Micro:bit – Space War

Rückmeldungen zur ersten Version

Allgemeiner Codestil

- Halte dich an **Python-Konventionen**
- **Sinnvolle Variablennamen**
- Code soll sauber **strukturiert & übersichtlich** sein:
  - Kommentare als Überschriften und Erklärungen
  - Zeilenumbrüche
  - Funktionen
- Tipp: Halte dich immer an **Checkliste** von Gruppenarbeit

```
1   from microbit import*
2   import random
3   a=0
4   b=0
5   c=0
6   d=0
7   e=2
8   x=0
9   y=0
10  z=0
11  t=0
12  q=0
13  r=0
```

Zeit

- Wie timed man z.B. Bewegung der Asteroiden?
- eigene Zeit definieren:
  - Haben viele gemacht
  - Problem?
    - Hängt davon ab, wie schnell CPU vom micro:bit / Computer / … ist

```
time += 1
```

- Sleep:
  - Eliminiert Problem von `time += 1`
  - Dafür: Macht Code schwerfällig, kann z.B. keine gedrückten Buttons erkennen, während am sleepen ist
  - Gute Lösung: Frame Rate von ca. 20 FPS (frames per second -> sleep(50)).
  - Ein Frame ist die kleinste Zeitscheibe, in der etwas passieren kann im Game.
- Andere Möglichkeit:
  - **running_time()**
  - Gibt an, seit wie vielen Millisekunden micro:bit schon läuft
  - aber Achtung: running_time kann Sprünge machen oder zweimal die gleiche Zahl ergeben!

Copy-Paste

- Viele Codes mit copy-paste Elementen
- Beispiel: 3 Asteroden, dann 3x copy-paste vom entsprechenden Code
- Besser: Code nur 1x, dafür in Schleife oder Funktion
- **Faustregel**: macht man *copy-paste*, so macht man etwas falsch
  -> verwende Schleife oder Funktion

Achtung vor Doppelgleich ==

- Beispiel: Nach 10s soll das Level erhöht werden
- ```
  if running_time() == 10000:
      level += 1
  ```
- Problem: running_time() ist höchstwahrscheinlich nie genau 10000.
- Beispiel: Beim letzten Durchgang war es noch 9991, beim nächsten dann 10042, da Code auch Zeit für das Ausführen benötigt
- Besser:
  ```
  if running_time() > 10000:
      level += 1
  ```
- Achtung: Muss dafür sorgen, dass nach 10s nicht in jedem Durchgang Level erhöht wird
  ```

Achtung vor Doppelgleich ==

- Beispiel: Immer *nach weiteren 10s* soll das *Level erhöht* warden
- ```
  level_increase_time = running_time() # save initial time
  while True:
      frame_time = running_time()
      if frame_time - level_increase_time > 10000: # >10s
          level += 1
          level_increase_time = frame_time
  ```
  ```

Wie besser programmieren?

Zwei Arten von Code-Elementen in m:bit

- **Normaler Python-Code:**
  - In jedem Python-Projekt
  - Auch ohne m:bit

```
if light == 0:
    #koordinaten erstellen
    x = random.randint(0,4)
    if level == 1:
```

```
def rand_down(L,count):
    L.insert(0,rand_mets(count))
    L.pop(len(L)-1)
    return L
```

```
while True:
```

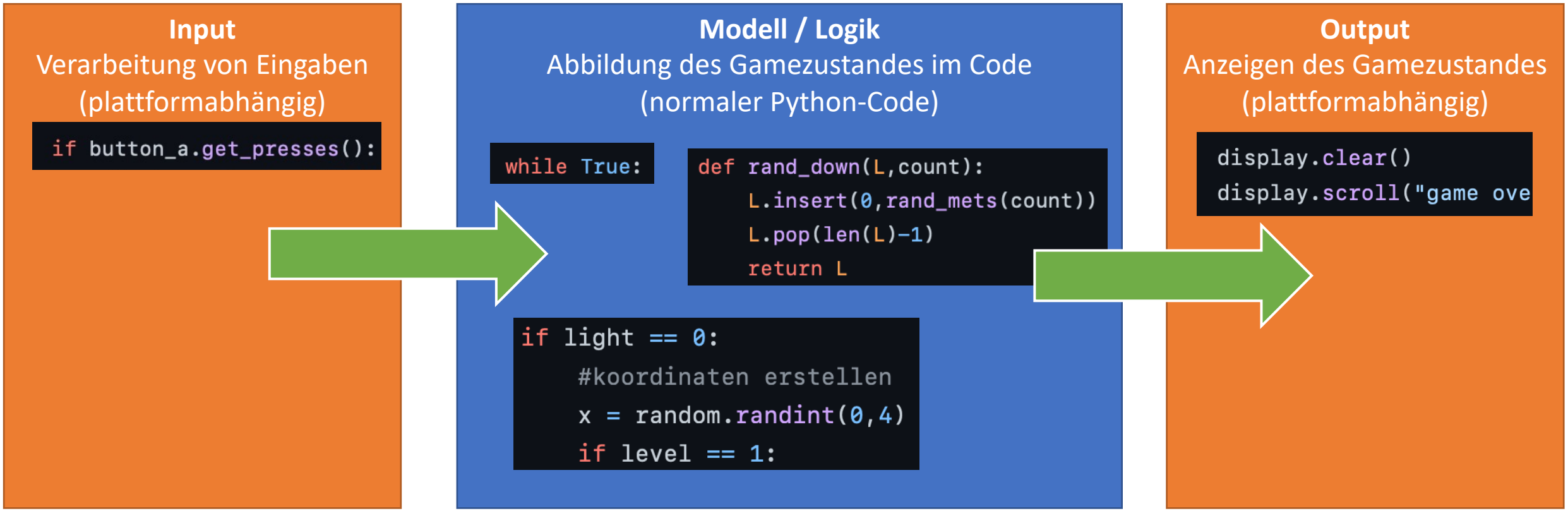
- **Spezifischer micro:bit-Code:**
  - Nur auf m:bit
  - Nicht in normalen Python-Projekt

```
display.clear()
display.scroll("game over level {0}".format(level))
```

```
t = running_time()
```

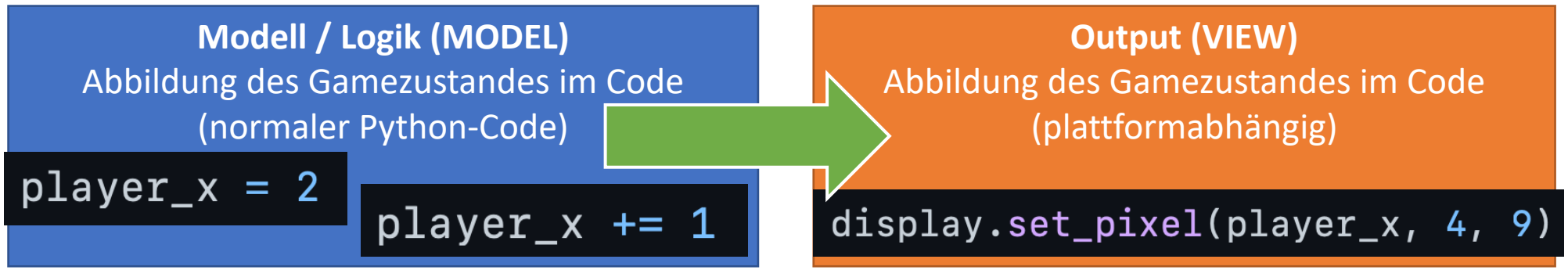
```
if button_a.get_presses():
    display.set_pixel(PLAYER_X, 4, 0)
```

Zwei Arten von Code-Elementen in m:bit



Model vs. View

- Programmier-Philosophie
- **Strikte Trennung Spiellogik (Model) und View (Output):**
  - Model:
    - Bildet Zustand des Spiels *im Code* ab
    - Nur normales Python
  - View:
    - Zeigt Spielzustand graphisch dar, übersetzt also Model in etwas, was angezeigt werden kann
    - Plattformspezifischer Code
- Beispiel:



Model für Asteroiden

- Vorgabe:
  - **Beliebig viele** Asteroiden
  - Sollen random erzeugt werden können
- Beispiel nicht optimal:
  - nur 3 Asteroiden
  - Könnte zwar erweitern, aber viel copy-paste

```
m1_x = random.randint(0, 4)
m1_y = 0
m2_x = random.randint(0, 4)
m2_y = 0
m3_x = random.randint(0, 4)
m3_y = 0
```

- Wichtig: Code des Models beinhaltet **nur normalen Python-Code**
- Muss **nicht umgeschrieben** werden, wenn z.B. von Konsolenprogramm auf micro:bit portiert wird
- **Ideen**, wie man Asteroiden mit Vorgaben oben umsetzen kann in View-Model? Besprecht in Gruppen.

Model für Asteroiden

- Asteroiden in **Liste** speichern:
  - Jeder Asteroid ist selbst wieder Liste [x_coord,y_coord]
  - Haben also Liste, die Listen beinhaltet
  - Vorteile:
    - Beliebige Anzahl
    - Mit append neue Asteroiden adden, mit pop/remove entfernen
  - Beispiel: Asteroids = [[3,2] , [4,1] , [1,2]]
    - Aktuell 3 Asteroiden
    - Einer hat Koordinaten: x-Koordinate 3 und y-Koordinate 2

Model für Asteroiden

- Asteroiden in **Liste** speichern
- **Funktionen** um Asteroiden:
  - Hinzuzufügen
  - Zu bewegen
  - Zu entfernen
- Funktion um neue Asteroiden **hinzufügen**

```
def add_random_asteroids(L):
    # IN: List L with asteroids
    # At each x-position creates new asteroid with certain probability
    # Check if identical doesn't exist yet. If doesn't, add to asteroid list
    # Note: can create multiple asteroids (different x-pos) at same time
    # RETURN: Updated asteroid list
    pass
```

Model für Asteroiden

- Funktion um Asteroiden zu **bewegen & entfernen:**

```
def move_and_remove_asteroids(L):
    ## IN: asteroid list
    ## updates y-coords of asteroids
    ## identifies all asteroids that are out of screen and removes them
    ## RETURN: updated asteroid list
    pass
```

Model für Asteroiden

- Haben bisher:
  - Liste mit Asteroiden: : `Asteroids = [[3,2],[4,1],[1,2]]`
  - Fkt. Um neue Asteroiden hinzuzufügen: `add_random_asteroids(L)`
  - Fkt. Um Asteroiden zu bewegen & entfernen `move_and_remove_asteroids(L)`
- Können nun Asteroiden updaten, in dem Funktionen aufgerufen werden:
  - `Asteroids = add_random_asteroids(Asteroids)`
  - `Asteroids = move_and_remove_asteroids(Asteroids)`
- Beachte: in Funktionen nenne **Liste** L und nicht Asteroids, ist good practice, diese **unterschiedlich zu benennen**
- Model-Funktionen beinhalten **nur normalen Python-Code**!

View für Asteroiden

- View: Funktion(en), die sich um Anzeige kümmern
- Beinhaltet **plattformspezifischen Code**
- Muss **umgeschrieben** werden, wenn z.B. von Konsolenprogramm auf micro:bit portiert wird

```
def display(L):
    # IN: all elements that need to be displayed
    # displays elements passed to function
    # RETURN: nothing
    pass
```

Auftrag

- **Idee:**
  - **Model/View für Asteroiden umsetzen**
  - Schreibe **Konsolenprogramm**, also normales Python (VSCode), NICHT auf mirco:bit
  - **Player** für Moment **ignorieren**
  - Code soll also einfach **‘Asteroiden regen’** lassen
  - Soll für **beliebige Bildschirmgrösse** funktionieren, definier dazu Konstanten:
    - WIDTH = 7
    - HEIGHT = 15
    - Beachte: GROSSBUCHSTABEN weil Konstanten (Werte die am Anfang gesetzt werden und nicht mehr verändert werden)
    - Für m:bit haben beide Konstanten Wert 5, da 5x5-LED-Matrix

Auftrag: Teil I

1. Lade das **Template** ‘space_war_ii_model_view.py’ (auf Wiki) herunter und füge es dem Repo vom letzten Auftrag hinzu. Das Template dient als Startpunkt.
2. Entwickle nun ein **Model** für deine Asteroiden und schreibe die entsprechenden Funktionen zu den ‘### MODEL FUNCTIONS`. Achte darauf, dass diese **nichts mit der Anzeige** zu tun haben und **nur normalen Python-Code** beinhalten.
3. Der Vorteil vom Programmieren mit **Funktionen** ist, dass man diese einzeln testen kann. **Teste** deine Model-Funktionen.
   Zum Beispiel: Erzeuge eine leere Asteroids-Liste. Wende dann deine Funktion, die random Asteroiden erzeugt, darauf an und schaue die Liste wieder an. Hat die Funktion gemacht, was sie soll?
4. **Bespreche** den Code mit der Lehrperson: persönlich oder per Teams (Link zu Repo schicken)

Bemerkungen

- Committe und pushe regelmässig auf **GitHub**.
- Achte darauf, dass dein Code ganz **sauber strukturiert** ist. Verhindere ein 'Gebastel'.
- Halte dich an die **Checkliste** vom letzten Auftrag.
- Ist **Einzelarbeit** (wichtig, dass du ALLEINE codest!)

Termine

- **Gruppenprojekt**:
  - Feedback umsetzen bis Freitag 18.2. oder gemäss Absprache
  - Inkl. Meldung per Teams an zuständige LP
- **Präsentation** Gruppenprojekte:
  - 5-10 Minuten pro Projekt
  - Demonstriert Projekt auf micro:bit (könnte im Vorfeld auch auf mehrere micro:bits geladen werden)
  - Stellt spannendes **Code-Element** über Beamer (z.B. PowerPoint) vor
  - 23.3.:
    - Fulin, Enise & Laurin
    - Enny & David
    - Livia & Jona
    - Jonas, Leonardo & Kajetan
  - 30.3.:
    - Dimitri & Dario
    - Paul & Lukas
    - Hanna & Anatol
- **Space War Game II:**
  - **Teil I bis So. 20.3**. **gepushed** (**geteilt** mit anschae & tkilla77)
  - File richtig benannt
  - **Nachricht** an LP (gleiche wie bei Gruppenprojekt) mit Status, z.B. «funktioniert meistens, gibt aber noch …»

Auftrag: Teil II

1. Schreibe nun eine **View** für deine Asteroiden. Die entsprechende(n) Funktion(en) gehören natürlich unter `### VIEW FUNCTIONS`.
   Dieser Code ist natürlich **plattformabhängig**.
2. **Teste** die View-Funktion(en).
3. **Finalisiere** das Programm: Es soll nun 'Asteroiden regnen'
4. **Bespreche** wieder mit der Lehrperson.