HOCHSCHULE DER MEDIEN

Termin 11: PySide6 (2)

Ziel des heutigen Übungstermins:

- Sie vertiefen Ihre PySide6-Kenntnisse.

Dieses Übungsblatt ist als Inspiration und abschließende Übung gedacht. Versuchen sie sich an diesen graphischen Übungen. Sie können das realisieren!

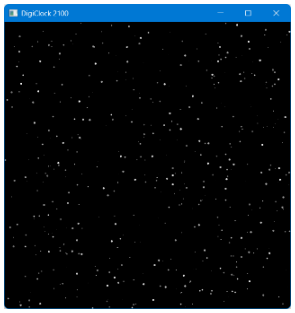
Die Datei `tools/pyside6.py` finden sie in den Musterlösungen. Kopieren sie diese Datei in Ihr eigenes Python-Projekt.

Aufgabe 67: Digitale Uhr ★

Erstellen Sie ein PySide6-Programm, das eine digitale Uhr anzeigt (s. nebenstehende Abbildung, Fenstergröße 800 x 200 Pixel). Die einzelnen Komponenten der Uhrzeit erhalten Sie über die Methoden `self.hours()`, `self.minutes()` und `self.seconds()`.

Beachten Sie, dass in der Textanzeige gegebenenfalls führende Nullen einzufügen sind, wenn die Ziffern beim Übergang von 9 zu 10 nicht horizontal springen sollen. Die trennenden Doppelpunkte sollen zudem im Sekundentakt blinken, d.h. nur sichtbar sein, wenn die Sekundenanzahl gerade ist.

Aufgabe 68: Sternhimmel ★

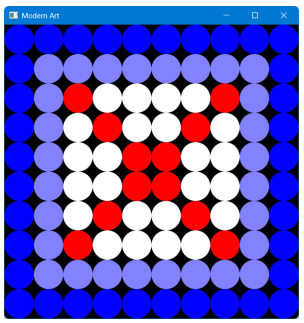


Erstellen Sie ein PySide6-Programm `starry_Sky`, das einen zufälligen Sternhimmel mit 500 Sternen auf einer Fläche von 600 x 600 Pixeln zeigt. Ein Stern ist ein Punkt mit einem Durchmesser von [1;5[Pixeln und einem Grauwert von [100;256[.

Zufällige Werte erhalten Sie mit dem Paket `random` der Python-Standardbibliothek (vgl. Aufgabe 38: Zahlenraten). Wichtig: Fügen Sie als erste Zeile Ihrer Methode `paint(…)` folgende mysteriöse Anweisung ein: `random.seed(42)`

Wenn Ihr Sternenhimmel zu Ihrer Zufriedenheit aussieht, lassen Sie die obige Zeile versuchsweise weg und vergleichen die resultierende Ausgabe. Welchen Zweck hat diese Anweisung? Versuchen Sie dies durch eigene Recherche herauszufinden bzw. lassen Sie es sich von Ihrem Copiloten erklären.

Aufgabe 69: Moderne Kunst ★



Erstellen Sie als Fingerübung ein weiteres PySide6-Programm `modern_art`, welches den nebenstehenden 600 x 600 Pixel großen Teppich zeichnet. Dargestellt werden 10 x 10 Kreise, die wie gezeigt eingefärbt werden sollen. Leiten Sie den Durchmesser eines Kreises von der Fensterbreite ab. Beim Verändern der Fenstergröße soll das Motiv zu jeder Zeit die komplette Fensterbreite ausfüllen.

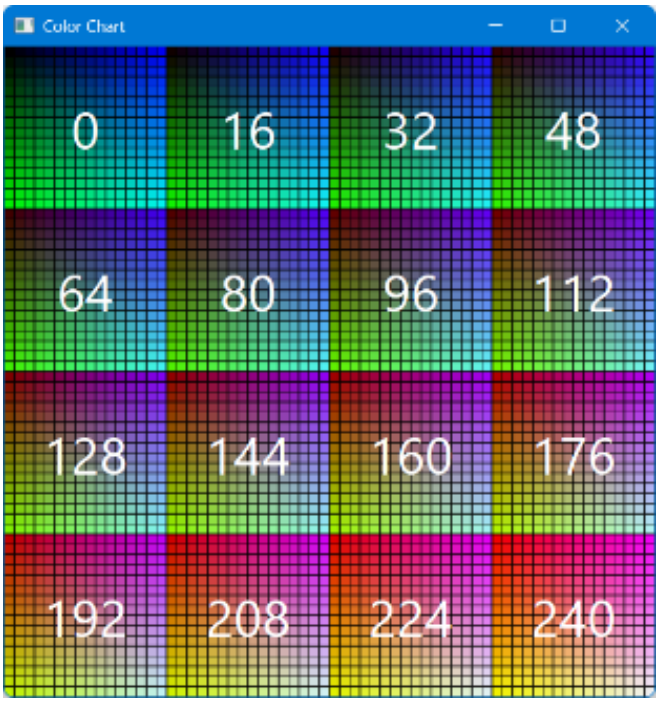
Bitte vermeiden Sie es, 100 `drawEllipse(…)`-Anweisungen einzutippen… Setzen Sie stattdessen – ganz schön gewitzt! – verschachtelte Schleifen zur Lösung ein.

Aufgabe 70: Farbkarte ★★

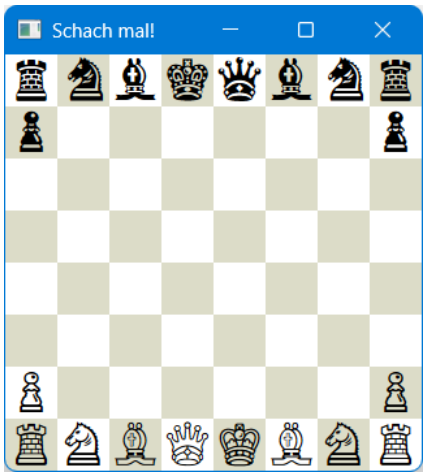
Erstellen Sie ein PySide6-Programm, das den kompletten RGB-Farbraum in Form einer Farbkarte darstellt. Es sollen alle RGB-Kombinationen dargestellt werden, die aus jeweils 16 Abstufungen pro Kanal hergestellt werden können (also insgesamt 16 x 16 x 16 = 4096 Kombinationen).

1. In einem **ersten** Schritt unterteilen Sie dazu ein Fenster mit 600 x 600 Pixeln in 16 beschriftete Quadrate. Für jedes Quadrat ist ein Rot-Anteil aus 0, 16, 32, 48, …, 224, 240 vorgesehen:

2. Im **zweiten** Schritt wird *jedes* dieser Quadrate anschließend in 16 Zeilen und 16 Spalten aufgeteilt. Von links nach rechts soll der Blauanteil steigen, von oben nach unten der Grünanteil. Die folgende Abbildung zeigt eine vollständige Farbkarte, so wie sie von Ihrem Programm angezeigt werden soll:



Aufgabe 71: Schachbrett reloaded★★



Die Figurenstellung auf einem Schachbrett kann durch eine Liste aus Codes angegeben werden, z.B. „`Ta8 kc6`“ für „Schwarzer Turm auf A8, weißer König auf C6“.

Die einzelnen Codes sind folgendermaßen aufgebaut:

1. Zuerst wird die **Art der Spielfigur** genannt. Für die schwarze Figuren gilt: K=König, D=Dame, T=Turm, L=Läufer, S=Springer, B=Bauer. Für die weiße Figuren kommen entsprechende Kleinbuchstaben zum Einsatz.
2. Dann folgt die **Koordinate** auf dem Schachbrett, wobei der Ursprung a1 in der unteren, linken Ecke liegt.

Beispiel: Der schwarze König auf `d8` wird als „`Kd8`“ kodiert.

Mehrere Codes sind durch jeweils ein Leerzeichen getrennt.

Schreiben Sie ein Programm, das beliebige Stellungen anzeigen kann, die als Strings vorliegen. Der String „`Ta8 Sb8 Lc8 Kd8 De8 Lf8 Sg8 Th8 Ba7 Bh7 ta1 sb1 lc1 dd1 ke1 lf1 sg1 th1 ba2 bh2`“ sollte die abgebildete Darstellung ergeben.

Tipps zur Implementierung:

1. Zeichnen Sie zunächst das Schachbrett mit einer Gesamtgröße von 320 x 320 Pixeln (vgl. Aufgabe 56 Schachbrett). Achten Sie darauf, dass das Feld `a1` in der unteren, linken Ecke dunkel ist.
2. Spalten Sie den Eingabestring der Spielstellung in zunächst in eine Liste aus Teilstrings auf, jeweils ein Teilstring pro Figur. Hierfür bietet sich die nützliche Funktion `split()` an. Beispiel:

```
setup = "Ta8 Sb8 Lc8"

pieces = setup.split()

for piece in pieces:

    # Position of one piece is encoded in three characters, e.g. "Kd8".

    # Extract type and position...

    # Draw piece...
```

3. Zeichnen Sie dann Figur für Figur über das bereits existierende Schachbrett. Die Symbole der Schachfiguren sind praktischerweise in der Unicode-Zeichentabelle festgelegt und können mit `drawText(…)` als Buchstaben ausgegeben werden, z.B. `"\u265a"` für den schwarzen König. Verwenden Sie eine Fallunterscheidung oder noch besser ein Dictionary, um den Typ der Figur auf den korrespondierenden Unicode-Codepunkt abzubilden:

\u2654	♔	k	Weißer König	\u2655	♕	D	Weiße Dame
\u2656	♖	t	Weißer Turm	\u2657	♗	L	Weißer Läufer
\u2658	♘	s	Weißer Springer	\u2659	♙	B	Weißer Bauer
\u265a	♚	K	Schwarzer König	\u265b	♛	d	Schwarze Dame
\u265c	♜	T	Schwarzer Turm	\u265d	♝	l	Schwarzer Läufer
\u265e	♞	S	Schwarzer Springer	\u265f	♟	b	Schwarzer Bauer

Aufgabe 72: Click me, if you can! ★★★

In dieser Aufgabe entwickeln Sie ein wahrlich minimalistisches Geschicklichkeitsspiel:

Rudi, das verspielte Rentier, hat sich im Dunklen versteckt. Sie sollen nun innerhalb von zehn Sekunden möglichst oft die rote Nase von Rudi anstupsen. Rudi ändert allerdings nach jedem Treffer flink seine Position und Abstand, so dass schnelle Reflexe gefragt sind.

So gehen Sie vor:

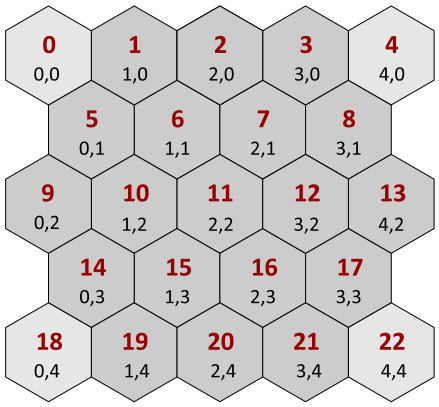
1. Erstellen Sie zunächst wieder ein PySide6-Programm wie in den vorausgegangenen Aufgaben. Die Fenstergröße beträgt 600 x 600 Pixel (Quadrat).
2. Deklarieren Sie die untenstehenden vier Variablen in der Methode `def setup(self)`:

   ```
   self.nose_x = self.width()/2
   self.nose_y = self.height()/2
   self.nose_radius = 50
   self.score = 0
   ```

   Die erste rote Nase wird damit immer im Zentrum des Fensters stehen und den Radius 50 haben. Das Spiel beginnt mit einem Punktestand von 0.
3. In der Methode `draw()`…
   - berechnen Sie die verbleibende Spielzeit. Tipp: Die Methode `self.elapsed()` liefert Ihnen die Anzahl der Sekunden seit Programmstart. Ist die verbleibende Spielzeit negativ, korrigieren Sie diese auf exakt 0.
   - Geben Sie die verbleibende Spielzeit und den aktuellen Punktestand als Text aus.
   - Zeichnen Sie eine rote Kreisschreibe um die Stelle (`self.nose_x`, `self.nose_y`) mit Radius `self.nose_radius`.
   - Ist die Spielzeit größer als 0, prüfen Sie, ob die linke Maustaste gedrückt ist (mittels `self.leftMouseButtonPressed`) und wenn ja, ob die Koordinaten des Mauszeigers (`self.mousePos().x()` bzw. `self.mousePos().y()`) innerhalb der roten Scheibe liegen. Tipp: Satz des Pythagoras verwenden. Wenn ja, wurde die Rudi erwischt. Dann inkrementieren Sie `self.score` und wählen für `self.nose_radius, self.nose_x` und `self.nose_y` zufällige neue Werte. Achten Sie dabei darauf, dass die neue Scheibe immer vollständig im Fenster zu sehen sein wird!
   - Ist die Spielzeit hingegen gleich 0, geben in eine „Game Over“-Meldung aus.

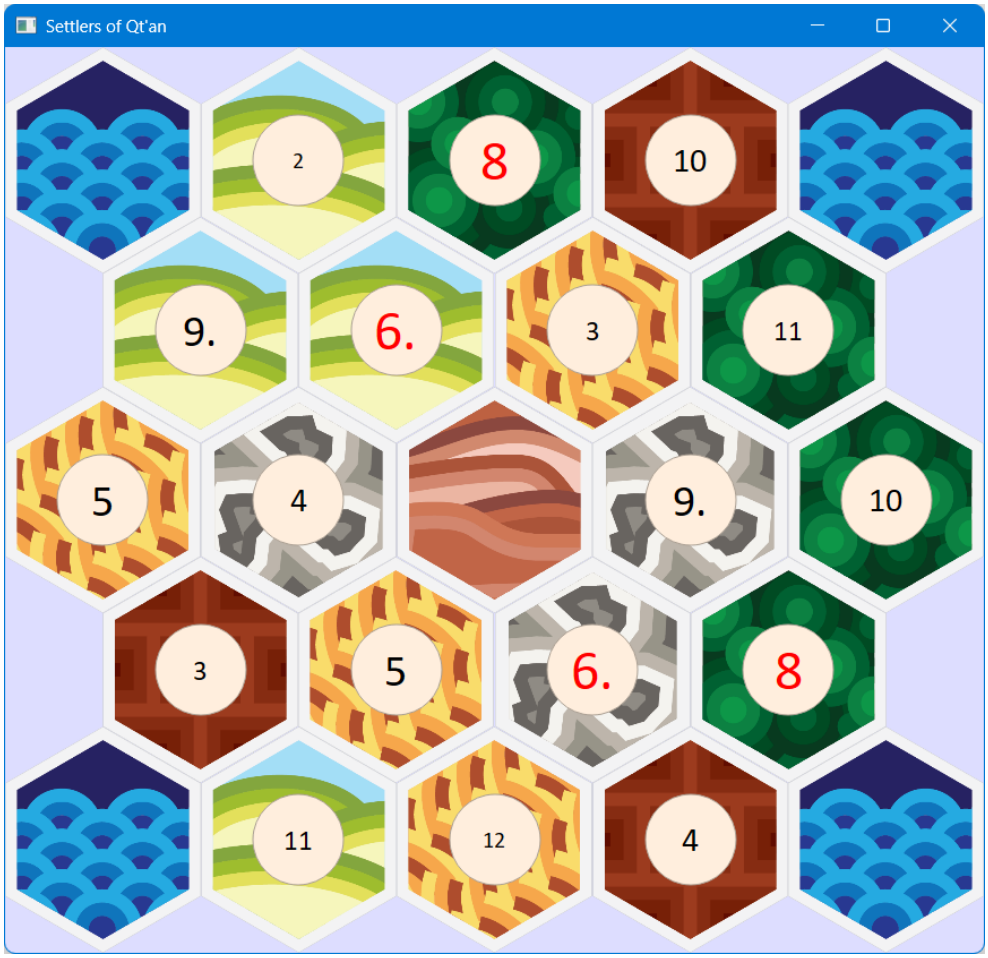
Aufgabe 73: Die Siedler von Qt'an ★★★

In dieser Übung zeichnen wir die Insel aus dem weltberühmten Brettspiel „Die Siedler von Qt'an“:



- Das Spielfeld ist wie links nebenstehend aus 23 Hexagon-Feldern aufgebaut. Jedes Feld kann sowohl durch einen Index (in Skizze rot dargestellt) als auch durch eine zweidimensionale Koordinate beschrieben werden (in Skizze schwarz dargestellt). Felder einer Spalte haben die gleiche x-Komponente und liegen in einer Zick-Zack-Linie untereinander. Felder einer Reihe haben die gleiche y-Komponente und liegen nebeneinander.
- Im Zentrum der Insel (Index 11) liegt immer ein Wüstenfeld.
- An den vier Ecken (Index 0, 4, 18 und 22) liegen Meeresfelder.
- Um das Zentrum herum verteilen sich auf zufällige Weise je vier Weizen-, Wald- und Weidefelder sowie je drei Lehm- und Erzfelder.
- Im Englischen heißen die Felder wheat, forest, pasture, brick, ore, desert und sea. Praktischerweise unterscheiden sich alle Anfangsbuchstaben dieser Namen, so dass Sie die Buchstaben im Programm als Kürzel bei der Definition des Bretts verwenden können.

Ein komplett gezeichnetes Spielbrett könnte wie folgt aussehen:



Gehen Sie aufbauend in drei Schritten vor und speichern Sie jeweils in ein eigenständiges Programm:

1. ★★ Schreiben Sie ein PySide6-Programm, das die oben gezeigte Insel in einem Fenster der Größe 865 x 800 Pixel zeichnet.

   Tipp: Speichern Sie die Verteilung der Karten zeilenweise in einer verschachtelten Liste. Sie haben dann den Spielplan in einer Datenstruktur, die Sie gut zum Zeichnen des Spielbretts einsetzen können:

```
# Define tile distribution on board (s = sea, f = forest, etc.)
self.board = []
self.board.append(list("spfbs"))
self.board.append(list("ppwf"))
self.board.append(list("wodof"))
self.board.append(list("bwof"))
self.board.append(list("spwbs"))
```

   Für die Felder können Sie die vorbereiteten PNG-Grafiken im Verzeichnis `images` aus der Musterlösung verwenden. Legen Sie ein Dictionary `self.images` an, dessen Schlüssel die Anfangsbuchstaben der englischen Namen und Werte die korrespondierenden `QPixmap`-Objekte sind. Zeichnen Sie Felder jeweils in ein Rechteck der Größe 173 x 200 Pixel. Der Abstand der Felder zueinander beträgt horizontal 200 Pixel und vertikal 150 Pixel.

2. ★★ Auf allen Feldern außer Wüste und Meer werden nun Zahlenplättchen verteilt (siehe obiger Screenshot). Es gibt Plättchen mit folgenden Werten: 2, 3, 3, 4, 4, 5, 5, 6, 6, 8, 8, 9, 9, 10, 10, 11, 11, 12.

   Tipp: Arbeiten Sie wie im vorherigen Schritt mit einer verschachtelten Liste, nur dieses Mal mit Zahlenwerten statt Buchstaben. Felder ohne Plättchen kennzeichnen Sie mit `None`:

```
# Define chip distribution on board tiles
self.chips = []
self.chips.append([None, 2, 8, 10, None])
self.chips.append([9, 6, 3, 11])
self.chips.append([5, 4, None, 9, 10])
self.chips.append([3, 5, 6, 8])
self.chips.append([None, 11, 12, 4, None])
```

   Es gelten folgende Anforderungen für die Grafik:

   - Die Zahlen 6 und 8 werden rot dargestellt.
   - Die Schriftgröße der Zahlen 2…6 wird aufsteigend größer und von 8…12 wieder kleiner.
   - Den Zahlen 6 und 9 folgt ein Punkt, um Verwechselungen zu vermeiden.

3. ★★★ Die Verteilung der Felder und der Zahlenplättchen soll nun per Zufall bestimmt werden, so dass Sie bei jedem Programmstart ein neues, gültiges Spielfeld erwartet. #Gänsehaut!

   Tipp: Starten Sie mit einer Liste, die für allen Felder außer Wüste und Meere jeweils einen Buchstaben enthält, also `list("wwwwffffppppbbbooo").` Diese können Sie mit der Funktion `random.shuffle(…)` des Pakets `random` aus der Python-Standardbibliothek zufällig durcheinanderwirbeln. Dann fügen Sie Wüste und die vier Meeresfelder an den

korrespondieren Indizes hinzu. Die konkreten Indizes können Sie der Skizze zu Beginn der Aufgabe entnehmen. Die nun entstandene Liste beschreibt den Spielplan, wenn man ihn zeilenweise von oben nach unten liest. Schließlich spalten Sie die Liste in Zeilen zu je 5, 4, 5, 4 und 5 Feldern auf und erhalten z.B. `[['s', 'p', 'f', 'b', 's'], ['p', 'p', 'w', 'f'], ['w', 'o', 'd', 'o', 'f'], ['b', 'w', 'o', 'f'], ['s', 'p', 'w', 'b', 's']]` – das ist genau das Format, das wir im ersten Schritt zum Zeichnen des Bretts verwenden!

Bei den zufälligen Zahlenplättchen gehen Sie ähnlich vor, starten aber mit einer Liste der gültigen Zahlenwerte. Nach Mischen, Einfügen von Wüste und Meer und zeilenweisem Auftrennen erhalten Sie wieder eine verschachtelte Liste, z.B. `[[None, 2, 8, 10, None], [9, 6, 3, 11], [5, 4, None, 9, 10], [3, 5, 6, 8], [None, 11, 12, 4, None]]`. Auch hier liegt wieder genau das Datenformat vor, das wir im zweiten Schritt bereits zum Zeichnen der Zahlenplättchen verwenden.

Diese Aufgabe ist eine Hommage an das Brettspiel „Die Siedler von Catan“ (Kosmos-Verlag, 1995). Die Grafiken in dieser Aufgabe stammen von Ryan Schenk unter der Lizenz CC BY-NC-SA 2.0.