

Game of Life

1 Grundlagen

Auf diesem Arbeitsblatt wollen wir das *Game of Life* programmieren. Bei diesem wird ein rechteckiges Spielfeld betrachtet, das aus einzelnen Zellen besteht. Diese Zellen sind entweder tot oder lebendig.

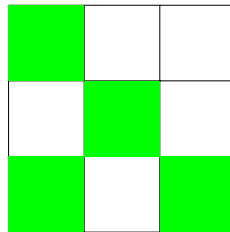


Abbildung 1: 3x3-Spielfeld mit 4 lebenden Zellen

Wir werden das Spielfeld mit einer 2D-Liste von Booleans darstellen. `true` steht für eine lebende und `false` für eine tote Zelle. Das Feld aus der letzten Skizze entspricht der folgenden 2D-Liste.

```
listOf(listOf(true, false, false),  
        listOf(false, true, false),  
        listOf(true, false, true))
```

```
[[true, false, false], [false, true, false], [true, false, true]]
```

Der Benutzer des Programms soll die Möglichkeit haben Zellen auszuwählen und ihren Zustand zu verändern. Der Zustand der ausgewählten Zelle kann nicht erkannt werden.

Tipp: Wenn du vor dem Bearbeiten dieses Arbeitsblatt das Spiel 2048 programmiert hast, kannst du die Funktionen `repeat`, `setAt`, und `repeat2D` für dieses Projekt verwenden.

2 Terminal-Grafik

2.1 Aufgabe

Implementiere eine Funktion `showCell`. Diese gibt die Darstellung einer Zelle als String zurück. Jede Zelle wird mit 2 Charactern dargestellt, um auszugleichen, dass ein Zeichen im

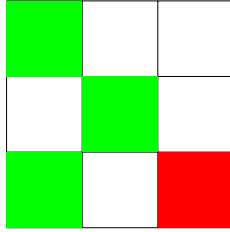


Abbildung 2: 3x3-Spielfeld mit 3 sichtbaren lebenden Zellen. Die Zelle rechts unten ist gerade ausgewählt.

Terminal ungefähr doppelt so hoch wie breit angezeigt wird.

Die Funktion hat die folgenden Parameter

- der Zeilenindex und
- der Spaltenindex

der aktuell ausgewählten Zelle.

- der Zeilenindex und
- der Spaltenindex
- der Wert

der Zelle, die dargestellt werden soll.

Die ausgewählte Zelle wird als ><, eine lebende Zelle als ■ und eine tote Zelle als zwei Leerzeichen dargestellt.

```
1 showCell(1, 1, 1, 1, true)
```

```
><
```

```
1 showCell(2, 3, 2, 3, false)
```

```
><
```

```
1 showCell(1, 1, 2, 3, false)
```

```
1 showCell(1, 1, 2, 3, true)
```

```
■
```

2.2 Aufgabe

Implementiere eine Funktion `showRow`. Diese gibt die Darstellung einer Zeile des Spielfelds als String zurück. Das erste und das letzte Zeichen in einer Zeile ist ein senkrechter Strich `|`. Die Funktion hat die folgenden Parameter:

- der Zeilenindex und
- der Spaltenindex der aktuell ausgewählten Zelle
- der Zeilenindex der Zeile, die dargestellt werden soll
- die Zeile selbst.

```
1 showRow(0, 1, 0, listOf(true, false))
```

```
|■><|
```

```
1 showRow(2, 0, 1, listOf(true, false, false, true))
```

```
|■ ■|
```

```
1 showRow(1, 2, 1, listOf(true, false, false, true))
```

```
|■ ><■|
```

Hinweis: Nutze `showCell`!

2.3 Aufgabe

Implementiere eine Funktion `rowsAsStrings`.

Die Funktion hat die folgenden Parameter:

- der Zeilenindex und
- der Spaltenindex der aktuell ausgewählten Zelle
- das Spielfeld

Sie gibt eine Liste zurück. In dieser sind die Zeilen des Spielfelds als Strings dargestellt. Der erste und der letzte String in der Liste bestehen aus Bindestrichen und sind genauso lang wie die übrigen Elemente der Liste.

```
1 rowsAsStrings(0, 1, listOf(listOf(true, false),
2                               listOf(false, true)))
```

```
[------, |■><|, | ■|, ------]
```

```

1 rowsAsStrings(0, 2,
2     listOf(listOf(true, false, false),
3         listOf(false, true, true),
4         listOf(true, true, true)))

[-----, |■ >|, | ■■■|, |■■■■|, -----]

```

```

1 rowsAsStrings(1, 1,
2     listOf(listOf(true, false, false),
3         listOf(false, true, true),
4         listOf(true, true, true)))

[-----, |■ |, | >■|, |■■■■|, -----]

```

Hinweis: Nutze showRow!

3 Zellen ändern

3.1 Aufgabe

Implementiere eine Funktion `toggle`. Dieser wird ein Zeilenindex, ein Spaltenindex und eine nicht leere 2D-Liste von Booleans übergeben. Sie gibt eine Liste zurück, in der das Boolean ausgetauscht wird, dessen Indizes mit den übergebenen Indizes übereinstimmen. Alle anderen Booleans werden übernommen.

```

1 toggle(0, 0, listOf(listOf(false)))

[[true]]

1 toggle(1, 1, listOf(listOf(true), listOf(false, true)))

[[true], [false, false]]

```

Hinweis: Nutze `setAt`!

4 Spielfeld anlegen

4.1 Aufgabe

Implementiere eine Funktion `startBoard`. Diese gibt ein Spielfeld aus toten Zellen zurück. Ihr werden die gewünschte Zeilen- und Spaltenanzahl übergeben.

```
1 startBoard(2, 3)
```

```
[[false, false, false], [false, false, false]]
```

```
1 startBoard(3, 4)
```

```
[[false, false, false, false], [false, false, false, false], [false, false, false, false]]
```

Hinweis: Nutze repeat2D!

5 Spiellogik

In den folgenden Aufgaben wollen wir die Spielregeln implementieren. Für die Regeln ist relevant, wie viele lebende Nachbarn eine Zelle hat.

Die Nachbarn einer Zelle sind die Zellen, die diese berühren. Z. B. hat die blaue Zelle in der folgenden Skizze acht Nachbarzellen.

1	2	3
4		5
6	7	8

Zellen am Rand oder in einem Eck haben weniger Nachbarn.

	1	
2	3	

Für die weiteren Regeln ist relevant, wie viele lebende Nachbarn eine Zelle hat. In der folgenden Skizze sind die lebenden Zellen grün markiert.

In diesem Beispiel hat die mittlere Zelle drei lebende Nachbarn. Die Zelle links oben hat nur einen lebenden Nachbarn.

5.1 Aufgabe

Implementiere eine Funktion `countLivingNeighbors`. Dieser werden ein Zeilenindex, ein Spaltenindex und das Spielfeld übergeben. Sie gibt die Anzahl der lebenden Nachbarn der Zelle mit diesen Indizes zurück.

```
1 countLivingNeighbors(1, 2,  
2     listOf(listOf(true, false, false),  
3             listOf(false, true, false),  
4             listOf(true, false, true)))
```

2

```
1 countLivingNeighbors(1, 1, listOf(listOf(true, false),  
2                                   listOf(false, true)))
```

1

```
1 countLivingNeighbors(1, 1,  
2     listOf(listOf(true, false, false),  
3             listOf(false, true, true),  
4             listOf(true, true, true)))
```

5

Hinweis Überlege dir welche Indizes die Nachbarn einer Zelle mit Zeilenindex i und Spaltenindex j haben.

	(i, j)	

5.2 Aufgabe

Eine momentan lebende Zelle lebt in der nächsten Runde des Spiels, wenn sie 2 oder 3 lebende Nachbarn hat. Tote Zellen leben in der nächsten Runde, wenn sie genau 3 lebende Nachbarn haben.

1	2	3
4	5	6
7	8	9

In diesem Beispiel gilt:

- Zelle 1 ist in der nächsten Runde tot, weil sie nur einen lebenden Nachbarn hat.
- Zelle 2 ist in der nächsten Runde immer noch tot, weil sie nur zwei lebende Nachbarn hat.
- Zelle 4 lebt in der nächsten Runde, weil sie genau drei lebende Nachbarn hat.
- Zelle 5 hat drei lebende Nachbarn und ist deshalb in der nächsten Runde am Leben.

Insgesamt sieht das Spielfeld in der nächsten Runde folgendermaßen aus.

Implementiere eine Funktion `livesNextRound`. Dieser werden ein Zeilenindex, ein Spaltenindex und das Spielfeld übergeben. Sie gibt zurück, ob die Zelle mit diesen Indizes in der nächsten Runde lebt.

```
1 livesNextRound(1, 1,  
2     listOf(listOf(true, false),  
3             listOf(false, true)))
```

false

```
1 livesNextRound(1, 1,  
2     listOf(listOf(true, false, false),  
3             listOf(false, true, true),  
4             listOf(true, true, true)))
```

false

```
1 livesNextRound(1, 2,  
2     listOf(listOf(true, false, false),  
3             listOf(false, true, true),  
4             listOf(true, true, true)))
```

true

Hinweis: Nutze `countLivingNeighbors`!

5.3 Aufgabe

Implementiere eine Funktion `computeUpdatedRow`. Dieser werden ein Zeilenindex und das Spielfeld übergeben. Sie berechnet diese Zeile für die nächste Runde.

```
1 computeUpdatedRow(1, listOf(listOf(true, false),
2                               listOf(false, true)))
```

[false, false]

```
1 computeUpdatedRow(2,
2   listOf(listOf(true, false, false),
3           listOf(false, true, true),
4           listOf(true, true, true)))
```

[true, false, true]

```
1 computeUpdatedRow(0,
2   listOf(listOf(true, false, false),
3           listOf(false, true, true),
4           listOf(true, true, true)))
```

[false, true, false]

Hinweis: Nutze `livesNextRound`!

5.4 Aufgabe

Implementiere eine Funktion `computeNextBoard`. Dieser wird das Spielfeld übergeben. Sie berechnet das nächste Spielfeld.

```
1 computeNextBoard(listOf(listOf(true, false),
2                             listOf(false, true)))
```

[[false, false], [false, false]]

```
1 computeNextBoard(
2   listOf(listOf(true, false, false),
3           listOf(false, true, true),
4           listOf(true, true, true)))
```

[[false, true, false], [false, false, true], [true, false, true]]


```

1 computeNextBoard(
2     listOf(listOf(true, false, false),
3             listOf(false, true, true),
4             listOf(true, true, true)))

```

```
[[false, true, false], [false, false, true], [true, false, true]]
```

Hinweis: Nutze computeUpdatedRow!

6 Vektoren

6.1 Aufgabe

Erstelle eine Datenklasse V2. Die Objekte dieser Klasse sind Vektoren im zweidimensionalen Raum mit ganzzahligen Komponenten. Beide Komponenten sind unveränderlich.

```
1 V2(3, 2)
```

```
V2(x=3, y=2)
```

```
1 V2(1, -5)
```

```
V2(x=1, y=-5)
```

6.2 Aufgabe

Erweitere die Datenklasse V2 um die Methode plus. Dieser wird ein Vektor übergeben. Es wird die Summe der beiden Vektoren zurückgegeben.

```
1 V2(3, 2).plus(V2(1, -5))
```

```
V2(x=4, y=-3)
```

```
1 V2(3, 4).plus(V2(-4, -5))
```

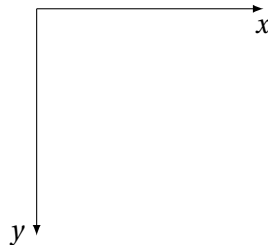
```
V2(x=-1, y=-1)
```

Hinweis:

$$\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} + \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} v_1 + w_1 \\ v_2 + w_2 \end{pmatrix}$$

6.3 Aufgabe

Implementiere eine Funktion `charToDirectionVector`. Diese bestimmt den Richtungsvektor zu einem Character, der für eine Richtung steht. Wenn nicht w, a, s oder d gedrückt wurde, wird der Nullvektor zurückgegeben. Dabei werden Terminal-Koordinaten verwendet. Der Unterschied zu dem Koordinatensystem aus dem Matheunterricht ist die Richtung der y-Achse.



```
1 charToDirectionVector('w')
```

V2(x=0, y=-1)

```
1 charToDirectionVector('a')
```

V2(x=-1, y=0)

```
1 charToDirectionVector('s')
```

V2(x=0, y=1)

```
1 charToDirectionVector('d')
```

V2(x=1, y=0)

```
1 charToDirectionVector('i')
```

V2(x=0, y=0)

6.4 Aufgabe

Implementiere eine Funktion `clipToListBounds`. Dieser werden eine Zahl und eine Liste übergeben. Wenn die Zahl ein Index der Liste ist, wird diese wieder zurückgegeben. Wenn sie kleiner als 0 ist, wird 0 zurückgegeben. Wenn sie größer als der größte Index der Liste ist, wird dieser Index zurückgegeben.

```
1 clipToListBounds(1, listOf(5, 6, 7))
```

1

```
1 clipToListBounds(-1, listOf('d', 'e'))
```

0

```
1 clipToListBounds(4, listOf(9, 29))
```

1

6.5 Aufgabe

Implementiere eine Funktion `clipToListBounds`. Dieser werden ein Vektor und eine 2D-Liste übergeben. Sie gibt einen Vektor zurück, in dem die y -Koordinate mit `clipToListBounds` an die Zeilenindizes der 2D-Liste angepasst wurde. Außerdem wurde die x -Koordinate mit `clipToListBounds` an die Spaltenindizes der 2D-Liste angepasst.

```
1 clipToListBounds(V2(2,1), listOf(listOf(5, 6, 7),listOf(2, 3, 4)))
```

V2(x=2, y=1)

```
1 clipToListBounds(V2(3,2), listOf(listOf(true, true),listOf(false, true), listOf(true, false))
```

V2(x=1, y=2)

```
1 clipToListBounds(V2(4,4), listOf(listOf(true, true),listOf(false, true), listOf(true, false))
```

V2(x=1, y=2)

Hinweis: Nutze `clipToListBounds`.

7 Klasse Model

7.1 Aufgabe

Implementiere eine Klasse `Model`. Die Attribute sind

- `gameBoard`: Eine zweidimensionale Liste aus Booleans
- `selected`: Ein Vektor der angibt, welche Zelle ausgewählt ist. Die y -Koordinate steht für die ausgewählte Zeile und die x -Koordinate steht für die ausgewählte Spalte.

```
1 Model(listOf(listOf(true, false), listOf(true, true)), V2(1, 0))
```

```
Model(gameBoard=[[true, false], [true, true]], selected=V2(x=1, y=0))
```

7.2 Aufgabe

Erweitere die Klasse `Model` um die Methode `moveSelection`. Dieser wird ein Character übergeben, der für eine Richtung steht. Sie gibt das neue `Model` mit eventuell anderen ausgewählten Zelle zurück.

```
1 Model(listOf(listOf(true, false), listOf(true, true)), V2(1, 0)).moveSelection('s')
```

```
Model(gameBoard=[[true, false], [true, true]], selected=V2(x=1, y=1))
```

```
1 Model(listOf(listOf(true, false), listOf(true, true)), V2(1, 0)).moveSelection('d')
```

```
Model(gameBoard=[[true, false], [true, true]], selected=V2(x=1, y=0))
```

Hinweis: Nutze die Vektoraddition und `clipToListBounds`.

7.3 Aufgabe

Erweitere die Klasse `Model` um die Methode `toggleSelectedCell`. Diese gibt ein neues `Model` zurück. Bei diesem wurde der Wert der aktuell ausgewählten Zelle geändert.

```
1 Model(listOf(listOf(true, false), listOf(true, true)), V2(1, 0)).toggleSelectedCell()
```

```
Model(gameBoard=[[true, true], [true, true]], selected=V2(x=1, y=0))
```

```
1 Model(listOf(listOf(true, false), listOf(true, true)), V2(0, 1)).toggleSelectedCell()
```

```
Model(gameBoard=[[true, false], [false, true]], selected=V2(x=0, y=1))
```

Hinweis: Nutze `toggle`.

7.4 Aufgabe

Erweitere die Klasse `Model` um die Methode `updateBoard`. Diese gibt ein neues `Model` zurück. Bei diesem wurde das Spielfeld mit `computeNextBoard` aktualisiert.

```
1 Model(listOf(listOf(true, false), listOf(true, true)), V2(1, 0)).updateBoard()
```

```
Model(gameBoard=[[true, true], [true, true]], selected=V2(x=1, y=0))
```

```

1 Model(listOf(listOf(true, true), listOf(false, true)), V2(0, 1)).updateBoard()

Model(gameBoard=[[true, true], [true, true]], selected=V2(x=0, y=1))

```

Hinweis: Nutze computeNextBoard.

7.5 Aufgabe

Erweitere die Klasse Model um die Methode update. Der Methode wird ein Character übergeben. Sie gibt ein neues Model zurück. .

- Wenn n eingegeben wurde, wurde die nächste Generation des Spielfelds berechnet.
- Wenn t eingegeben wurde, wurde der Wert der aktuell ausgewählten Zelle gewechselt.
- Bei einer Richtungstaste ändert sich die Position der aktuell ausgewählten Zelle. Ansonsten wird das unveränderte Board zurückgegeben.

```

1 Model(listOf(listOf(true, false), listOf(true, true)), V2(1, 0)).update('n')

Model(gameBoard=[[true, true], [true, true]], selected=V2(x=1, y=0))

1 Model(listOf(listOf(true, false), listOf(true, true)), V2(1, 0)).update('t')

Model(gameBoard=[[true, true], [true, true]], selected=V2(x=1, y=0))

1 Model(listOf(listOf(true, false), listOf(true, true)), V2(1, 0)).update('d')

Model(gameBoard=[[true, false], [true, true]], selected=V2(x=1, y=0))

1 Model(listOf(listOf(true, false), listOf(true, true)), V2(1, 0)).update('o')

Model(gameBoard=[[true, false], [true, true]], selected=V2(x=1, y=0))

```

Hinweis: Nutze die bisher geschriebenen Methoden der Klasse Model.

7.6 Aufgabe

Erweitere die Klasse Model um die Methode rowsAsStrings. Sie gibt eine Liste zurück. In dieser sind die Zeilen des Spielfelds als Strings dargestellt

```

1 Model(listOf(listOf(true, false), listOf(true, true)), V2(1, 0)).rowsAsStrings()

[-----, |■><|, |■■■|, -----]

```

Hinweis: Nutze rowsAsStrings!

7.7 Aufgabe

Erweitere die Klasse `Model` um einen sekundären Konstruktor. Diesem werden eine Zeilen- und eine Spaltenanzahl übergeben. Er erzeugt eine Instanz von `Model`, bei der das Spielfeld die gewünschten Dimensionen hat. Auf dem Spielfeld gibt es keine lebenden Zellen. Die Zelle links oben ist ausgewählt.

```
1 Model(3, 4)
```

```
Model(gameBoard=[[false, false, false, false], [false, false, false, false], [false, false, false, false], [false, false, false, false]])
```

Hinweis: Nutze `startBoard`!

8 Ausgabe

8.1 Aufgabe

Implementiere eine Funktion `printStrings`. Diese gibt eine Liste von Strings zeilenweise in der Konsole aus.

```
1 printStrings(listOf("akjefhkajsd", "adfsf", "12e124"))
```

```
akjefhkajsd
adfsf
12e124
```

```
1 printStrings(listOf("", "90798"))
```

```
90798
```

8.2 Aufgabe

Schreibe eine Funktion `readChar`. Diese gibt den ersten Buchstaben einer Eingabe des Benutzers als `Char` zurück.

9 Zusammenfügen

9.1 Aufgabe

Implementiere eine Funktion `playGOLTUI`. Mit dieser Funktion wird das Spiel des Lebens mit einem 10x10 Spielfeld gespielt.

- Zu Beginn sind alle Zellen tot und die Zelle links oben ist ausgewählt.

- Wenn q eingegeben wird, wird das Spiel beendet.

Funktionen printStrings und reachChar!


|><

d


t

><




d


 > <

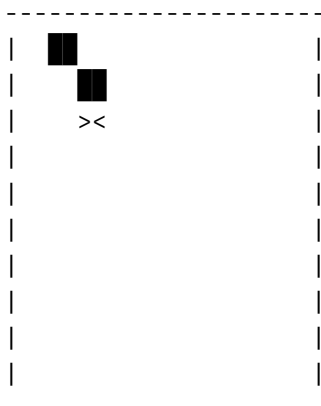
S

 $><$

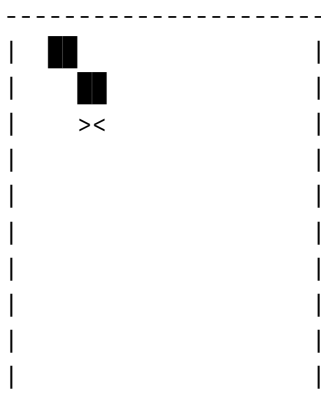
t

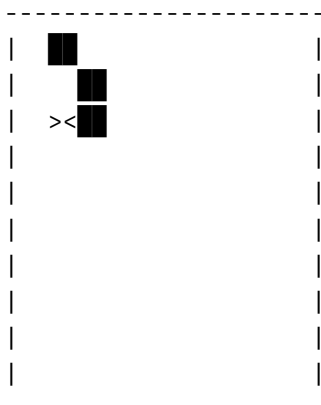
S



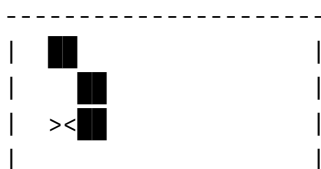
t



a

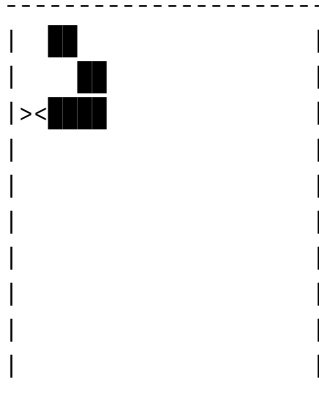


t

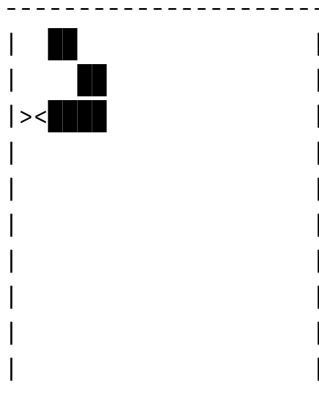




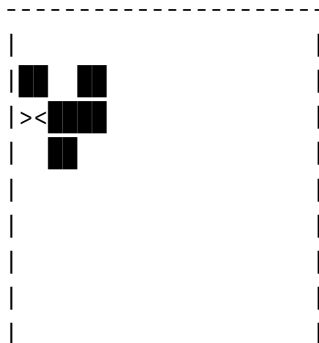
a



t

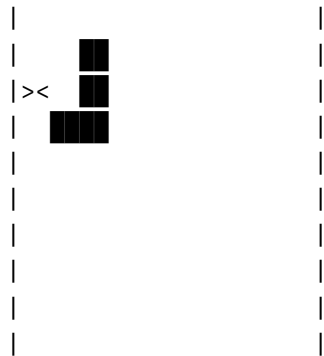


n

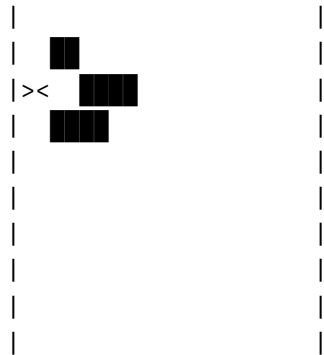


| |

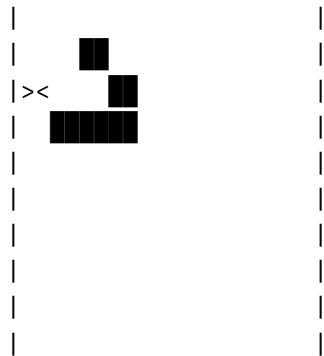
n



n

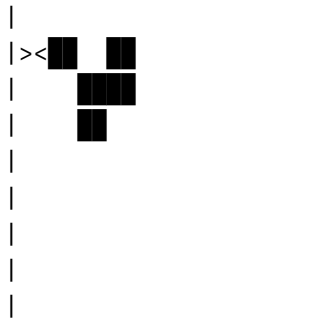


n

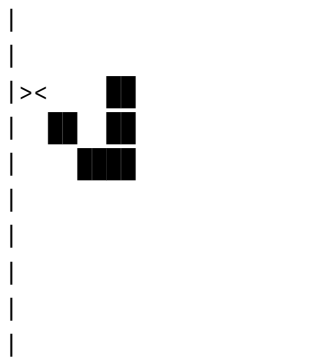


n

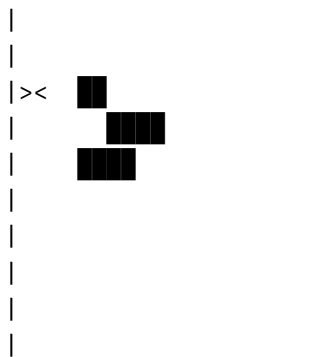
| |



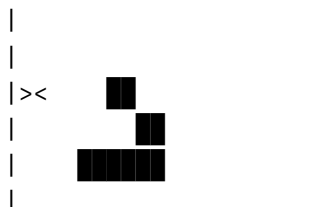
n



n

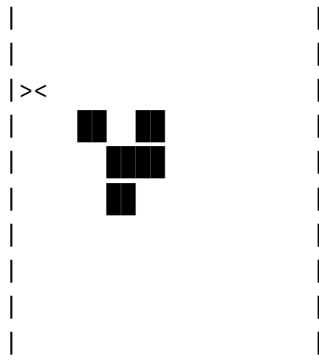


n

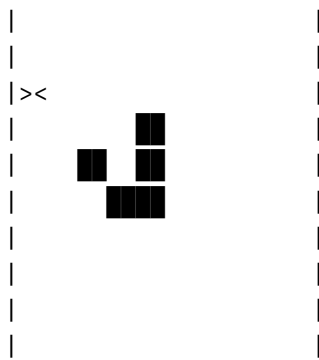




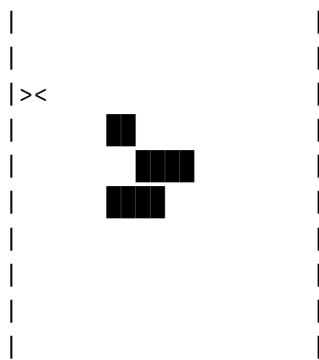
n



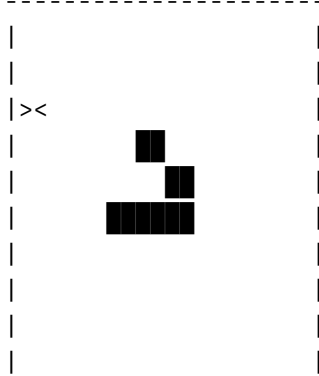
n



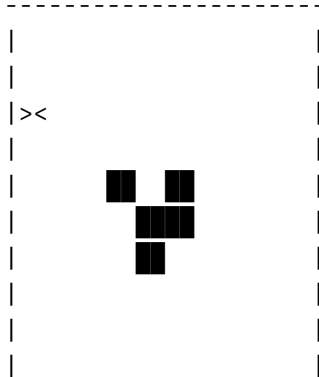
n



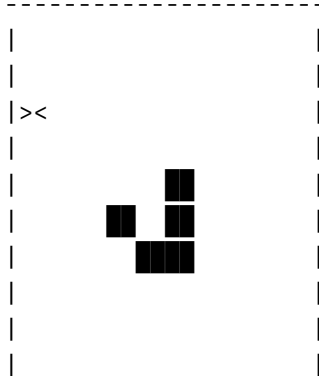
n



n

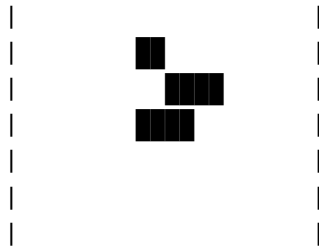


n

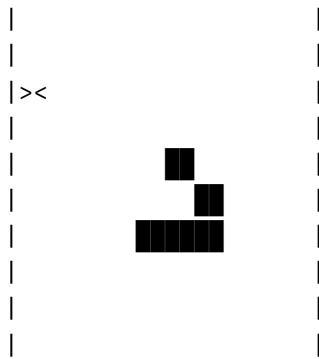


n

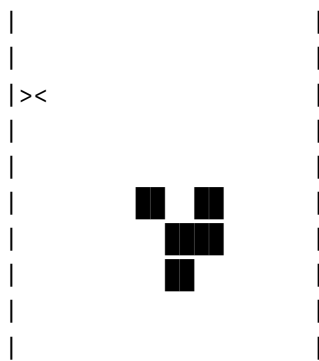




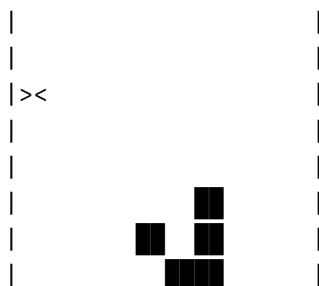
n



n



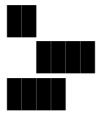
n



|
|

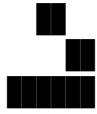
n

|
|
|><
|
|
|
|
|
|
|
|
|
|
|



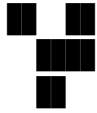
n

|
|
|><
|
|
|
|
|
|
|
|
|
|
|

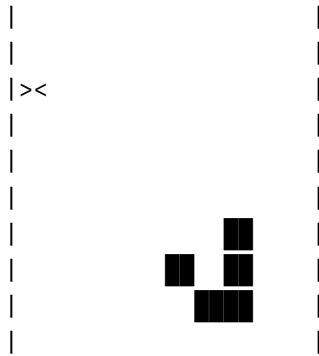


n

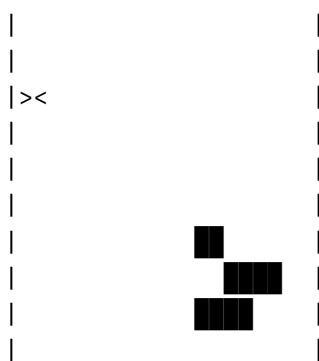
|
|
|><
|
|
|
|
|
|
|
|
|
|
|



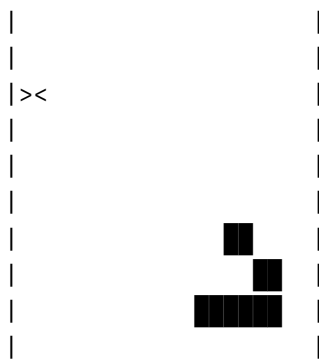
n



n

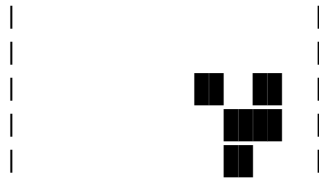


n

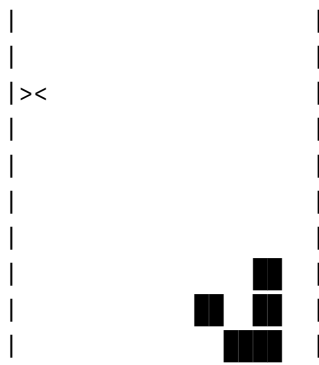


n

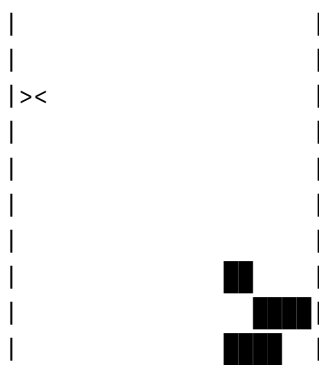




n



n



n

