

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.

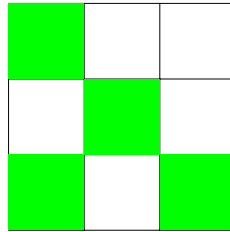


Abbildung 1: 3x3-Spielfeld mit 4 lebenden Zellen

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.

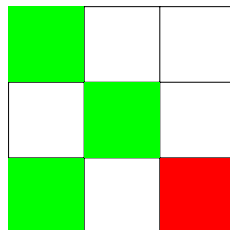


Abbildung 2: 3x3-Spielfeld mit 3 sichtbaren lebenden Zellen. Die Zelle rechts unten ist gerade ausgewählt.

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 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!

2.4 Aufgabe

Implementiere eine Funktion `printBoard`.

Die Funktion hat dieselben Parameter wie die letzte Funktion. Sie gibt die Darstellung des Spielfelds an der Konsole aus.

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

```
-----
|  ■  |
|><■  |
|-----|
```

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

```
-----
|  ■  ><|
|  ■■■  |
|■■■■■  |
|-----|
```

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

```
-----
|  ■  |
|  ><■  |
|■■■■■  |
|-----|
```

Hinweis: Nutze `rowsAsStrings`!

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

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 Steuerung

6.1 Aufgabe

Implementiere eine Funktion `updateBoard`. Die Funktion hat die folgenden Parameter:

- der Zeilenindex und
- der Spaltenindex der aktuell ausgewählten Zelle
- das Spielfeld
- der Buchstabe, den der Benutzer eingegeben hat.

Der Rückgabewert wird folgendermaßen bestimmt:

- Wenn `n` eingegeben wurde, wird das Spielfeld für die nächste Runde zurückgegeben.
- Wenn `t` eingegeben wurde, wird der Wert der aktuell ausgewählten Zelle gewechselt.
- Ansonsten wird das Spielfeld unverändert zurückgegeben.

```

1 updateBoard(1,
2             1,
3             listOf(listOf(true, false),
4                     listOf(false, true)),
5             't')

```

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

```

1 updateBoard(2, 0,
2     listOf(listOf(true, false, false),
3         listOf(false, true, true),
4         listOf(true, true, true)),
5     't')

```

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

```

1 updateBoard(2, 0,
2     listOf(listOf(true, false, false),
3         listOf(false, true, true),
4         listOf(true, true, true)),
5     'n')

```

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

```

1 updateBoard(2, 0,
2     listOf(listOf(true, false, false),
3         listOf(false, true, true),
4         listOf(true, true, true)),
5     'y')

```

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

Hinweis: Nutze toggle und computeNextBoard!

7 Auswahl von Zellen

Bei der Benutzung des Programms ist immer eine Zelle des Spielfelds ausgewählt. Durch die Eingabe einer Richtung mit w, a, s oder d kann eine andere Zelle ausgewählt werden. Hierfür implementieren wir erst eine sehr allgemeine Hilfsfunktion, die für beliebige Listen funktioniert.

7.1 Aufgabe

Implementiere eine Funktion moveSelectedIndex. Mit dieser kann der neue Index des ausgewählten Elements in einer Liste berechnet werden. Die Funktion hat die folgenden Parameter:

- der Buchstabe, der vom Benutzer zuletzt eingegeben wurde
- der aktuelle ausgewählte Index
- die Liste
- der Buchstabe mit dem ein niedriger Index gewählt wird

- der Buchstabe mit dem ein höherer Index gewählt wird

Der Index wird nur erhöht/reduziert, wenn das Ergebnis immer noch ein Index der Liste ist.

```
1 moveSelectedIndex('i', 0, listOf(8, 6, 9), 'k', 'i')
```

1

```
1 moveSelectedIndex('k', 0, listOf(8, 6), 'k', 'i')
```

0

```
1 moveSelectedIndex('s', 1, listOf(true, false), 's', 'w')
```

0

```
1 moveSelectedIndex('s', 0, listOf(true), 's', 'w')
```

0

```
1 moveSelectedIndex('x', 1, listOf(true, false), 's', 'w')
```

1

7.2 Aufgabe

Implementiere eine Funktion `moveSelectedRowIndex`. Mit dieser kann der neue Zeilenindex des ausgewählten Elements in einer 2D-Liste berechnet werden. Die Funktion hat die folgenden Parameter:

- der Buchstabe, der vom Benutzer zuletzt eingegeben wurde
- der Zeilenindex der aktuell ausgewählten Zelle
- die 2D-Liste

Der Index wird reduziert wenn `w` eingegeben wird und erhöht, wenn `s` eingegeben wird. Falls ein anderer Buchstabe eingegeben wird, wird der unveränderte Index zurückgegeben. Der Index wird nur erhöht/reduziert, wenn das Ergebnis immer noch ein Zeilenindex der 2D-Liste ist.

```
1 moveSelectedRowIndex('w',
2     0,
3     listOf(listOf(true, false),
4             listOf(false, true))
5     )
```

0

```
1 moveSelectedRowIndex('s',  
2     0,  
3     listOf(listOf(10),  
4             listOf(-3))  
5 )
```

1

```
1 moveSelectedRowIndex('r',  
2     0,  
3     listOf(listOf(10),  
4             listOf(-3))  
5 )
```

0

Hinweis: Nutze moveSelectedIndex!

7.3 Aufgabe

Implementiere eine Funktion moveSelectedColumnIndex. Mit dieser kann der neue Zeilenindex des ausgewählten Elements in einer 2D-Liste berechnet werden. Die Funktion hat die folgenden Parameter:

- der Buchstabe, der vom Benutzer zuletzt eingegeben wurde
- der Spaltenindex der aktuell ausgewählten Zelle
- die 2D-Liste

Der Index wird reduziert wenn a eingegeben wird und erhöht, wenn d eingegeben wird. Falls ein anderer Buchstabe eingegeben wird, wird der unveränderte Index zurückgegeben. Der Index wird nur erhöht/reduziert, wenn das Ergebnis immer noch ein Spaltenindex der 2D-Liste ist. Du kannst davon ausgehen, dass alle Zeilen gleich lang sind.

```
1 moveSelectedColumnIndex('d',  
2     0,  
3     listOf(listOf(true, false),  
4             listOf(false, true))  
5 )
```

1

```

1 moveSelectedColumnIndex('a',
2     0,
3     listOf(listOf(10),
4             listOf(-3))
5 )

```

0

Hinweis: Nutze moveSelectedIndex!

7.4 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.
- Mit der Eingabe von w, a, s oder d kann eine andere Zelle ausgewählt werden.
- Wenn t eingegeben wird, wird ändert sich der Zustand der ausgewählten Zelle.
- Wenn n eingegeben wird, wird das Spielfeld aktualisiert.
- Wenn q eingegeben wird, wird das Spiel beendet.

Hinweis: Nutze startBoard, printBoard, moveSelectedRowIndex, moveSelectedColumnIndex und updateBoard!

```

-----
|><      |
|         |
|         |
|         |
|         |
|         |
|         |
|         |
|         |
|         |
-----
d
-----
|  ><    |
|         |
|         |
|         |
|         |

```


t


><



d



■ ><


s

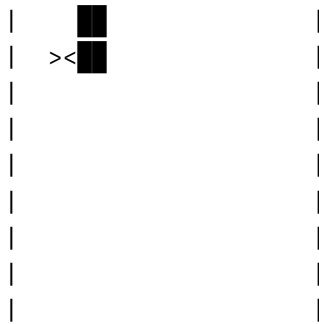
■ ><

t	
	
><	

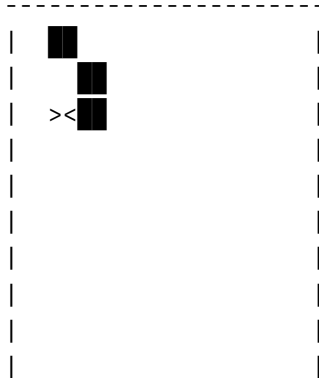
s	
	
	
><	

t	
	
	
><	

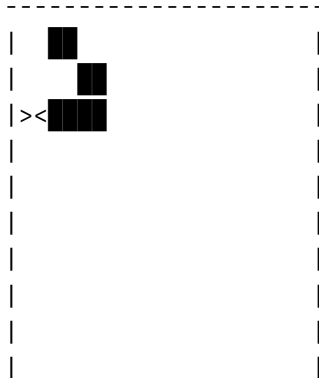
a	
	



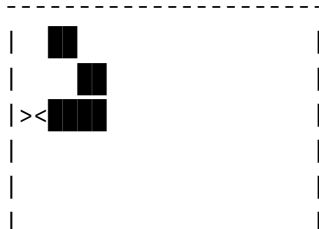
t



a

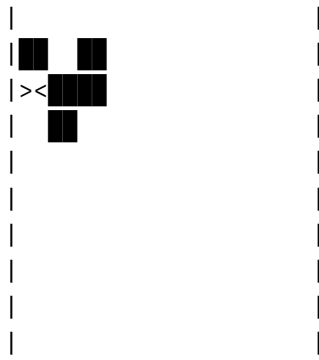


t

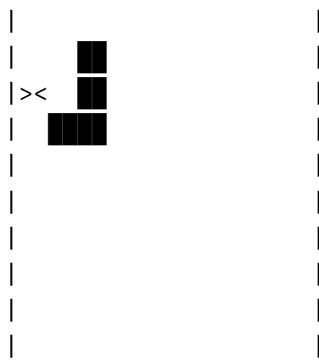




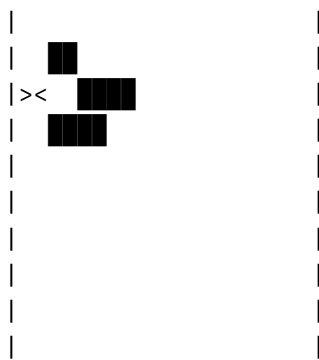
n



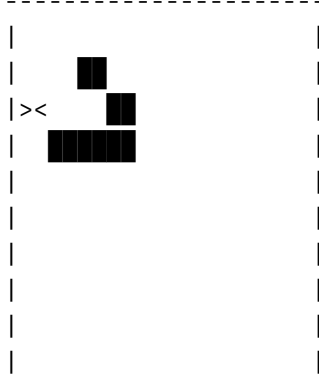
n



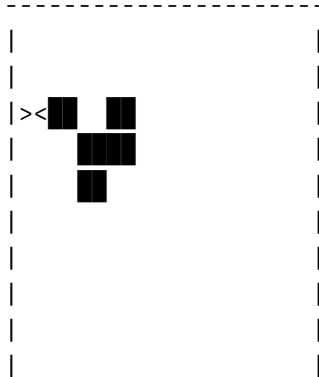
n



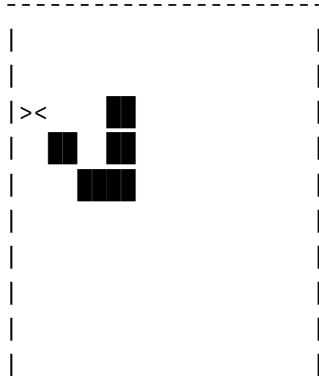
n



n

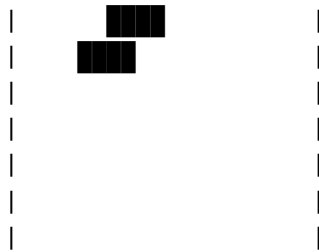


n

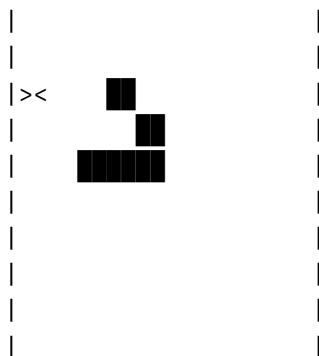


n

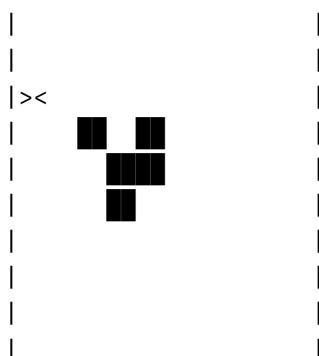




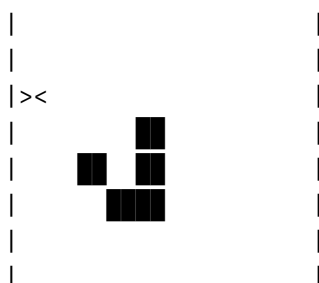
n



n

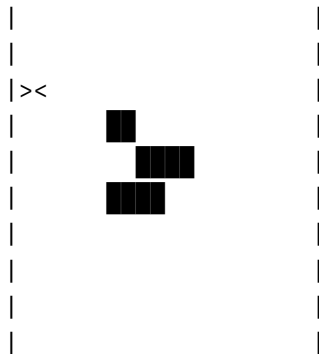


n

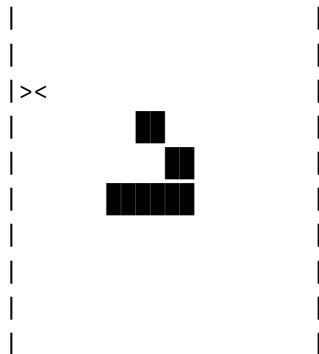


|
|

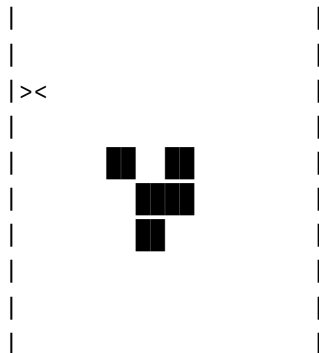
n



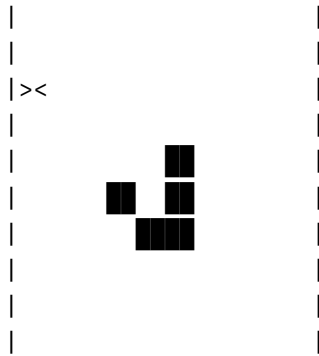
n



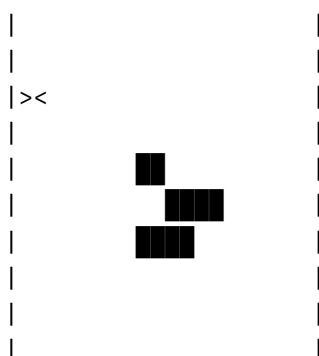
n



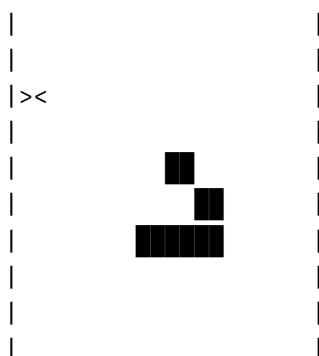
n



n

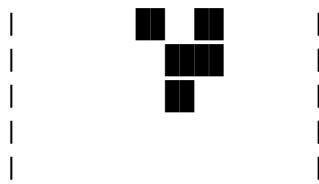


n

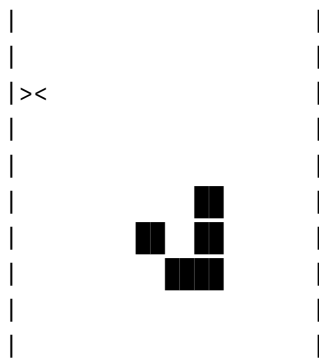


n

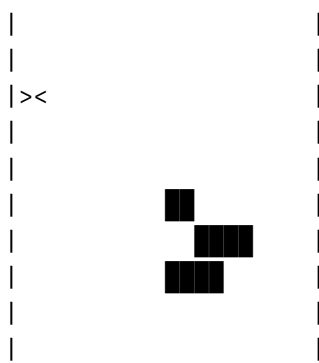




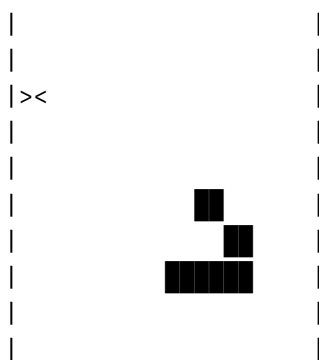
n



n

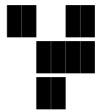


n



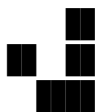
n

|><



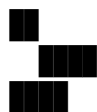
n

|><



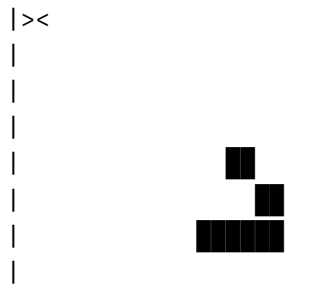
n

|><

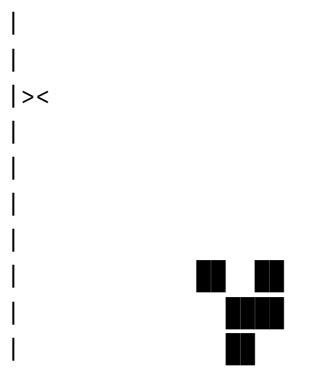


n

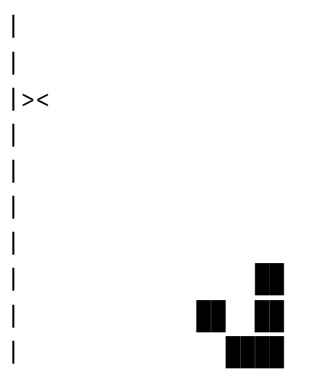
|



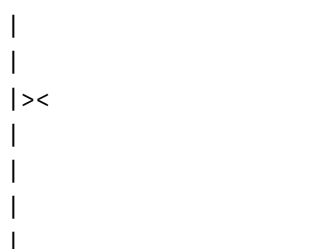
n

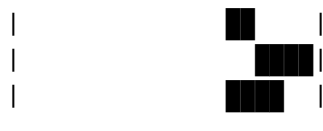


n

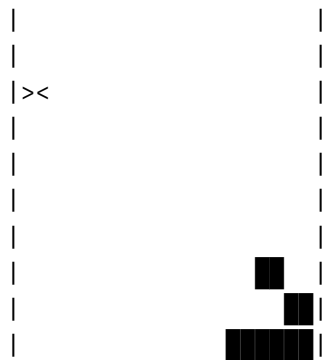


n

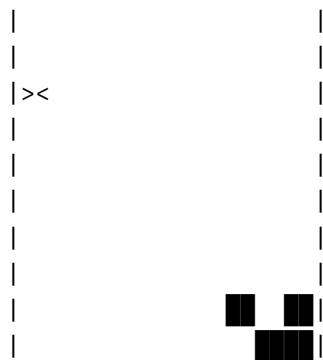




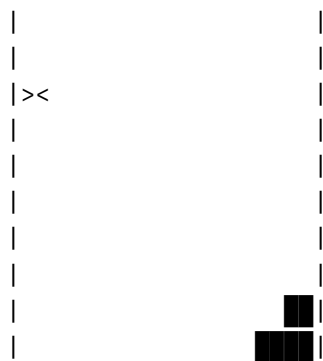
n



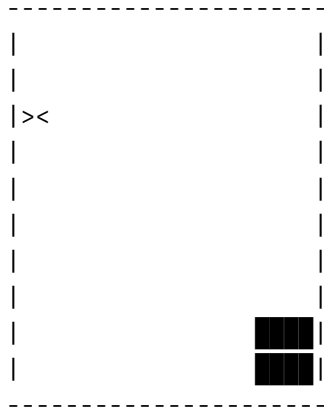
n



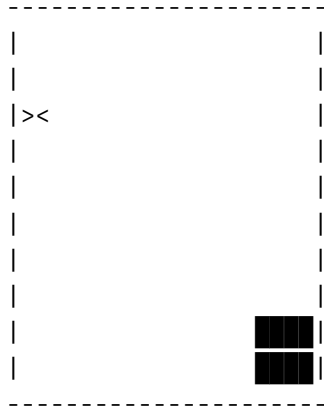
n



n



n



q