Snake

1 Aufgabe

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

```
new V2(3, 2);
```

```
V2[x=3, y=2]
```

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

```
V2[x=1, y=-5]
```

2 Aufgabe

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

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

```
V2[x=4, y=-3]
```

```
new V2(3, 4).plus(new V2(-4, -5));
```

```
V2[x=-1, y=-1]
```

Hinweis:

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

3 Aufgabe

Erweitere die Datenklasse `V2` um die Methode `times`. Dieser wird ein Vektor übergeben. Es wird das Skalarprodukt der beiden Vektoren zurückgegeben.

```
new V2(1, 2).times(new V2(3, 4))
```

```
11
```

```
new V2(3, 2).times(new V2(1, -5))
```

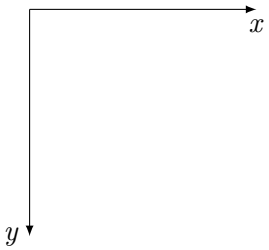
```
-7
```

Hinweis:

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

4 Aufgabe

Implementiere eine Funktion `keyToV2`. 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.




```
keyToV2('w')
```

```
V2[x=0, y=-1]
```

```
keyToV2('a')
```

```
V2[x=-1, y=0]
```

```
keyToV2('s')
```

```
V2[x=0, y=1]
```

```
keyToV2('d')
```

```
V2[x=1, y=0]
```

```
keyToV2('i')
```

```
V2[x=0, y=0]
```

5 Aufgabe

Implementiere eine Funktion `computeNewDirection`. Dieser wird die aktuelle Richtung der Schlange als `V2` und die zuletzt drückte Taste als Character übergeben. Sie gibt die neue Richtung zurück. Die Richtung wird nur geändert, wenn eine Richtungstaste gedrückt wurde und die gewünschte Richtung orthogonal zur aktuellen Richtung ist. Außerdem darf die neue Richtung nicht der Nullvektor sein.

```
computeNewDirection(new V2(1, 0), 'w');
```

```
V2[x=0, y=-1]
```

```
computeNewDirection(new V2(0, 1), 's');
```

```
V2[x=0, y=1]
```

```
computeNewDirection(new V2(0, 1), 't');
```

```
V2[x=0, y=1]
```

Hinweis: Zwei Vektoren sind orthogonal zueinander, wenn ihr Skalarprodukt 0 ist.

6 Aufgabe

Implementiere eine Funktion `dropLast`. Dieser wird eine Liste übergeben. Sie gibt die Liste ohne das letzte Element zurück. Wenn die übergebene Liste leer ist, wird auch die leere Liste zurückgegeben.

```
dropLast(List.of('i', 'a'));
```

```
[i]
```

```
dropLast(new ArrayList<>());
```

```
[]
```

```
dropLast(List.of(new V2(5, 5), new V2(4, 5), new V2(4, 6)));
```

```
[V2[x=5, y=5], V2[x=4, y=5]]
```

7 Aufgabe

Implementiere eine Funktion `isOnBoard`. Dieser werden

- die Koordinaten einer Zelle als `V2` und
- die Anzahl der Spalten und Zeilen des Spielfelds

übergeben. Sie gibt zurück, ob diese Zelle auf dem Spielfeld ist. Die x- und y-Koordinaten der Zellen des Spielfelds werden hierfür mit 0 beginnend durchnummeriert.

```
isOnBoard(new V2(3, 4), 4, 5)
```

```
true
```

```
isOnBoard(new V2(6, 5), 6, 7)
```

```
false
```

```
isOnBoard(new V2(6, 5), 7, 5)
```

```
false
```

```
isOnBoard(new V2(3, 2), 7, 5)
```

```
true
```

```
isOnBoard(new V2(3, -2), 7, 5)
```

```
false
```

8 Aufgabe

In der nächsten Aufgabe brauchst du die Klasse `Random`

```
var random = new Random();
```

Diese hat eine Methode `nextInt`. Dieser werden zwei positive ganze Zahl n und m übergeben. Sie gibt eine zufällige natürliche Zahl zurück, die mindestens n und **echt kleiner** als m ist.

```
random.nextInt(0, 5);
```

```
1
```

```
random.nextInt(0, 5);
```

```
1
```

9 Aufgabe

Implementiere eine Funktion `generateRandomFreeCoordinates`. Dieser werden

- eine Liste der Koordinaten aller belegten Zellen und
- die Anzahl der Zeilen
- die Anzahl der Spalten des Spielfelds

übergeben. Sie gibt die Koordinaten einer zufällig ausgewählten freien Zelle zurück.

```
generateRandomFreeCoordinates(List.of(new V2(0,0), new V2(1,0), new V2(0,1)), 2, 2);
```

```
V2[x=1, y=1]
```

```
generateRandomFreeCoordinates(List.of(new V2(0,0), new V2(1,0), new V2(0,1)), 3, 2);
```

```
V2[x=0, y=2]
```

```
generateRandomFreeCoordinates(List.of(new V2(0,0), new V2(1,0), new V2(0,1)), 3, 2);
```

```
V2[x=0, y=2]
```

Hinweis: Nutze `nextInt` und die Listen-Methode `contains`!

```
List.of(1, 2, 3).contains(4);
```

```
false
```

```
List.of(1, 2).contains(2);
```

```
true
```

10 Aufgabe

Implementiere eine Daten-Klasse `Snake`. Die Eigenschaften der Klasse sind

- die Postion des Schlangenkopfes als Vektor
- die Postion aller anderen Teile der Schlange als Liste von Vektoren
- ein Boolean, das angibt, ob die Schlange gerade verdaut

```
new Snake(new V2(6, 5), List.of(new V2(5, 5), new V2(4, 5), new V2(4, 6)), true);
```

```
Snake[head=V2[x=6, y=5], tail=[V2[x=5, y=5], V2[x=4, y=5], V2[x=4, y=6]], digesting=true]
```

```
new Snake(new V2(1, 0), new ArrayList<V2>(), false);
```

```
Snake[head=V2[x=1, y=0], tail=[], digesting=false]
```

11 Aufgabe

Erweitere die Klasse `Snake` um eine Methode `getCoordinates`. Diese gibt eine Liste aller Koordinaten der Schlange zurück. Das erste Element dieser Liste ist der Kopf der Schlange. Die Reihenfolge der übrigen Elemente wird nicht verändert.

```
new Snake(new V2(6, 5), List.of(new V2(5, 5), new V2(4, 5), new V2(4, 6)), true).getCoordina
```

```
[V2[x=6, y=5], V2[x=5, y=5], V2[x=4, y=5], V2[x=4, y=6]]
```

```
new Snake(new V2(1, 0), List.of(), false).getCoordinates();
```

```
[V2[x=1, y=0]]
```

12 Aufgabe

Erweitere die Klasse `Snake` um eine Methode `tailBitten`. Diese gibt zurück, ob sich die Schlange selbst gebissen hat.

```
new Snake(new V2(6, 5), List.of(new V2(5, 5), new V2(4, 5), new V2(4, 6)), true).tailBitten()
```

```
false
```

```
new Snake(new V2(2, 3),
      List.of(new V2(3, 3), new V2(4, 3), new V2(4,2), new V2(4, 1),
             new V2(3, 1), new V2(2, 1), new V2(2, 2), new V2(2, 3)),
      false).tailBitten()
```

```
true
```

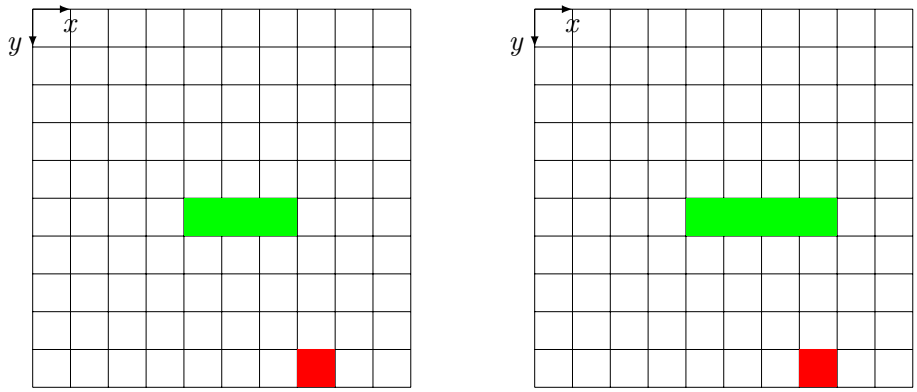
13 Aufgabe

Erweitere die Klasse `Snake` um eine Methode `move`. Dieser wird die Richtung der Schlange und die Position des Apfels gegeben. Sie gibt die neue Schlange zurück.

- Die Schlange bewegt sich in die übergebene Richtung.
- Wenn die Schlange gerade verdaut, behält sie die letzte Zelle. Ansonsten verliert sie diese.
- Wenn der Kopf der Schlange nach der Bewegung auf dem Apfel ist, isst die Schlange den Apfel und verdaut diesen. Ansonsten verdaut sie nicht mehr.

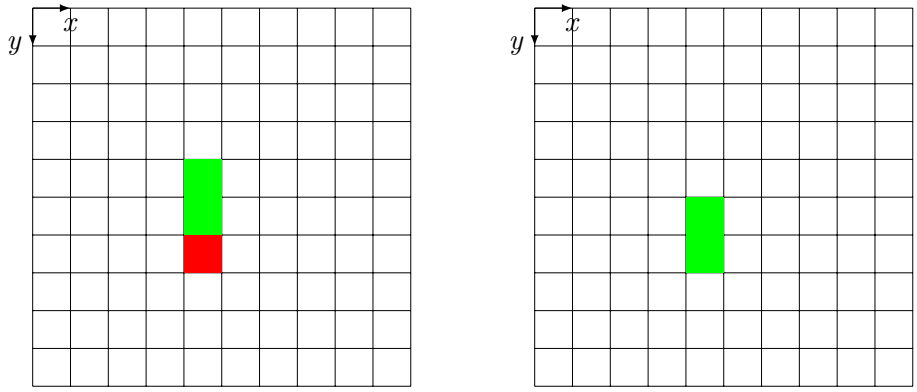
```
new Snake(new V2(6, 5), List.of(new V2(5, 5), new V2(4, 5)), true).move(new V2(1, 0), new V2
```

```
Snake[head=V2[x=7, y=5], tail=[V2[x=6, y=5], V2[x=5, y=5], V2[x=4, y=5]], digesting=false]
```




```
new Snake(new V2(4, 5), List.of(new V2(4, 4)), false).move(new V2(0, 1), new V2(4, 6))
```

```
Snake[head=V2[x=4, y=6], tail=[V2[x=4, y=5]], digesting=true]
```




```
new Snake(new V2(6, 5), List.of(new V2(5, 5), new V2(4, 5)), false).move(new V2(1, 0), new V
```

```
Snake[head=V2[x=7, y=5], tail=[V2[x=6, y=5], V2[x=5, y=5]], digesting=false]
```

```
new Snake(new V2(4, 5), List.of(new V2(4, 4)), true).move(new V2(0, 1), new V2(4, 6))
```

```
Snake[head=V2[x=4, y=6], tail=[V2[x=4, y=5], V2[x=4, y=4]], digesting=true]
```

Hinweis: Nutze `dropLast`, `getCoordinates` und die Vektoraddition! Berechne nacheinander

- den neuen Kopf der Schlange
- ob die Schlange nach der Bewegung verdaut
- den neuen `tail`

14 Aufgabe

Schreibe einen zweiten Konstruktor für die Klasse `Snake`. Für diesen muss nur der Kopf übergeben werden. Falls dieses Konstruktor verwendet wird, besteht die Schlange nur aus dem Kopf und verdaut nicht.

```
new Snake(new V2(3, 2))
```

```
Snake[head=V2[x=3, y=2], tail=[], digesting=false]
```

15 Aufgabe

Implementiere die Klasse `Model`. Die Eigenschaften der Klasse sind

- die Anzahl der Spalten des Spielfelds

- die Anzahl der Zeilen des Spielfelds
- die Schlange
- die Richtung der Schlange als Vektor
- die Position des Apfels als Vektor

Dem Konstruktor der Klasse werden Werte für alle Eigenschaften übergeben.

```
new Model(8, 9, new Snake(new V2(6, 5), List.of(), true), new V2(1, 0), new V2(3, 2))
```

```
REPL.$JShell$21$Model@3611ca86
```

```
var m1 = new Model(10, 5);
System.out.println(m1.cols);
System.out.println(m1.rows);
System.out.println(m1.snake);
System.out.println(m1.direction);
System.out.println(m1.applePosition);
```

```
10
5
Snake[head=V2[x=5, y=2], tail=[], digesting=false]
V2[x=1, y=0]
V2[x=9, y=4]
```

```
var m2 = new Model(30, 20);
System.out.println(m2.cols);
System.out.println(m2.rows);
System.out.println(m2.snake);
System.out.println(m2.direction);
System.out.println(m2.applePosition);
```

```
30
20
Snake[head=V2[x=15, y=10], tail=[], digesting=false]
V2[x=1, y=0]
V2[x=29, y=19]
```

Der Kopf der Schlange ist zu Beginn des Spiels mitten auf dem Spielfeld. Die Schlange besteht dann nur aus dem Kopf, verdaut nicht und bewegt sich nach rechts. Der Apfel ist rechts unten.

16 Aufgabe

Erweitere die Klasse `Model` um eine Methode `snakeIsAlive`. Diese bestimmt, ob die Schlange noch lebt. Die Schlange stirbt, wenn sie sich selbst beißt, oder ihr Kopf nicht mehr auf dem Spielfeld ist.

```
new Model(8, 9, new Snake(new V2(6, 5), List.of(), true), new V2(1, 0), new V2(3, 2)).
    snakeIsAlive()
```

```
true
```

```
new Model(8, 9, new Snake(new V2(-1, 5), List.of(), true), new V2(1, 0), new V2(3, 2)).
    snakeIsAlive()
```

```
false
```

```
new Model(5, 5, new Snake(new V2(2, 3),
                  List.of(new V2(3, 3), new V2(4, 3), new V2(4,2), new V2(4, 1),
                          new V2(3, 1), new V2(2, 1), new V2(2, 2), new V2(2, 3)),
                  false), new V2(1, 0), new V2(0, 1)).snakeIsAlive()
```

```
false
```

Hinweis: Nutze `isOnBoard` und `tailBitten`!

17 Aufgabe

Erweitere die Klasse `Model` um eine Methode `boardIsFull`. Diese gibt zurück, ob die Schlange das ganze Spielfeld ausfüllt.

```
new Model(2, 2, new Snake(new V2(0, 0), List.of(new V2(0, 1), new V2(1, 1), new V2(1, 0)), true), new V2(1, 0), new V2(0, 0)).boardIsFull()
```

```
true
```

```
new Model(2, 2, new Snake(new V2(0, 0), List.of(new V2(0, 1), new V2(1, 1)), true), new V2(1, 0), new V2(0, 0)).boardIsFull()
```

```
false
```

Hinweis: Nutze `getCoordinates` und überlege dir wie lang die Schlange ist, wenn sie das ganze Spielfeld ausfüllt.

18 Aufgabe

Erweitere die Klasse `Model` um eine Methode `gameOngoing`. Diese gibt zurück, ob das Spiel weitergeht. Das ist der Fall, wenn es auf dem Spielfeld noch freie Plätze gibt und die Schlange noch lebt.

```
new Model(2, 2, new Snake(new V2(0, 0), List.of(new V2(0, 1), new V2(1, 1), new V2(1, 0)), true), new V2(1, 0), new V2(0, 0)).gameOngoing()
```

```
false
```

```
new Model(2, 2, new Snake(new V2(0, 0), List.of(new V2(0, 1), new V2(1, 1)), true), new V2(1, 0), new V2(0, 0)).gameOngoing()
```

```
true
```

```
new Model(2, 2, new Snake(new V2(-1, 0), List.of(new V2(0, 0)), true), new V2(-1, 0), new V2(0, 0)).gameOngoing()
```

```
false
```

Hinweis: Nutze `snakeIsAlive` und `boardIsFull`.

19 Aufgabe

Erweitere die Klasse `Model` um eine Methode `getEndMessage`. Diese gibt einen String zurück, indem steht ob der Spieler gewonnen oder verloren hat. Du kannst davon ausgehen, dass diese Methode nur aufgerufen wird, wenn das Spiel zu Ende ist.

```
new Model(2, 2, new Snake(new V2(0, 0), List.of(new V2(0, 1), new V2(1, 1), new V2(1, 0)), true), new V2(1, 0), new V2(0, 0)).getEndMessage()
```

```
won
```

```
new Model(2, 2, new Snake(new V2(-1, 0), List.of(new V2(0, 0)), true), new V2(-1, 0), new V2(0, 0)).getEndMessage()
```

```
lost
```

Hinweis: Nutze `boardIsFull` oder `snakeIsAlive`.

20 Aufgabe

Implementiere die Klasse `UIState`. Die Eigenschaften der Klasse sind

- der Kopf der Schlange
- die Koordinaten der restlichen Schlangenzellen
- die Position des Apfels als Vektor

```
new UIState(new V2(6, 5), List.of(new V2(5, 5), new V2(5, 4)), new V2(1, 0))
```

```
UIState[snakeHead=V2[x=6, y=5], snakeTail=[V2[x=5, y=5], V2[x=5, y=4]], applePosition=V2[x=1, y=0]]
```

```
new UIState(new V2(6, 5), List.of(), new V2(2, 3))
```

```
UIState[snakeHead=V2[x=6, y=5], snakeTail=[], applePosition=V2[x=2, y=3]]
```

21 Aufgabe

Erweitere die Klasse `Model` um eine Methode `getUIState`. Diese gibt den `UIState` des `new Models` zurück.

```
new Model(8, 9, new Snake(new V2(6, 5), List.of(), true), new V2(1, 0), new V2(3, 2)).getUIS
```

```
UIState[snakeHead=V2[x=6, y=5], snakeTail=[], applePosition=V2[x=3, y=2]]
```

```
new Model(8, 9, new Snake(new V2(6, 5), List.of(new V2(5, 5), new V2(5, 4)), false), new V2(1, 0), ne
```

```
UIState[snakeHead=V2[x=6, y=5], snakeTail=[V2[x=5, y=5], V2[x=5, y=4]], applePosition=V2[x=0, y=1]]
```

22 Aufgabe

Erweitere die Klasse `Model` um eine Methode `moveSnake`. Beim Ausführen wird die Schlange mit der `move` Methode bewegt. Falls die Schlange verdaut, wird die Position des Apfels neu berechnet.

```
var m = new Model(8, 9, new Snake(new V2(6, 5), List.of(new V2(5, 5)), false), new V2(1, 0), new V2(3, 2));
m.moveSnake();
System.out.println(m.snake);
System.out.println(m.applePosition);
```

```
Snake[head=V2[x=7, y=5], tail=[V2[x=6, y=5]], digesting=false]
V2[x=3, y=2]
```

```
var m = new Model(8, 9, new Snake(new V2(4, 5), List.of(new V2(5, 5)), true), new V2(-1, 0), new V2(3, 5));
m.moveSnake();
System.out.println(m.snake);
System.out.println(m.applePosition);
```

```
Snake[head=V2[x=3, y=5], tail=[V2[x=4, y=5], V2[x=5, y=5]], digesting=true]
V2[x=5, y=3]
```

Falls die Schlange nach der Bewegung das ganze Spielfeld ausfüllt, wird die Position des Apfels nicht neu berechnet.

```
var m = new Model(2, 2, new Snake(new V2(0, 0), List.of(new V2(0, 1), new V2(1, 1)), true), new V2(1, 0), new V2(1, 1));
m.moveSnake();
m.applePosition;
```

```
V2[x=1, y=1]
```

Nutze die Methoden `boardIsFull` der Klasse `Model`, `move` und `getCoordinates` der Klasse `Snake` und die Funktion `generateRandomFreeCoordinate`.

23 Aufgabe

Erweitere die Klasse `Model` um eine Methode `setDirection`. Dieser wird ein Character übergeben. Wenn die Richtung, für die der Character steht, orthogonal zur aktuellen Richtung der Schlange ist, wird die Richtung der Schlange durch diese Richtung ersetzt.

```
var m = new Model(8, 9, new Snake(new V2(6, 5), List.of(new V2(5, 5)), false), new V2(1, 0), new V2(3, 2));
m.setDirection('w');
m.direction
```

```
V2[x=0, y=-1]
```

```
var m = new Model(8, 9, new Snake(new V2(6, 5), List.of(new V2(5, 5)), false), new V2(1, 0), new V2(3, 2));
m.setDirection('a');
m.direction
```

```
V2[x=1, y=0]
```

Hinweis: Nutze `computeNewDirection`!

24 Aufgabe

Die Methoden `setDirection`, `moveSnake`, `getUIState`, `getEndMessage` und `gameOngoing` werden außerhalb der Klasse gebraucht. Schütze alle anderen Methoden durch einen Zugriff außerhalb der Klasse `Model`! Schütze außerdem alle Eigenschaften!

24.1 Aufgabe

Ergänze in `build.gradle.kts` die folgende Zeile in den geschweiften Klammern hinter `dependencies`:

```
implementation("com.googlecode.lanterna:lanterna:3.1.3")
```

Klicke anschließend auf den Elefanten.

25 Aufgabe

Erstelle die Klasse `TUI` aus dem Spiel `pong`!
Diese hat die folgenden Eigenschaften:

name	typ
cols	int
rows	int
screen	TerminalScreen
textGraphics	TextGraphics

Kopiere dafür die Klasse `TUI` aus dem Spiel `pong`! Erstelle einen Konstruktor, dem die Spalten und Zeilenanzahl übergeben wurden. Er initialisiert alle vier Eigenschaften. Die restlichen Methoden musst du nur an ein paar Stellen anpassen. Nutze bei den Aufrufen von `sleep` relativ hohe Werte um das Spiel zu testen. In Java musst du angeben, welche Fehler eine Methode werfen kann. IntelliJ kann dies automatisch ergänzen, wenn du auf die entsprechende Fehlermeldung klickst.

26 Aufgabe

Implementiere eine Methode `print` an. Dieser wird ein Objekt der Klasse `UIState` übergeben. Wie die letzte Methode werden

- die Methode `clear` der Klasse `TerminalScreen`
- die Methode `refresh` der Klasse `TerminalScreen`
- die Funktion `sleep` mit dem Argument 1

aufgerufen.
Nach dem Aufruf von `clear`, werden an den Rändern des Spielfelds mit `putString` Blöcke als ▒ gezeichnet.
Anschließend werden

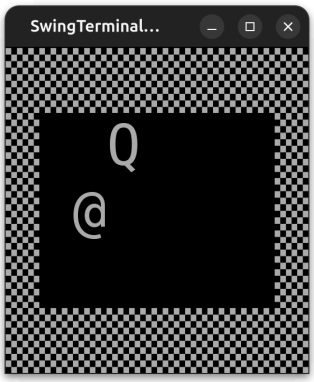
- der Kopf der Schlange als `Q`
- der Apfel als `@`

- die übrigen Zellen als `0`

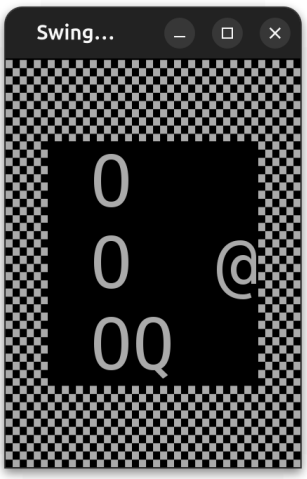
gezeichnet.

Die Lage der Blöcke im Terminal ist wegen der Ränder um jeweils eine Einheit in $x-$ und $y-$Richtung verschoben.

```
1 TUI(7, 3).print(UIState(new V2(2,0), List.of(), new V2(1, 1)))
```




```
1 TUI(5, 3).print(UIState(new V2(2,2), List.of(new V2(1, 2), new V2(1,1), new V2(1, 0)),new V2
```



27 Aufgabe

Schreibe eine Funktion `playSnake`. Dieser werden eine gewünschte Spalten- und Zeilenanzahl übergeben. Sie spielt Snake mit den entsprechenden Dimensionen.

Hinweis: Nutze die Methoden`setDirection`, `moveSnake`, `getUIState`, `getEndMessage` und `gameOngoing` der Klasse `Model` und die Methoden `getPressedKey` `print`, `printString` und `close` der Klasse `TUI`!