randfeld, auf dem das Laub akkumuliert wird, Nachbar eines Eckfelds ist, dann ist es auch möglich, alle Randfelder bis auf eines zu leeren (Eckfelder nicht miteinbezogen)).

### Muster zum Leeren von Rand- und Eckfeldern

Zum Leeren der Ecke links oben wird dieses Muster definiert, das Laub vom Source-Feld (0,0) auf das Target-Feld (1,0) schafft:

|  |
| --- |
| 1. blase(Feld (0,1), (1,0))   *In der Darstellung rechts ist das source-Feld rot markiert und das target-Feld grün markiert.* |

Das Muster lässt sich auf jede Ecke anwenden, wenn es entsprechend gespiegelt und gedreht wird. Hierfür wird ein Algorithmus entwickelt:

**Algorithmus 9: Algorithmus, der ein Muster zum Leeren einer Ecke ermittelt**

Parameter: source\_corner\_field = Der Index des Eckfelds, das geleert werden soll

Parameter: target\_field = Der Index des Randfelds, auf das das Laub geblasen werden soll

1. orthogonal\_direction = Ermitteln der Richtung orthogonal zum   
   Vektor (target\_field-source\_field) mithilfe von Algorithmus 1
2. feld0 = Finde das Feld, von dem aus in die Ecke geblasen werden muss, um Laub aus der Ecke zu befördern. Für feld0 kommen die Felder target\_field-orthogonal\_direction und target\_field+orthogonal\_direction infrage, die durchprobiert werden.
3. **return** ein Muster mit folgenden Eigenschaften:  
   - Das Source-Feld des Musters ist source\_corner\_field  
   - Das Target-Feld ist target\_field  
   - Das Muster speichert eine Blasoperation, die von Feld 0 aus in die zu leerende Ecke bläst

Zum Leeren des Randfelds wird dieses Muster definiert, das Laub von den Source-Feldern (1,0), (2,0) und (3,0) auf das Target-Feld (0,1) schafft:

|  |
| --- |
| 1. blase((2,1), (0,-1)) 2. blase((0,0),(1,0)) 3. blase((4,0),(-1,0)) |

Wenn das zu leerende Randfeld ein Nachbarfeld eines Eckfelds ist, muss das Muster leicht abgeändert. Es wird ein neues Muster definiert, das Laub von den Source-Feldern (0,0), (1,0) und (2,0) auf das Target-Feld (0,1) schafft:

|  |
| --- |
| 1. blase((1,1), (0,-1)) 2. blase((0,1),(0,-1)) 3. blase((3,0),(-1,0)) |

Auch dieses Muster lassen sich wieder auf jedes Randfeld anwenden, wenn es entsprechend gespiegelt und gedreht wird. Hierfür wird ein Algorithmus entwickelt, der aber nicht mehr in Pseudocode formalisiert wird. Das Prinzip, basierend auf dem Randfelder geleert werden, sollte inzwischen klar geworden sein.

### Bauen des generalisierten Ablaufplans

Dieses Kapitel beschäftigt sich mit den Algorithmen, die den generalisierten Ablaufplan zusammenbauen (Schritt 1 von Algorithmus 8). Grundsätzlich lässt sich sagen, dass der Ablaufplan immer aus diesen **4 Phasen** besteht:

1. Phase: Unterste Reihe des Hofs befreien
2. Phase: Blasen des gesamten Laubs in die oberste Reihe
3. Phase: Konzentrieren des gesamten Laubs der obersten Reihe auf dem Randfeld, das sich in der selben Spalte wie Q befindet.
4. Phase: Transferieren des auf Feld (Q[0],0) gesammelten Laubs auf Feld Q.

Für jede dieser Phasen wird ein Algorithmus entwickelt, der die während der Phase auszuführenden Blasoperationen und Muster erstellt und zum generalisierten Ablaufplan hinzufügt.

**Rotation des Hofs:** Die Algorithmen zur Erstellung des generalisierten Ablaufplans sind am effektivsten, wenn sich das Randfeld Feld, auf dem das Laub in der 3. Phase konzentriert wird, möglichst nahe am Feld Q befindet. Dies liegt daran, dass Phase 4 sehr „kostenintensiv“ (im Sinne von viele Blasoperationen erfordernd ist), wenn Feld Q weit von dem Feld entfernt ist, auf dem in der 3. Phase das Laub konzentriert wurde (mehr dazu später). Zur Vermeidung unnötiger Blasoperationen muss dies berücksichtigt werden.

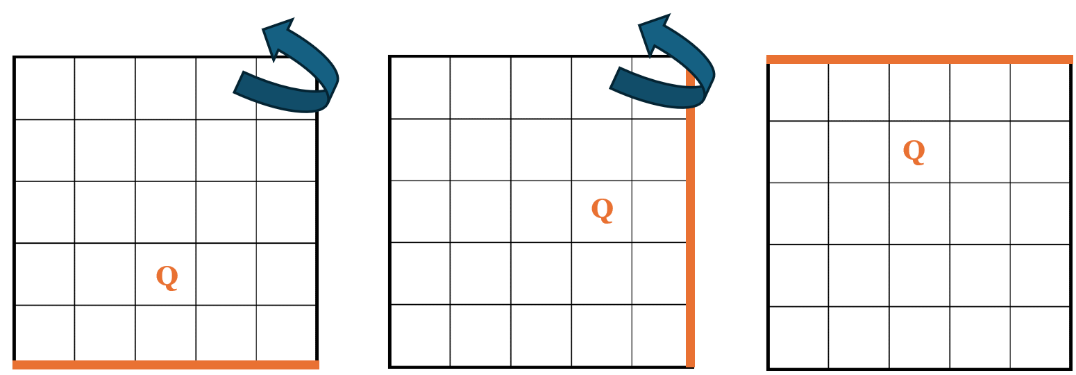
Um die Algorithmen zur Erstellung des Ablaufplans zu vereinfachen, wird vor ihrer Durchführung der Hof mitsamt Feld Q solange um 90° gegen den Uhrzeigersinn rotiert, bis die Kante, die am nächsten an Feld Q ist, die obere Kante des Hofs ist. Dies hat zur Folge, dass das Laub nun immer auf dem Feld (Q[0], 0) (das Feld am oberen Rand, das sich am nächsten an Q befindet) konzentriert werden kann und sich die Algorithmen nicht dynamisch an die Kante anpassen müssen, die am nächsten an Q ist.  
Diese Rotation wird im Folgenden als „Initialrotation“ bezeichnet, die Anzahl an durchgeführten 90°-Rotationen wird gespeichert.

Abbildung 16: Veranschaulichung der Initialrotation: Der Hof wird solange um 90° gegen den Uhrzeigersinn gedreht, bis die Kante, die Feld Q am nächsten ist (rot markiert), die obere Kante ist bzw. bis Feld Q in der oberen Hälfte des Kreises ist. Die Algorithmen zum Erstellen des Ablaufplans werden anschließend auf den rotierten Hof angewendet.

Wenn nun der Hof rotiert wurde und anschließend ein Muster oder eine Blasoperation zum generalisierten Ablaufplan hinzugefügt wird, dann muss immer zunächst die Initialrotation rückgängig gemacht werden, d.h. Feld 0 und Blasrichtung der jeweiligen Blasoperation müssen mit dem Uhrzeigersinn rotiert werden.

Die Rotation eines sich auf dem Feld befindenden Punkts um 90° gegen den Uhrzeigersinn kann mit folgender Formel berechnet werden, die von der alten Position des Punkts und der Seitenlänge n des Hofs abhängt:

Eine Rotation um 90° kann ausgedrückt werden, indem die Umkehrung der obigen Formel gebildet wird:

Im folgenden wird für jede Phase des Conquer-Verfahrens ein Algorithmus vorgestellt, der die in der Phase auszuführenden Blasoperationen und Muster ermittelt. Die Algorithmen arbeiten dabei unter der Voraussetzung, dass sich Feld Q in der oberen Hälfte des Hofs befindet. Diese Voraussetzung ist über die Initialrotation sichergestellt.

### Phase: Unterste Reihe des Hofs befreien

In der ersten Phase wird die unterste Reihe des Hofs so vollständig wie möglich geleert.  
Zunächst wird das Laub der Nicht-Eckfelder, die sich in der untersten Reihe befinden, mit einem nachhaltigen Verfahren auf das Eckfeld in der Ecke links unten geblasen, sodass die Nicht-Eckfelder in der untersten Reihe vollständig geleert sind.

Ein Bild, das Reihe, Quadrat, Rechteck, Diagramm enthält.

Automatisch generierte Beschreibung

Abbildung 17: Die Blasoperationen, die hierbei ausgeführt, am Beispiel eines Hofs der Seitenlänge 5. Die eingezeichneten Blasoperationen werden von rechts nach links ausgeführt. Dabei werden die grün eingefärbten Felder vollständig geleert.

Anschließend werden zwei Muster ausgeführt, die die beiden Eckfelder in der untersten Reihe asymptotisch leeren und das Laub der Eckfelder dabei auf die Randfelder in der zweit-untersten Reihe transferieren.

Ein Bild, das Text, Screenshot, Quadrat, Diagramm enthält.

Automatisch generierte Beschreibung

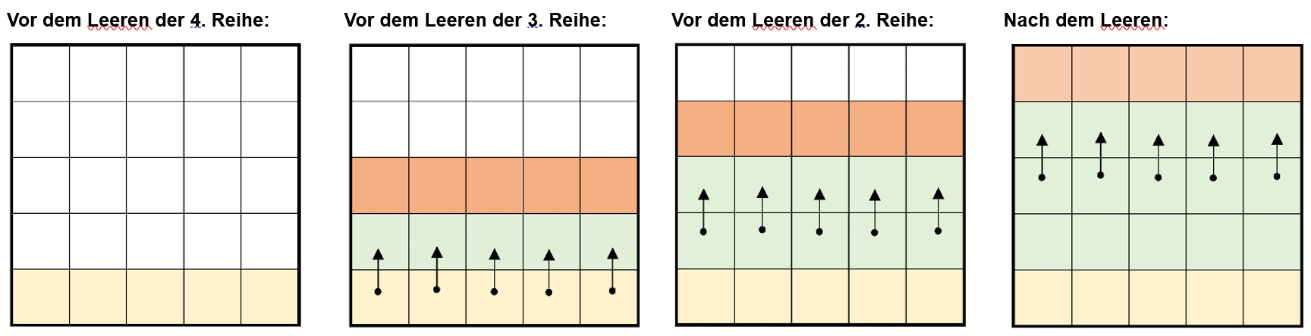
*Abbildung 18: Beispiel - ein Hof der Seitenlänge 6 nach Durchführen der 1. Phase. Die unterste Reihe ist bis auf die Eckfelder vollständig geleert.*  
  
**Algorithmus 10: clear\_bottom\_line bzw. Algorithmus zum Hinzufügen der Blasoperationen und Muster von Phase 1 zum generalisierten Ablaufplan**

Globale Variable: strategy = Liste, die den generalisierten Ablaufplan speichert  
Parameter: n = Seitenlänge des Hofs

1. **für jedes** start\_x **im Bereich** [n-1,0] (Änderung von start\_x pro Iteraton: -1):
2. **füge** Blasoperation, die vom Feld (start\_x, n-1) in Richtung (-1,0) bläst **zu** strategy **hinzu**
3. **füge** Muster, das das Laub vom Eckfeld (0, n-1) auf das Randfeld (0, n-2) transferiert **zu** strategy **hinzu**
4. **füge** Muster, das das Laub vom Eckfeld (n-1,n-1) auf das Randfeld (n-1, n-2) transferiert **zu** strategy **hinzu**

### Phase: Laub auf oberste Reihe bringen

In der zweiten Phase wird das gesamte Laub auf die oberste Reihe gebracht. Hierfür wird von unten nach oben Reihe für Reihe vollständig geleert. Das Vorgehen, das bereits in einem Vorkapitel erläutert wurde (auf eine Formalisierung als Pseudocode wird daher verzichtet), wird in der folgenden Grafik an einem Hof der Dimensionen (5,5) veranschaulicht:



### Phase: Laub auf der obersten Reihe konzentrieren

In der 3. Phase wird das sich auf der oberen Reihe befindende Laub auf den Feldern (Q[0],0), (Q[0]-1,0) und (Q[0]+1,0) versammelt.

Zunächst werden hierfür die Eckfelder (0,0) und (n-1,0) über Muster geleert, das Laub wird dabei auf die Nachbarfelder der Eckfelder, die sich auf der obersten Reihe befinden, geblasen. Anschließend wird das Laub, das sich auf der obersten Reihe befindet, auf Feld (Q[0],0) geblasen.  
Durch die Anwendung von Mustern wird versucht, die entstehenden dabei Seitenabtriebe (die auf Felder in der Reihe unter der obersten Reihe fliegen) zu eliminieren bzw. die von den Seitenabtrieben betroffenen Felder asymptotisch zu leeren.  
Folgende Grafik veranschaulicht ein Beispiel, in dem ein Muster verwendet wird, das das Feld (1,0) asymptotisch leert, gleichzeitig das Feld (2,1), auf das bei der Blasoperation blase((1,0),(1,0)) Seitenabtriebe gelangen, ebenfalls asymptotisch leert und die Laubmenge auf Feld (2,0) kontinuierlich erhöht:

Ein Bild, das Text, Reihe, Diagramm, Screenshot enthält.

Automatisch generierte Beschreibung

Wenn die schwarz eingezeichneten Blasoperationen sehr oft durchgeführt werden, dann werden die Felder (1,0) und (2,1) asymptotisch gelehrt. Nach Beendigung des Musters kann anschließend das Feld (2,0) mit einem neuen Muster, das auf dem gleichen Prinzip basiert, geleert werden und dabi das Laub auf das Feld (3,0) geschafft werden usw., bis das gesamte Laub der oberen Reihe auf den Feldern (Q[0],0), (Q[0]-1,0) und (Q[0]+1,0) versammelt ist.

**Algorithmus 11: concentrate\_top\_line bzw. Algorithmus zum Hinzufügen der Blasoperationen und Muster von Phase 3 zum generalisierten Ablaufplan**

Globale Variable: strategy = Liste, die den generalisierten Ablaufplan speichert  
Parameter: n = Seitenlänge des Hofs

1. source\_fields\_for\_edgeclear = [(Q[0]-1,0), (Q[0],0), (Q[0]+1,0)] (es handelt sich hierbei um die Randfelder, auf denen das Laub versammelt werden soll)
2. **falls** (0,0) nicht in der Liste source\_fields\_for\_edgeclear: (wenn das Laub auf einem dieser Randfelder versammelt werden soll, dann macht es keinen Sinn, es von ebendiesem Randfeld herunterzublasen)
3. **füge** Muster, das das Laub vom Eckfeld (0,0) auf das Randfeld (1,0) transferiert **zu** strategy   
    **hinzu**
4. **falls** (n-1,0) nicht in der Liste source\_fields\_for\_edgeclear:
5. **füge** Muster, das das Laub vom Eckfeld (n-1,0) auf das Randfeld (n-2,0) transferiert **zu** strategy **hinzu**
6. **für jedes** start\_x **im Bereich** [0;n-1] (Änderung von start\_x pro Iteraton: 1):
7. **falls** (start\_x+1, 0) in der Liste source\_fields\_for\_edgeclear:
8. **Schleife abbrechen**
9. **andernfalls:**
10. m = Muster, das nach dem oben erläuterten Prinzip das Laub vom Randfeld (start\_x+1,0)   
     auf das Randfeld (start\_x+2,0) schafft und dabei die Seitenabtriebe eliminiert
11. **füge** m **zu** strategy **hinzu**
12. **für jedes** start\_x **im Bereich** [n-1;0] (Änderung von start\_x pro Iteraton: -1):
13. **falls** (start\_x-1, 0) in der Liste source\_fields\_for\_edgeclear:
14. **Schleife abbrechen**
15. **andernfalls:**
16. m = Muster, das nach dem oben erläuterten Prinzip das Laub vom Randfeld (start\_x-1,0)   
     auf das Randfeld (start\_x-2,0) schafft und dabei die Seitenabtriebe eliminiert
17. **füge** m **zu** strategy **hinzu**

Ein Bild, das Screenshot, Text, Rechteck, Diagramm enthält.

Automatisch generierte Beschreibung

Abbildung 19: Ein Beispielhof nach Vollendung von Phase 3

### Phase: Gesamtes Laub auf Feld Q verschieben

Wenn Q ein Nachbarfeld eines Randfelds ist, dann kann über das Muster zum Transferieren des Laubs eines Randfelds auf ein Nicht-Randfeld die Laubmenge auf Q asymptotisch maximiert bzw. die Laubmenge des benachbarten Randfelds asymptotisch minimiert werden.

Dies geht aber nur, wenn Q Nachbar eines Randfelds ist – für andere Fälle muss ein anderes Verfahren entwickelt werden.

**Warum ist es nicht sinnvoll, direkt auf Q zu blasen?**

Beim direkten Blasen auf Q (sodass Feld Q Feld B der Blasoperation ist und das Nachbarfeld von Feld Q, dessen Laub auf Q geblasen werden soll, Feld A der Blasoperation ist) gelangt zwar Laub auf Q, allerdings …

* gelangt nicht das gesamte Laub von Feld A auf Q, da ca. 20% des Laubs von Feld A seitlich abgetrieben wird
* werden außerdem 10% des Laubs von Feld Q nach vorne abgetrieben. Wenn sich bereits viel Laub auf Q befindet, kann es sein, dass der Zusammenhang  
  erfüllt ist. In diesem Fall hätte ein direktes Blasen auf Feld Q also sogar zur Folge, dass sich die Blattanzahl auf Feld Q reduzieren würde. Wenn man nur durch direktes Blasen auf Feld Q Laub auf Q befördert, dann kann man folglich die Laubmenge auf Q nicht asymptotisch maximieren bzw. die Laubmengen auf den anderen Feldern nicht asymptotisch minimieren, da sich diese Laubmengen durch die zuvor erläuterten Abtriebe nicht Null annähern werden.

Sinnvoller ist es, das Laub so auf Feld Q zu blasen, dass das zu leerende Feld mit Feld B der Blasoperation korrespondiert (im Folgenden als „Verfahren zum abtriebsfreien Transportieren“ bezeichnet):

Ein Bild, das Text, Screenshot, Diagramm, Schrift enthält.

Automatisch generierte Beschreibung

Hierbei hat man keine Seitenabtriebe, und es gelangen nur die 10% des Laubs, die auf Feld B nach vorne abgetrieben werden, auf Feld Q. Die Blasoperation muss dabei sooft durchgeführt werden, bis die Laubmenge auf B kleiner als *tolerated\_amount* ist – dies kann über ein Muster umgesetzt werden.

**Problem:** Dies ist nur möglich, wenn sich unter Feld Q drei weitere Felder befinden, da nur dann ein Feld 0 existiert, von dem aus so wie in der Grafik dargestellt geblasen werden kann. Auf einem Hof der Maße (5,5) oder (6,6) und Q=(2,2) ist dem nicht der Fall – daher handelt es sich bei diesem Hof um einen Sonderfall, der gesondert zu betrachten ist.

**Bauen des generalisierten Ablaufplans der 4. Phase**

Zunächst wird mit einem Muster das Laub vom Randfeld (Q[0],0) auf das Nicht-Randfeld (Q[0],1) gebracht.

Das Laub ist nach diesem Schritt auf dem Feld (Q[0],1) gesammelt - wenn Q[1] == 1, dann hat man also sein Ziel erreicht und der Algorithmus kann beendet werden.

**Transferieren des Laubs vom Feld (Q[0],1) auf das Feld (Q[0],2):**

Wenn Q[1] != 1, dann ist man aber leider noch nicht fertig. Da Q[1] != 1 gilt nun zwangsläufig Q[1] > 1, d.h. das Laub muss in eine tiefere Zeile gebracht werden. Um das Laub vom Feld (Q[0],1) auf (Q[0],2) zu bringen und dabei (Q[0],1) vollständig und alle anderen beteiligten Felder (außer Q[0],2) asymptotisch zu leeren, kann wie folgt vorgegangen werden:

1. Zuerst stellt sich der Hausmeister auf Feld (Q[0],0) und bläst in die Richtung (0,1) bzw. nach unten. Hierdurch wird das Feld (Q[0],1) vollständig geleert und ein Großteil des Laubs gelangt auf (Q[0],2), es gelangen aber auch Seitenabtriebe auf die Felder (Q[0],3), (Q[0]+1,2) und (Q[0]-1,2).

Schritt 2 und Schritt 3 werden nur dann ausgeführt, wenn n != 5 ist (im Sonderfall n = 5 wird anders vorgegangen):

1. Das Laub vom Feld (Q[0]+1,2) muss nun seitenabtriebsfrei auf (Q[0]+1,3) geblasen werden und (Q[0]+1,2) muss dabei asymptotisch geleert werden. Hierfür wird ein Muster verwendet, das die Blasoperation blase((Q[0]+1,0), (0,1)) solange ausführt, bis die Laubmenge auf Feld (Q[0]+1,2) kleiner als tolerated\_amount ist.
2. Das Laub vom Feld (Q[0]-1,2) muss nun seitenabtriebsfrei auf (Q[0]-1,3) geblasen werden und (Q[0]-1,2) muss dabei asymptotisch geleert werden. Hierfür wird ein Muster verwendet, das die Blasoperation blase((Q[0]-1,0), (0,1)) solange ausführt, bis die Laubmenge auf Feld (Q[0]-1,2) kleiner als tolerated\_amount ist.

Nach Schritt 3 sieht die Blattverteilung auf dem Hof wie folgt aus:

Ein Bild, das Text, Screenshot, Reihe, Zahl enthält.

Automatisch generierte Beschreibung

Abbildung 20: Die hellrot markierten Felder sind asymptotisch geleert worden, auf den braunen Feldern befindet sich viel Laub

Für die nächsten Schritte muss eine Fallunterscheidung vorgenommen werden.

**Fall n >= 7:**

In diesem Fall sind unter (Q[0],2), noch mind. vier weitere Felder. Daher kann folgendes Vorgehen verwendet werden, um das gesamte Laub auf Feld (Q[0],2) zu bringen:

1. Die Blasoperationen blase((Q[0]-2,3), (1,0)) und blase((Q[0]-2,3), (1,0)) eingezeichnet. Nach den beiden Blasoperationen sieht die Blattverteilung folgendermaßen aus (die beiden Blasoperationen sind ebenfalls in dem Hofausschnitt eingezeichnet):

Ein Bild, das Text, Screenshot, Reihe, Diagramm enthält.

Automatisch generierte Beschreibung

1. Mit dem zuvor erläuterten Verfahren zum abtriesfreien Transportieren wird zunächst das Laub von Feld (Q[0],4) auf (Q[0],3) befördert, indem die Blasoperation blase((Q[0], 6), (0,-1)) sooft wiederholt wird, bis die Blattmenge auf (Q[0],4) kleiner als tolerated\_amount ist.
2. Anschließend wird das Laub von Feld (Q[0],3) auf (Q[0],2) befördert, indem die Blasoperation blase((Q[0], 5), (0,-1)) sooft wiederholt wird, bis die Blattmenge auf (Q[0],3) kleiner als tolerated\_amount ist.

Nach diesen Schritten befindet sich das Laub auf (Q[0],2) - wenn Q[1] == 2, dann hat man also sein Ziel erreicht und der Algorithmus kann beendet werden. Wenn nicht, dann muss das Laub noch von (Q[0],2) auf Feld Q transportiert werden – dies kann mit dem Verfahren zum abtriebsfreien Transportieren seitenverlustfrei durchgeführt werden.

**Fall n = 6:**

In diesem Fall kann die Blasoperation blase((Q[0], 6), (0,-1)) nicht durchgeführt werden, da die das Feld (Q[0],6) nicht im Hof existiert. Stattdessen muss ein anderes Verfahren verwendet werden, dass leider mehr Blasoperationen erfordert als das im Fall n = 7 verwendete Verfahren.

Es wird folgendes Muster ausgeführt, um das gesamte Laub auf Q zu bringen - die Source-Felder des Musters sind dabei (Q[0], 3), (Q[0], 4), (Q[0]-1, 3) und (Q[0]+1, 3):

|  |
| --- |
| 1. blase((Q[0]-2,3),(1,0)) 2. blase((Q[0]+2,3),(-1,0)) 3. blase((Q[0],5),(0,-1)) |

Nach dem Durchführen dieses Musters befindet sich das Laub auf (Q[0],2) - wenn Q[1] == 2, dann hat man also sein Ziel erreicht und der Algorithmus kann beendet werden. Wenn nicht, dann muss das Laub noch von (Q[0],2) auf Feld Q transportiert werden – dies kann mit dem Verfahren zum abtriebsfreien Transportieren seitenverlustfrei durchgeführt werden.

**Fall n = 5:**

Der Sonderfall n = 5 ist am schwierigsten zu lösen bzw. man benötigt hier am meisten Blasoperationen, um die Laubmenge auf Q asymptotisch zu maximieren und die anderen Felder asymptotisch zu leeren. Im Fall n = 5 ist Q auf jeden Fall (2,2), da alle anderen möglichen Positionen von Q Nachbarn von Randfeldern sind - und in diesem Fall wäre das Laub schon längst auf Q gebracht worden (zur Erinnerung siehe Seite 37: „Das Laub ist nach diesem Schritt auf dem Feld (Q[0],1) gesammelt - wenn Q[1] == 1, dann hat man also sein Ziel erreicht und der Algorithmus kann beendet werden.“)

Es wird zunächst die Blasoperation blase((2,4),(0,-1) ausgeführt. Diese Blasoperation bläst Laub direkt auf Feld Q. Dies soll ja eigentlich vermieden werden – für einen Hof mit den Seitenlängen (5,5) und Q=(2,2) wurde durch Testen bzw. Ausprobieren herausgefunden, dass diese Blasoperation die Laubmenge auf Q erhöht.

Anschließend wird mit dem Verfahren zum abtriebsfreien Transportieren das Laub von den Feldern (Q[0]-1,2) und (Q[0]+1,2) auf die Felder (Q[0]-1,1) und (Q[0]+1,1). Der Grund dafür, dass im Sonderfall n = 5 die dritte, sondern die erste Reihe zum Sammeln des Laubs verwendet wird: Das im nächsten Schritt definierte Muster, das die Laubmenge auf Q asymptotisch maximiert, geht mit sehr vielen Feldern einher, die nur asymptotisch geleert werden können. Ziel des 2. Ansatzes ist es, die Anzahl an Feldern, die nur asymptotisch und nicht vollständig geleert werden können, möglichst gering zu halten. Daher ist es sinnvoll, für dieses Muster die Felder (Q[0]-1, 0), (Q[0],0) und (Q[0]+1,0) zu verwenden, die ohnehin schon zu Beginn der 4. Phase nur asymptotisch geleert werden konnten und daher quasi „vorbelastet“ sind, anstatt die Felder in der unterstehen Reihe zu verwenden, die alle vollständig geleert sind und „neu belastet würden“, würde das Muster auf diesen Feldern ausgeführt werden.

Dieses Muster, das als nächstes auszuführen ist, hat die Source-Felder (1,1), (2,1), (3,1), (2,0) und beinhaltet folgende Blasoperationen:

|  |  |
| --- | --- |
| 1. Muster, das die Source-Felder (1,1),(2,1),(3,1) hat und folgende Blasoperationen beinhaltet:  |  | | --- | | 1. blase((0,1),(1,0)) 2. blase((4,1),(-1,0)) |  1. Muster, das das Laub vom Randfeld (2,0) auf das Nicht-Randfeld (2,1) transferiert |

Das im Sonderfall n = 5 durchgeführte Muster ähnelt konzeptionell dem im Sonderfall n = 6 durchgeführten Muster, es ist aber an den kleineren Hof angepasst.

**Anzahl durchzuführender Blasoperationen – Korrelation zu Hofgröße:**

Es fällt auf: Je kleiner der Hof, desto komplizierter ist die Durchführung von Phase 4 bzw. desto mehr Blasoperationen müssen durchgeführt werden. Dies mag einem antiintuitiv vorkommen. Ursächlich für den Umstand ist, dass auf einem großen Hof das Verfahren zum abtriebsfreien Transportieren von Laub auf Q uneingeschränkt funktioniert, was auf einem kleinen Hof nicht mehr der Fall ist, da dort die entsprechenden Felder außerhalb des Hofs liegen,

### Optimierungen

Im den vorherigen Kapiteln zum 2. Ansatz wurden einige umgesetzte Optimierungen noch nicht angesprochen. Diese werden im Folgenden aufgelistet:

* *Muster*-Konzept: Wenn während der Ausführung eines Musters festgestellt wird, das sich die Gesamtlaubmenge auf den Source-Feldern nicht mehr ändert und die Erwartungswerte zur Wahrscheinlichkeitsmodellierung angewendet werden, dann wird das Muster frühzeitig abgebrochen – auf die Art werden unnötige Blasoperationen, die eh nichts mehr bringen, vermieden. Dieser Fall kann aber nur auftreten, wenn irgendwo anders in der Implementierung ein Bug ist.
* *Phase 4 mit Sonderfall n = 5:* Die Durchführung von Phase 4 geht schneller, wenn das Muster zum Maximieren der Laubmenge auf Q von einer anderen Kante aus durchgeführt wird – dies hat aber eine Verringerung der Anzahl an vollständig geleerten Feldern zur Folge. Über den Parameter choose\_faster\_path kann eingestellt werden, wie in dem Fall verfahren wird – ist choose\_faster\_path wahr, dann wird die Geschwindikeit priorisiert, ansonsten wird die Maximierung der Anzahl vollständig geleerter Felder priorisiert.

### Gesamt-Algorithmus und Kritik

**Der Gesamt-Algorithmus nimmt folgende Parameter**:

* Q (Tupel): Ein Tupel, das den Index von Feld Q angibt
* hof\_size (Tupel): Ein Tupel, das die Größe des Hofs angibt
* startwert (ganze Zahl): Gibt die Anzahl an Blättern an, die sich zu Beginn auf den Feldern befindet
* use\_binomial (Wahrheitswert): Gibt an, ob bei der Simulation der Blasvorgänge am tatsächlichen Hof in Algorithmus 7 die Erwartungswerte verwendet werden sollen oder ob die tatsächlichen Wahrscheinlichkeiten zu simulieren sind
* tolerated\_amount (ganze oder rationale Zahl): in den Vorkapiteln erläuert
* max\_muster\_operations (ganze Zahl): Anzahl an Operationen, die pro Muster maximal durchgeführt werden (ein Muster wird auf jeden Fall abgebrochen, wenn es max\_muster\_operations Blasoperationen durchgeführt hat
* choose\_faster\_path (Wahrheitswert): Im Vorkapitel erläutert

**Der zweite Ansatz ist aus mehreren Gründen deutlich positiver zu bewerten als der erste Ansatz:**

* Deutlich bessere Ergebnisse: Das Ziel des Hausmeisters, so viel Laub wie möglich auf Q zu versammeln lässt sich mit dem 2. Ansatz optimal erreichen, da er dafür sorgt, dass auf allen Feldern außer Q die Laubmenge gegen Null konvergiert (Voraussetzung: es werden die deterministischen Erwartungswerte zur Wahrscheinlichkeitsmodellierung verwendet[[5]](#footnote-6)). Es lässt sich also sagen, dass der zweite Ansatz die Aufgabenstellung optimal löst – dies ist beim 1. Ansatz nicht der Fall.
* Der Hausmeister kann über den Parameter tolerated\_amount steuern, wie viele Blätter er auf einem nur asymptotisch leerbaren Feld toleriert.
* Deutlich systemtatischeres Vorgehen als beim ersten Ansatz.
* Die Laufzeit ist auch deutlich besser (siehe nächstes Kapitel)

Einzig und allein der Umstand, dass der generalisierte Programmablaufplan zu Teilen nicht dynamisch ist und diese nicht-dynamischen Teile bereits vor Beginn des Blasprozesses endgültig festgelegt werden, ist möglicherweise etwas kontrovers zu betrachten, denn in der Aufgabe heißt es explizit: „Der Hausmeister soll sich vor jedem Blasvorgang entscheiden …“, dies legt ein dynamisches Verfahren nahe.

Die Aufgabenstellung verbietet allerdings nicht, dass der Hausmeister ein Gedächtnis hat, und wie zuvor gezeigt handelt es sich bei der in Ansatz 2 verwendeten Vorgehensweise um eine Vorgehensweise, die das Problem optimal löst. Außerdem kommen in Ansatz 2 ja durchaus dynamische Teile vor (Muster). Von daher halte ich den zweiten Ansatz für den besten Ansatz bzw. **der zweite Ansatz, der zuvor erläutert wurde und in der Datei aufgabe1\_solver2.py implementiert ist, soll das zu bewertende Verfahren sein.** Die anderen Verfahren dienen als Vergleichsgegenstände bzw. als andere mögliche Verfahren, die sich aber als schlechter erwiesen haben.

### Laufzeit

Die Laufzeiten der einzelnen Bestandteile des 2. Lösungsverfahren:

1. Laufzeit des Verfahrens, das die in Phase 1 auszuführenden Operationen ermittelt: O(n), da einmal über alle Felder der untersten Reihe iteriert wird
2. Laufzeit des Verfahrens, das die in Phase 2 auszuführenden Operationen ermittelt: O(n² - n) = O(n²), da einmal über alle Felder bis auf die Felder der obersten Reihe iteriert wird
3. Laufzeit des Verfahrens, das die in Phase 3 auszuführenden Operationen ermittelt: O(n)
4. Laufzeit des Verfahrens, das die in Phase 4 auszuführenden Operationen ermittelt: Schwer zu quantifizieren. Wenn alle Parameter außer n als konstant angenommen werden, dann ist die Laufzeit O(1). Wenn alle Parameter außer n und Q als konstant angenommen werden, dann ist die Laufzeit O(max(0,Q[1]-2)) (wegen dem Programmteil, der das Laub vom Feld (Q[0],2) auf das Feld (Q[0],3) transferiert.
5. Laufzeit des Verfahrens, was die Operationen des generalisierten Ablaufplans ausführt: O(m) mit m := Anzahl an Operationen, die durchgeführt werden müssen, um den generalisierten Ablaufplan einmal zu durchlaufen. m ist jedoch schwer zu generalisieren und außerdem vom Zufall abhängig, wenn der tatsächliche Zufall simuliert wird.

Die Gesamtlaufzeit wird dominiert durch O(n²). Sie ist deutlich besser als die Gesamtlaufzeit des 1. Verfahrens, die O(n² \* max\_operation) beträgt. Anders ausgedrückt: In derselben (theoretischen) Zeit, in der der 1. Ansatz eine einzige Blasoperation simuliert, kann der 2. Ansatz einmal ganz durchlaufen.

## Kombination der Verfahren (3. Ansatz)

Es stellt sich die Frage, ob der 1. Ansatz verbessert werden kann, indem Teile des 2. Ansatz in ihn eingebunden werden. Es wird also ein 3. Ansatz entwickelt, der auf dem 1. Ansatz aufbaut und konkret folgende Bestandteile des 2. Ansatz in den 1. Ansatz einbindet:

* Die Muster zum Leeren von Eck- und Randfeldern
* Das Muster aus Phase 4 des 2. Ansatz (Transferieren des Laubs auf Q).

Als der dritte Ansatz entwickelt wurde, bestand die Hoffnung, durch die Kombination ein Verfahren schaffen zu können, das sowohl über die Steuerbarkeit des 1. Ansatzes als auch über die systematische Herangehensweise des 2. Ansatzes verfügt. Die Ergebnisse zeigen jedoch auf, dass dies nicht wirklich gelungen ist.

Der 3. Ansatz ist kaum optimiert und auf keinen Fall das Verfahren, das bewertet werden soll. Es dient vielmehr als Vergleichsgegenstand zum 2. Ansatz (dem besten Ansatz).

### Parametrisierung

Durch die Kombination des 1. und 2. Ansatzes verfügt der 3. Ansatz über alle Parameter, über die der 1. Ansatz und der 2. Ansatz verfügen. Parameter sind grundsätzlich gut, da sie dem Nutzer Anpassungsmöglichkeiten geben. Bei einer zu großen Anzahl an Parametern ist allerdings fraglich, ob es wirklich sinnvoll, so viele Parameter zu verwenden – insbesondere dann, wenn eine Veränderung verschiedener Parameter ähnliche Auswirkungen auf das Resultat hat.

## Zusammenfassung

Es wurden drei Ansätze entwickelt. Ansatz 1 verwendet dabei eine Greedy-Heuristik, deren Heuristik eine Kombination zweier Heuristik-Kenngrößen ist, die über Parameter gewichtet werden können.

Ansatz 2 verwendet einen systematischen Conquer-Algorithmus, um möglichst viele Felder vollständig zu leeren und alle anderen Felder (außer Q) asymptotisch zu leeren. Dieser Ansatz liefert die besten Ergebnisse und ist der zu bewertende Ansatze.

Ansatz 3 versucht, Ansatz 1 und 2 zu kombinieren, um die Vorteile beider Ansätze auszunutzen, was aber nur bedingt gelingt und zu einem Parameterchaos führt.

## Bereits in Solver 1 bis 3 enthaltene Erweiterungen

* Ansatz 1 und 3 lassen sich problemlos auch auf Höfe anwenden, die nicht quadratisch, sondern rechteckig sind. Diese Erweiterung wird direkt in dem Programm, das die Grundaufgabe löst, implementiert, da mit ihr keine Komplexitätszunahme einhergeht.
* An Ansatz 2 müssen einige kleine Änderungen vorgenommen werden, damit er anwendbar auf rechteckige Höfe wird: Die Initialrotation muss bei einem Hof mit den Maßen (3,x) mit x > 3 so lange durchgeführt werden, bis die obere Kante die Länge x hat. Die obere Kante darf nämlich nicht die Länge 3 haben, da am oberen Rand das Verfahren zum Transferieren des Randlaubs auf ein Nicht-Randlaub durchgeführt wird, und dieses Verfahren funktioniert nur an Randsegmenten, die länger als 4 Felder sind.
* Aus der Aufgabe geht nicht klar hervor, ob erwartet wird, dass ein Feld Q als Parameter gegeben wird, oder ob das Programm automatisch das am besten geeignete Feld Q auswählen soll. Die Programme sind auf jeden Fall so konzipiert, dass Q als Parameter mitgegeben wird, und können mit allen möglichen Q umgehen.  
  Wenn zum Durchführen des Laubblasprozesses Ansatz 2 verwendet wird, dann sind für Q Felder geeignet, die Nachbarn von Randfeldern sind und die zum nächsten Eckfeld eine Manhattan-Distanz größer als 2 haben. Grund hierfür: Der Prozess, mit dem das Laub in Phase 4 von (Q[0],1) auf (Q[0],2) befördert wird, ist in Hinblick auf die benötigten Blasoperationen gerade bei Höfen mit n = 5 oder n = 6, aber auch bei anderen Hofgrößen aufwändig. Bei Feldern, die Nachbarn von Randfeldern sind, muss dieser Prozess nicht durchgeführt werden.

## Erweiterung 1: Keine Mauer-Umrandung des Hofs

Diese Erweiterung basiert auf dem 2. Ansatz.

Zunächst müssen die Blasregeln an die neuen Gegebenheiten angepasst werden: Wenn der Hof kein Mauer mehr hat, dann ist es möglich, dass Laub den Hof verlässt.

Die zu Beginn des Dokuments beschriebenen Regeln legen in Sonderfällen normalerweise fest, dass das Laub, das überlicherweise auf ein nicht existierendes Feld geflogen wäre, nun stattdessen vollständig auf dem Feld neben der Mauer verbleibt. Diese Festlegung ergibt keinen Sinn mehr, wenn der Hof nicht mehr von einer Mauer umgeben ist. „Kein Mauer“ bedeutet aber auch nicht zwangsläufig, dass die Umrandung des Hofs komplett offen ist: Was, wenn der Hof von einem Maschendrahtzaun umgeben ist, durch den zwar manche Blätter durchfliegen können, der aber immer noch einen gewissen Widerstand auf die seitlich abtreibenden Blätter ausübt?

Wenn Laub gegen die Umrandung geblasen oder an der Umrandung entlanggeblasen wird, dann wird zumindest der dabei entstehende Seitenabtrieb auf jeden Fall mit der Umrandung in Berührung kommen. Es wird der Parameter *wall\_resistance* eingeführt, der angibt, wie groß der Anteil der Blätter, die den Hof nicht verlassen, an den Blättern ist, die insgesamt mit der Umrandung in Berührung kommen. Der Parameter kann durch Kalibrieren an die entsprechenden örtlichen Gegebenheiten angepasst werden. Im Folgenden wird mit wall\_resistance = 0,5 gearbeitet (das heißt, 50% der Blätter, die gegen die Umrandung fliegen, verlassen den Hof – die anderen 50% verbleiben im Hof und fliegen auf das Feld, auf das sie den regulären Blasregeln nach fliegen würden.

Nach Anpassung der Blasregeln gelangt man schnell zu der Feststellung, dass auch die Algorithmen, die die in den einzelnen Phasen durchzuführenden Operationen zum generalisierten Ablaufplan hinzufügen, angepasst werden müssen. Es ist jetzt, wo der auf die Umrandung treffende Seitenabtrieb teilweise verloren geht, definitiv keine gute Idee mehr, Laub unnötigerweise am Rand entlangzublasen. Das Ziel des Hausmeisters ist schließlich klar definiert als „möglichst viel Laub auf Q bringen. Laub, das einmal den Hof verlassen hat, kann nie wieder auf ihn zurückgebracht werden und wirkt sich daher negativ auf das Erreichen des Ziel des Hausmeisters aus.

Es müssen also alle Phasen so abgeändert werden, dass möglichst wenig Operationen durchgeführt werden, die Laub am Rand entlangblasen – hierbei handelt es sich nun um das primäre Ziel. Das erste Ziel des 2. Ansatzes (möglichst viele Felder vollständig leeren) ist jetzt erst mal sekundär, denn noch nicht vollständig geleerte Felder können später immer noch geleert werden, aber Laub, das den Hof einmal verlassen hat, kann nie wieder zurückgewonnen werdens. Das tertiäre Ziel, die Anzahl an Blasoperationen zu verringern, rückt dabei nun vollständig in den Hintergrund.

Hierfür werden die Phasen wie folgt abgeändert: (die neue Funktionsweise wird an einem Beispielhof visualisiert, der die Dimensionen (10,10) hat).

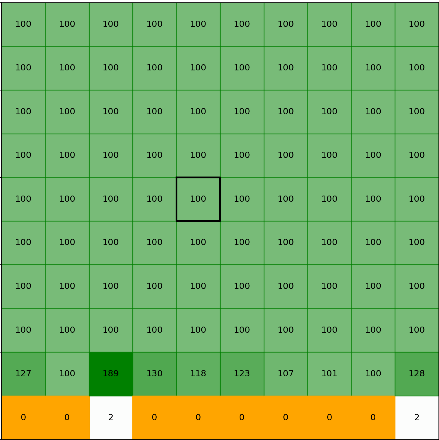
**Abgeänderte 1. Phase (jetzt: Ränder (bis auf den oberen Rand) entlauben):**

In der 1. Phase werden nun zunächst die Blätter, die sich auf dem unteren Rand befinden und keine Eckfelder sind, alle auf die Felder (1,n-1) und (2,n-1) geblasen:

Ein Bild, das Text, Kalender, Quadrat, Zahl enthält.

Automatisch generierte Beschreibung

Anschließend werden die Blätter von Feld (2, n-1) mit einem Muster auf das Nicht-Randfeld (2, n-2) transferiert, und die Ecken (0, n-1) sowie (n-1, n-1) werden geleert bzw. das Laub dieser Ecken wird auf die Felder (0, n-2) sowie (n-1, n-2) transferiert. Hierdurch erreicht man eine teilweise vollständige und teilweise asymptotische Leerung der unteren Reihe:

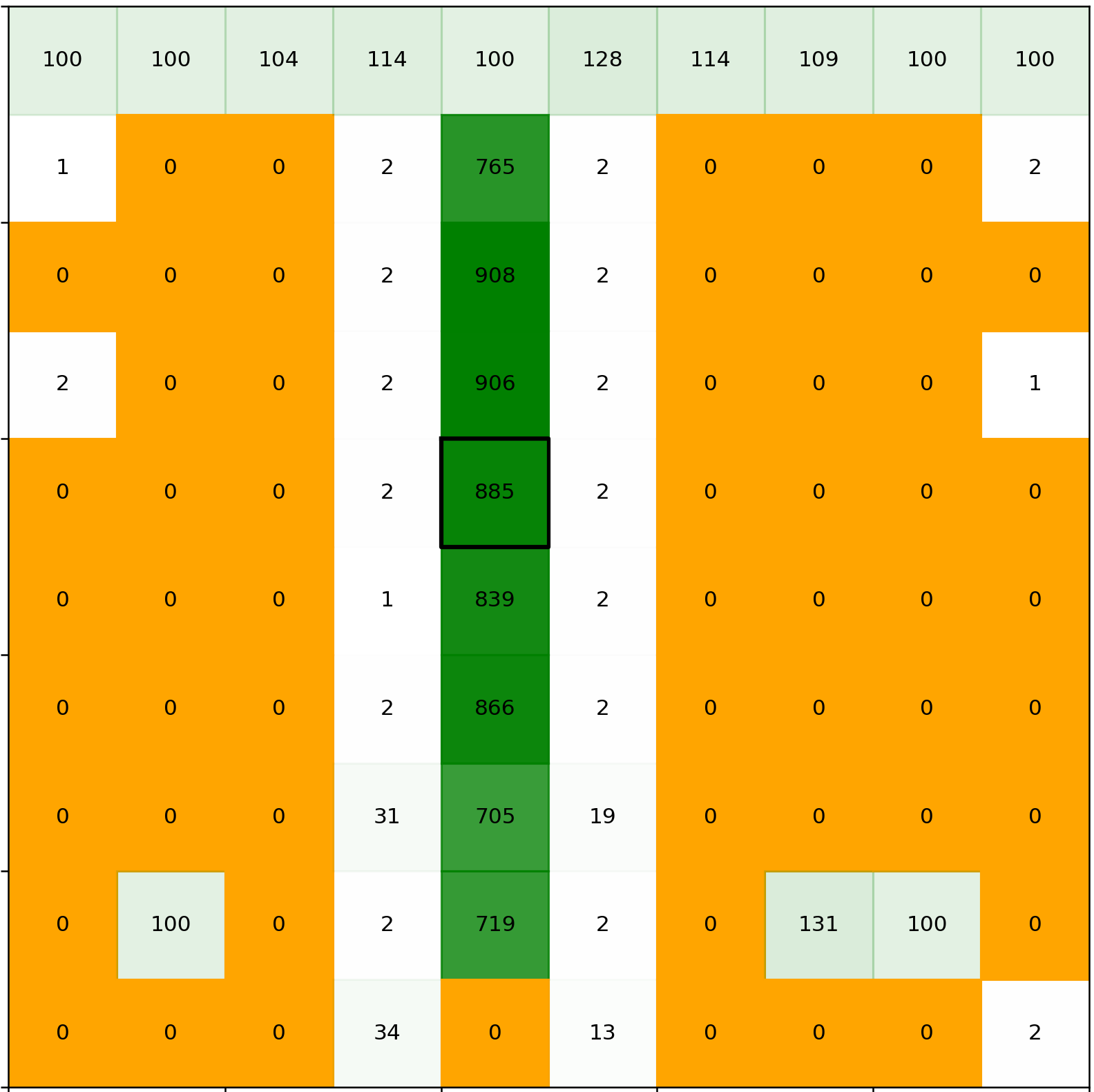


Dasselbe Verfahren wird nun angewendet, um die Ränder rechts und links zu leeren:

Ein Bild, das Text, Quadrat, Screenshot, Kalender enthält.

Automatisch generierte Beschreibung

Abschließend wird das Laub der Nicht-Randfelder auf den Feldern, die sich in derselben Spalte wie Q befinden, konzentrieren, indem das Laub vom rechten und vom linken Rand aus dorthin geblasen wird.



**2. Phase (Das gesamte Laub in die oberste Reihe blasen):**An dieser Phase wurde fast nichts abgeändert. Das Laub wird weiterhin Zeile für Zeile nach oben geblasen, allerdings wird das Laub, das sich in der Spalte von Q befindet, nicht über Feld Q hinausgeblasen. Dadurch sieht der Hof nach Durchführung der 2. Phase so aus:

Ein Bild, das Screenshot, Text, Rechteck, gelb enthält.

Automatisch generierte Beschreibung

Die restlichen Phasen bleiben alle unverändert. Die wesentliche Änderung erfolgte in Phase 1, wo das Laub zunächst in der Spalte von Q gesammelt wird, bevor es Richtung Q geblasen wird. Hierdurch wird verhindert, dass beim Transferieren des Laubs vom oberen zum unteren Rand durch Seitenabtriebe Blätter am Rand des Hofs verloren gehen: Wenn die Blätter nicht am Rand, sondern in der Mitte des Hofs geblasen werden, dann können dort logischerweise auch keine Blätter verloren gehen.

### Laufzeit

Auf die Laufzeit hat die Abänderung des Verfahrens zur Implementierung dieses Sonderfalls keine direkte Auswirkung. Eine indirekte Auswirkung ist allerdings, dass man durch das veränderte Verfahren bei einem Hof gleicher Größe jetzt deutlich mehr Blasoperationen braucht, um den vom abgeänderten Verfahren generierten generalisierten Programmablaufplan durchzuführen und man daher wahrscheinlich länger braucht, um die Simulation durchzuführen.

## Erweiterung 2: Mehrere Laubbläser

Diese Erweiterung basiert ebenfalls auf dem 2. Ansatz. Ich habe mir überlegt: Was, wenn der Hausmeister einen Kollegen bittet, ihm beim Laubblasen zu helfen? Wie groß wären die Auswirkungen, die dies auf den Laubblasprozess hätte? Treten Synergieeffekte zwischen den beiden Laubbläsern auf, die den Laubblasprozess erleichtern?

Im Folgenden wird erläutert, wie ein zweiter Laubbläser hinzugefügt werden kann, der gleichzeitig bzw. zusammen mit dem ersten Laubbläser bläst. Es wird dabei davon ausgegangen, dass die beiden Blasgeräte baugleich sind und für sie somit die gleichen Blasregeln gelten. Außerdem wird festgelegt, dass die beiden Hausmeister nicht auf demselben Feld stehen können (es wäre einfach zu unrealistisch, dass zwei je einen Laubbläser haltende Hausmeister auf dasselbe Feld passen).

Auch für die zweite Erweiterung muss zunächst der Algorithmus, der eine Blasoperation simuliert, angepasst werde. Da die Hausmeister beide gleichzeitig blasen, gilt es ab sofort als „eine Blasoperation“, wenn die beiden Hausmeister ihre Laubbläser zum selben Zeitpunkt je einmal betätigen.

Der Algorithmus nimmt daher ab sofort nicht mehr nur die Argumente feld0 und blow\_direction. Er simuliert jetzt stattdessen zwei Laubbläser, die gleichzeitig blasen, und nimmt daher die Argumente feld0\_b1 (Feld 0 des ersten Laubbläsers), blow\_direction\_b1 (Blasrichtung des ersten Laubbläsers), feld0\_b2 (Feld 0 des zweiten Laubbläsers) und blow\_direction\_b2 (Blasrichtung des zweiten Laubbläsers).

Um eindeutig zu definieren, wie die beiden Laubblasgeräte sich beeinflussen, müssen die folgenden drei Fälle betrachtet werden. Basierend auf den Blasregeln kann genau bestimmt werden, welche Felder durch den ersten Laubbläser beeinflusst werden, wenn er sich auf feld0\_b1 bläst und in die Richtung blow\_direction\_b1 bläst, und welche Felder durch zweiten Laubbläser beeinflusst werden, wenn er sich auf das Feld feld0\_b2 stellt und in die Richtung blow\_direction\_b2 bläst.

**Allgemeine Regel:**

Grundidee: Wie aus den Blasregeln deutlich hervorgeht, übt der Laubbläser auf Feld A mit Abstand am meisten Kraft aus, da dieses Feld vollständig geleert wird. Es wird daher vereinfachend davon ausgegangen, dass der erste Laubbläser kein Laub auf das Feld A des zweiten Laubbläsers blasen kann, da der zweite Laubbläser dieses Laub sofort wieder wegblasen würde (und umgekehrt). Hierfür wird bei der Simulation der Blaswirkung des ersten Laubbläsers eine unsichtbare Barriere um das Feld A des zweiten Laubbläsers errichtet, die wie die Umrandung des Hofs behandelt wird. Analog dazu wird bei der Simulation der Blaswirkung des zweiten Laubbläsers eine unsichtbare Barriere um das Feld A des ersten Laubbläsers errichtets.

Diese Modellierung entspricht nicht ganz der Realität: Wenn Laubbläser 1 Laub auf das Feld A von Laubbläser 2 bläst und sich die beiden Laubbläser dabei parallel gegenüberstehen, dann ergibt es Sinn, dass Feld A von Laubbläser 2 wie die Hofumrandung behandelt wird, da es wahrscheinlich zu ähnlichen Seitenabtrieben wie beim Blasen von Laub gegen die Hofumrandung kommen wird. Stehen sich die Laubbläser jedoch orthogonal gegenüber, dann ist diese Modellierung allerdings nicht realitätsgetreu. Zur Vereinfachung wird dies hingenommen – der Sonderfall, in dem die nicht realitätsgetreue Modellierung auftritt, spielt bei der Entwicklung einer Strategie für zwei Laubbläser ohnehin eine untergeordnete Rolle.

Bei der Simulation der Blaswirkung des ersten Laubbläsers wird außerdem überprüft, ob der zweite Laubbläser in die Richtung der ersten Laubbläsers bläst (dem ist der Fall, wenn die Bedingung *manhatten\_distance(feld0\_b2, feld0\_b1) >   
manhatten\_distance(feld0\_b1, feld0\_b2+blow\_direction\_b2)*erfüllt ist). Wenn der zweite Laubbläser in die Richtung des ersten Laubbläsers bläst, dann würde es wenig Sinn machen, dass Seitenabtriebe des ersten Laubbläsers auf dem Feld 0 des zweiten Laubbläsers landen, obwohl von diesem Feld ein starker Luftstrom in die Gegenrichtung ausgeht. Daher wird in dem Fall auch um das Feld 0 des zweiten Laubbläsers eine unsichtbare Barriere errichtet. Analog dazu wird bei der Simulation der Blaswirkung des zweiten Laubbläsers vorgegangen.

**Sonderfall: Feld A des ersten Laubbläsers überschneidet sich mit Feld A des zweiten Laubbläsers**

Da feld0\_b1 != feld0\_b2 gibt es genau 2 Situationen, in denen der Fall auftreten kann:

Ein Bild, das Reihe, Diagramm, Quadrat enthält.

Automatisch generierte Beschreibung

Abbildung 21: 1. Situation, in der feldA\_b1 = feldA\_b2 auftreten kann: Die Laubbläser stehen sich gegenüber

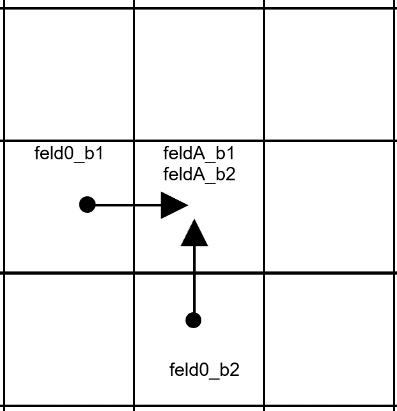


Abbildung 22: 2. Situation, in der feldA\_b1 = feldA\_b2 auftreten kann: Die Laubbläser stehen sich orthogonal gegenüber

Den allgemeinen Regeln zu folge würde in diesem Sonderfall nichts passieren: Für den ersten Laubbläser wäre Feld A blockiert, da es dem Feld A des 2. Laubbläsers entspricht – umgekehrt wäre für den 2. Laubbläser Feld A blockiert, da es dem Feld A des 1. Laubbläsers entspricht. Dies ist aber natürlich nicht realistisch.

Daher wird festgelegt, dass in diesem Sonderfall keine unsichtbare Barriere um Feld A errichtet wird. Stattdessen wird das sich auf dem gemeinsamen Feld A befindliche Laub zwischen den beiden Laubbläsern aufgeteilt. Jeder Laubbläser bläst dabei 50% des Laubs basierend auf der Blaswirkung seines Laubbläsers.

In der 1. Situation (Abb. 21) hat dies zur Folge, dass das gemeinsame Feld A vollständig geleert wird. Dadurch, dass Laubbläser 1 um feld0\_b2 eine unsichtbare Barriere errichtet und Laubbläser 2 um feld0\_b1 eine unsichtbare Barriere erreichtet, wenden beide Bläser die Regeln für das Blasen auf Randfelder orthogonal zum Laub an, was dazu führt, dass 50% des Laubs auf das Feld über A und 50% des Laubs auf das Feld unter A gerät.

In der 2. Situation werden die ganz normalen Blasregeln aus der Aufgabenstellung für jeden Laubbläser mit jeweils 50% der Blätter von Feld A angewendet:

Ein Bild, das Screenshot, Text, Quadrat, Diagramm enthält.

Automatisch generierte Beschreibung

### Synergien ausnutzen

Durch die **Möglichkeit, zwei Blasoperationen gleichzeitig durchzuführen**, können Phase 1 (Leeren der untersten Reihe) und Phase 2 (Transferieren des Laubs auf die oberste Reihe) deutlich schneller ausgeführt werden. **Außerdem können zwei Ecken parallel zueinander geleert werden**, wenn jeder Hausmeister sich um eine Ecke kümmert. Nicht immer ist es sinnvoll, zwei Blasoperationen gleichzeitig auszuführen – z.B. kann es passieren, dass die Anzahl an zu leerenden Felder in Phase 2 nicht gerade ist und daher bei der letzten Blasoperation der Hausmeister nichts zu tun hat. Es gibt grundsätzlich diese Möglichkeiten, mit den Hausmeister, der nichts zu tun hat, umzugehen:

* Man könnte den Hausmeister eine Pause machen lassen,
* Man könnte den Hausmeister auch genauso gut irgendwo in die Wüste schicken und dort irgendeiner sinnlosen Beschäftigung zuführen.
* Sinnvoller ist es aber, Blasoperationen festzulegen, die immer entweder gar keinen Effekt haben oder zumindest eine geringe Menge Laub auf Q befördern und daher zu jedem Zeitpunkt ohne Probleme ausgeführt werden können. Sollte der Hausmeister mal nichts zu tun haben, dann kann er eine dieser Blasoperationen zufällig auswählen und ausführen

**Deutlich schnelleres Aufräumen von Randfeldern möglich:** Ein Randfeld kann nun deutlich schneller geleert werden, indem diese Blasoperation in einem Muster ausgeführt wird:

Ein Bild, das Text, Screenshot, Reihe, Diagramm enthält.

Automatisch generierte Beschreibung

Abbildung 23: Eine Blasoperation ist eingezeichnet. Die beiden Punkte markieren feld0\_b1 und feld0\_b2, die beiden Pfeile markieren blow\_direction\_b1 und blow\_direction\_b2. Das orangene Randfeld wird geleert, Target-Feld ist das grüne Feld

**Deutlich schnelleres Transferieren des Laubs von Feld (Q[0],1) auf (Q[0],2) möglich:** Das Laub kann sehr effektiv transferiert werden, indem in einem Muster abwechselnd die rote und die schwarze Blasoperation durchgeführt werden (siehe Abbildung):

Ein Bild, das Text, Screenshot, Reihe, Diagramm enthält.

Automatisch generierte Beschreibung

### Laufzeit

Die theoretische Laufzeit des Algorithmus verändert sich durch diese Erweiterung nicht. Die praktische Evaluierungsdauer verbessert sich jedoch, da mit zwei Laubbläsern weniger Blasoperationen simuliert werden müssen, bis der gewünschte Zielzustand erreicht ist.

### j Laubbläser

Eine Implementierung von j verschiedenen Laubbläsern (für beliebige natürliche j), die alle gleichzeitig blasen, habe ich nicht vorgenommen. Eine solche Erweiterung wäre aber im Grunde genommen nicht schwer umzusetzen; die einzige wesentliche Änderung bestünde darin, dass jeder Laubbläser vor der Simulation der Blaswirkung Barrieren um alle Felder A aller anderen Laubbläser errichten müsste (wie im Vorkapitel für zwei Laubbläser erläutert).

Trotzdem steht die theoretische Frage im Raum: **Welche zusätzlichen Synergieeffekte würden entstehen, wenn man 3 oder mehr Laubbläser gleichzeitig blasen ließe?**

* Wenn drei Laubbläser gleichzeitig blasen, dann entsteht hierdurch erstmals die Möglichkeit, das gesamte Laub von einem beliebigen Nicht-Randfeld in nur einem Schritt auf ein anderes Feld zu blasen. Hierfür müssen die Laubbläser U-förmig um das zu leerende Feld angeordnet werden, d.h. die Laubbläser stellen sich auf die drei Nachbarfelder vom zu leerenden Feld, auf die das Laub nicht gelangen soll, nur das Feld, auf das das Laub geblasen werden soll, wird offen gelassen. Das zu leerende Feld wird dabei vollständig geleert.  
  Dies hat zur Folge, dass jedes beliebige Nicht-Randfeld nun mit geringem Aufwand vollständig geleert werden kann. Hierdurch wird der letzte Teil von Phase 4 deutlich einfacher.
* Wenn noch mehr Laubbläser hinzukommen, dann beschleunigt das zwar natürlich den Blasprozess und verstärkt bestehende Synergien, es entstehen aber keine wesentlichen neuen Synergieeffekte mehr. Der „letzte große Sprung“ findet also beim Erhöhen der Laubbläserzahl von 2 auf 3 statt.
* Bei einem Hof mit fixen Seitenlängen ist irgendwann der Punkt erreicht, wo eine weitere Erhöhung der Laubbläserzahl keinen Sinn mehr macht, da die neuen Laubbläser nur alte blockieren würden (spätestens dann, wenn es mehr Bläser als Felder gibt, ist dieser Punkt auf jeden Fall erreicht).

## Erweiterung 3: Verschiedene Laubtypen

Die Erweiterung 3 führt zwei verschiedene Laubtypen ein, die jeweils unterschiedlichen Regeln unterliegen. Dies erfordert Änderungen der Datenstrukturen, die die Blattverteilungen speichern, und der Funktionen, die die Blasoperationen durchführen.

Diese Erweiterung ist besonders interessant wegen ihrem Realitätsbezug: Auf einem Schulhof wird in der Regel nicht nur monokulturartig eine Baumart gepflanzt. Stattdessen muss davon ausgegangen werden, dass verschiedene Baumarten wachsen und somit verschiedene Laubtypen wegzublasen sind.

Die Erweiterung wird so umgesetzt, dass für jede Blattart die geltenden Blasregeln angewendet werden können, Bei der Simulation des Blasvorgangs wird anschließend auf jede Blattart die passende Regel angewendet.

## Weitere Erweiterungsideen (mit Lösungsansätzen)

### Andere Hofformen: Rechtwinklige Polygone

Hierbei handelt es sich um eine Erweiterungsidee, die eigentlich implementieren wollte, aufgrund von Zeitdruck (Abiturvorbereitung) aber leider abbrechen musste. Ich werde hier trotzdem beschreiben, wie die erweiterte Aufgabe ausgesehen hätte und was für eine Art von Lösung ich implementiert hätte.

Ziel war die Entwicklung eines Verfahrens, dass auch auf Höfen, die nicht rechteckig geformt sind, so viel Laub wie möglich auf Q sammeln kann. Die Höfe sollten die Form eines Polygons haben, das ausschließlich rechte Innen- oder Außenwinkel hat. Beispiele für solche Formen:

Ein Bild, das Diagramm, Rechteck, Entwurf, Reihe enthält.

Automatisch generierte Beschreibung

Ich plante, zur Lösung dieses Problems einen Divide-and-conquer Algorithmus zu verwenden. Dieser Algorithmus sollte das Polygon zunächst in zusammenhängende Rechtecke aufteilen, die in einer Baumstruktur (Graph) gespeichert werden, damit ersichtlich ist, welches Rechteck sich mit welchen Rechteck überlappt (Divide-Teil):

**Algorithmus 12: Divide-Teil des Divide-and-conquer Algorithmus**

Parameter: Q = Index von Feld Q  
Parameter: Hof = Polygon, das den Hof darstellt

1. rechteck\_tree = Initialisiere einen leeren Baum
2. initial\_rechteck = Finde das größte Rechteck, das in das Polygon Hof passt und Q beinhaltet
3. Füge initial\_rechteck als erste bode zu rechteck\_tree hinzu
4. **Endlosschleife, die bis zum Abbruch ausgeführt wird:**
5. unbound\_field = Finde ein Feld, dass Nachbarfeld eines der sich in rechteck\_tree befindenden   
    Rechtecke ist und noch auf keinem der in Hof vorhandenen Rechtecke liegt
6. origin\_rechteck = Finde das Rechteck, von dem unbound\_field ein Nachbarfeld ist
7. new\_rechteck = Finde das größte Rechteck, das in das Polygon Hof passt, origin\_rechteck   
    schneidet und unbound\_field beinhaltet
8. **füge** new\_rechteck **am Baum** rechteck\_tree **an der Node** von origin\_rechteck **an**

**…**

Anschließend werden die im Baum gespeicherten Rechtecke „von außen nach innen“ geleert. Das bedeutet, es wird mit der optimalen Leerung des Rechtecks, das im Baum am weitesten vom Rechteck initial\_rechteck entfernt ist, begonnen. Das Laub dieses Rechtecks wird auf dem Feld gesammelt (unter Verwendung von Ansatz 2), das das Rechteck mit dem Rechteck, das im Baum durch die Parent-Node des zu leerenden Rechtecks repräsentiert ist, schneidet. Dieser Prozess wird so lange fortgesetzt (in jeder Iteration wird immer das noch nicht geleerte Rechteck ausgewählt, das im Baum am weitesten von initial\_rechteck entfernt ist), bis alle Rechtecke aus dem Baum bis auf das Rechteck, das Q enthält, leer sind. Dieses Rechteck kann nun unter Verwendung von Ansatz 2 so aufgeräumt werden, dass das Laub auf Q geblasen wird.

### Weitere Ideen

Ich hatte noch viele weitere Ideen für mögliche Erweiterungen, die aus zeitlichen Gründen nicht umsetzbar waren:

* Höfe mit sechseckigen Feldern
* Seitenwinde
* Eine Variante der Aufgabe, bei der der Hausmeister den Laubbläser dauerhaft eingeschaltet lassen muss

# Umsetzung

Die Lösungsidee wird in Python 3.10 implementiert. Verwendete Python-Bibliotheken:  
- *math:* Verwendet für mathematische Operationen (atan, Quadratwurzel, comb), Python standard library  
- *matplotlib:* Verwendet zum Visualisieren des Hofs in einem Plot  
- *random*: Simulieren des Zufalls  
- *numpy:* Modellierung des Hofs als numpy-Array

Insgesamt werden diese Dateien erstellt:

|  |  |  |
| --- | --- | --- |
| **Ordner** | **Datei** | **Inhalt** |
| Aufgabe 1 | aufgabe1\_solver1.py | Implementiert den 1. Ansatz |
| aufgabe1\_solver2.py | Implementiert den 2. Ansatz (bester Ansatz bzw. zu bewertendes Verfahren) |
| aufgabe1\_solver3.py | Implementiert den 3. Ansatz |
| hof.py | Enthält die Klassen Rules und Hof (Modellierung eines Schulhofs und der Regeln, die auf ihm gelten) |
| binomial\_util.py | Enthält Hilfsfunktionen zum Berechenen von Binomialverteilungen |
| Aufgabe 1 E1 | aufgabe1\_solver2.py | Erste Erweiterung (keine Hofmauer), basiert auf 2. Ansatz |
| hof.py | Enthält die Klassen Rules und Hof (Modellierung eines Schulhofs und der Regeln, die auf ihm gelten) |
| binomial\_util.py | Enthält Hilfsfunktionen zum Berechenen von Binomialverteilungen |
| Aufgabe 1 E2 | aufgabe1\_solver2.py | Zweite Erweiterung (zwei Laubbläser), basiert auf 2. Ansatz |
| hof.py | Enthält die Klassen Rules und Hof (Modellierung eines Schulhofs und der Regeln, die auf ihm gelten) |
| binomial\_util.py | Enthält Hilfsfunktionen zum Berechenen von Binomialverteilungen |
| Aufgabe 1 E3 | aufgabe1\_solver2.py | Dritte Erweiterung (mehrere verschiedene Laubtypen), basiert auf 2. Ansatz |
| hof.py | Enthält die Klassen Rules und Hof (Modellierung eines Schulhofs und der Regeln, die auf ihm gelten) |
| binomial\_util.py | Enthält Hilfsfunktionen zum Berechenen von Binomialverteilungen |

Im Folgenden werden die besonders wichtigen und interessanten Teile der Implementierung erläutert.

## Hilfsfunktionen zur Berechnung von Binomialverteilungen (binomial\_util.py)

Ich definiere die Funktion binomialpdf(\*, n, p, k), die den Wert von P(X=k) mit den Parametern n und p und einer binomialverteilte Größe X berechnet und als *float* zurückgibt. Hierfür wird zunächst der Binomialkoeffizient mithilfe von *math* library bestimmt:

bincomb = math.comb(n, k)

Anschließend wird über die logarithmische Darstellung P(X=k) berechnet und zurückgegeben.

log\_binom = math.log(bincomb) + k \* math.log(p) + (n - k) \* math.log(1 - p)

return math.exp(log\_binom)

Es wird außerdem die Funktion binomialdist(\*, n, p, relevant\_threshold=0.01) definiert, die P(xX=k) für alle k im Bereich [0;n] und den Parametern n und p, die der Funktion als Keyword-Argumente gegeben werden, für eine binomialverteilten Größe X berechnet. Die Funktion gibt eine Liste *dist* mit der Wahrscheinlichkeitsverteilung zurück, der Index in der Liste korrespondiert mit dem jeweiligen k-Wert, dessen Eintrittswahrscheinlichkeit am entsprechenden Index in der Liste beschrieben wird.  
Da es für große n sehr aufwändig wäre, alle P(X=k) zu berechnen, wird stattdessen ein anderes Vorgehen genutzt: Beim Berechnen von P(X=k) wird mit dem k-Wert begonnen, für den P(X=k) maximal wird. Dann wird sich das Monotoniekriterium der Binomialverteilung zunutze gemacht:

dist = np.zeros(n+1) # Mit Nullen gefüllte Liste erzugen

for k in range(math.floor(n\*p),-1,-1):

pdf = binomialpdf(n=n, p=p, k=k)

dist[k] = pdf

if pdf < relevant\_threshold:

break

for k in range(math.ceil(n\*p),n+1,1):

pdf = binomialpdf(n=n, p=p, k=k)

dist[k] = pdf

if pdf < relevant\_threshold: # der restliche Bereich ist   
 vernachlässigbar, da die Wahrscheinlichkeiten   
 verschwinden gering werden

break

…

Sobald sehrs kleine Werte von P(X=k) werden nicht berechnet, stattdessen werden diese Listenelemente „abgeflacht“ bzw. es wird statt dem korrekten P(X=k) Wert ein konstanter Wert in der Verteilungsliste *dist* gespeichert, sodass die Liste aufsummiert immer noch 1 ergibt:

…

if len(dist[dist==0]) != 0:

fill\_rest = (1- sum(dist)) / len(dist[dist==0])

dist[dist==0] = fill\_rest

return dist

Es wird außerdem eine *biomial\_likeliest* *(\*, n, p, rank=0, handle\_ties="higher")* -Funktion definiert. Diese Funktion bestimmt den wahrscheinlichsten fall, der bei einer Binomialverteilung mit n=n und p=p auftritt. Wird das Keyword Argument *rank = „random“* übergeben, dann simuliert die Funktion den tatsächlichen Zufall bzw. wählt basierend auf der mit *binomialdist* berechneten Binomialverteilung als Wahrscheinlichkeits- bzw. Gewichtefunktion ein k mithilfe von *random.choices* zufällig aus:

def binomial\_likeliest(\*, n, p, rank=0, handle\_ties="higher"):   
 if rank == "random":

dist = binomialdist(n=n, p=p)

k = random.choices(population=[i for i in range(n+1)], k=1,   
 weights=dist)[0]

return k, dist[k]

## Definieren einer Rules-Klasse (hof.py)

Die Rules-Klasse dient zum Speichern der Regeln, die auf einem Hof gelten. Der Konstrukor der Klasse nimmt diese Keyword-Argumente, die als Attribute der Klasse gespeichert werden:

* A\_seitenabtrieb = 0.1,
* B\_vorne\_abtrieb = 0.1,
* A\_noB\_seitenabtrieb=0.5\*0.95,
* s1=0.9,
* s4=0.05,
* use\_binomial=True,
* binomial\_rank="random",
* binomial\_handle\_ties="higher”

Noch im Konstruktor der Klasse wird sichergestellt, dass die angegebenen Argumente alle im gültigen Wertebereich liegen:

….

assert 0 < A\_seitenabtrieb < 0.5 # Implementierungsbedingte Einschränkung: An dieser Stelle wird festgelegt, dass niemals alle Blätter auf abgetrieben werden

assert 0 < B\_vorne\_abtrieb < 1

assert 0 < A\_noB\_seitenabtrieb <= 0.5 # Implementierungsbedingt wird in diesem Fall allerdings erlaubt, dass das ganze Laub zur Seite wegfliegt

...

assert isinstance(binomial\_rank, int) or binomial\_rank == "random"

assert binomial\_handle\_ties == "higher" or binomial\_handle\_ties == "lower" or binomial\_handle\_ties == "random"

## Definieren einer Hof-Klasse (hof.py)

Es wird außerdem eine Klasse namens Hof erstellt, die einen aus Planquadraten bestehenden Hof repräsentiert, auf dem Laub geblasen werden kann.

**Attribute der Klasse:**

*Hof.x\_size –* speichert die x-Größe des Hofs

*Hof.y\_size –* speichert die y-Größe des Hofs

*Hof.rules –* speichert die ein Rules-Objekt, in dem die Regeln festgesetzt sind, die für Blasoperationen auf dem Hof gelten

*Hof.startwert* – speichert die Anzahl an Blättern, die sich zu Beginn auf jedem Feld befindet

*Hof.blas\_counter* – Integer, der die Anzahl an auf dem Hof durchgeführten Blasoperationen zählt

*Hof.blas\_log* – Liste, in der die durchgeführten Blasoperationen als dictionaries gespeichert werden

**Funktionen der Klasse:**

*Hof.render(\*, title=““, plot\_last\_op=False)* – plottet den Hof mit matplotlib und zeigt ihn in einem Popup an. Dabei wird Feld Q schwarz umkastet. Felder, die vollständig geleert sind, werden orange eingefärbt. Felder, die nicht vollständig eigefärbt sind, werden grün mit verschiedenen Sättigungsstufen eingefärbt. Dabei entspricht die Sättigungsstufe der Blattmenge auf dem Feld geteilt durch die Blattmenge auf dem Feld, auf dem am meisten Blätter liegen:

saturation = min(1, self.felder[i][j] / max\_value)

Über das *title* Argument kann die Überschrift des Plots eingestellt werden. Wenn die zuletzt durchgeführte Blasoperation als Pfeil geplottet werden soll, dann muss *plot\_last\_op* auf True gesetzt werden.

*Hof.\_\_str\_\_(\*, round\_digits=5) –* gibt eine formatierte Darstellung des Hofs zurück, der entnommen werden kann, wie viele Blätter sich auf den jedem Feld befinden.

*Hof.print\_blas\_log()* – Gibt den Blas-Log als String zurück

*Hof.is\_corner(feld : tuple)* – Gibt als boolean zurück, ob das Feld am Index feld ein Eckfeld ist

*Hof.is\_edge(feld : tuple)* – Gibt als boolean zurück, ob das Feld am Index feld ein Randfeld ist

*Hof.does\_exist(feld : tuple)* – Gibt als boolean zurück, ob das Feld am Index feld im Hof existiert

*Hof.are\_adjacent(feldA: tuple, feldB : tuple)* – Gibt als boolean zurück, ob feldA Nachbarfeld von feldB ist

*Hof.orthogonal\_direction(direction)* – Gibt mit folgendem Code einen Richtungsvektor zurück, der zu direction orthogonal ist:

return (0,1) if direction[1] == 0 else (1,0)

Es wird außerdem eine *blase(feld0, blow\_direction)-Funktion* implementiert, die basierend auf den Blasregeln einen Blasvorgang simuliert. Die Funktion loggt die Blasoperation, danach ermittelt sie zunächst die Richtung, die orthogonal zur Blasrichtung ist, und anschließend Feld A und Feld B:

# Richtung, die orthogonal zur Blasrichtung ist, ermitteln:

orthogonal\_direction = self.get\_orthogonal\_direction(blow\_direction)

# Feld A (Feld unmittelbar vor dem Laubbläser) ermitteln:

feldA = (feld0[0]+blow\_direction[0], feld0[1]+blow\_direction[1])

if not self.does\_exist(feldA):

return # -> Es gibt kein Feld vor dem Laubbläser bzw. der Laubbläser bläst gegen die  
 Umrandung. Für diesen Fall ist definiert, dass sich die Verteilung des   
 Laubs nicht verändert

new\_feldA\_value = 0 # In dieser Variable wird die neue Anzahl an Blättern auf Feld A   
 gespeichert

# Feld B (Feld hinter Feld A) ermitteln:

feldB = (feldA[0]+blow\_direction[0], feldA[1]+blow\_direction[1])

Anschließend überprüft sie, ob Feld B existert:

if self.does\_exist(feldB):

Je nachdem ob B existiert werden basierend auf den entsprechenden Regeln die Seitenabtriebe bestimtm, danach werden alle betroffenen Felder aktualisiert (sofern sie existieren).

Bei der Bestimmung der Blattanzahl, die abgetrieben wird, wird dabei – je nachdem was in Hof.rules festgelegt ist – der Erwartungswert oder eine Zufallssimulation über die Binomialverteilung verwendet, was z.B. hier sichtbar ist:

if self.rules.use\_binomial:

A\_noB\_seitenabtrieb\_1, \_ = binomial\_likeliest(n=self.felder[feldA],   
 p=self.rules.A\_noB\_seitenabtrieb,   
 rank=self.rules.binomial\_rank,   
 handle\_ties=self.rules.binomial\_handle\_ties)

A\_noB\_seitenabtrieb\_2, \_ = binomial\_likeliest(n=self.felder[feldA]-  
 A\_noB\_seitenabtrieb\_1, p=self.rules.A\_noB\_seitenabtrieb/(1-  
 self.rules.A\_noB\_seitenabtrieb), rank=self.rules.binomial\_rank,   
 handle\_ties=self.rules.binomial\_handle\_ties)

else:

A\_noB\_seitenabtrieb\_1 = self.felder[feldA] \*   
 self.rules.A\_noB\_seitenabtrieb

A\_noB\_seitenabtrieb\_2 = A\_noB\_seitenabtrieb\_1

Da der dieser Funktion zugrundeliegene Algorithmus in der Lösungsidee bereits sehr ausführlich als Pseudocode analysiert wurde, wird auf eine weitere Analyse der Funktion verzichtet.

## Grundaufgabe Ansatz 1

Es handelt sich hierbei nicht um meinen besten bzw. den zu bewertenden Ansatz, daher bleibt die Analyse relativ oberflächlich.

Die Parameter werden in Zeile 24 bis 36 im Code festgelegt:

# Hof-Eigenschaften festlegen:

Q = (3,4) # Index von Feld Q festlegen

hof\_size = (7,7) # Hofseitenlängen festlegen

startwert = 100 # Anfangsanzahl an Blättern pro Feld

# Wahrscheinlichkeitsmodellierung festlegen:

use\_binomial = True # Festlegen, ob die Wahrscheinlichkeiten basierend auf der Binomialverteilung simuliert oder ob die Erwartungswerte verwendet werden sollen

weight\_avg = 0.1 # Gewichtung des 1. Heuristik-Maßes (durchschnittlicher Laubabstand zu Feld Q)

weight\_varianz = 0.9 # Gewichtung des 2. Heuristik-Maßes (Varianz der Laubabstände zu Feld Q)

# Abbruchbedingungen feslegen:

satisfied\_constraint = 0.8 # Bei erreichen dieser prozentualen Laubmenge (im Verhältnis zum Gesamtlaub) wird das Programm auf jeden Fall abgebrochen

max\_operations = 300 # Maximal durchgeführte Anzahl an Operationen, nach denen der Blasprozess auf jeden Fall abgebrochen wird

# Datei zum Speichern der durchgeführten Blasvorgänge festlegen:

output\_file = ""

Auf oberster Ebene befinden sich zwei Hilfsfunktionen (*squared\_std* und *manhattan\_distance*).

Es wird außerdem eine Solver1-Klasse definiert, die alle zum Lösen des Hofs über die Greedy-Heuristik relevanten Funktionen enthält. Die *Solver1.edge\_distance* Funktion gibt die Randdistanz bzw. die kleinste Verbindungsstrecke zwischen den Randfeldern feld0 und feld1, die nur über Rand- und Eckfelder läuft, zurück.

Die Greedy-Heuristik zum Leeren der Randfelder wird als Funktion namens *Solver1.greedy\_edges* implementiert, die Greedy-Heuristik, die auf Nicht-Randfelder anzuwenden ist, wird als Solver1.greedy\_mid definiert und nimmt als Argumente die Gewichte *weight\_varianz* und *weight\_avg.*

Die Solver1.step() Funktion führt bei Aufruf den nächsten Schritt auf, der basierend auf einer der beiden Greedy-Heuristiken ausgewählt wird. Dabei speichert das Solver1.clear\_edges Attribut, ob als nächstes ein Randfeld oder ein Nicht-Randfeld zu bearbeiten ist:

def step(self):

if self.clear\_edges:

next\_op = self.greedy\_edges()

else:

next\_op = self.greedy\_mid(weight\_varianz=self.weight\_varianz, weight\_avg=self.weight\_avg)

Wenn eine der Abbruchbedingungen erfüllt ist wird False, ansonsten wird True zurückgegeben:

self.hof.blase(next\_op["feld0"], next\_op["blow\_direction"])

if self.satisfied\_constraint is not None:

if self.hof.felder[self.Q] / self.sum\_laub >= self.satisfied\_constraint:

return False

if self.max\_operations is not None:

if self.hof.blas\_counter >= self.max\_operations:

return False

return True

## Grundaufgabe Ansatz 2:

Auch hier werden zunächst in Zeile 8 bis 17 die Hyperparameter des Programms festgelegt:

# Hyperparameter festlegen

Q = (2,2) # Index von Feld Q festlegen

hof\_size = (5,5) # Hofseitenlängen festlegen

use\_binomial = True # Festlegen, ob die Wahrscheinlichkeiten basierend auf der Binomialverteilung simuliert oder ob die Erwartungswerte verwendet werden sollen

tolerated\_amount = 5 # Blattmenge, die auf nicht vollständig leerbaren Feldern als vernachlässigbar gilt

max\_muster\_operations = 5000 # Anzahl an Operation, die pro Muster maximal durchgeführt werden

startwert = 100 # Anfangsanzahl an Blättern pro Feld

choose\_faster\_path = True

# Datei zum Speichern der durchgeführten Blasvorgänge festlegen:

output\_file = ""

Es wird eine *Muster* Klasse erstellt, in der das Muster-Konzept implementiert ist. Dabei wird im Muster.operations Attribut die Abfolge an Blasoperationen des Musters als Liste gespeichert. Eine einzelne Blasoperation wird hierbei als dictionary der Form {„feld0“: …, „blow\_direction“:…} gespeichert.

Das Muster.next\_op\_index Attribut speichert den Index, den die als nächstes auszuführende Operation in Muster.operations hat.

Die Muster.reset() Methode setzt das Muster auf seinen Anfangszustand zurück.

Die Muster.step() Methode führt basierend auf Muster.operations die nächste Operation des Musters aus, hierfür wird überprüft, ob diese Operation eine Blasoperation oder ein anderes Muster ist:

if isinstance(self.operations[self.next\_op\_index], dict):

# -> Eine Blasoperation liegt vor, die ausgeführt wird

self.hof.blase(self.operations[self.next\_op\_index]["feld0"], self.operations[self.next\_op\_index]["blow\_direction"])

self.next\_op\_index += 1

self.num\_operations += 1

elif isinstance(self.operations[self.next\_op\_index], Muster):

# -> Ein anderes Muster liegt vor, das ausgeführt wird

run\_another\_step = self.operations[self.next\_op\_index].step()

if run\_another\_step:

return True

self.next\_op\_index += 1

self.num\_operations += 1

if self.next\_op\_index == len(self.operations):

# -> Einmal durch alle Operationen des Musters durchgelaufen -> i zurücksetzen

self.next\_op\_index = 0

if self.check\_for\_changes:

# Überprüfen, ob noch Veränderungen stattfinden

new\_sum = sum([self.hof.felder[index] for index in self.source\_fields])

if new\_sum == self.current\_sum:

return False

self.current\_sum = int(new\_sum)

if isinstance(self.operations[self.next\_op\_index], Muster):

self.operations[self.next\_op\_index].reset()

Anschließend werden die Abbruchbedingungen überprüft:

if self.num\_operations >= self.num\_max\_operations:

return False

if max([self.hof.felder[index] for index in self.source\_fields]) <=   
 self.tolerated\_amount:

return False

Die Solver2-Klasse speichert alle Informationen und Funktionen, die zum Erstellen und Ausführen eines generalisierten Ablaufplans benötigt werden. Der generalisierte Ablaufplan wird dabei in Solver2.strategy gespeichert.

**Die folgenden Funktionen sind an der Erstellung des generalisierten Ablaufplans besonders stark beteiligt:**

*Solver2.add\_operation(operation) –* fügt eine Blasoperation oder ein Muster zum Ablaufplan hinzu und macht davor die Initialrotation rückgängig

*Solver2.corner\_to\_edge(source\_corner\_field, target\_field)* – Gibt ein Muster zurück, das bei Anwendung Laub vom Eckfeld source\_corner\_field auf das Randfeld target\_field bläst

*Solver2.edge\_to\_mid (source\_edge\_field, target\_field)* – Gibt Muster zurück, das bei Anwendung Laub vom Randfeld source\_edge\_field und seinen beiden auf dem Rand liegenden Nachbarn auf das Nicht-Rand-oder-Eckfeld target\_field bläst

*Solver2.clear\_bottom\_line()* – leert die unterste Zeile des Hofs (Phase 1)

*Solver2.move\_to\_top\_line()* – leert die unterste Zeile des Hofs (Phase 2)

*Solver2.concentrate\_top\_line()* – leert die unterste Zeile des Hofs (Phase 3)

*Solver2.transfer\_to\_Q()* – leert die unterste Zeile des Hofs (Phase 4)

*Solver2.build\_strategy()* – führt die Initialrotation durch und führt danach die vier zuvor genannten Funktionen aus

# Beispiele – Grundaufgabe

Zum Demonstrieren der Funktionstüchtigkeit meiner Programme habe ich sie mit verschiedenen Parametern ausgeführt. Die vollständigen Beispielausgaben sowie die vom Programm erstellten Dateien, die die durchgeführten Blasoperationen enthalten, sind im Ordner *Aufgabe 1* zu finden.

Im Ordner Aufgabe 1/Outputs sind für jedes Beispiel die Blasoperationen zu finden, die die Programme bei der Simulation durchgeführt haben. Sie sind in Textdateien gespeichert. Dabei sind die Blasoperationen, die von Ansatz 1 bei Beispiel 1 durchgeführt wurden in der Textdatei „output\_{ansatz}\_{beispiel}.txt“ gespeichert.

**Anmerkung:** Wenn *use\_binomial True* ist und *binomial\_rank == „random“,* dann wird bei der Simulation der Blasvorgänge der tatsächliche Zufall simuliert (wie im Kapitel *Lösungsidee* erläutert). Die Programmausgabe hängt somit vom Zufall ab und ist somit bei jedem Ausführen anders bzw. nicht reproduzierbar, auch wenn beim erneuten Ausführen dieselben Parameter gegeben werden.

## Ansatz 1 (aufgabe1\_solver.py)

### Beispiel 1

**Eingabeparameter:**

Q = (2,2)

hof\_size = (5,5)

use\_binomial = True  
binomial\_rank = “random”  
startwert = 100

weight\_avg = 0.5

weight\_varianz = 0.5

max\_operations = 1000

satisfied\_constraint = 0.8

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 229

Anteil der Blätter auf Q an der Gesamtlaubmenge: 74.44 %

Laubverteilung nach dem Blasprozess:

15 | 0 | 12 | 0 | 14

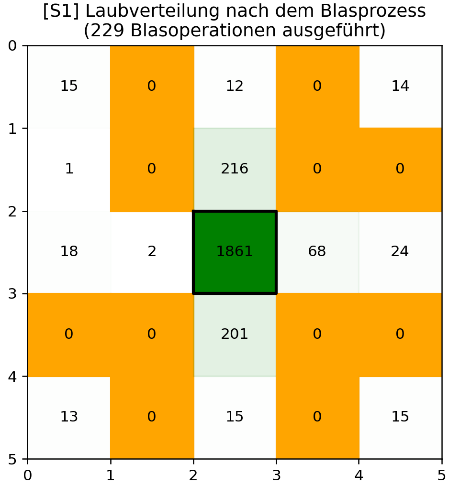
1 | 0 | 216 | 0 | 0

18 | 2 | 1861 | 68 | 24

0 | 0 | 201 | 0 | 0

13 | 0 | 15 | 0 | 15

**Ausgabeplots:**



**Beobachtung:**

Wie man sieht, gelingt es nicht, mehr als 74 % des Laubs auf Feld Q zu versammeln. Ansatz 2 schneidet da deutlich besser ab.

### Beispiel 2

**Eingabeparameter:**

Q = (2,2)

hof\_size = (5,5)

use\_binomial = True  
binomial\_rank = “random”  
startwert = 100

weight\_avg = 0.5

weight\_varianz = 0.5

max\_operations = 1000

satisfied\_constraint = 0.8

*Diese Eingabeparameter entsprechen denen aus Beispiel 1: Um zu zeigen, dass das Programm tatsächlich den Zufall simuliert, wurde es mit denselben Eingabeparametern zweifach ausgeführt.*

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 186

Anteil der Blätter auf Q an der Gesamtlaubmenge: 74.52 %

Laubverteilung nach dem Blasprozess:

14 | 0 | 13 | 0 | 10

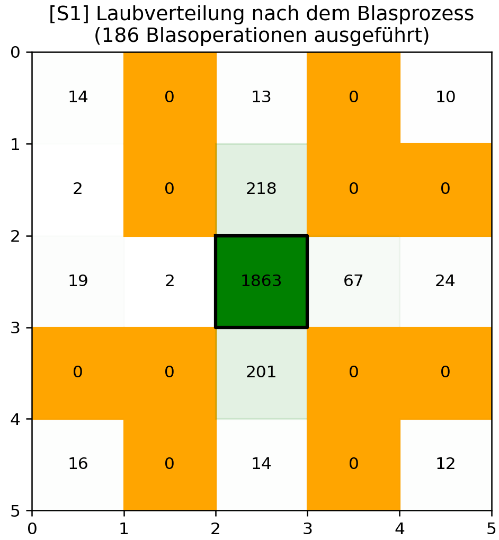
2 | 0 | 218 | 0 | 0

19 | 2 | 1863 | 67 | 24

0 | 0 | 201 | 0 | 0

16 | 0 | 14 | 0 | 12

**Ausgabeplots:**



### Beispiel 3

**Eingabeparameter:**

Q = (1,1)

hof\_size = (3,4)

use\_binomial = True  
binomial\_rank = “random”  
startwert = 100

weight\_avg = 0.5

weight\_varianz = 0.5

max\_operations = 1000

satisfied\_constraint = 0.8

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100

100 | 100 | 100

100 | 100 | 100

100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 455

Anteil der Blätter auf Q an der Gesamtlaubmenge: 80.08333333333333 %

Laubverteilung nach dem Blasprozess:

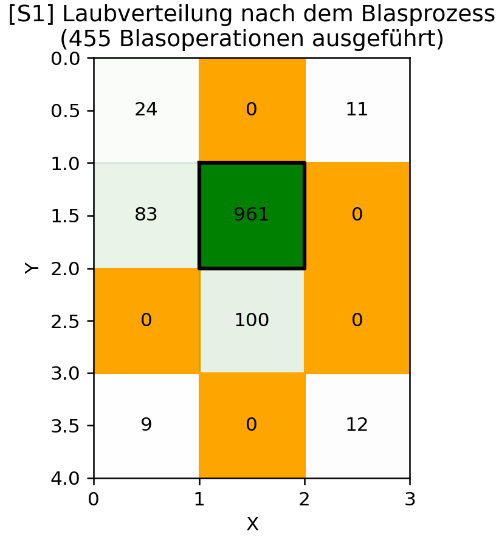
24 | 0 | 11

83 | 961 | 0

0 | 100 | 0

9 | 0 | 12

**Ausgabeplots:**



### Beispiel 4

**Eingabeparameter:**

Q = (3,3)

hof\_size = (8,8)

use\_binomial = True  
binomial\_rank = “random”  
startwert = 100

weight\_avg = 0.5

weight\_varianz = 0.5

max\_operations = 1000

satisfied\_constraint = 0.8

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 610

Anteil der Blätter auf Q an der Gesamtlaubmenge: 80.0625 %

Laubverteilung nach dem Blasprozess:

17 | 0 | 1 | 73 | 0 | 0 | 0 | 10

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 218 | 0 | 0 | 0 | 0

40 | 0 | 218 | 5124 | 0 | 0 | 0 | 0

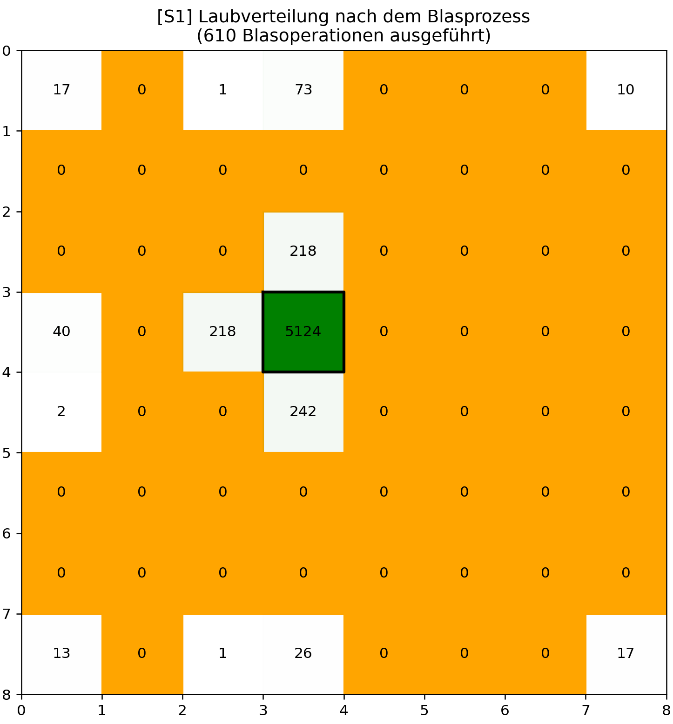
2 | 0 | 0 | 242 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

13 | 0 | 1 | 26 | 0 | 0 | 0 | 17

**Ausgabeplots:**



**Beobachtung:**

Auffällig ist, dass bereits bei einer Hofgröße von gerade einmal (8,8) das Programm bereits ca. 10 Sekunden braucht, bis die Simulation abgeschlossen ist. Dies ist auf die schlechte Laufzeit zurückzuführen. Bei den anderen Ansätzen, insbesondere bei Ansatz 2, ist die Laufzeit deutlich besser.

### Beispiel 5

**Eingabeparameter:**

Q = (3,4)

hof\_size = (7,7)

use\_binomial = True  
binomial\_rank = “random”  
startwert = 100

weight\_avg = 1

weight\_varianz = 0

max\_operations = 200

satisfied\_constraint = 0.8

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 283

Anteil der Blätter auf Q an der Gesamtlaubmenge: 80.08163265306122 %

Laubverteilung nach dem Blasprozess:

18 | 0 | 0 | 17 | 0 | 0 | 14

0 | 0 | 1 | 4 | 1 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0

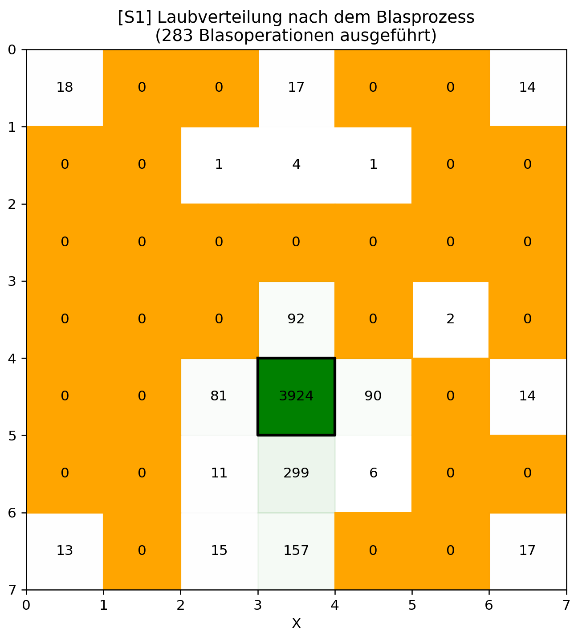
0 | 0 | 0 | 92 | 0 | 2 | 0

0 | 0 | 81 | 3924 | 90 | 0 | 14

0 | 0 | 11 | 299 | 6 | 0 | 0

13 | 0 | 15 | 157 | 0 | 0 | 17

**Ausgabeplots:**



**Beobachtung:**

In diesem Beispiel wurden die Parameter so eingestellt, das maximal 300 Operationen durchgeführt werden. Das Gewicht für die Distanzheuristik wurde auf 1 und das für die Varianzheuristik auf 0 gestellt. Das Ergebnis ist, das in nur 300 Schritten relativ viel Laub auf Q versammelt wurde, das restliche Laub aber ungeordnet auf den restlichen Feldern verstreut ist.

### Beispiel 6

**Eingabeparameter:**

Q = (3,4)

hof\_size = (7,7)

use\_binomial = True  
binomial\_rank = “random”  
startwert = 100

weight\_avg = 0.1

weight\_varianz = 0.9

max\_operations = 200

satisfied\_constraint = 0.8

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 300

Anteil der Blätter auf Q an der Gesamtlaubmenge: 8.26530612244898 %

Laubverteilung nach dem Blasprozess:

18 | 0 | 0 | 0 | 0 | 0 | 14

0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0

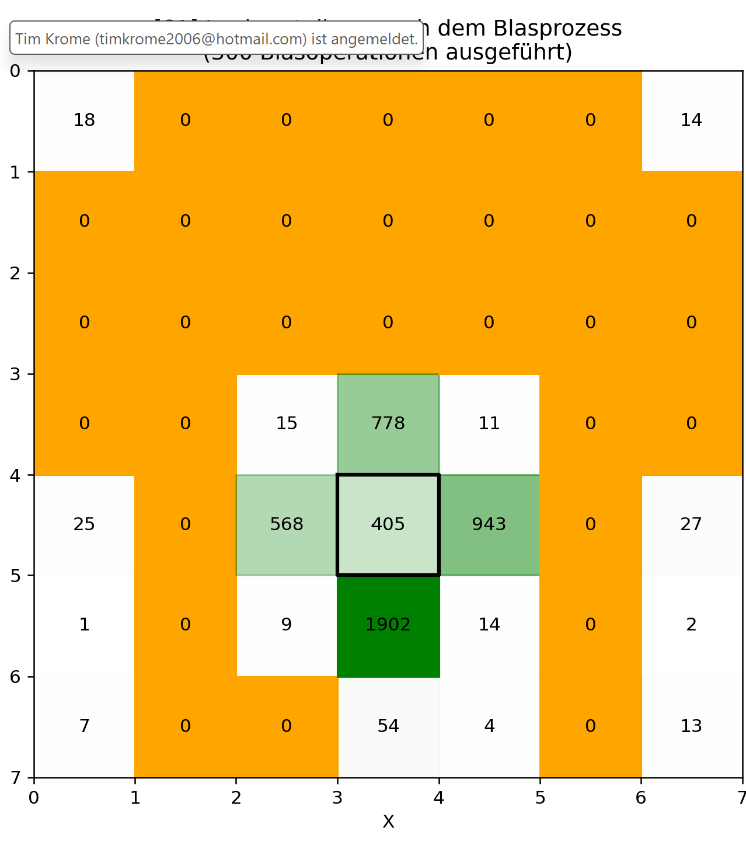
0 | 0 | 15 | 778 | 11 | 0 | 0

25 | 0 | 568 | 405 | 943 | 0 | 27

1 | 0 | 9 | 1902 | 14 | 0 | 2

7 | 0 | 0 | 54 | 4 | 0 | 13

**Ausgabeplots:**



**Beobachtung:**

In diesem Beispiel wurden die Parameter so eingestellt, das maximal 300 Operationen durchgeführt werden. Das Gewicht für die Distanzheuristik wurde auf 0.1 und das für die Varianzheuristik auf 0.9 gestellt. Das Ergebnis ist, das in nur 300 Schritten deutlich weniger Laub auf Feld Q gebracht wurde, das übrige Laub aber sehr geordnet um Feld Q herum angeordnet ist und bei Durchführung der entsprechenden Blasoperationen sehr schnell auf Feld Q gebracht werden könnte. (Da die Gewichtung der Distanzheuristik deutlich niedriger als die Gewichtung der Varianzheuristik ist, geschieht dies aber nicht).

## Ansatz 2 (aufgabe2\_solver.py)

### Beispiel 1

**Eingabeparameter:**

Q = (2,2)

hof\_size = (5,5)

use\_binomial = True  
tolerated\_amount = 5

max\_muster\_operations = 5000

startwert = 100

choose\_faster\_path = True

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 1686

Anteil der Blätter auf Q an der Gesamtlaubmenge: 98.24 %

Laubverteilung nach dem Blasprozess:

5 | 5 | 2 | 5 | 5

0 | 0 | 5 | 1 | 0

0 | 5 | 2456 | 5 | 0

0 | 0 | 0 | 0 | 0

2 | 0 | 0 | 0 | 4

GENERALISIERTER ABLAUFPLAN:

[Phase 1: Unterste Reihe entlauben]

blase(Feld0: (4, 4), nach: links)

blase(Feld0: (3, 4), nach: links)

blase(Feld0: (2, 4), nach: links)

blase(Feld0: (1, 4), nach: links)

Muster(Sourcefelder: [[(4, 4)]]) {

blase(Feld0: (3, 4), nach: rechts)

}

Muster(Sourcefelder: [[(0, 4)]]) {

blase(Feld0: (1, 4), nach: links)

}

[Phase 2: Laub auf oberste Reihe blasen]

blase(Feld0: (4, 4), nach: oben)

blase(Feld0: (3, 4), nach: oben)

blase(Feld0: (1, 4), nach: oben)

blase(Feld0: (0, 4), nach: oben)

blase(Feld0: (4, 3), nach: oben)

blase(Feld0: (3, 3), nach: oben)

blase(Feld0: (1, 3), nach: oben)

blase(Feld0: (0, 3), nach: oben)

blase(Feld0: (4, 2), nach: oben)

blase(Feld0: (3, 2), nach: oben)

blase(Feld0: (1, 2), nach: oben)

blase(Feld0: (0, 2), nach: oben)

[Phase 3: Laub auf oberster Reihe konzentrieren]

Muster(Sourcefelder: [[(0, 0)]]) {

blase(Feld0: (0, 1), nach: oben)

}

Muster(Sourcefelder: [[(4, 0)]]) {

blase(Feld0: (4, 1), nach: oben)

}

[Phase 4: Laub nach Feld Q transferieren]

Muster(Sourcefelder: [[(2, 0), (3, 0), (1, 0)]]) {

blase(Feld0: (2, 1), nach: oben)

blase(Feld0: (4, 0), nach: links)

blase(Feld0: (0, 0), nach: rechts)

}

blase(Feld0: (2, 0), nach: unten)

blase(Feld0: (0, 2), nach: rechts)

blase(Feld0: (2, 4), nach: oben)

Muster(Sourcefelder: [[(1, 2)]]) {

blase(Feld0: (1, 4), nach: oben)

}

Muster(Sourcefelder: [[(3, 2)]]) {

blase(Feld0: (3, 4), nach: oben)

}

Muster(Sourcefelder: [[(1, 1), (2, 1), (3, 1), (2, 0)]]) {

Muster(Sourcefelder: [[(1, 1), (2, 1), (3, 1)]]) {

blase(Feld0: (0, 1), nach: rechts)

blase(Feld0: (4, 1), nach: links)

}

Muster(Sourcefelder: [[(2, 0), (3, 0), (1, 0)]]) {

blase(Feld0: (2, 1), nach: oben)

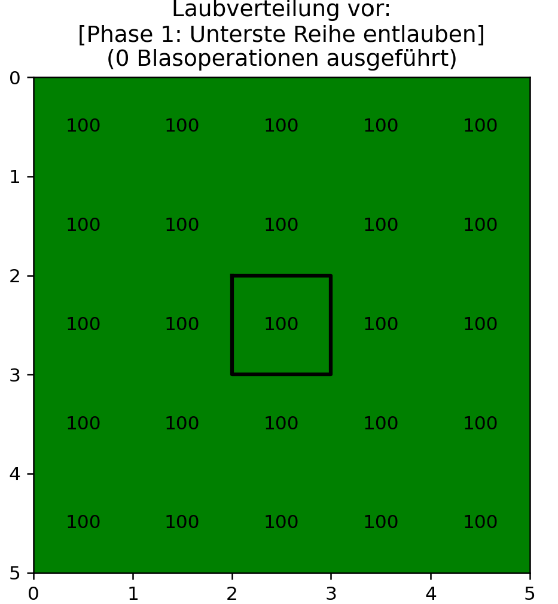
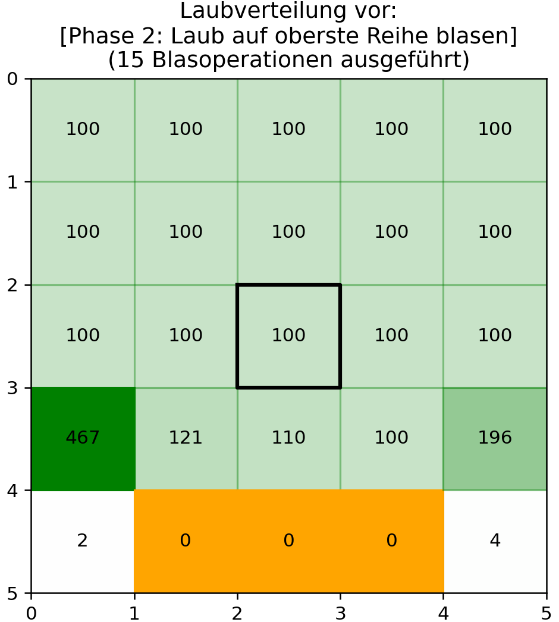
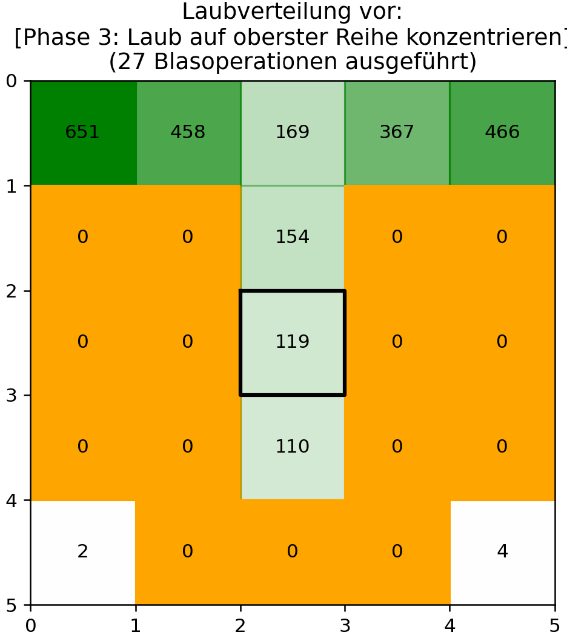
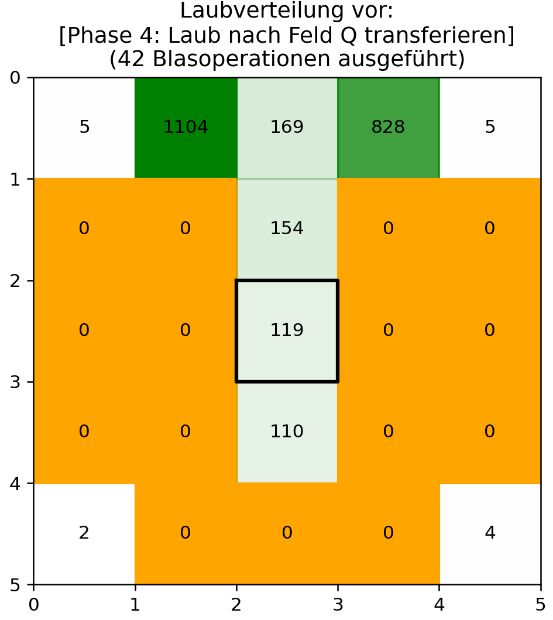
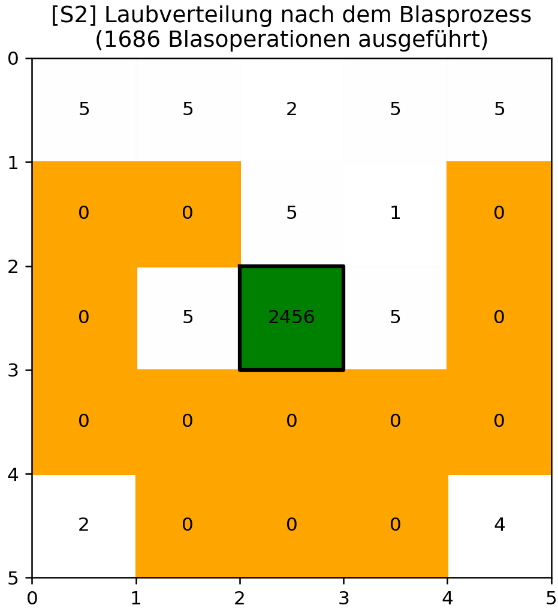
blase(Feld0: (4, 0), nach: links)

blase(Feld0: (0, 0), nach: rechts)

}

}

**Ausgabeplots:**

*Anmerkung: In späteren Beispielen wird teilweise nur noch der letzte Ausgabeplot, der den Zustand des Hofs nach Abschluss des Blasprozesses darstellt, abgebildet. Außerdem wird der generalisierte Programmablaufplan aus der Prorammausgabe herausgekürzt.*

**Beobachtung:**

Wie man sieht, gelingt es dem Programm, fast das gesamte Laub auf Feld Q zu versammeln. Es braucht hierfür sehr viele Schritte, die fast alle in der letzten Phase ausgeführt werden. Dies ist dadurch begründet, dass ein Hof der Größe (5,5) mit Q = (2,2) ein Sonderfall ist, der wie zuvor erläutert besonders kompliziert zu lösen ist.

### Beispiel 2

**Eingabeparameter:**

Q = (1,3)

hof\_size = (5,5)

use\_binomial = True  
tolerated\_amount = 5

max\_muster\_operations = 5000

startwert = 100

choose\_faster\_path = True

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 193

Anteil der Blätter auf Q an der Gesamtlaubmenge: 99.24 %

Laubverteilung nach dem Blasprozess:

1 | 3 | 2 | 4 | 3

0 | 0 | 2481 | 0 | 0

0 | 0 | 0 | 0 | 0

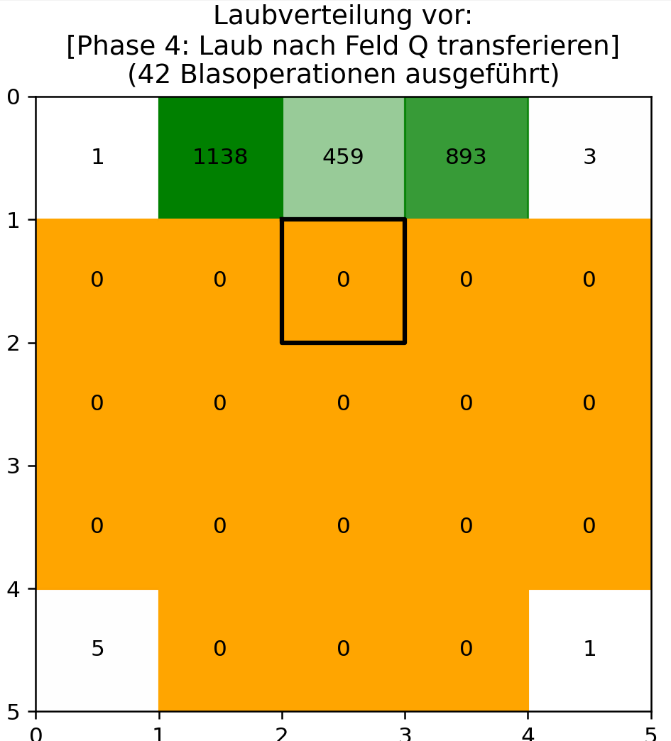
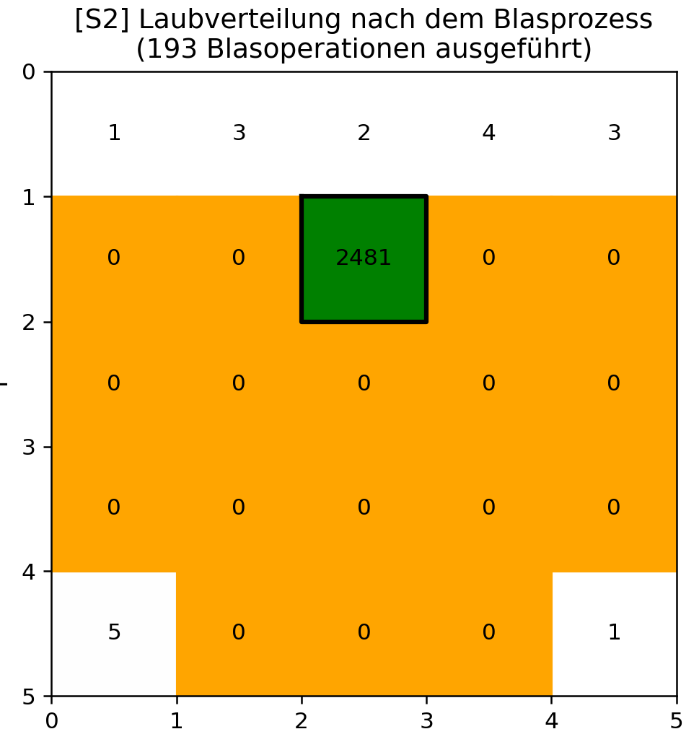
0 | 0 | 0 | 0 | 0

5 | 0 | 0 | 0 | 1

GENERALISIERTER ABLAUFPLAN:

(...)

**Ausgabeplots:**

**** ****

**Beobachtung:**

Wie man sieht, nimmt Phase 4 deutlich weniger Blasoperationen weniger in Anspruch, wenn Feld Q nicht mehr das Feld in der Mitte, sondern ein Randfeld ist.

### Beispiel 3

**Eingabeparameter:**

Q = (2,2)

hof\_size = (6,6)

use\_binomial = True  
tolerated\_amount = 5

max\_muster\_operations = 5000

startwert = 100

choose\_faster\_path = True

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 402

Anteil der Blätter auf Q an der Gesamtlaubmenge: 98.80555555555556 %

Laubverteilung nach dem Blasprozess:

5 | 4 | 0 | 5 | 0 | 5

0 | 0 | 0 | 1 | 0 | 0

0 | 5 | 3557 | 5 | 0 | 0

0 | 0 | 5 | 1 | 0 | 0

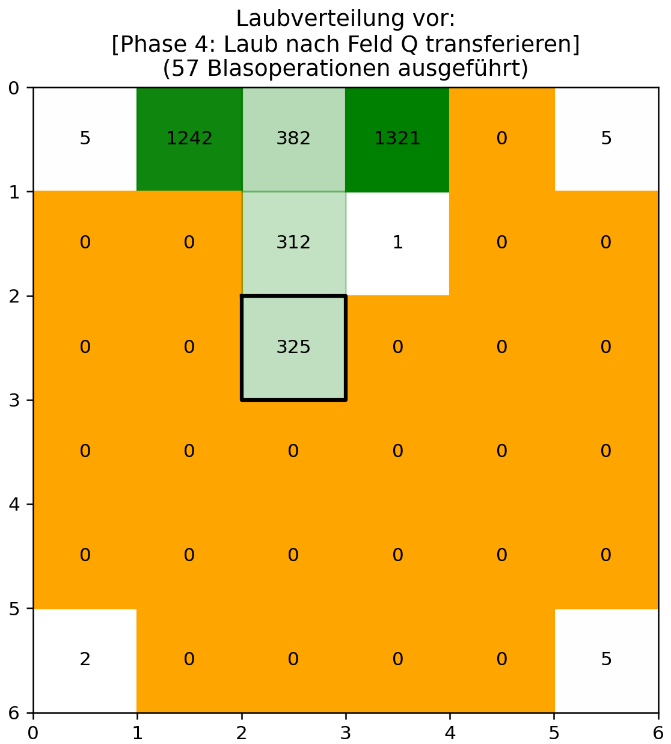
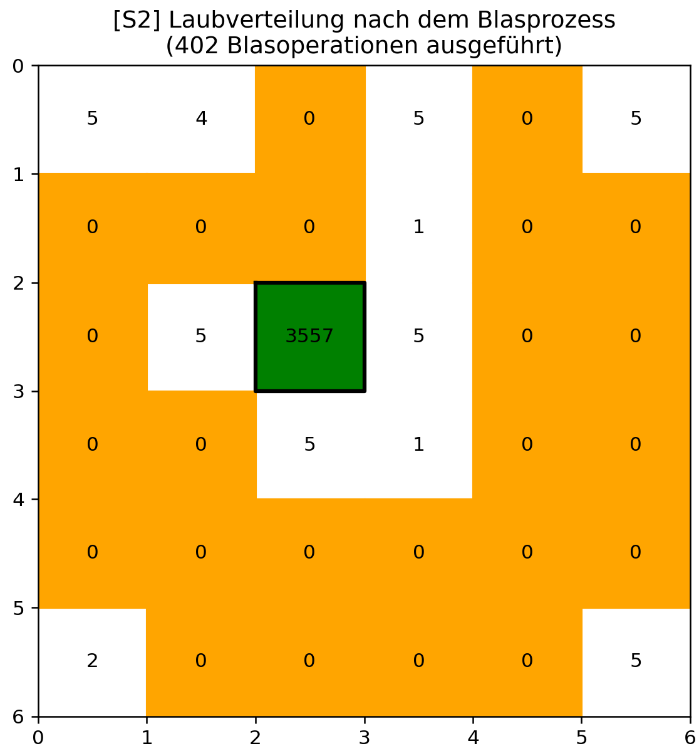
0 | 0 | 0 | 0 | 0 | 0

2 | 0 | 0 | 0 | 0 | 5

GENERALISIERTER ABLAUFPLAN:

(...)

**Ausgabeplots:**

**** ****

**Beobachtung:**

Im Vergleich zu Beispiel 1, wo ein Hof der Größe (5,5) aufgeräumt wurde, nimmt der Blasprozess bei einem Hof der Größe (6,6) deutlich weniger Blasoperationen in Anspruch. Dies ist ebenfalls darauf zurückzuführen, das der Hof (5,5) mit Q = (2,2) ein besonders schwerer Sonderfall ist.

### Beispiel 4

**Eingabeparameter:**

Q = (2,2)

hof\_size = (6,6)

use\_binomial = True  
tolerated\_amount = 5

max\_muster\_operations = 5000

startwert = 5000

choose\_faster\_path = True

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

5000 | 5000 | 5000 | 5000 | 5000 | 5000

5000 | 5000 | 5000 | 5000 | 5000 | 5000

5000 | 5000 | 5000 | 5000 | 5000 | 5000

5000 | 5000 | 5000 | 5000 | 5000 | 5000

5000 | 5000 | 5000 | 5000 | 5000 | 5000

5000 | 5000 | 5000 | 5000 | 5000 | 5000

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 621

Anteil der Blätter auf Q an der Gesamtlaubmenge: 99.97722222222222 %

Laubverteilung nach dem Blasprozess:

2 | 4 | 1 | 5 | 0 | 4

0 | 0 | 0 | 3 | 0 | 0

0 | 5 | 179959 | 5 | 0 | 0

0 | 1 | 5 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0

2 | 0 | 0 | 0 | 0 | 4

GENERALISIERTER ABLAUFPLAN:

(...)

**Ausgabeplots:**

Ein Bild, das Text, Screenshot, Rechteck, Diagramm enthält.

Automatisch generierte Beschreibung**Ein Bild, das Text, Screenshot, Diagramm, Rechteck enthält.

Automatisch generierte Beschreibung**

**Ein Bild, das Text, Screenshot, Diagramm, Rechteck enthält.

Automatisch generierte Beschreibung** Ein Bild, das Text, Diagramm, Screenshot, Rechteck enthält.

Automatisch generierte Beschreibung

**Beobachtung:**

Auch bei Höfen, auf denen sich zu Beginn deutlich mehr Blätter auf den einzelnen Feldern befinden (in diesem Beispiel sind es zu Beginn 5 000 Blätter pro Feld), wird dank der logarithmischen Darstellung der Formel von Bernoulli in binomial\_util.py innerhalb weniger Sekunden die Simulation durchgeführt. Im Vergleich zu Beispiel 3, wo sich auf jedem Feld zu Beginn 100 Blätter befanden, braucht das Programm hier ca. 200 Blasoperationen mehr, um den Zielzustand zu erreichen.

### Beispiel 5

**Eingabeparameter:**

Q = (2,2)

hof\_size = (6,6)

use\_binomial = False  
tolerated\_amount = 5

max\_muster\_operations = 5000

startwert = 100

choose\_faster\_path = True

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: Erwartungswert-basiert

Laubverteilung vor dem Blasprozess:

100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0

100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0

100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0

100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0

100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0

100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 437

Anteil der Blätter auf Q an der Gesamtlaubmenge: 98.81743334969002 %

Laubverteilung nach dem Blasprozess:

4.6982 | 4.50521 | 4.925 | 0.0 | 0.0 | 3.2743

0.0 | 0.0 | 0.0 | 1.78055 | 0.0 | 0.0

0.0 | 4.82349 | 3557.4276| 4.82349 | 0.0 | 0.0

0.0 | 0.5307 | 4.68416 | 0.01131 | 0.0 | 0.0

0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0

4.52761 | 0.0 | 0.0 | 0.0 | 0.0 | 3.98838

GENERALISIERTER ABLAUFPLAN:

(...)

**Ausgabeplots:**

**Ein Bild, das Text, Screenshot, Diagramm, Zahl enthält.

Automatisch generierte Beschreibung** Ein Bild, das Text, Screenshot, Diagramm, Rechteck enthält.

Automatisch generierte Beschreibung

**Beobachtung:**

In diesem Beispiel wurden die Erwartungswerte zur Wahrscheinlichkeitsmodellierung verwendet. Dies sorgt dafür, dass gut sichtbar ist, welche Felder vollständig und welche nur asymptotisch geleert werden. An den im Plot angezeigten Zahlen lässt es sich zwar nicht immerzu ablesen, da diese gerundet werden. Allerdings werden nur die Felder, die vollständig leer sind, orange eingefärbt, die asymptotisch geleerten Felder sind weiß eingefärbt.

### Beispiel 6

**Eingabeparameter:**

Q = (1,1)

hof\_size = (3,4)

use\_binomial = True  
tolerated\_amount = 5

max\_muster\_operations = 5000

startwert = 100

choose\_faster\_path = True

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100

100 | 100 | 100

100 | 100 | 100

100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 553

Anteil der Blätter auf Q an der Gesamtlaubmenge: 98.5 %

Laubverteilung nach dem Blasprozess:

3 | 0 | 5

0 | 1182 | 1

0 | 0 | 4

3 | 0 | 2

GENERALISIERTER ABLAUFPLAN:

[Phase 1: Unterste Reihe entlauben]

blase(Feld0: (0, 3), nach: oben)

blase(Feld0: (0, 2), nach: oben)

blase(Feld0: (0, 1), nach: oben)

Muster(Sourcefelder: [[(0, 3)]]) {

blase(Feld0: (0, 2), nach: unten)

}

Muster(Sourcefelder: [[(0, 0)]]) {

blase(Feld0: (0, 1), nach: oben)

}

[Phase 2: Laub auf oberste Reihe blasen]

blase(Feld0: (0, 3), nach: rechts)

blase(Feld0: (0, 2), nach: rechts)

blase(Feld0: (0, 1), nach: rechts)

blase(Feld0: (0, 0), nach: rechts)

[Phase 3: Laub auf oberster Reihe konzentrieren]

Muster(Sourcefelder: [[(2, 3)]]) {

blase(Feld0: (1, 3), nach: rechts)

}

[Phase 4: Laub nach Feld Q transferieren]

Muster(Sourcefelder: [[(2, 1), (2, 2), (2, 0)]]) {

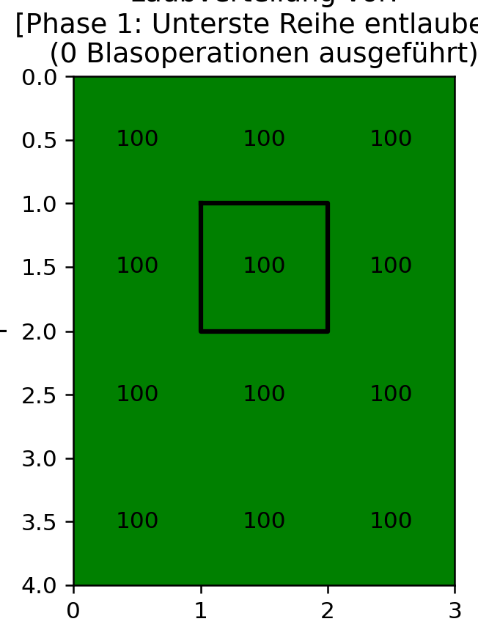
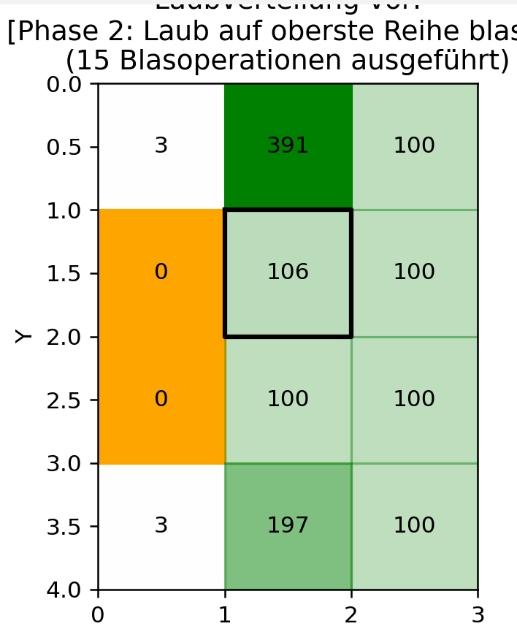
blase(Feld0: (1, 0), nach: rechts)

blase(Feld0: (1, 1), nach: rechts)

blase(Feld0: (2, 3), nach: oben)

}

**Ausgabeplots:**

****  Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Screenshot, Diagramm, Reihe enthält.

Automatisch generierte Beschreibung

**Beobachtung:**

Auch bei sehr kleinen Höfen, die noch dazu unterschiedliche Seitenlängen haben, funktioniert das Programm einwandfrei.

### Beispiel 7

**Eingabeparameter:**

Q = (8,9)

hof\_size = (15,19)

use\_binomial = False  
tolerated\_amount = 5

max\_muster\_operations = 5000

startwert = 100

choose\_faster\_path = True

**Ausgabe in der Konsole:**

Ausgabe in der Konsole:

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 981

Anteil der Blätter auf Q an der Gesamtlaubmenge: 98.78947368421052 %

Laubverteilung nach dem Blasprozess:

4 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 2 | 4 | 0 | 3 | 0 | 0 | 2

0 | 0 | 3 | 5 | 4 | 5 | 1 | 1 | 0 | 3 | 0 | 1 | 5 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 5 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 122 | 0 | 147 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 5 | 2 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 28155 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

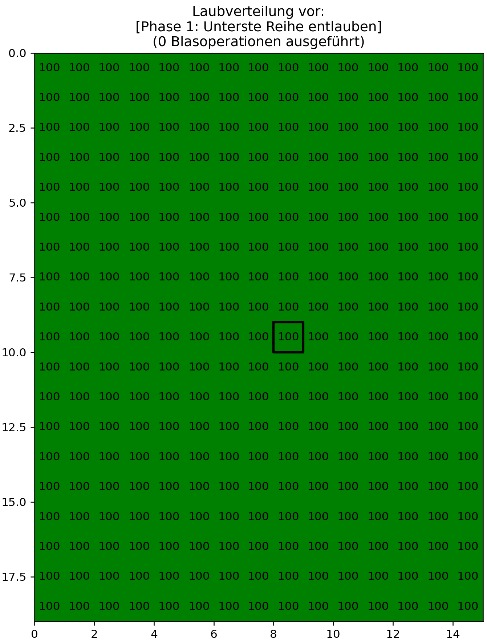
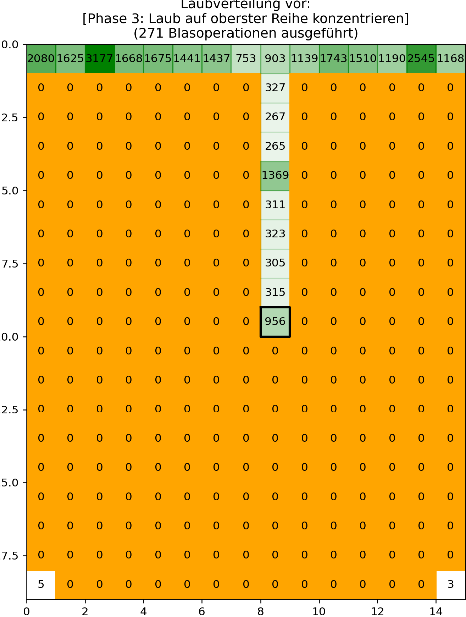
0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

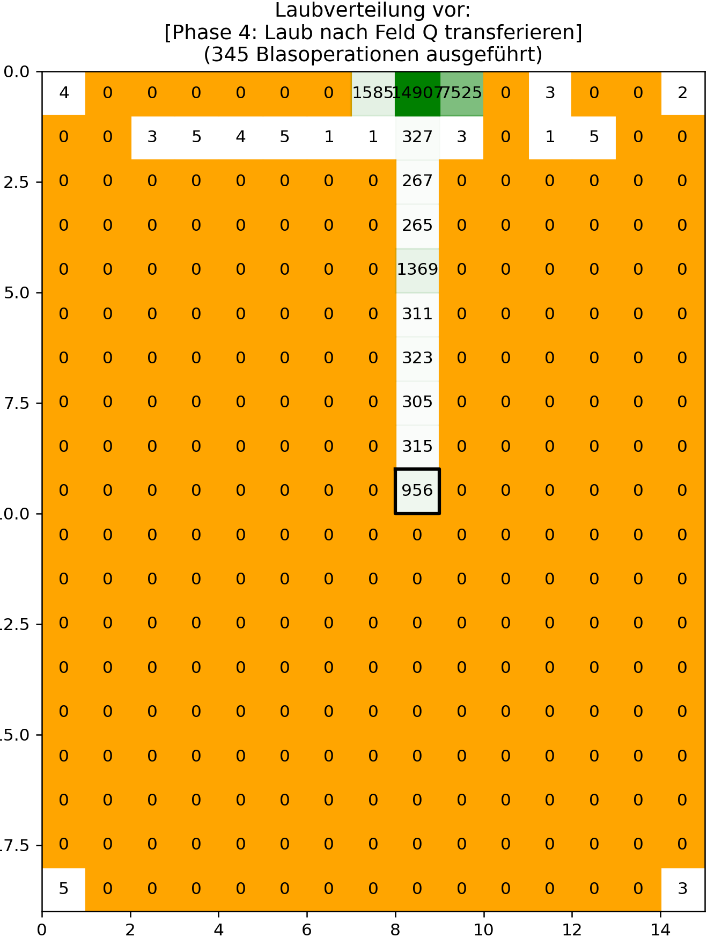
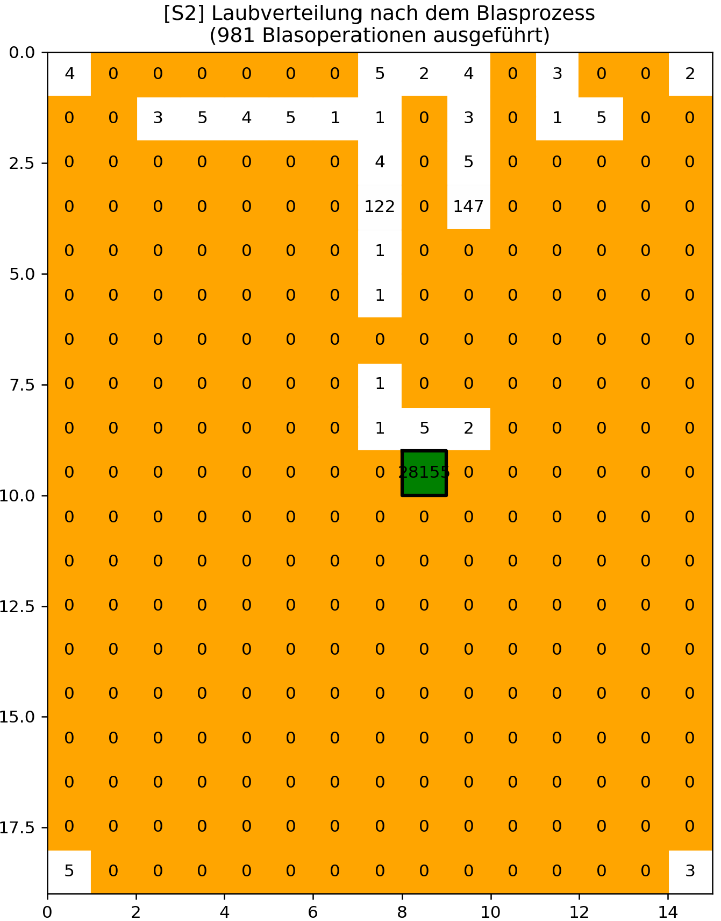
5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

GENERALISIERTER ABLAUFPLAN:

(...)

**Ausgabeplots:**

****  

**** ****

**Beobachtung:**

Auch für große Höfe können trotzdem innerhalb von weniger als einer Sekunde sehr gute Ergebnisse gefunden werden. All dies macht deutlich, dass Ansatz 2 sowohl in Hinblick auf die Laufzeit als auch in Hinblick auf die Qualität weit besser als Ansatz 1 ist, der einen Hof dieser Größe gar nicht mehr lösen könnte.

## Ansatz 3 (aufgabe3\_solver.py)

### Beispiel 1

**Eingabeparameter:**

Q = (2,2)

hof\_size = (5,5)

use\_binomial = True  
binomial\_rank = “random”  
startwert = 100

weight\_avg = 0.5

weight\_varianz = 0.5

max\_operations = 1000

satisfied\_constraint = 0.8

max\_operations = 1000

tolerated\_amount = 5

clear\_edges = True

max\_muster\_operations = 1000

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 294

Anteil der Blätter auf Q an der Gesamtlaubmenge: 80.0 %

Laubverteilung nach dem Blasprozess:

3 | 0 | 117 | 0 | 2

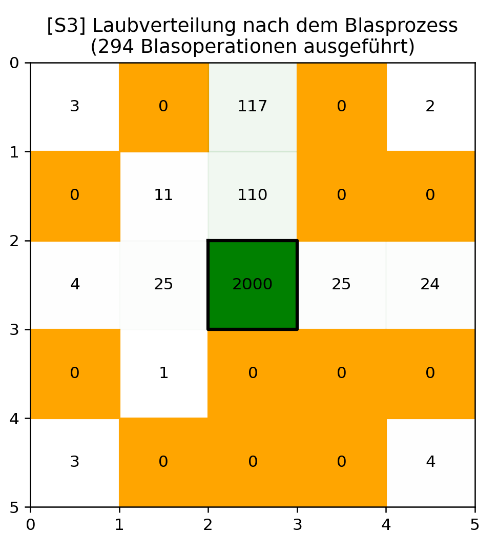
0 | 11 | 110 | 0 | 0

4 | 25 | 2000 | 25 | 24

0 | 1 | 0 | 0 | 0

3 | 0 | 0 | 0 | 4

**Ausgabeplots:**



**Beobachtung:**

Wie man sieht, kann Ansatz 3 etwas mehr Blätter als Ansatz 1 auf Feld Q befördern – das Ergebnis ist dennoch deutlich schlechter als das von Ansatz 2.

### Beispiel 2

**Eingabeparameter:**

Q = (1,1)

hof\_size = (4,4)

use\_binomial = True  
binomial\_rank = “random”  
startwert = 100

weight\_avg = 0.5

weight\_varianz = 0.5

max\_operations = 1000

satisfied\_constraint = 0.8

max\_operations = 1000

tolerated\_amount = 5

clear\_edges = True

max\_muster\_operations = 1000

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100 | 100

100 | 100 | 100 | 100

100 | 100 | 100 | 100

100 | 100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 467

Anteil der Blätter auf Q an der Gesamtlaubmenge: 80.0625 %

Laubverteilung nach dem Blasprozess:

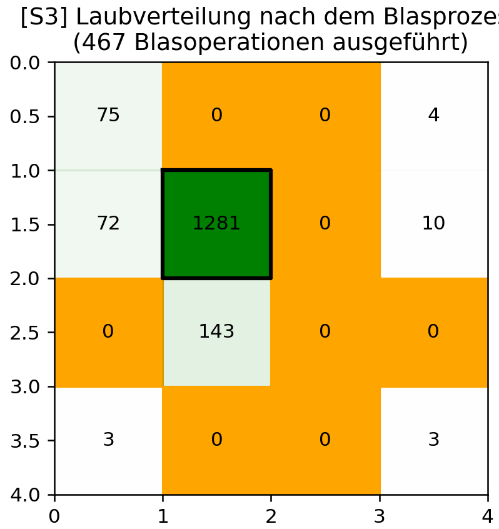
75 | 0 | 0 | 4

72 | 1281 | 0 | 10

0 | 143 | 0 | 0

3 | 0 | 0 | 3

**Ausgabeplots:**



### Beispiel 3

**Eingabeparameter:**

Q = (3,3)

hof\_size = (4,4)

use\_binomial = True  
binomial\_rank = “random”  
startwert = 100

weight\_avg = 0.5

weight\_varianz = 0.5

max\_operations = 1000

satisfied\_constraint = 0.3

max\_operations = 1000

tolerated\_amount = 5

clear\_edges = True

max\_muster\_operations = 1000

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 288

Anteil der Blätter auf Q an der Gesamtlaubmenge: 37.5 %

Laubverteilung nach dem Blasprozess:

2 | 0 | 0 | 9 | 0 | 5

0 | 0 | 0 | 0 | 0 | 0

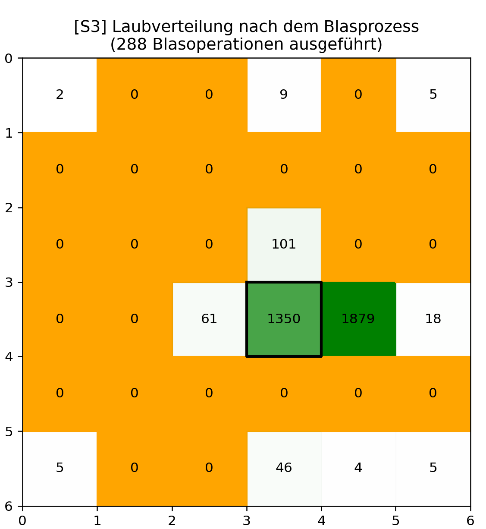
0 | 0 | 0 | 101 | 0 | 0

0 | 0 | 61 | 1350 | 1879 | 18

0 | 0 | 0 | 0 | 0 | 0

5 | 0 | 0 | 46 | 4 | 5

**Ausgabeplots:**



# Beispiele – Erweiterung 1: Keine Mauer-Umrandung

Zum Demonstrieren der Funktionstüchtigkeit des Programms, das diese Erweiterung implementiert (Aufgabe 1 E1/aufgabe1\_solver2.py) habe ich es ebenfalls mit verschiedenen Parametern ausgeführt. Die vollständigen Beispielausgaben sowie die vom Programm erstellten Dateien, die die durchgeführten Blasoperationen enthalten, sind im Ordner *Aufgabe 1 E1* zu finden.

Im Ordner Aufgabe 1 E1/Outputs sind für jedes Beispiel die Blasoperationen zu finden, die die Programme bei der Simulation durchgeführt haben. Sie sind in Textdateien gespeichert. Dabei sind die Blasoperationen, die von Beispiel x durchgeführt wurden, in der Textdatei „output\_E1\_{beispielx}.txt“ gespeichert.

## Beispiel 1

**Eingabeparameter:**

Q = (2,2)

hof\_size = (5,5)

use\_binomial = True  
tolerated\_amount = 2

max\_muster\_operations = 500

startwert = 100

choose\_faster\_path = True

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 455

Anteil der Blätter auf Q an der ursprünglichen Gesamtlaubmenge: 30.36 %

Gesamtlaubmenge, die sich noch auf dem Hof befindet: 779 von urspr. 2500

Laubverteilung nach dem Blasprozess:

0 | 1 | 0 | 0 | 0

0 | 0 | 2 | 0 | 0

0 | 8 | 759 | 2 | 2

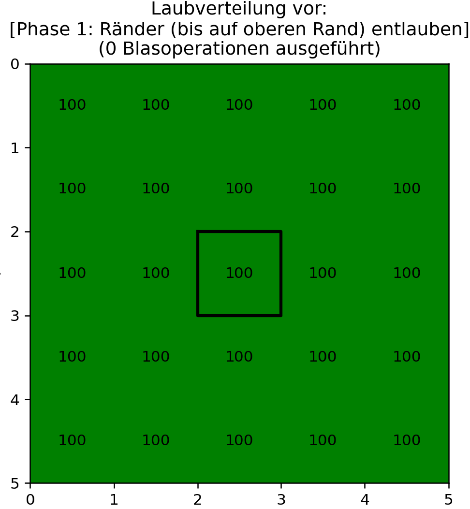
0 | 0 | 0 | 2 | 1

0 | 2 | 0 | 0 | 0

GENERALISIERTER ABLAUFPLAN:

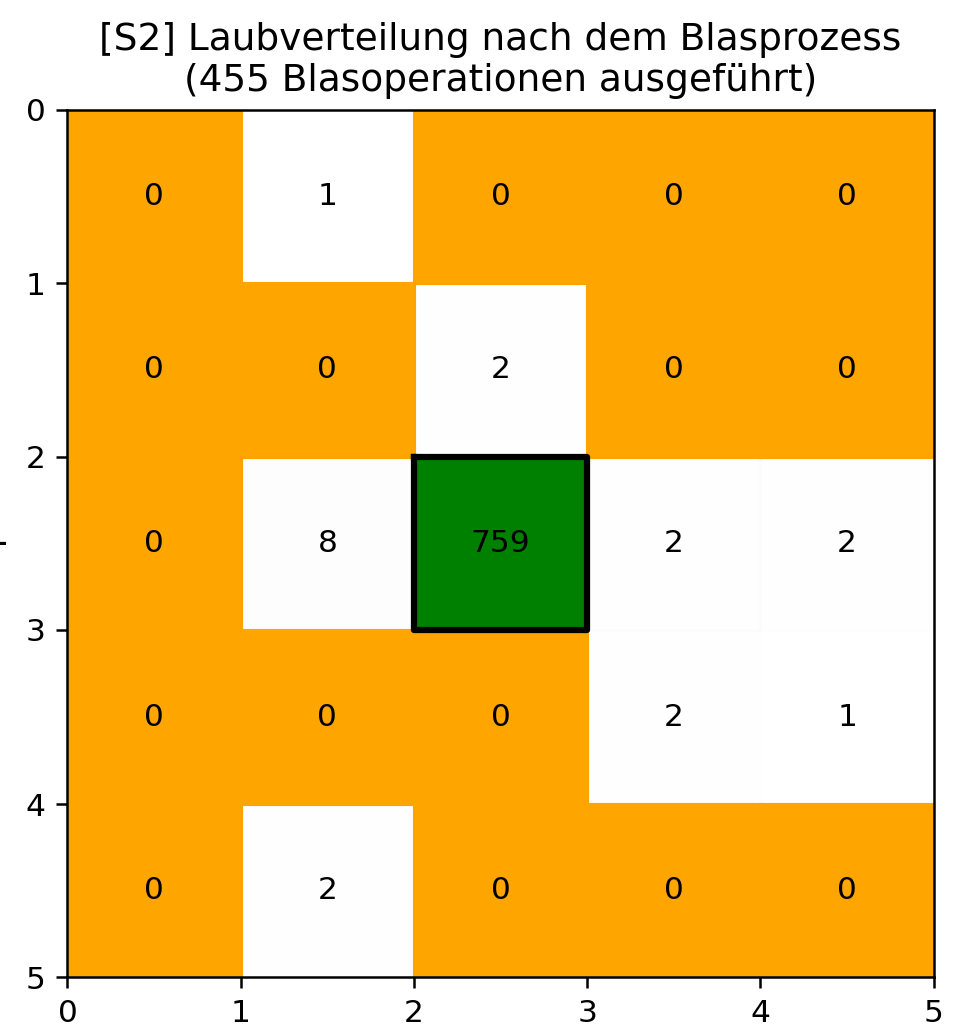
(...)

**Ausgabeplots:**

 Ein Bild, das Text, Screenshot, Diagramm, Quadrat enthält.

Automatisch generierte Beschreibung Ein Bild, das Text, Screenshot, Diagramm, Rechteck enthält.

Automatisch generierte Beschreibung Ein Bild, das Text, Screenshot, Diagramm, Rechteck enthält.

Automatisch generierte Beschreibung 

## Beispiel 2

**Eingabeparameter:**

Q = (4,4)

hof\_size = (10,10)

use\_binomial = True  
tolerated\_amount = 2

max\_muster\_operations = 5000

startwert = 100

choose\_faster\_path = True

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 1153

Anteil der Blätter auf Q an der ursprünglichen Gesamtlaubmenge: 29.38 %

Gesamtlaubmenge, die sich noch auf dem Hof befindet: 2984 von urspr. 10000

Laubverteilung nach dem Blasprozess:

1 | 2 | 0 | 1 | 0 | 2 | 0 | 0 | 1 | 2

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 2938 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

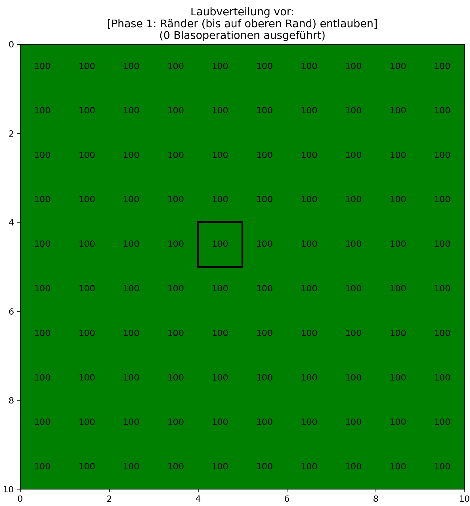
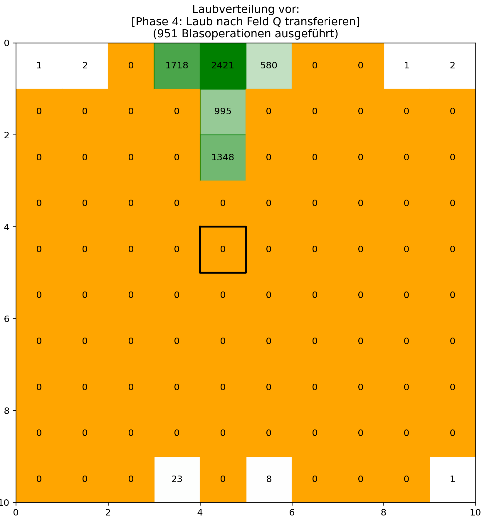
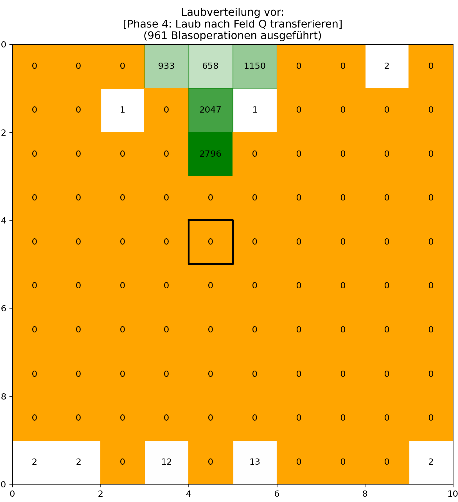
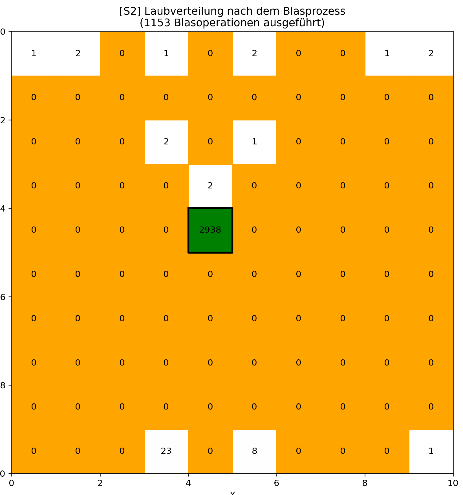
0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 23 | 0 | 8 | 0 | 0 | 0 | 1

GENERALISIERTER ABLAUFPLAN:

(...)

**Ausgabeplots:**

# Beispiele – Erweiterung 2: Zwei Laubbläser

Zum Demonstrieren der Funktionstüchtigkeit des Programms, das diese Erweiterung implementiert (Aufgabe 1 E2/aufgabe1\_solver2.py) habe ich es ebenfalls mit verschiedenen Parametern ausgeführt. Die vollständigen Beispielausgaben sowie die vom Programm erstellten Dateien, die die durchgeführten Blasoperationen enthalten, sind im Ordner *Aufgabe 1 E2* zu finden.

Im Ordner Aufgabe 1 E2/Outputs sind für jedes Beispiel die Blasoperationen zu finden, die die Programme bei der Simulation durchgeführt haben. Sie sind in Textdateien gespeichert. Dabei sind die Blasoperationen, die von Beispiel x durchgeführt wurden, in der Textdatei „output\_E2\_{beispielx}.txt“ gespeichert.

## Beispiel 1

**Eingabeparameter:**

Q = (2,2)

hof\_size = (5,5)

use\_binomial = True  
tolerated\_amount = 5

max\_muster\_operations = 500

startwert = 100

choose\_faster\_path = True

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 59

Anteil der Blätter auf Q an der Gesamtlaubmenge: 96.04 %

Laubverteilung nach dem Blasprozess:

1 | 0 | 3 | 0 | 5

0 | 0 | 5 | 0 | 0

0 | 0 | 2401 | 0 | 0

0 | 0 | 29 | 0 | 0

2 | 50 | 0 | 0 | 4

GENERALISIERTER ABLAUFPLAN:

[Phase 1: Unterste Reihe entlauben]

blase(

BLÄSER1: Feld0: (0, 4), nach: rechts BLÄSER2: Idle-Operation)

blase(

BLÄSER1: Feld0: (1, 4), nach: rechts BLÄSER2: Idle-Operation)

blase(

BLÄSER1: Feld0: (2, 4), nach: rechts BLÄSER2: Idle-Operation)

blase(

BLÄSER1: Feld0: (3, 4), nach: links BLÄSER2: Feld0: (1, 4), nach: rechts)

blase(

BLÄSER1: Feld0: (2, 4), nach: links BLÄSER2: Feld0: (2, 4), nach: rechts)

Muster(Sourcefelder: [[(0, 4), (4, 4)]]) {

blase(

BLÄSER1: Feld0: (1, 4), nach: links BLÄSER2: Feld0: (3, 4), nach: rechts)

}

[Phase 2: Laub auf oberste Reihe blasen]

blase(

BLÄSER1: Feld0: (4, 4), nach: oben BLÄSER2: Feld0: (3, 4), nach: oben)

blase(

BLÄSER1: Feld0: (1, 4), nach: oben BLÄSER2: Feld0: (0, 4), nach: oben)

blase(

BLÄSER1: Feld0: (4, 3), nach: oben BLÄSER2: Feld0: (3, 3), nach: oben)

blase(

BLÄSER1: Feld0: (1, 3), nach: oben BLÄSER2: Feld0: (0, 3), nach: oben)

blase(

BLÄSER1: Feld0: (4, 2), nach: oben BLÄSER2: Feld0: (3, 2), nach: oben)

blase(

BLÄSER1: Feld0: (1, 2), nach: oben BLÄSER2: Feld0: (0, 2), nach: oben)

[Phase 3: Laub auf oberster Reihe konzentrieren]

Muster(Sourcefelder: [[(0, 0), (4, 0)]]) {

blase(

BLÄSER1: Feld0: (0, 1), nach: oben BLÄSER2: Feld0: (4, 1), nach: oben)

}

blase(

BLÄSER1: Feld0: (4, 0), nach: links BLÄSER2: Feld0: (0, 0), nach: rechts)

[Phase 4: Laub nach Feld Q transferieren]

Muster(Sourcefelder: [[(2, 0), (2, 1)]]) {

Muster(Sourcefelder: [[(2, 0)]]) {

blase(

BLÄSER1: Feld0: (1, 0), nach: rechts BLÄSER2: Feld0: (3, 0), nach: links)

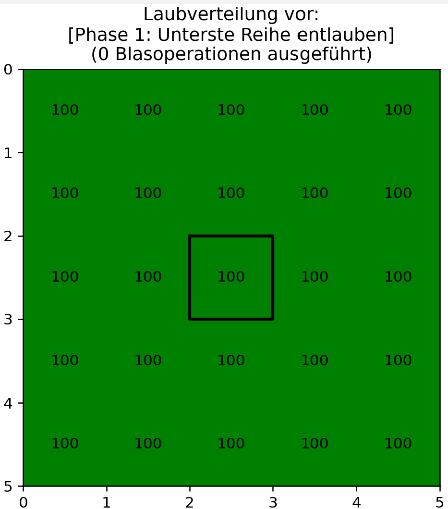
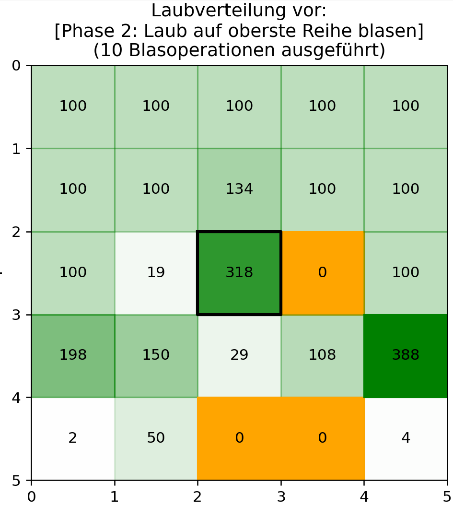
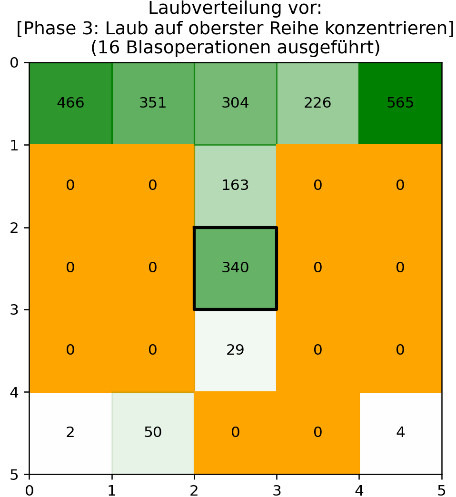
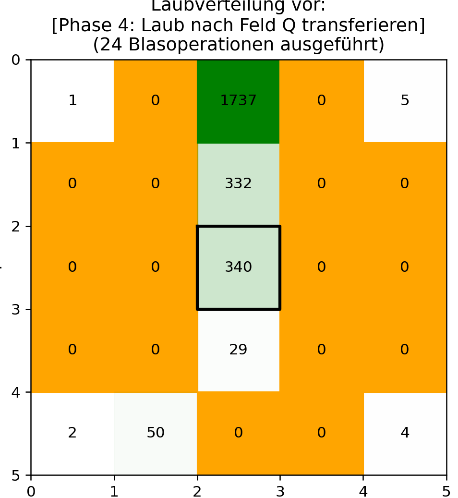
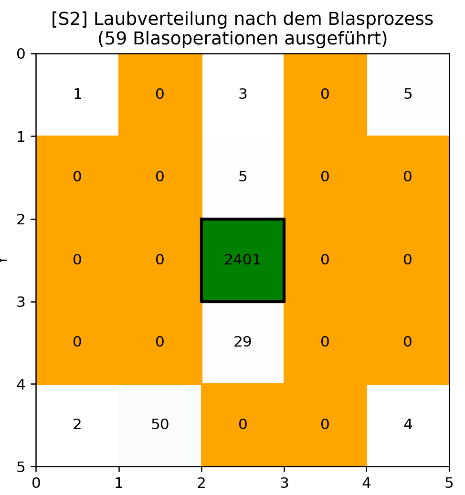
}

blase(

BLÄSER1: Feld0: (1, 1), nach: rechts BLÄSER2: Feld0: (3, 1), nach: links)

}

**Ausgabeplots:**

## Beispiel 2

**Eingabeparameter:**

Q = (5,7)

hof\_size = (15,16)

use\_binomial = True  
tolerated\_amount = 5

max\_muster\_operations = 5000

startwert = 100

choose\_faster\_path = True

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 190

Anteil der Blätter auf Q an der Gesamtlaubmenge: 99.90416666666667 %

Laubverteilung nach dem Blasprozess:

5 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 23977 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

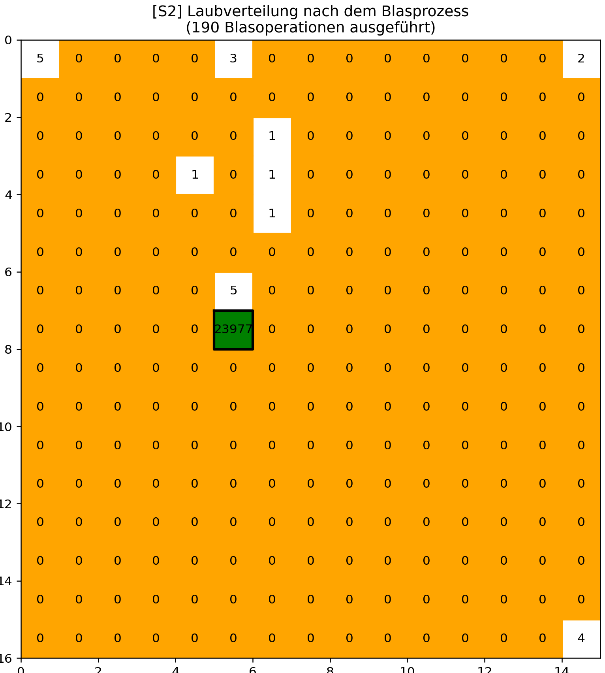
0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4

GENERALISIERTER ABLAUFPLAN:

(...)

**Ausgabeplots:**



# Beispiele – Erweiterung 3: Verschiedene Laubtypen

Zum Demonstrieren der Funktionstüchtigkeit des Programms, das diese Erweiterung implementiert (Aufgabe 1 E3/aufgabe1\_solver2.py) habe ich es ebenfalls mit verschiedenen Parametern ausgeführt. Die vollständigen Beispielausgaben sowie die vom Programm erstellten Dateien, die die durchgeführten Blasoperationen enthalten, sind im Ordner *Aufgabe 1 E3* zu finden.

Im Ordner Aufgabe 1 E3/Outputs sind für jedes Beispiel die Blasoperationen zu finden, die die Programme bei der Simulation durchgeführt haben. Sie sind in Textdateien gespeichert. Dabei sind die Blasoperationen, die von Beispiel x durchgeführt wurden, in der Textdatei „output\_E3\_{beispielx}.txt“ gespeichert.

## Beispiel 1

**Eingabeparameter:**

Q = (2,2)

hof\_size = (5,5)

use\_binomial = True  
tolerated\_amount = 5

max\_muster\_operations = 500

startwert = 100

choose\_faster\_path = True

Regeln, die für den zweiten Laubtyp gelten: Rules(

use\_binomial=use\_binomial, binomial\_rank="random“,

A\_seitenabtrieb = 0.3, B\_vorne\_abtrieb = 0.3, A\_noB\_seitenabtrieb=0.5\*1, s1=0.7, s4=0)  
Für den ersten Laubtyp gelten die Standardregeln.

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 59

Anteil der Blätter auf Q an der Gesamtlaubmenge: 96.04 %

Laubverteilung nach dem Blasprozess:

1 | 0 | 3 | 0 | 5

0 | 0 | 5 | 0 | 0

0 | 0 | 2401 | 0 | 0

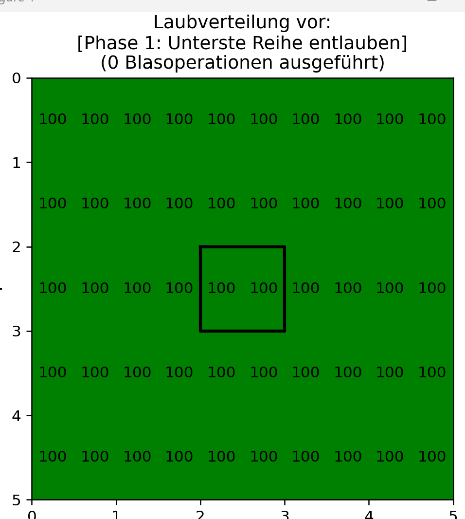
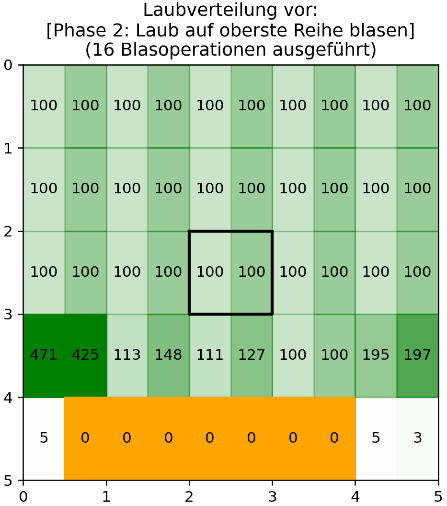
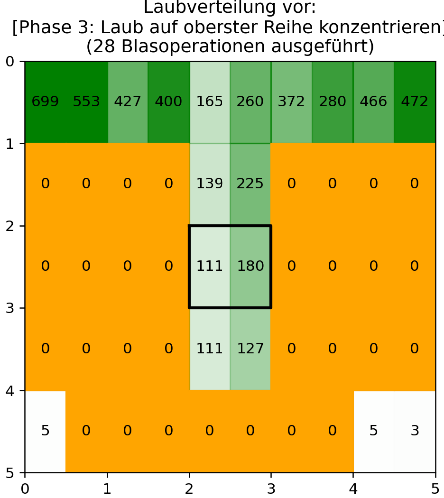
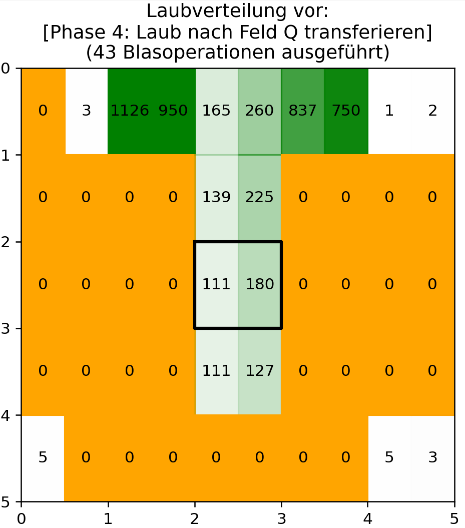
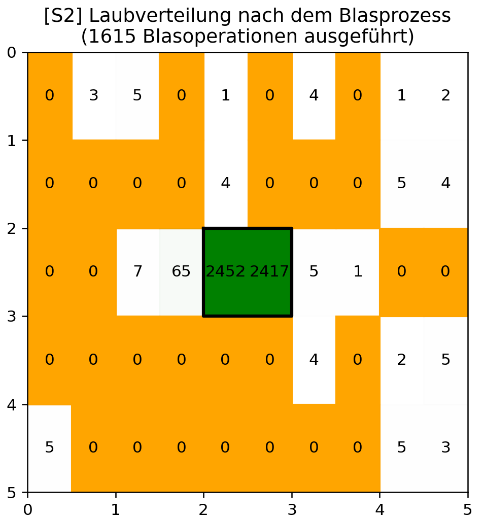
0 | 0 | 29 | 0 | 0

2 | 50 | 0 | 0 | 4

GENERALISIERTER ABLAUFPLAN:

(...)

**Ausgabeplots:**

## Beispiel 2

**Eingabeparameter:**

Q = (1,2)

hof\_size = (8,4)

use\_binomial = True  
tolerated\_amount = 5

max\_muster\_operations = 500

startwert = 100

choose\_faster\_path = True

Regeln, die für den zweiten Laubtyp gelten: Rules(

use\_binomial=use\_binomial, binomial\_rank="random“,

A\_seitenabtrieb = 0.3, B\_vorne\_abtrieb = 0.3, A\_noB\_seitenabtrieb=0.5\*1, s1=0.7, s4=0)  
Für den ersten Laubtyp gelten die Standardregeln.

**Ausgabe in der Konsole:**

AUSGANGSPUNKT:

Verwendete Wahrscheinlichkeitsmodellierung: binomial

Laubverteilung vor dem Blasprozess:

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

Führe Simulation durch ...

ERGEBNIS DER SIMULATION:

Ausgeführte Blasoperationen: 190

Anteil der Blätter auf Q an der Gesamtlaubmenge: 99.90416666666667 %

Laubverteilung nach dem Blasprozess:

5 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 23977 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

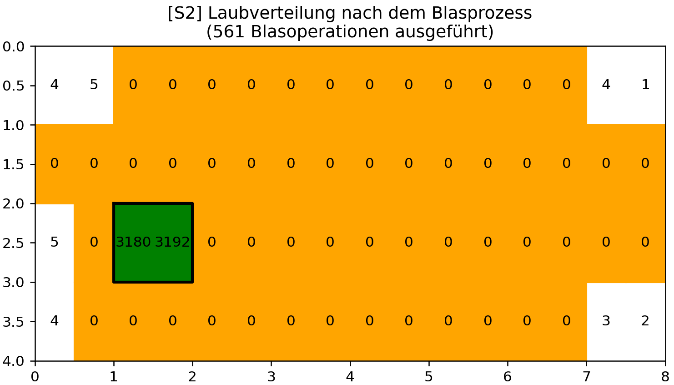
0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4

GENERALISIERTER ABLAUFPLAN:

(...)

**Ausgabeplots:**



# Quellcode

Im Folgenden einige Auszüge aus dem Quellcode meiner Implementierung.

## Auszüge aus binomial\_util.py

In binomial\_util.py sind Wahrscheinlichkeitsfunktionen definiert, die von allen Ansätzen / Erweiterungen verwendet werden.

def binomialpdf(\*, n, p, k):

"""

Returns:

float: P(X=k) mit den Parametern n und p und einer   
 binomialverteilte Größe X

"""

# Diese Sonderfälle würden wegen der logarithmischen Implementierung der Benoulli-Formel für P(X=k) einen DomainError erzeugen und werden daher seperat abgehandelt:

if p == 1:

return n == k

elif p == 0:

return n == 0 # Berechneter Wert wird gecached:

bincomb = math.comb(n, k)

if bincomb == 0:

return 0 # Dieser Fall tritt auf, wenn k > n oder k < 0

log\_binom = math.log(bincomb) + k \* math.log(p) + (n - k) \* math.log(1 - p)

return math.exp(log\_binom)

def binomialdist(\*, n, p, relevant\_threshold=0.01):

"""

Returns:

list: P(x=k) für alle k und den Parametern n und p und einer binomialverteilten Größe X. Der Index in der Liste korrespondiert mit dem jeweiligen k-Wert

"""

dist = np.zeros(n+1) # Mit Nullen gefüllte Liste erzugen

for k in range(math.floor(n\*p),-1,-1):

pdf = binomialpdf(n=n, p=p, k=k)

dist[k] = pdf

if pdf < relevant\_threshold:

break

for k in range(math.ceil(n\*p),n+1,1):

pdf = binomialpdf(n=n, p=p, k=k)

dist[k] = pdf

if pdf < relevant\_threshold: # der restliche Bereich ist   
 vernachlässigbar, da die   
 Wahrscheinlichkeiten verschwinden   
 gering werden

break

if len(dist[dist==0]) != 0:

fill\_rest = (1- sum(dist)) / len(dist[dist==0])

dist[dist==0] = fill\_rest

return dist

def binomial\_likeliest(\*, n, p, rank=0, handle\_ties="higher"):

"""

Bestimmt das k, für das P(X=k) maximal wird (also das k, das am   
 wahrscheinlichsten Eintritt).

Args:

n (int), p (float): Parameter für die Binomialverteilung

rank (int oder str): Gibt an, welches bzw. das wieviel-wahrscheinlichste k zurückgegeben werden soll. z.B. wird im Fall rank==0 das wahrscheinlichste k zurückgegeben.

Wenn rank == "random", dann wird den tatsächlichen Wahrscheinlichkeiten entsprechend ein k zufällig ausgewählt

handle\_ties (str): Entweder "higher", "lower" oder "random". Gibt an, ob im Falle zweier gleich wahrscheinlicher Fälle (tritt auf wenn p=0.5 und floor(n/2) <= k <= ceil(n/2)) das größere oder kleinere k zurückgegeben werden soll (wenn handle\_ties=="random", dann wird mit 50%-iger Wahrscheinlichkeit zufällig gewählt).

Returns:

int: Das gesuchte k

float: Die Wahrscheinlichkeit, zu der k eintritt, bzw. P(X=k)

"""

if rank == "random":

dist = binomialdist(n=n, p=p)

k = random.choices(population=[i for i in range(n+1)], k=1,   
 weights=dist)[0]

return k, dist[k]  
 (…)

## Auszüge aus hof.py

In hof.py ist eine Rules-Klasse und eineHof-Klasse definiert, die in allen Ansätzen für die Grundaufgabe und in den meisten Erweiterungen unverändert verwendet wird.

class Rules:

"""

Dient zum Speichern der Regeln, die auf einem Hof für den Laubblasprozess   
 gelten.

"""

def \_\_init\_\_(self, \*, A\_seitenabtrieb = 0.1, B\_vorne\_abtrieb = 0.1, A\_noB\_seitenabtrieb=0.5\*0.95, s1=0.9, s4=0.05, use\_binomial=True, binomial\_rank="random", binomial\_handle\_ties="higher"):

assert 0 < A\_seitenabtrieb < 0.5

assert 0 < B\_vorne\_abtrieb < 1

assert 0 < A\_noB\_seitenabtrieb <= 0.5

self.A\_seitenabtrieb = A\_seitenabtrieb

self.B\_vorne\_abtrieb = B\_vorne\_abtrieb

self.s1 = s1

self.s4 = s4

self.A\_noB\_seitenabtrieb = A\_noB\_seitenabtrieb

assert isinstance(binomial\_rank, int) or binomial\_rank == "random"

assert binomial\_handle\_ties == "higher" or binomial\_handle\_ties ==   
 "lower" or binomial\_handle\_ties == "random"

self.use\_binomial = use\_binomial

self.binomial\_rank = binomial\_rank

self.binomial\_handle\_ties = binomial\_handle\_ties

class Hof:

"""

Repräsentiert einen aus Planquadraten bestehenden Hof, auf dem Laub   
 geblasen werden kann

"""

def \_\_init\_\_(self, size, rules = Rules(), \*, startwert=100, felder=None, blas\_counter=0):

self.x\_size = size[0] # Hof quadratisch -> x\_size und y\_size sind   
 gleich

self.y\_size = size[1]

self.rules = rules

self.startwert = startwert

if felder is None:

self.felder = np.full((self.x\_size, self.y\_size), startwert,   
 dtype=int if self.rules.use\_binomial else float)

else:

self.felder = np.array(felder, dtype=int if self.rules.use\_binomial else float)

self.blas\_counter = blas\_counter

self.blas\_log = []

def blase(self, feld0, blow\_direction):

"""

Simuliert einen Blasvorgang und aktualisiert self.felder auf die   
 resultierende Blattverteilung.

Args:

feld0 (int): Index des Felds, auf dem der Hausmeister steht

blow\_direction (tuple): Richtung, in die der Hausmeister bläst.   
 Kann folgende Werte annehmen: (1,0) =rechts, (-1,0) (=links),   
 (0,1) (=unten), (0,-1) (=oben)

"""

if not blow\_direction in [(0,1),(0,-1),(1,0),(-1,0)]:

return

self.blas\_counter += 1

self.blas\_log.append(dict(feld0=feld0, blow\_direction=blow\_direction))

# Richtung, die orthogonal zur Blasrichtung ist, ermitteln:

orthogonal\_direction = self.get\_orthogonal\_direction(blow\_direction)

# Feld A (Feld unmittelbar vor dem Laubbläser) ermitteln:

feldA = (feld0[0]+blow\_direction[0], feld0[1]+blow\_direction[1])

if not self.does\_exist(feldA):

return

new\_feldA\_value = 0

# Feld B (Feld hinter Feld A) ermitteln:

feldB = (feldA[0]+blow\_direction[0], feldA[1]+blow\_direction[1])

if self.does\_exist(feldB):

# -> Es gibt ein Feld B

if self.rules.use\_binomial:

A\_seitenabtrieb\_1, \_ = binomial\_likeliest(n=self.felder[feldA], p=self.rules.A\_seitenabtrieb, rank=self.rules.binomial\_rank, handle\_ties=self.rules.binomial\_handle\_ties)

A\_seitenabtrieb\_2, \_ = binomial\_likeliest(n=self.felder[feldA]-A\_seitenabtrieb\_1, p=self.rules.A\_seitenabtrieb/(1-self.rules.A\_seitenabtrieb), rank=self.rules.binomial\_rank, handle\_ties=self.rules.binomial\_handle\_ties)

B\_vorne\_abtrieb, \_ = binomial\_likeliest(n=self.felder[feldB], p=self.rules.B\_vorne\_abtrieb, rank=self.rules.binomial\_rank, handle\_ties=self.rules.binomial\_handle\_ties) # Anzahl an Blättern von Feld B, die nach vorne abgetrieben warden

else:

A\_seitenabtrieb\_1 = self.felder[feldA] \*   
 self.rules.A\_seitenabtrieb

A\_seitenabtrieb\_2 = A\_seitenabtrieb\_1

B\_vorne\_abtrieb = self.felder[feldB] \*   
 self.rules.B\_vorne\_abtrieb

new\_feldB\_value = self.felder[feldA] - (A\_seitenabtrieb\_1 + A\_seitenabtrieb\_2) + self.felder[feldB] - B\_vorne\_abtrieb

# Nachbarfeld von Feld B, das Feld A gegenüberliegt, aktualisieren   
 (sofern vorhanden):

if self.does\_exist((feldB[0]+blow\_direction[0],   
 feldB[1]+blow\_direction[1])):

B\_seitenabtrieb\_1 = 0

B\_seitenabtrieb\_2 = 0

self.felder[(feldB[0]+blow\_direction[0],   
 feldB[1]+blow\_direction[1])] += B\_vorne\_abtrieb

else:

if self.rules.use\_binomial:

B\_seitenabtrieb\_1, \_ = binomial\_likeliest(n=B\_vorne\_abtrieb, p=(1-self.rules.s1)/2, rank=self.rules.binomial\_rank, handle\_ties=self.rules.binomial\_handle\_ties)

B\_seitenabtrieb\_2, \_ = binomial\_likeliest(n=B\_vorne\_abtrieb-B\_seitenabtrieb\_1, p=((1-self.rules.s1)/2)/(1-(1-self.rules.s1)/2), rank=self.rules.binomial\_rank, handle\_ties=self.rules.binomial\_handle\_ties)

new\_feldB\_value += B\_vorne\_abtrieb - B\_seitenabtrieb\_1 - B\_seitenabtrieb\_2

else:

new\_feldB\_value += B\_vorne\_abtrieb \* self.rules.s1

B\_seitenabtrieb\_1 =   
 B\_vorne\_abtrieb \* (1-self.rules.s1) \* 0.5

B\_seitenabtrieb\_2 = B\_seitenabtrieb\_1

# Nachbarfelder von Feld B, die Feld A nicht gegenüberliegen,   
 aktualisieren (sofern vorhanden):

if self.does\_exist((feldB[0]+orthogonal\_direction[0],   
 feldB[1]+orthogonal\_direction[1])):

self.felder[(feldB[0]+orthogonal\_direction[0],  
 feldB[1]+orthogonal\_direction[1])  
 ] += A\_seitenabtrieb\_1 + B\_seitenabtrieb\_1

else:

new\_feldB\_value += A\_seitenabtrieb\_1 + B\_seitenabtrieb\_1

if self.does\_exist((feldB[0]-orthogonal\_direction[0], feldB[1]-orthogonal\_direction[1])):

self.felder[(feldB[0]-orthogonal\_direction[0], feldB[1]-  
 orthogonal\_direction[1])  
 ] += A\_seitenabtrieb\_2 + B\_seitenabtrieb\_2

else:

new\_feldB\_value += A\_seitenabtrieb\_2 +B\_seitenabtrieb\_2

# Feld B aktualsieren

self.felder[feldB] = new\_feldB\_value

else:

# -> Es gibt kein Feld B.

if self.rules.use\_binomial:

A\_noB\_seitenabtrieb\_1, \_ = binomial\_likeliest(n=self.felder[feldA], p=self.rules.A\_noB\_seitenabtrieb, rank=self.rules.binomial\_rank, handle\_ties=self.rules.binomial\_handle\_ties)

A\_noB\_seitenabtrieb\_2, \_ = binomial\_likeliest(n=self.felder[feldA]-A\_noB\_seitenabtrieb\_1, p=self.rules.A\_noB\_seitenabtrieb/(1-self.rules.A\_noB\_seitenabtrieb), rank=self.rules.binomial\_rank, handle\_ties=self.rules.binomial\_handle\_ties)

else:

A\_noB\_seitenabtrieb\_1 = self.felder[feldA] \*   
 self.rules.A\_noB\_seitenabtrieb

A\_noB\_seitenabtrieb\_2 = A\_noB\_seitenabtrieb\_1

new\_feldA\_value = self.felder[feldA] - (A\_noB\_seitenabtrieb\_1 +   
 A\_noB\_seitenabtrieb\_2)

# Nachbarfelder von Feld A, die dem laubblasenden Hausmeister nicht   
 gegenüberliegen, aktualisieren, sofern vorhanden:

if self.does\_exist((feldA[0]+orthogonal\_direction[0],  
 feldA[1]+orthogonal\_direction[1])):

self.felder[(feldA[0]+orthogonal\_direction[0],   
 feldA[1]+orthogonal\_direction[1])] += A\_noB\_seitenabtrieb\_1

else:

# -> Feld A ist ein Eckfeld, ein Teil des Laubs verbleibt also   
 auf Feld A.

new\_feldA\_value += A\_noB\_seitenabtrieb\_1

if self.does\_exist((feldA[0]-orthogonal\_direction[0], feldA[1]-  
 orthogonal\_direction[1])):

self.felder[(feldA[0]-orthogonal\_direction[0], feldA[1]-  
 orthogonal\_direction[1])] += A\_noB\_seitenabtrieb\_2

else:

# -> Feld A ist ein Eckfeld, ein Teil des Laubs verbleibt also   
 auf Feld A.

new\_feldA\_value += A\_noB\_seitenabtrieb\_2

# Feld A aktualisieren:

self.felder[feldA] = new\_feldA\_value

## Grundaufgabe: Ansatz 1

**Hilfsfunktionen zur Berechnung von Standardabweichung und Manhattan-Distanz:**

def squared\_std(values : list[float]):

"""

Returns:

float: Die quadrierte Standardabweichung (= Varianz) der Werte   
 in values

"""

mean = sum(values) / len(values)

squared\_diff = [(value - mean) \*\* 2 for value in values]

return sum(squared\_diff)

def manhattan\_distance(feld0, feld1) -> int:

"""

Returns:

int: Die Manhatten-Distanz zwischen den Tupeln feld0 und feld1,   
 definiert als d = (feld1[0] - feld0[0]) + (feld1[1] - feld1[0])

"""

return abs(feld1[0] - feld0[0]) + abs(feld1[1] - feld0[1])

**Klasse Solver1, die Methoden zum Durchführen des Blasprozesses (wie z.B. den Greedy-Algorithmus und Methoden zum Berechnen der Heuristiken) beinhaltet:**

class Solver1:

def \_\_init\_\_(  
self, hof, \*, satisfied\_constraint, max\_operations=1000, Q, weight\_avg, weight\_varianz  
 ):

"""

Args:

hof (Hof): Der Hof, auf den sich die Strategie beziehen soll

max\_operations (int): Anzahl an Blasoperationen, nach der die   
 Strategie auf jeden Fall abbricht

satisfied\_constraint (float): Anteil des Laubs auf Q am   
 Gesamtlaub, nach dessen Erreichen die Strategie auf jeden   
 Fall abbricht

Q (tuple): Der Index von Feld Q

weight\_avg (float), weight\_varianz (float): Die Gewichte, mit   
 denen die Größen durchschnittl. Blattabstand zu Q und   
 Varianz der Blattabstände zu Q in der Greedy Heuristik   
 gewichtet bzw. multipliziert werden

"""

self.hof = hof

self.max\_operations = max\_operations

self.satisfied\_constraint = satisfied\_constraint

self.Q = Q

if not self.hof.does\_exist(Q):

print("Q existiert nicht.")

exit()

if self.hof.is\_edge(Q):

print("Q darf kein Rand-/Eckfeld sein.")

exit()

self.clear\_edges = True

self.sum\_laub = np.sum(self.hof.felder)   
 # Gesamtanzahl an Blättern im Schulhof / im System bestimmen

self.weight\_avg = weight\_avg

self.weight\_varianz = weight\_varianz

# Variablen, die den Fortschritt beim Leeren der Ränder speichern:

self.edge\_fields\_to\_clear = [  
(x,0) for x in range(self.hof.x\_size)] + [

(x,self.hof.y\_size-1) for x in range(self.hof.x\_size)] + [

(0,y) for y in range(self.hof.y\_size)] +[

(self.hof.x\_size-1,y) for y in range(self.hof.y\_size)]

possible\_edge\_target\_fields = []

if not self.hof.y\_size == 3:

possible\_edge\_target\_fields += [(0,self.Q[1]),(self.hof.x\_size-1,self.Q[1])]

if not self.hof.x\_size == 3:

possible\_edge\_target\_fields += [(self.Q[0],0),(self.Q[0],self.hof.y\_size-1)]

if possible\_edge\_target\_fields == []:

self.edge\_target\_field = None

else:

self.edge\_target\_field = min(

possible\_edge\_target\_fields, key=lambda x : manhattan\_distance(x, self.Q)

)

self.clear\_edge\_last\_field\_index = 0 # Im Prozess des Verschiebens von Laub vom Feldrand   
 auf ein Nicht-Randfeld (Teil der self.greedy\_edge   
 Methode) wird hier die zuletzt durchgeführte   
 Operation gespeichert

self.edge\_target\_field\_neighbors = [] # Nachbarfelder das "Rand-Zielfelds"

**Hilfsfunktion, die die Randdistanz zwischen zwei Randfeldern bestimmt (gehört zur Solver1-Klasse):**

def edge\_distance(self, feld0, feld1):

"""

returns:

int: Die kleinste Verbindungsstrecke zwischen den Randfeldern   
 feld0 und feld1, die nur über Rand- und Eckfelder läuft

"""

if feld0[0] == 0 or feld0[0] == self.hof.x\_size -1:

# -> Feld 0 liegt auf unterem Rand

if feld0[0] == feld1[0]:

# -> Feld 0 und Feld 1 liegen in derselben "Zeile" - in   
 diesem Fall ist der gesuchte Abstand der x-  
 Koordinatenunterschied zwischen den Feldern

return abs(feld0[1] - feld1[1])

# Idee: Die beiden Ecken ermitteln, die zu Feld0 den   
 geringsten Abstand haben.

# min(distance(feld0, c) + distance(c, feld1)) entspricht   
 nämlich (für das Eckeld c, für das der Ausdruck minimal   
 wird) dem gesuchten Abstand zwischen Feld 0 und Feld 1.

c1 = (feld0[0],0)

d1 = feld0[1]

c2 = (feld0[0],self.hof.y\_size-1)

d2 = abs(self.hof.y\_size-1-feld0[1])

else:

if feld0[1] == feld1[1]:

return abs(feld0[0] - feld1[0])

c1 = (0,feld0[1])

d1 = feld0[0]

c2 = (self.hof.x\_size-1,feld0[1])

d2 = abs(self.hof.x\_size-1-feld0[0])

return min([d1+manhattan\_distance(c1,feld1),d2+manhattan\_distance(c2,feld1)])

**Greedy-Algorithmus, der auf Randfelder angewendet wird (gehört zur Solver1-Klasse):**

Hinweis: Zur Verbesserung der Lesbarkeit des Dokuments wurden sämtliche Kommentare aus dem Quellcode entfernt. Es wird daher empfohlen, den Quellcode (aufgabe1\_solver1.py) in einer geeigneten IDE zu öffnen und die Funktion dort zu lesen.

def greedy\_edges(self):

"""

Findet die beste Blasoperation zum Leeren des Rands (systematisches Verfahren)

"""

if self.edge\_target\_field is None:

return None

if self.edge\_fields\_to\_clear == []:

return None

possible\_blow\_directions = [(0,1),(1,0),(0,-1),(-1,0)]

field\_to\_clear = max(self.edge\_fields\_to\_clear, key=lambda x : self.edge\_distance(x, self.edge\_target\_field))

if self.hof.felder[field\_to\_clear] > self.hof.startwert \* (1-self.satisfied\_constraint):

if manhattan\_distance(field\_to\_clear, self.edge\_target\_field) == 1:

if not field\_to\_clear in self.edge\_target\_field\_neighbors:

self.edge\_target\_field\_neighbors.append(field\_to\_clear)

elif field\_to\_clear == self.edge\_target\_field:

self.edge\_fields\_to\_clear += self.edge\_target\_field\_neighbors

if self.clear\_edge\_last\_field\_index == 0:

self.clear\_edge\_last\_field\_index = 1

elif self.clear\_edge\_last\_field\_index == 1:

self.clear\_edge\_last\_field\_index = 2

if self.edge\_target\_field[0] == 0:

return dict(feld0=(1,self.edge\_target\_field[1]), blow\_direction=(-1,0))

elif self.edge\_target\_field[0] == self.hof.x\_size-1:

return dict(feld0=(self.hof.x\_size-2,self.edge\_target\_field[1]), blow\_direction=(1,0))

elif self.edge\_target\_field[1] == 0:

return dict(feld0=(self.edge\_target\_field[0],1), blow\_direction=(0,-1))

elif self.edge\_target\_field[1] == self.hof.y\_size-1:

return dict(feld0=(self.edge\_target\_field[0],self.hof.y\_size-2), blow\_direction=(0,1))

else:

self.clear\_edge\_last\_field\_index = 0

field\_to\_clear = self.edge\_target\_field\_neighbors[self.clear\_edge\_last\_field\_index]

for blow\_direction in possible\_blow\_directions:

feld0=(field\_to\_clear[0]-blow\_direction[0],field\_to\_clear[1]-blow\_direction[1])

if not self.hof.does\_exist(feld0):

continue

if self.hof.is\_corner(field\_to\_clear):

orthogonal\_direction = self.hof.get\_orthogonal\_direction(blow\_direction)

target\_field = (field\_to\_clear[0]+orthogonal\_direction[0],field\_to\_clear[1]+orthogonal\_direction[1])

if not (self.hof.is\_edge(target\_field) and target\_field in self.edge\_fields\_to\_clear):

target\_field = (field\_to\_clear[0]-orthogonal\_direction[0],field\_to\_clear[1]-orthogonal\_direction[1])

if not (self.hof.is\_edge(target\_field) and target\_field in self.edge\_fields\_to\_clear):

continue

else:

if field\_to\_clear in self.edge\_fields\_to\_clear:

self.edge\_fields\_to\_clear.remove(field\_to\_clear)

target\_field = (field\_to\_clear[0]+blow\_direction[0],field\_to\_clear[1]+blow\_direction[1])

if not self.hof.does\_exist(target\_field):

continue

if self.edge\_distance(target\_field, self.edge\_target\_field) < self.edge\_distance(field\_to\_clear, self.edge\_target\_field):

return dict(

feld0=feld0,

blow\_direction=blow\_direction

)

else:

self.edge\_fields\_to\_clear.remove(field\_to\_clear)

return self.greedy\_edges()

**Greedy-Algorithmus, der auf Nicht-Randfelder angewendet wird (gehört zur Solver1-Klasse):**

def greedy\_mid(self, \*, weight\_varianz, weight\_avg):

"""

Findet die beste Blasoperation zum Verschieben von Laub von Nicht-Randfeldern Richtung Q.   
 (unter Verwendung einer Greedy-Heuristik)

Returns:

dict: Die als nächstes auszuführende Blasoperation.

"""

possible\_blow\_directions = [(0,1),(1,0),(0,-1),(-1,0)]

blattdistanzen = self.blattdistanzen(self.hof, self.Q, ignore\_edge=False)  
 # Berechnet alle Blattdistanzen vom Feld Q aus

current\_mw\_bd = sum(blattdistanzen) / len(blattdistanzen)  
 # Die mittlere Blattdistanz am aktuellen Hof

best\_score = float("inf")  
 # Bester (= niedrigeste) Wert für die STABW der Blattdistanz an einem Hof, der aus einer   
 möglichen Blasoperation resultiert

best\_op = None # Die Blasoperation, aus der der best\_varianz\_bd Wert resultiert ist

for x in range(0,self.hof.x\_size):

for y in range(0,self.hof.y\_size):

for blow\_direction in possible\_blow\_directions:

if not self.hof.is\_edge((x,y)):

if self.hof.is\_edge((x+blow\_direction[0],y+blow\_direction[1])) or self.hof.is\_edge((x+2\*blow\_direction[0],y+2\*blow\_direction[1])):

continue # Laub auf Rand blasen ist verboten

if not manhattan\_distance((x,y), self.Q) < manhattan\_distance((x+blow\_direction[0], y+blow\_direction[1]), self.Q):

hof\_copy = self.hof.copy()

hof\_copy.rules.use\_binomial=False # Beim Finden der besten Blasoperation  
 mit den Erwartungswerten arbeiten, um   
 die Zufallskomponente zu entfernen

hof\_copy.blase((x,y), blow\_direction)

blattdistanzen = self.blattdistanzen(hof\_copy, self.Q, ignore\_edge=False)

if sum(blattdistanzen) / len(blattdistanzen) < current\_mw\_bd:

# Erstes, priorisiertes Maß der Heuristik. Nur Operationen, die   
 dieses Maß verringern, werden zugelassen

score = squared\_std(blattdistanzen) \* weight\_varianz + (sum(blattdistanzen) / len(blattdistanzen)) \* weight\_avg # Gewichtetes Produkt aus erstem und zweitem Maß bilden

if score < best\_score:

# Blasoperation mit dem besten Score finden

best\_score = score

best\_op = dict(  
feld0=tuple((x,y)), blow\_direction=tuple(blow\_direction)  
 )

return best\_op

**step-Funktion, die bei Aufruf die Greedy-Heuristik ausführt, um die als nächstes auszuführende Blasoperation zu bestimmen, und diese anschließend ausführt:**

def step(self):

"""

Führt den nächsten Schritt des Blasprozesses aus:

Mithilfe von self.greedy wird die nächste Blasoperation bestimmt, die anschließend   
 ausgeführt wird

"""

if self.clear\_edges:

next\_op = self.greedy\_edges()

else:

next\_op = self.greedy\_mid(weight\_varianz=self.weight\_varianz, weight\_avg=self.weight\_avg)

if next\_op is None:

self.clear\_edges = not self.clear\_edges

if self.clear\_edges:

self.edge\_fields\_to\_clear = [(x,0) for x in range(self.hof.x\_size)] + [(x,self.hof.y\_size-1) for x in range(self.hof.x\_size)] + [(0,y) for y in range(self.hof.y\_size)] +[(self.hof.x\_size-1,y) for y in range(self.hof.y\_size)]

self.edge\_target\_field = min([(0,self.Q[1]),(self.hof.x\_size-1,self.Q[1]),(self.Q[0],0),(self.Q[0],self.hof.y\_size-1)], key=lambda x : manhattan\_distance(x, self.Q))

next\_op = self.greedy\_edges()

else:

next\_op = self.greedy\_mid(weight\_varianz=self.weight\_varianz, weight\_avg=self.weight\_avg)

if next\_op is not None:

return True

else:

self.hof.blase(next\_op["feld0"], next\_op["blow\_direction"])

if self.satisfied\_constraint is not None:

if self.hof.felder[self.Q] / self.sum\_laub >= self.satisfied\_constraint:

return False

if self.max\_operations is not None:

if self.hof.blas\_counter >= self.max\_operations:

return False

return True

## Grundaufgabe: Ansatz 2

**Klasse zur Implementierung des Muster-Konzepts:**

class Muster:

"""

Implementierung des theoretischen Konzept eines Musters   
 (siehe Dokumentation)

"""

def \_\_init\_\_(self, strategy, source\_fields, operations, tolerated\_amount : float, \*, num\_max\_operations=None):

self.strategy = strategy # Die Strategie, zu der das Muster gehört

self.hof = self.strategy.hof

if num\_max\_operations is None:

self.num\_max\_operations = strategy.max\_muster\_operations

else:

self.num\_max\_operations = num\_max\_operations

self.source\_fields = [  
 strategy.rotate\_field\_index(f) for f in source\_fields]

self.operations = [  
 strategy.rotate\_blasoperation(o) if isinstance(o, dict) else o for   
 o in operations]

self.tolerated\_amount = tolerated\_amount

self.reset()

def reset(self):

"""

Zurücksetzen des Musters auf den Zustand vor seiner Ausführung

"""

self.num\_operations = 0 # Bisher durchgeführte Blasoperationen

self.check\_for\_changes = not (self.hof.rules.use\_binomial is True and  
 self.hof.rules.binomial\_rank == "random")

if self.check\_for\_changes:

# Es wird nach jedem vollständigen Musterdurchlauf überprüft, ob   
 nach Veränderungen an der Gesamtlaubmenge auf den source-Feldern   
 stattfinden

# Diese Überprüfung findet aber nicht statt, wenn die   
 Laubblassimulation den tatsächlichen Zufall simuliert

self.current\_sum = 0

self.next\_op\_index = 0 # Index der als nächst. auszuführenden Operation

def step(self):

"""

Führt basierend auf self.operations die nächste Blasoperation aus

"""

if isinstance(self.operations[self.next\_op\_index], dict):

self.next\_op\_index += 1

self.num\_operations += 1

elif isinstance(self.operations[self.next\_op\_index], Muster):

# -> Ein anderes Muster liegt vor, das ausgeführt wird

run\_another\_step = self.operations[self.next\_op\_index].step()

if run\_another\_step:

return True

self.next\_op\_index += 1

self.num\_operations += 1

if self.next\_op\_index == len(self.operations):

# -> Einmal durch alle Operationen des Musters durchgelaufen   
 # -> i zurücksetzen

self.next\_op\_index = 0

if self.check\_for\_changes:

# Überprüfen, ob noch Veränderungen stattfinden

new\_sum = sum(  
 [self.hof.felder[index] for index in self.source\_fields])

if new\_sum == self.current\_sum:

return False

self.current\_sum = int(new\_sum)

if isinstance(self.operations[self.next\_op\_index], Muster):

self.operations[self.next\_op\_index].reset()

# Überprüfen, ob maximale Anzahl an Operationen erreicht wurde:

if self.num\_operations >= self.num\_max\_operations:

return False

if max(  
 [self.hof.felder[index] for index in self.source\_fields]  
 ) <= self.tolerated\_amount:

return False

return True

def run(self):

"""

Ruft self.step() solange auf, bis die Abbruchbedingungen erfüllt sind

"""

self.reset()

while self.step():

pass

**Klasse Solver2, die Methoden zum Durchführen des Blasprozesses beinhaltet:**Auch hier wurden wieder einige Kommentare entfernt. In der Original-Python-Datei (aufgabe1\_solver2.py) ist der vollständig kommentierte Code zu finden.

class Solver2:

"""

Dient zum Erstellen, Speichern, Ausführen von Strategien bzw. generalisierten Ablaufplänen  
 nach Verfahren 2 (siehe Dokumentation)

"""

def \_\_init\_\_(self, hof, \*, tolerated\_amount, max\_muster\_operations=100, choose\_faster\_path=False):

self.hof = hof

self.tolerated\_amount = tolerated\_amount

if tolerated\_amount <= 0:

print("tolerated\_amount muss größer als Null sein.")

exit()

self.strategy = []

self.max\_muster\_operations = max\_muster\_operations

self.running\_op\_index = 0

# Bekommen später einen Wert zugewiesen:

self.Q = None # Hier wird der Index von Feld Q als Tupel gespeichert werden

self.num\_rotations = 0

self.choose\_faster\_path = choose\_faster\_path

def add\_operation(self, operation):

"""

Fügt eine Blasoperation oder ein Muster zum Ablaufplan hinzu. Macht zuvor die Initialrotation rückgängig (beim späteren Ausführen wird der Hof in seiner ursprünglichen Ausrichtung betrachtet, die Rotation erfolgt nur, da dies die Generierung des generalisierten Ablaufplans / der Strategie erleichtert).

"""

self.strategy.append(self.rotate\_blasoperation(operation) if isinstance(operation, dict) else operation)

def rotate\_field\_index(self, field\_index) -> tuple:

"""

Ermittelt den ursprünglichen Index des Felds (den es vor der Initialrotation hatte), das derzeit am Index field\_index ist

"""

x\_size, y\_size = int(self.hof.x\_size), int(self.hof.y\_size)

for i in range(self.num\_rotations):

field\_index = (y\_size-1-field\_index[1], field\_index[0]) # field\_index um 90° im Uhrzeigersinn rotieren

helper = int(y\_size) # Eine Rotation um 90° sorgt dafür, dass sich x-Größe (Breite)   
 und y-Größe (Höhe) des Hofs tauschen

y\_size = int(x\_size)

x\_size = int(helper)

return field\_index

def rotate\_direction\_vector(self, vector : tuple) -> tuple:

"""

Ermittelt die ursprüngliche Richtung des Vektors vector (den er vor der Initialrotation hatte)

"""

for r in range(self.num\_rotations):

# Vektor um 90° im Uhrzeigersinn rotieren:

if vector[0] == 0:

vector = (-vector[1],0)

elif vector[1] == 0:

vector = (0,vector[0])

return vector

def rotate\_blasoperation(self, blasoperation) -> dict:

return dict(

feld0=self.rotate\_field\_index(blasoperation["feld0"]),

blow\_direction=self.rotate\_direction\_vector(blasoperation["blow\_direction"])

)

**Funktionen, die Muster zum Transferieren von Blättern zurückgeben (gehören beide zur Solver2-Klasse):**

def corner\_to\_edge(self, source\_corner\_field, target\_field) -> Muster:

"""

Returns:

Muster: Ein Muster, das bei Anwendung Laub vom Eckfeld source\_corner\_field auf das  
 Randfeld target\_field bläst

"""

if not self.hof.are\_adjacent(source\_corner\_field, target\_field):

return

if not (self.hof.is\_corner(source\_corner\_field) and self.hof.is\_edge(target\_field)):

return

orthogonal\_direction = (target\_field[0]-source\_corner\_field[0], target\_field[1]-  
 source\_corner\_field[1])   
 # Die Richtung, die orthogonal zur Blasrichtung ist (entspricht dem Vektor target\_field-  
 source\_corner)

# Beide zu orthogonal\_direction orthogonale richtungen werden als mögliche Blasrichtungen   
 durchprobiert:

anti\_blow\_direction = self.hof.get\_orthogonal\_direction(orthogonal\_direction)  
 # Gegenrichtung / Gegenvektor zur Blasrichtung

feld0 = (source\_corner\_field[0]+anti\_blow\_direction[0], source\_corner\_field[1]+anti\_blow\_direction[1]) # Erstes mögliches Startfeld

blow\_direction = (-anti\_blow\_direction[0], -anti\_blow\_direction[1])

if not self.hof.does\_exist(feld0): # Wenn das erste mögliche Startfeld nicht existiert,   
 dann muss das andere mögliche Startfeld das Startfeld sein

anti\_blow\_direction = blow\_direction

feld0 = (source\_corner\_field[0]+anti\_blow\_direction[0],   
 source\_corner\_field[1]+anti\_blow\_direction[1])

blow\_direction = (-anti\_blow\_direction[0], -anti\_blow\_direction[1])

muster = Muster(self, [source\_corner\_field], [dict(feld0=feld0,   
 blow\_direction=blow\_direction)], self.tolerated\_amount)

return muster

def edge\_to\_mid(self, source\_edge\_field, target\_field) -> Muster:

"""

Returns:

Muster: Ein Muster, das bei Anwendung Laub vom Randfeld source\_edge\_field und seinen  
 beiden auf dem Rand liegenden Nachbarn auf das Nicht-Rand-oder-Eckfeld   
 target\_field bläst

"""

if not self.hof.are\_adjacent(source\_edge\_field, target\_field):

return

if (not self.hof.is\_edge(source\_edge\_field)) or self.hof.is\_edge(target\_field):

return

orthogonal\_direction = (source\_edge\_field[0]-target\_field[0], source\_edge\_field[1]-  
 target\_field[1]) # Die Richtung, die zum Rand hinzeigt

blow\_direction = self.hof.get\_orthogonal\_direction(orthogonal\_direction)

operations = []

operations.append(dict(feld0=target\_field, blow\_direction=orthogonal\_direction)) # Den Blasvorgang von der Mitte zur Kante hin ermitteln (dieser verteilt das Laub durch die Seitenabtriebe auf den beiden Feldern neben source\_edge\_field)

# Die Blasvorgänge, die Laub durch Seitenabtriebe auf target\_field transportieren,   
 ermitteln

feld0\_1 = (source\_edge\_field[0]+blow\_direction[0]\*2,   
 source\_edge\_field[1]+blow\_direction[1]\*2)

blow\_direction = (-blow\_direction[0], -blow\_direction[1])

if self.hof.does\_exist(feld0\_1):

operations.append(dict(feld0=feld0\_1, blow\_direction=blow\_direction))

else:

operations.insert(0, dict(feld0=(source\_edge\_field[0]-blow\_direction[0]-orthogonal\_direction[0], source\_edge\_field[1]-blow\_direction[1]-orthogonal\_direction[1]), blow\_direction=orthogonal\_direction))

feld0\_2 = (source\_edge\_field[0]+blow\_direction[0]\*2,   
 source\_edge\_field[1]+blow\_direction[1]\*2)

blow\_direction = (-blow\_direction[0], -blow\_direction[1])

if self.hof.does\_exist(feld0\_2):

operations.append(dict(feld0=feld0\_2, blow\_direction=blow\_direction))

else:

operations.insert(0, dict(feld0=(source\_edge\_field[0]-blow\_direction[0]-orthogonal\_direction[0], source\_edge\_field[1]-blow\_direction[1]-orthogonal\_direction[1]), blow\_direction=orthogonal\_direction))

if not (self.hof.does\_exist(feld0\_1) or self.hof.does\_exist(feld0\_2)):

return # Beide möglichen Startfelder existieren nicht -> Hof hat Kantenlänge 3 ->   
 Kante kann nicht gecleared werden

source\_fields = [source\_edge\_field, (source\_edge\_field[0]+blow\_direction[0], source\_edge\_field[1]+blow\_direction[1]), (source\_edge\_field[0]-blow\_direction[0], source\_edge\_field[1]-blow\_direction[1])]

muster = Muster(self, source\_fields, operations, self.tolerated\_amount)

return muster

**Die Funktionen, die die in den Phasen des generalisierten Ablaufplans auszuführenden Muster und Blasoperationen festlegen (gehören alle zur Solver2-Klasse):**

def clear\_bottom\_line(self):

"""

1. Phase des generalisierten Ablaufs: Befreien der untersten Reihe des Hofs. Fügt die   
 hierfür notwendigen Operationen zu self.strategy hinzu.

"""

for start\_x in range(self.hof.x\_size-1,0,-1):

self.add\_operation(dict(feld0=(start\_x,self.hof.y\_size-1),blow\_direction=(-1,0)))   
 # Nicht-Eckfelder am unteren Rand leeren

self.add\_operation(self.corner\_to\_edge((self.hof.x\_size-1, self.hof.y\_size-  
 1),(self.hof.x\_size-1,self.hof.y\_size-2))) # Ecke rechts unten leeren

self.add\_operation(self.corner\_to\_edge((0, self.hof.y\_size-1),(0,self.hof.y\_size-2)))   
 # Ecke links unten leeren

def move\_to\_top\_line(self):

"""

2. Phase des generalisierten Ablaufs: Bläst das gesamte Laub in die oberste Reihe. Fügt  
 die hierfür notwendigen Operationen zu self.strategy hinzu.

"""

for start\_y in range(self.hof.y\_size-1,1,-1):

for start\_x in range(self.hof.x\_size-1,-1,-1):

if not (start\_x == self.Q[0] and (start\_y <= self.Q[1]+1 or self.hof.y\_size == 5) and (self.Q[1] != 1)): # Feld Q und Felder über Q nicht unnötigerweise leeren, da hier nachher das Laub sowieder wieder hintransportiert werden muss

self.add\_operation(dict(feld0=(start\_x, start\_y), blow\_direction=(0,-1)))

def concentrate\_top\_line(self):

"""

3. Phase des generalisierten Ablaufs: Konzentriert das Laub der obersten Reihe auf dem   
 Randfeld, das sich in der selben Spalte wie Q befindet. Fügt die hierfür notwendigen   
 Operationen zu self.strategy hinzu.

"""

Q = self.Q

source\_fields\_for\_edgeclear = [(Q[0]-1,0), (Q[0],0), (Q[0]+1,0)]

if not (0,0) in source\_fields\_for\_edgeclear:

self.add\_operation(self.corner\_to\_edge((0,0),(1,0)))   
 # Ecke links oben clearen, wenn sie in Phase 4 nicht Source-Feld wird

if not (self.hof.x\_size-1,0) in source\_fields\_for\_edgeclear:

self.add\_operation(self.corner\_to\_edge((self.hof.x\_size-1,0),(self.hof.x\_size-2,0)))   
 # Ecke rechts oben clearen, wenn sie in Phase 4 nicht Source-Feld wird

# Nicht-Eckfelder der obersten Reihe auf Source-Felder konzentrieren:

for start\_x in range(0,self.hof.x\_size-1):

if (start\_x+1,0) in source\_fields\_for\_edgeclear:

break

else:

operationen = [dict(feld0=(start\_x,0), blow\_direction=(1,0)), dict(feld0=(start\_x+2,2), blow\_direction=(0,-1))]

muster = Muster(self, [(start\_x+1,0), (start\_x+2,1)], operationen, self.tolerated\_amount) # Muster zum Leeren des Eckfelds (start\_x+1,0) und zum gleichzeitigen Beseitigen des entstehenden Seitenabtriebs auf Feld (start\_x+2,0)

self.add\_operation(muster)

for start\_x in range(self.hof.x\_size-1,0,-1):

if (start\_x-1,0) in source\_fields\_for\_edgeclear:

break

else:

operationen = [dict(feld0=(start\_x,0), blow\_direction=(-1,0)),   
 dict(feld0=(start\_x-2,2), blow\_direction=(0,-1))]

muster = Muster(self, [(start\_x-1,0), (start\_x-2,1)], operationen,   
 self.tolerated\_amount)

self.add\_operation(muster)

def transfer\_to\_Q(self):

"""

4. Phase des generalisierten Ablaufs: Verschiebt das gesamte Laub auf Feld Q. Fügt die   
 hierfür notwendigen Operationen zu self.strategy hinzu.

"""

Q = self.Q

self.add\_operation(self.edge\_to\_mid((Q[0],0), (Q[0],1))) # Muster hinzufügen, dass Laub   
 vom Eckfeld (Q[0],0) auf das Feld (Q[0],1) bläst

if Q != (Q[0],1): # Wenn (Q[0],1) == Q, dann befindet sich das Laub jetzt bereits auf Q -  
 wenn nicht, müssen weitere Schritte ausgeführt werden

self.add\_operation(dict(feld0=(Q[0],0), blow\_direction=(0,1))) # Das Laub wird vom   
 Rand aus auf (Q[0],2) geblasen - dabei entstehen aber Seitenabtriebe. Mit   
 diesen wird im Folgenden umgegangen.

# Mit den in der vorherigen Blasoperation entstandenen Seitenabtrieben umgehen: Wenn möglich wird das Laub verlustfrei durch den Abtrieb bei B vorne auf das Feld (Q[0],2) geblasen. Wenn nicht, dann wird das im vorherigen Schritt abgetriebene Laub auf den Felder (Q[0]-1,3), (Q[0],3) und (Q[0]+1,3) versammelt.

cleared\_fields = []

if self.hof.does\_exist((Q[0]+3,2)):

muster = Muster(self, [(Q[0]+1,2)], [dict(feld0=(Q[0]+3,2),blow\_direction=(-  
 1,0))], self.tolerated\_amount)

self.add\_operation(muster)

cleared\_fields.append((Q[0]+1,2))

elif not (self.hof.x\_size == 5 and self.hof.y\_size == 5 and Q == (2,2)): # In diesem   
 Fall wird anders vorgegangen (siehe Code unten)

muster = Muster(self, [(Q[0]+1,2)],   
 [dict(feld0=(Q[0]+1,0),blow\_direction=(0,1))], self.tolerated\_amount)

self.add\_operation(muster)

if self.hof.does\_exist((Q[0]-3,2)):

muster = Muster(self, [(Q[0]-1,2)], [dict(feld0=(Q[0]-  
 3,2),blow\_direction=(1,0))], self.tolerated\_amount)

self.add\_operation(muster)

cleared\_fields.append((Q[0]-1,2))

elif not (self.hof.x\_size == 5 and self.hof.y\_size == 5 and Q == (2,2)):  
 # In diesem Fall wird anders vorgegangen (siehe Code unten)

muster = Muster(self, [(Q[0]-1,2)], [dict(feld0=(Q[0]-  
 1,0),blow\_direction=(0,1))], self.tolerated\_amount)

self.add\_operation(muster)

if self.hof.does\_exist((Q[0],5)):

if Q != (Q[0],3):

muster = Muster(self, [(Q[0],3)], [dict(feld0=(Q[0],5),blow\_direction=(0,-  
 1))], self.tolerated\_amount)

self.add\_operation(muster)

cleared\_fields.append((Q[0],3))

elif Q == (Q[0],3):

cleared\_fields.append((Q[0],3))

if len(cleared\_fields) != 3:

# Wenn dies nicht möglich war, dann wird stattdessen ein komplexeres System aus   
 Mustern und Blasoperationen verwendet, um das Laub auf Feld Q zu befärdern

if (self.hof.does\_exist((Q[0],6)) or ((Q[0], 3) == Q and self.hof.does\_exist((Q[0],5)))) and (self.hof.does\_exist((Q[0]-3, 3)) or self.hof.does\_exist((Q[0]+3, 3))):

# In diesem Sonderfall sind unter (Q[0],2) noch 4 weitere Felder, was ein   
 verlustfreies Blasen des unteren Seitenabtriebs auf Feld (Q[0],2)   
 ermöglicht

if self.hof.does\_exist((Q[0]-3, 3)):

self.add\_operation(dict(feld0=(Q[0]+2,3), blow\_direction=(-1,0)))

self.add\_operation(Muster(self, [(Q[0]-1, 3)], [dict(feld0=(Q[0]-3,3),   
 blow\_direction=(1,0))], self.tolerated\_amount))

else:

self.add\_operation(dict(feld0=(Q[0]-2,3), blow\_direction=(1,0)))

self.add\_operation(Muster(self, [(Q[0]+1, 3)], [dict(feld0=(Q[0]+3,3),  
 blow\_direction=(-1,0))], self.tolerated\_amount))

self.add\_operation(Muster(self, [(Q[0], 4)], [dict(feld0=(Q[0],6),   
 blow\_direction=(0,-1))], self.tolerated\_amount))

if not Q != (Q[0], 3):

# Das Laub befindet sich nach den vorherigen Schritten auf Feld (Q[0], 3)   
 - Wenn (Q[0], 3) == 3 dann müssen folglich keine weiteren Operationen   
 mehr durchgeführt werden

self.add\_operation(Muster(self, [(Q[0], 3)], [dict(feld0=(Q[0],5),  
 blow\_direction=(0,-1))], self.tolerated\_amount))

elif self.hof.does\_exist((Q[0],5)):

# Wenn unter Feld (Q[0], 2) drei Felder frei sind, dann wird mit folgendem   
 Muster kontinuierlich Laub auf Feld (Q[0], 2 geblasen)

operations = []

operations.append(dict(feld0=(Q[0]-2,3), blow\_direction=(1,0)))

operations.append(dict(feld0=(Q[0]+2,3), blow\_direction=(-1,0)))

operations.append(dict(feld0=(Q[0],5), blow\_direction=(0,-1)))

source\_fields = [(Q[0]-1,3), (Q[0],3), (Q[0]+1,3), (Q[0],4)]

self.add\_operation(Muster(self, source\_fields, operations,   
 self.tolerated\_amount))

elif self.hof.x\_size == 5 and self.hof.y\_size == 5 and Q == (2,2):

# Im übrigbleibenden Fall sind unter Feld (Q[0], 2) nur zwei Felder frei, es   
 muss also damit umgegangen werden, dass das Feld (Q[0], 4) ein Randfeld ist

# Da der Hof so rotiert wird, dass die lange Seite die y-Dimension ist, tritt   
 dieser Fall nur für Höfe mit den Maßen (5,5) und einem sich in der Hofmitte   
 befindenenden Feld Q auf

self.add\_operation(dict(feld0=(0,2), blow\_direction=(1,0)))

if self.choose\_faster\_path:

self.add\_operation(dict(feld0=(2,4), blow\_direction=(0,-1)))

self.num\_rotations -= 1

self.add\_operation(Muster(self, [(1,2)], [dict(feld0=(1,4),   
 blow\_direction=(0,-1))], self.tolerated\_amount))

self.add\_operation(Muster(self, [(3,2)], [dict(feld0=(3,4),   
 blow\_direction=(0,-1))], self.tolerated\_amount))

operations = []

operations.append(Muster(self, [(1,1),(2,1),(3,1)], [

dict(feld0=(0,1), blow\_direction=(1,0)),

dict(feld0=(4,1), blow\_direction=(-1,0)),

], self.tolerated\_amount)

)

operations.append(self.edge\_to\_mid((2,0), (2,1)))

self.add\_operation(Muster(self, [(1,1), (2,1), (3,1), (2,0)], operations,   
 self.tolerated\_amount))

if Q != (Q[0], 2):

# Das Laub befindet sich nach Durchführen der vorherigen Schritte auf (Q[0], 2).   
 Wenn Q[1] > 2, dann muss das Laub noch nach unten geblasen werden. Da über   
 (Q[0], 2) zwei Felder sind, ist das nun verlustfrei durch den Abtrieb bei B   
 vorne problemlos möglich

for y\_start in range(0,Q[1]-2):

self.add\_operation(Muster(self, [(Q[0], y\_start+2)], [dict(feld0=(Q[0],   
 y\_start), blow\_direction=(0,1))], self.tolerated\_amount))

**Die Funktion, die die Erstellung des generalisierten Ablaufplans initiiert und die Initalrotation durchführt (gehört zur Solver2-Klasse):**

def build\_strategy(self, \*, Q):

"""

Entwirft eine Strategie bzw. einen generalisierten Ablaufplan

"""

self.strategy = []

if not self.hof.does\_exist(Q):

print("Q existiert nicht.")

exit()

if self.hof.is\_edge(Q):

print("Q darf kein Rand-/Eckfeld sein.")

exit()

if self.hof.x\_size == 3 and self.hof.y\_size == 3:

return # For Höfe, bei denen eine Dimension kleiner 3 ist, gibt es keine Lösung. Für Höfe der Dimensionen (3,3) ist es am besten, gar nichts zu machen

# Hof rotieren, sodass sich Feld Q im Bereich oben links befindet (Initialrotation)

num\_rotations = 0

for i in range(3):

if self.hof.y\_size == 3 and self.hof.x\_size != 3:

break

if not self.hof.x\_size == 3:

if Q[1] <= math.ceil(self.hof.y\_size/2)-1:

if Q[0] == 1:

break

if not ((Q[0] == 1 and Q[1] != 1) or (Q[0] == self.hof.x\_size-2 and Q[1] !=  
 1)):

if self.hof.x\_size <= self.hof.y\_size:

break

self.hof.felder = np.rot90(self.hof.felder, k=1, axes=(0, 1))

self.hof.x\_size = self.hof.felder.shape[0]

self.hof.y\_size = self.hof.felder.shape[1]

Q = (Q[1], self.hof.y\_size-1-Q[0])

num\_rotations += 1

self.num\_rotations = num\_rotations

self.Q = Q

# Strategie speichern

self.strategy.append("\n[Phase 1: Unterste Reihe entlauben]")

self.clear\_bottom\_line()

self.strategy.append("\n[Phase 2: Laub auf oberste Reihe blasen]")

self.move\_to\_top\_line()

self.strategy.append("\n[Phase 3: Laub auf oberster Reihe konzentrieren]")

self.concentrate\_top\_line()

self.strategy.append("\n[Phase 4: Laub nach Feld Q transferieren]")

self.transfer\_to\_Q()

# Initialrotation rückgängig machen

for i in range(self.num\_rotations):

self.hof.felder = np.rot90(self.hof.felder, k=-1, axes=(0, 1))

Q = (self.hof.y\_size-1-Q[1], Q[0])

self.hof.x\_size = self.hof.felder.shape[0]

self.hof.y\_size = self.hof.felder.shape[1]

self.Q = Q

def step(self, \*, render\_zwischenschritte=False):

"""

Führt basierend auf self.strategy die nächste Blasoperation aus

"""

if self.running\_op\_index < len(self.strategy):

operation = self.strategy[self.running\_op\_index]

if isinstance(operation, str):

self.hof.blas\_log.append(operation)

if render\_zwischenschritte:

self.hof.render(title="Laubverteilung vor:"+operation+f"\n({self.hof.blas\_counter} Blasoperationen ausgeführt)", Q=self.Q)

self.running\_op\_index += 1

operation = self.strategy[self.running\_op\_index]

if isinstance(operation, dict):

self.hof.blase(operation["feld0"], operation["blow\_direction"])

elif isinstance(operation, Muster):

run\_another\_step = operation.step()

if run\_another\_step:

return

self.running\_op\_index += 1

## Grundaufgabe: Ansatz 3

Bei Ansatz 3 handelt es sich um eine Kombination von Ansatz 1 und 2. Daher kommen keine neuen Funktionen hinzu, die hier dokumentiert werden müssten.

Auch bei den vorgenommenen Erweiterungen (die alle auf Ansatz 2 basieren) werden nur die Funktionen, Klassen und Datenstrukturen dokumentiert, die sich verändern bzw. neu dazukommen.

## Erweiterung 1 (basierend auf Ansatz 2)

**Veränderte Funktionen, die die in den Phasen des generalisierten Ablaufplans auszuführenden Muster und Blasoperationen festlegen (Teil der Solver2-Klasse):**

def clear\_edges(self):

"""

1.Phase: Ränder (bis auf oberen Rand) entlauben, um zu vermeiden, dass beim Leeren der Ecken Laub verloren geht

"""

for start\_x in range(self.hof.x\_size-1,3,-1):

self.add\_operation(dict(feld0=(start\_x,self.hof.y\_size-  
 1),blow\_direction=(-1,0))) # Nicht-Eckfelder am unteren Rand leeren

if self.hof.x\_size > 4:

self.add\_operation(self.edge\_to\_mid((2, self.hof.y\_size-1), (2,   
 self.hof.y\_size-2)))

elif self.hof.x\_size > 3:

self.add\_operation(self.edge\_to\_mid((1, self.hof.y\_size-1), (1,   
 self.hof.y\_size-2)))

self.add\_operation(self.corner\_to\_edge((self.hof.x\_size-1,  
 self.hof.y\_size-1),(self.hof.x\_size-1,self.hof.y\_size-2))) # Ecke   
 rechts unten leeren

self.add\_operation(self.corner\_to\_edge((0, self.hof.y\_size-  
 1),(0,self.hof.y\_size-2))) # Ecke links unten leeren

for start\_y in range(self.hof.y\_size-1,3,-1):

self.add\_operation(dict(feld0=(0,start\_y),blow\_direction=(0,-1)))   
 # Nicht-Eckfelder am unteren Rand leeren

for start\_y in range(self.hof.y\_size-1,3,-1):

self.add\_operation(dict(feld0=(self.hof.x\_size-  
 1,start\_y),blow\_direction=(0,-1))) # Nicht-Eckfelder am unteren   
 Rand leeren

if self.hof.y\_size > 4:

self.add\_operation(self.edge\_to\_mid((self.hof.x\_size-1, 2),  
 (self.hof.x\_size-2, 2)))

self.add\_operation(self.edge\_to\_mid((0, 2), (1, 2)))

elif self.hof.y\_size > 3:

self.add\_operation(self.edge\_to\_mid((self.hof.x\_size-1, 1),   
 (self.hof.x\_size-2, 1)))

self.add\_operation(self.edge\_to\_mid((0, 1), (1, 1)))

for start\_x in range(0, self.Q[0]-1):

for start\_y in range(self.hof.y\_size-3,0,-1):

if start\_x == self.Q[0]-2 and self.hof.does\_exist((start\_x-1,   
 start\_y)):

self.add\_operation(Muster(self, [(start\_x+1, start\_y)],   
 [dict(feld0=(start\_x-1, start\_y), blow\_direction=(1,0))],   
 self.tolerated\_amount))

else:

self.add\_operation(dict(feld0=(start\_x,start\_y),   
 blow\_direction=(1,0)))

for start\_x in range(self.hof.x\_size-1, self.Q[0]+1, -1):

for start\_y in range(self.hof.y\_size-3,0,-1):

if start\_x == self.Q[0]+2 and self.hof.does\_exist((start\_x+1,   
 start\_y)):

self.add\_operation(Muster(self, [(start\_x-1, start\_y)],   
 [dict(feld0=(start\_x+1, start\_y), blow\_direction=(-1,0))],   
 self.tolerated\_amount))

else:

self.add\_operation(dict(feld0=(start\_x,start\_y),   
 blow\_direction=(-1,0)))

if self.hof.does\_exist((Q[0]-3,self.hof.y\_size-2)):

self.add\_operation(Muster(self, [(Q[0]-1, self.hof.y\_size-2)], [dict(feld0=(Q[0]-3, self.hof.y\_size-2), blow\_direction=(1,0))], self.tolerated\_amount))

if self.hof.does\_exist((Q[0]+3,self.hof.y\_size-2)):

self.add\_operation(Muster(self, [(Q[0]+1, self.hof.y\_size-2)], [dict(feld0=(Q[0]+3, self.hof.y\_size-2), blow\_direction=(-1,0))], self.tolerated\_amount))

def move\_to\_top\_line(self):

"""

2. Phase des generalisierten Ablaufs: Bläst das gesamte Laub in die oberste Reihe. Fügt die hierfür notwendigen Operationen zu self.strategy hinzu.

"""

for start\_y in range(self.hof.y\_size-1,1,-1):

for start\_x in range(self.hof.x\_size-1,-1,-1):

if not (start\_x == self.Q[0] and (start\_y <= self.Q[1]+1 or   
 self.hof.y\_size == 5)):

if not (start\_x == self.Q[0] and start\_y-1 == self.Q[1]):

self.add\_operation(dict(feld0=(start\_x, start\_y),   
 blow\_direction=(0,-1)))

elif start\_y > 3:

self.add\_operation(Muster(

self, [(start\_x, start\_y-1)], [dict(feld0=(start\_x,   
 start\_y), blow\_direction=(0,-1))],   
 self.tolerated\_amount

))

def concentrate\_top\_line(self):

"""

3. Phase des generalisierten Ablaufs: Konzentriert das Laub der obersten Reihe auf dem Randfeld, das sich in der selben Spalte wie Q befindet. Fügt die hierfür notwendigen Operationen zu self.strategy hinzu.

"""

Q = self.Q

source\_fields\_for\_edgeclear = [(Q[0]-1,0), (Q[0],0), (Q[0]+1,0)]

if not (0,0) in source\_fields\_for\_edgeclear:

self.add\_operation(self.corner\_to\_edge((0,0),(1,0))) # Ecke links oben clearen, wenn sie in Phase 4 nicht Source-Feld wird

if not (self.hof.x\_size-1,0) in source\_fields\_for\_edgeclear:

self.add\_operation(self.corner\_to\_edge((self.hof.x\_size-1,0),(self.hof.x\_size-2,0))) # Ecke rechts oben clearen, wenn sie in Phase 4 nicht Source-Feld wird

# Nicht-Eckfelder der obersten Reihe auf Source-Felder konzentrieren:

for start\_x in range(0,self.hof.x\_size-1):

if (start\_x+1,0) in source\_fields\_for\_edgeclear:

break

else:

operationen = [dict(feld0=(start\_x,0), blow\_direction=(1,0)), dict(feld0=(start\_x+2,2), blow\_direction=(0,-1))]

muster = Muster(self, [(start\_x+1,0), (start\_x+2,1)], operationen, self.tolerated\_amount) # Muster zum Leeren des Eckfelds (start\_x+1,0) und zum gleichzeitigen Beseitigen des entstehenden Seitenabtriebs auf Feld (start\_x+2,0)

self.add\_operation(muster)

for start\_x in range(self.hof.x\_size-1,0,-1):

if (start\_x-1,0) in source\_fields\_for\_edgeclear:

break

else:

operationen = [dict(feld0=(start\_x,0), blow\_direction=(-1,0)), dict(feld0=(start\_x-2,2), blow\_direction=(0,-1))]

muster = Muster(self, [(start\_x-1,0), (start\_x-2,1)], operationen, self.tolerated\_amount)

self.add\_operation(muster)

def transfer\_to\_Q(self):

"""

4. Phase des generalisierten Ablaufs: Verschiebt das gesamte Laub auf Feld Q. Fügt die hierfür notwendigen Operationen zu self.strategy hinzu.

"""

Q = self.Q

self.add\_operation(self.edge\_to\_mid((Q[0],0), (Q[0],1))) # Muster hinzufügen, dass Laub vom Eckfeld (Q[0],0) auf das Feld (Q[0],1) bläst

if Q == (Q[0],1):

if not self.hof.is\_edge((Q[0],2)):

if self.hof.does\_exist((Q[0],4)):

self.add\_operation(Muster(self, [(Q[0],2)], [dict(feld0=(Q[0],4), blow\_direction=(0,-1))], self.tolerated\_amount))

else:

self.add\_operation(dict(feld0=(Q[0],3), blow\_direction=(0,-1)))

else:

self.add\_operation(dict(feld0=(Q[0],0), blow\_direction=(0,1))) # Das Laub wird vom Rand aus auf (Q[0],2) geblasen - dabei entstehen aber Seitenabtriebe. Mit diesen wird im Folgenden umgegangen.

# Mit den in der vorherigen Blasoperation entstandenen Seitenabtrieben umgehen: Wenn möglich wird das Laub verlustfrei durch den Abtrieb bei B vorne auf das Feld (Q[0],2) geblasen. Wenn nicht, dann wird das im vorherigen Schritt abgetriebene Laub auf den Felder (Q[0]-1,3), (Q[0],3) und (Q[0]+1,3) versammelt.

cleared\_fields = []

if self.hof.does\_exist((Q[0]+3,2)):

muster = Muster(self, [(Q[0]+1,2)], [dict(feld0=(Q[0]+3,2),blow\_direction=(-1,0))], self.tolerated\_amount)

self.add\_operation(muster)

cleared\_fields.append((Q[0]+1,2))

elif not (self.hof.x\_size == 5 and self.hof.y\_size == 5 and Q == (2,2)): # In diesem Fall wird anders vorgegangen (siehe Code unten)

muster = Muster(self, [(Q[0]+1,2)], [dict(feld0=(Q[0]+1,0),blow\_direction=(0,1))], self.tolerated\_amount)

self.add\_operation(muster)

if self.hof.does\_exist((Q[0]-3,2)):

muster = Muster(self, [(Q[0]-1,2)], [dict(feld0=(Q[0]-3,2),blow\_direction=(1,0))], self.tolerated\_amount)

self.add\_operation(muster)

cleared\_fields.append((Q[0]-1,2))

elif not (self.hof.x\_size == 5 and self.hof.y\_size == 5 and Q == (2,2)): # In diesem Fall wird anders vorgegangen (siehe Code unten)

muster = Muster(self, [(Q[0]-1,2)], [dict(feld0=(Q[0]-1,0),blow\_direction=(0,1))], self.tolerated\_amount)

self.add\_operation(muster)

if self.hof.does\_exist((Q[0],5)):

if Q != (Q[0],3):

muster = Muster(self, [(Q[0],3)],  
 [dict(feld0=(Q[0],5),blow\_direction=(0,-1))],   
 self.tolerated\_amount)

self.add\_operation(muster)

cleared\_fields.append((Q[0],3))

elif Q == (Q[0],3):

cleared\_fields.append((Q[0],3))

if len(cleared\_fields) != 3:

if (self.hof.does\_exist((Q[0],6)) or ((Q[0], 3) == Q and   
 self.hof.does\_exist((Q[0],5)))) and   
 (self.hof.does\_exist((Q[0]-3, 3)) or  
 self.hof.does\_exist((Q[0]+3, 3))):

if self.hof.does\_exist((Q[0]-3, 3)):

self.add\_operation(dict(feld0=(Q[0]+2,3),   
 blow\_direction=(-1,0)))

self.add\_operation(Muster(self, [(Q[0]-1, 3)],   
 [dict(feld0=(Q[0]-3,3), blow\_direction=(1,0))],   
 self.tolerated\_amount))

else:

self.add\_operation(dict(feld0=(Q[0]-2,3),   
 blow\_direction=(1,0)))

self.add\_operation(Muster(self, [(Q[0]+1, 3)],   
 [dict(feld0=(Q[0]+3,3), blow\_direction=(-1,0))],  
 self.tolerated\_amount))

self.add\_operation(Muster(self, [(Q[0], 4)],   
 [dict(feld0=(Q[0],6), blow\_direction=(0,-1))],  
 self.tolerated\_amount))

if not Q != (Q[0], 3):

self.add\_operation(Muster(self, [(Q[0], 3)],   
 [dict(feld0=(Q[0],5), blow\_direction=(0,-1))],   
 self.tolerated\_amount))

elif self.hof.does\_exist((Q[0],5)):

operations = []

operations.append(dict(feld0=(Q[0]-2,3),   
 blow\_direction=(1,0)))

operations.append(dict(feld0=(Q[0]+2,3), blow\_direction=(-  
 1,0)))

operations.append(dict(feld0=(Q[0],5), blow\_direction=(0,-  
 1)))

source\_fields = [(Q[0]-1,3), (Q[0],3), (Q[0]+1,3), (Q[0],4)]

self.add\_operation(Muster(self, source\_fields, operations, self.tolerated\_amount))

elif self.hof.x\_size == 5 and self.hof.y\_size == 5 and Q == (2,2):

self.add\_operation(dict(feld0=(0,2), blow\_direction=(1,0)))

if self.choose\_faster\_path:

self.add\_operation(dict(feld0=(2,4), blow\_direction=(0,-1)))

self.num\_rotations += 1

self.add\_operation(Muster(self, [(1,2)], [dict(feld0=(1,4), blow\_direction=(0,-1))], self.tolerated\_amount))

self.add\_operation(Muster(self, [(3,2)], [dict(feld0=(3,4), blow\_direction=(0,-1))], self.tolerated\_amount))

operations = []

operations.append(Muster(self, [(1,1),(2,1),(3,1)], [

dict(feld0=(0,1), blow\_direction=(1,0)),

dict(feld0=(4,1), blow\_direction=(-1,0)),

], self.tolerated\_amount)

)

operations.append(self.edge\_to\_mid((2,0), (2,1)))

self.add\_operation(Muster(self, [(1,1), (2,1), (3,1), (2,0)], operations, self.tolerated\_amount))

if Q != (Q[0], 2):

for y\_start in range(0,Q[1]-2):

self.add\_operation(Muster(self, [(Q[0], y\_start+2)], [dict(feld0=(Q[0], y\_start), blow\_direction=(0,1))], self.tolerated\_amount))

## Erweiterung 2 (basierend auf Ansatz 2)

**Veränderte Funktionen zum Laubblasen, die jetzt zwei Laubbläser gleichzeitig behandeln (Teil der Hof-Klasse):**

def run\_single\_blasoperation(self, feld0 : tuple, blow\_direction : tuple, blocked\_fields : list[tuple], \*, remaining\_single\_ops=0):

"""

Führt die Blasoperation eines einzigen Laubbläsers aus und aktualisiert self.felder entsprechend.

Args:

feld0 (tuple), blow\_direction (tuple): Index von Feld 0 und Blasrichtung

blocked\_fields (list[tuple]): Felder, auf die kein Laub gelangen darf (da sie im Einflussgebiets eines anderen Laubbläsers liegen) und die daher wie Randfelder behandelt werden

"""

if not blow\_direction in [(0,1),(0,-1),(1,0),(-1,0)]:

return # -> Ungültige Blasrichtung. Der Laubbläser kann nur nach rechts, links, oben und unten blasen.

# Richtung, die orthogonal zur Blasrichtung ist, ermitteln:

orthogonal\_direction = self.get\_orthogonal\_direction(blow\_direction)

# Feld A (Feld unmittelbar vor dem Laubbläser) ermitteln:

feldA = (feld0[0]+blow\_direction[0], feld0[1]+blow\_direction[1])

if not self.does\_exist(feldA):

return # -> Es gibt kein Feld vor dem Laubbläser bzw. der Laubbläser bläst gegen die Umrandung. Für diesen Fall ist definiert, dass sich die Verteilung des Laubs nicht verändert

new\_feldA\_value = 0 # In dieser Variable wird die neue Anzahl an Blättern auf Feld A gespeichert

# Feld B (Feld hinter Feld A) ermitteln:

feldB = (feldA[0]+blow\_direction[0], feldA[1]+blow\_direction[1])

if self.does\_exist(feldB) and feldB not in blocked\_fields:

# -> Es gibt ein Feld B. Neue Anzahl an Blättern auf Feld B ermitteln:

if self.rules.use\_binomial:

# -> Zum Modellieren der Blätteranzahlen sollen ganze Zahlen verwendet werden, die über die Binomialverteilung bestimmt werden

A\_seitenabtrieb\_1, \_ = binomial\_likeliest(n=self.felder[feldA], p=self.rules.A\_seitenabtrieb, rank=self.rules.binomial\_rank, handle\_ties=self.rules.binomial\_handle\_ties) # Anzahl an Blättern von Feld A, die auf dem einen Feld neben Feld B landen (Seitenabtrieb 1)

A\_seitenabtrieb\_2, \_ = binomial\_likeliest(n=self.felder[feldA]-A\_seitenabtrieb\_1, p=self.rules.A\_seitenabtrieb/(1-self.rules.A\_seitenabtrieb), rank=self.rules.binomial\_rank, handle\_ties=self.rules.binomial\_handle\_ties) # Anzahl an Blättern von Feld A, die auf dem anderen Feld neben Feld B landen (Seitenabtrieb 2)

B\_vorne\_abtrieb, \_ = binomial\_likeliest(n=self.felder[feldB], p=self.rules.B\_vorne\_abtrieb, rank=self.rules.binomial\_rank, handle\_ties=self.rules.binomial\_handle\_ties) # Anzahl an Blättern von Feld B, die nach vorne abgetrieben werden

else:

# -> Zum Modellieren der Blätteranzahlen werden die Erwartungswerte (als Fließkommazahlen) verwendet

A\_seitenabtrieb\_1 = self.felder[feldA] \* self.rules.A\_seitenabtrieb

A\_seitenabtrieb\_2 = A\_seitenabtrieb\_1

B\_vorne\_abtrieb = self.felder[feldB] \* self.rules.B\_vorne\_abtrieb

new\_feldB\_value = self.felder[feldA] - (A\_seitenabtrieb\_1 + A\_seitenabtrieb\_2) + self.felder[feldB] - B\_vorne\_abtrieb # Neue Anzahl an Blättern auf Feld B

# Nachbarfeld von Feld B, das Feld A gegenüberliegt, aktualisieren (sofern vorhanden):

n1 = (feldB[0]+blow\_direction[0], feldB[1]+blow\_direction[1]) # Index des Nachbarfelds

if self.does\_exist(n1) and n1 not in blocked\_fields:

self.felder[n1] += B\_vorne\_abtrieb

else:

new\_feldB\_value += B\_vorne\_abtrieb

# Nachbarfelder von Feld B, die Feld A nicht gegenüberliegen, aktualisieren (sofern vorhanden):

n1 = (feldB[0]+orthogonal\_direction[0], feldB[1]+orthogonal\_direction[1])

if self.does\_exist(n1) and n1 not in blocked\_fields:

self.felder[n1] += A\_seitenabtrieb\_1

else:

new\_feldB\_value += A\_seitenabtrieb\_1

n2 = (feldB[0]-orthogonal\_direction[0], feldB[1]-orthogonal\_direction[1])

if self.does\_exist(n2) and n2 not in blocked\_fields:

self.felder[n2] += A\_seitenabtrieb\_2

else:

new\_feldB\_value += A\_seitenabtrieb\_2

# Feld B aktualsieren

self.felder[feldB] = new\_feldB\_value

else:

# -> Es gibt kein Feld B.

if self.rules.use\_binomial:

A\_noB\_seitenabtrieb\_1, \_ = binomial\_likeliest(n=self.felder[feldA], p=self.rules.A\_noB\_seitenabtrieb, rank=self.rules.binomial\_rank, handle\_ties=self.rules.binomial\_handle\_ties)

A\_noB\_seitenabtrieb\_2, \_ = binomial\_likeliest(n=self.felder[feldA]-A\_noB\_seitenabtrieb\_1, p=self.rules.A\_noB\_seitenabtrieb/(1-self.rules.A\_noB\_seitenabtrieb), rank=self.rules.binomial\_rank, handle\_ties=self.rules.binomial\_handle\_ties)

else:

A\_noB\_seitenabtrieb\_1 = self.felder[feldA] \* self.rules.A\_noB\_seitenabtrieb

A\_noB\_seitenabtrieb\_2 = A\_noB\_seitenabtrieb\_1

new\_feldA\_value = self.felder[feldA] - (A\_noB\_seitenabtrieb\_1 + A\_noB\_seitenabtrieb\_2)

# Nachbarfelder von Feld A, die dem laubblasenden Hausmeister nicht gegenüberliegen, aktualisieren, sofern vorhanden:

n1 = (feldA[0]+orthogonal\_direction[0], feldA[1]+orthogonal\_direction[1])

if self.does\_exist(n1) and n1 not in blocked\_fields:

self.felder[n1] += A\_noB\_seitenabtrieb\_1

else:

# -> Feld A ist ein Eckfeld, ein Teil des Laubs verbleibt also auf Feld A.

new\_feldA\_value += A\_noB\_seitenabtrieb\_1

n2 = (feldA[0]-orthogonal\_direction[0], feldA[1]-orthogonal\_direction[1])

if self.does\_exist(n2) and n2 not in blocked\_fields:

self.felder[n2] += A\_noB\_seitenabtrieb\_2

else:

# -> Feld A ist ein Eckfeld, ein Teil des Laubs verbleibt also auf Feld A.

new\_feldA\_value += A\_noB\_seitenabtrieb\_2

# Neuen Wert von A zurückgeben (Aktualisierung des Feldwerts erfolgt am Ende des insgesamten Blasvorgangs, damit die verschiedenen Blasvorgänge alle vom selben Ausgangspunkt ausgehen):

return new\_feldA\_value

def blase(self, feld0\_b1, blow\_direction\_b1, feld0\_b2, blow\_direction\_b2):

"""

Simuliert einen Blasvorgang (mit zwei Laubbläsern) und aktualisiert self.felder auf die resultierende Blattverteilung.

Args:

feld0\_b1 (int): Index des Felds, auf dem der Hausmeister 1 mit Bläser 1 steht

blow\_direction\_b1 (tuple): Richtung, in die der Hausmeister 1 mit Bläser 1 bläst. Kann folgende Werte annehmen: (1,0) =rechts, (-1,0) (=links), (0,1) (=unten), (0,-1) (=oben)

feld0\_b2 (int): Index des Felds, auf dem der Hausmeister 2 mit Bläser 2 steht

blow\_direction\_b2 (tuple): Richtung, in die der Hausmeister 2 mit Bläser 2 bläst. Kann folgende Werte annehmen: (1,0) =rechts, (-1,0) (=links), (0,1) (=unten), (0,-1) (=oben)

"""

if feld0\_b2 is None:

feldA\_b1 = (feld0\_b1[0]+blow\_direction\_b1[0], feld0\_b1[1]+blow\_direction\_b1[1])

self.felder[feldA\_b1] = self.run\_single\_blasoperation(feld0\_b1, blow\_direction\_b1, [])

return

if feld0\_b1 == feld0\_b2:

return None # Laubbläser dürfen nicht auf selbem Feld stehen

pd = [(0,1),(0,-1),(1,0),(-1,0)] # Mögliche Blasrichtungen

if not blow\_direction\_b1 in pd and blow\_direction\_b2 in pd:

return # -> Ungültige Blasrichtung. Der Laubbläser kann nur nach rechts, links, oben und unten blasen.

self.blas\_counter += 1 # Zähler, der durchgeführte Blasoperationen zählt, erhöhen

self.blas\_log.append(dict(feld0\_b1=feld0\_b1, blow\_direction\_b1=blow\_direction\_b1, feld0\_b2=feld0\_b2, blow\_direction\_b2=blow\_direction\_b2)) # Blasoperation loggen

feldA\_b1 = (feld0\_b1[0]+blow\_direction\_b1[0], feld0\_b1[1]+blow\_direction\_b1[1])

if not self.does\_exist(feldA\_b1): # Überprüfen, ob das ermittelte Feld tatsächlich existiert

feldA\_b1 = None

feldA\_b2 = (feld0\_b2[0]+blow\_direction\_b2[0], feld0\_b2[1]+blow\_direction\_b2[1])

if not self.does\_exist(feldA\_b2):

feldA\_b2 = None

if feldA\_b1 == feld0\_b2 and feld0\_b1 == feldA\_b2:

return

# Blasoperation für den ersten Laubbläser (b1) ausführen:

if feldA\_b2 == feldA\_b1:

half\_feldA\_value = round(self.felder[feldA\_b2]/2)

self.felder[feldA\_b2] -= half\_feldA\_value

blocked\_fields = [feldA\_b2]

if manhatten\_distance(feld0\_b2, feld0\_b1) < manhatten\_distance(feld0\_b2, (feld0\_b1[0]+blow\_direction\_b2[0], feld0\_b1[1]+blow\_direction\_b2[1])):

blocked\_fields.append(feld0\_b2)

new\_feldA\_b1\_value = self.run\_single\_blasoperation(feld0\_b1, blow\_direction\_b1, blocked\_fields)

if feldA\_b2 == feldA\_b1:

self.felder[feldA\_b2] = half\_feldA\_value

blocked\_fields = [feldA\_b1]

if manhatten\_distance(feld0\_b1, feld0\_b2) < manhatten\_distance(feld0\_b1, (feld0\_b2[0]+blow\_direction\_b1[0], feld0\_b2[1]+blow\_direction\_b1[1])):

blocked\_fields.append(feld0\_b1)

new\_feldA\_b2\_value = self.run\_single\_blasoperation(feld0\_b2, blow\_direction\_b2, blocked\_fields)

if feldA\_b2 == feldA\_b1:

self.felder[feldA\_b1] = new\_feldA\_b1\_value + new\_feldA\_b2\_value

else:

if not new\_feldA\_b1\_value is None:

self.felder[feldA\_b1] = new\_feldA\_b1\_value

if not new\_feldA\_b2\_value is None:

self.felder[feldA\_b2] = new\_feldA\_b2\_value

1. Allgemein lässt sich mit dem im Folgenden beschriebenen Vorgehen ein Zufallsexperiment mit n möglichen Ereignissen in n-1 miteinander verknüpfte Bernoulli-Experimente umformen. [↑](#footnote-ref-2)
2. P(¬E1 ∩ E2) = P(E2), da auf alle Blätter, auf die E2 (Blatt liegt auf dem Feld über B) zutrifft, E1 (Blatt liegt auf B) automatisch nicht zutrifft – das Blatt kann (dem Modell nach) schließlich nicht auf beiden Feldern gleichzeitig liegen. [↑](#footnote-ref-3)
3. Die Modellierung als Bernoulli-Kette ist zulässig unter der Annahme, dass sich die Blätter in ihrer Flugbahn nicht gegenseitig beeinflussen und es sich somit um voneinander unabhängige Bernoulli-Experimente handelt. [↑](#footnote-ref-4)
4. Als “Blasprozess” bezeichne ich den ganzen Prozess, während dem der Hof aufgeräumt wird. Ein Blasprozess besteht aus vielen Blasoperationen / Blasvorgängen. [↑](#footnote-ref-5)
5. Bei Simulation des Zufalls würde eine Aussage wie „die Blattmenge konvergiert gegen …“ gar keinen Sinn machen. [↑](#footnote-ref-6)