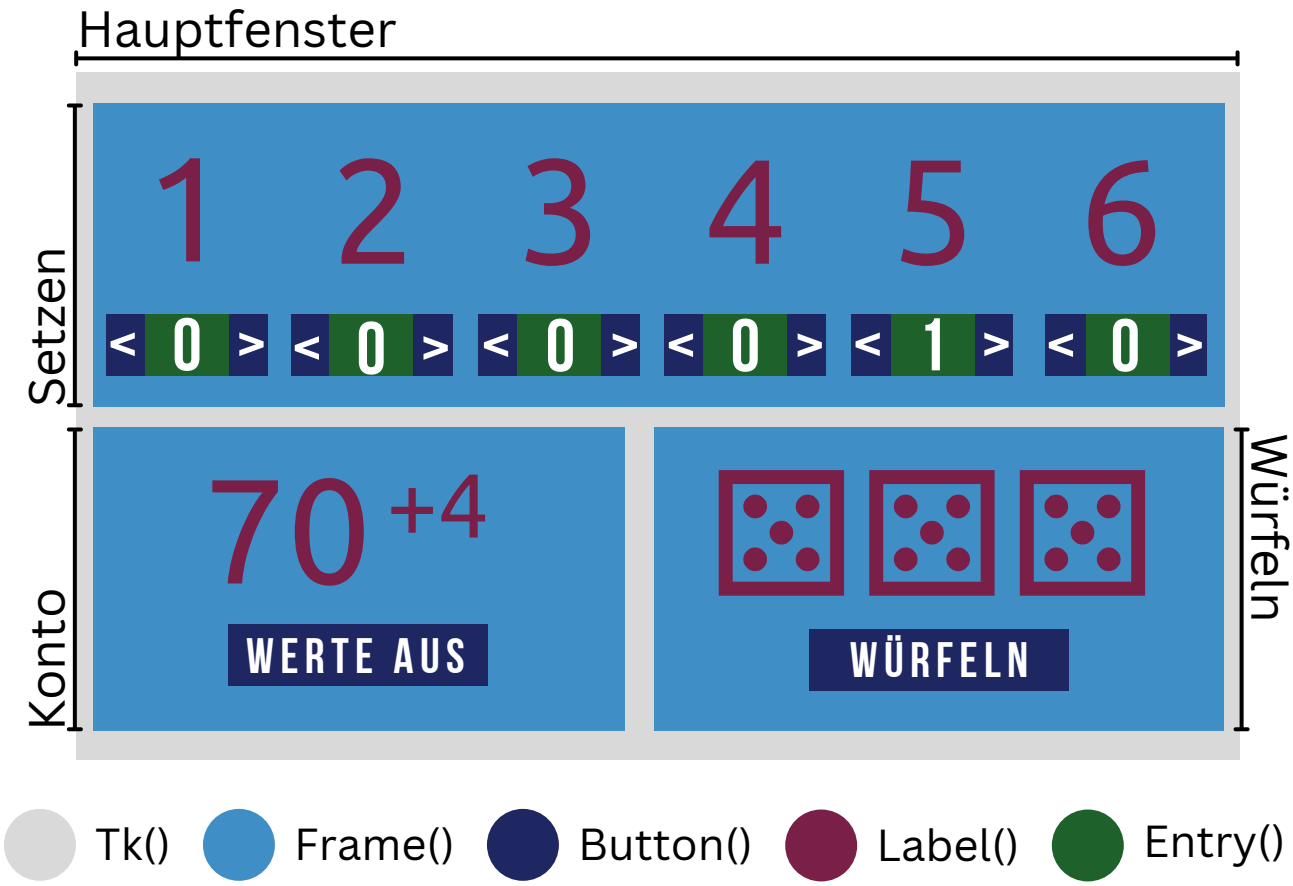
CHUCK A LUCK

KONZEPT



THEMEN

Spielmechanik
- Implementierung
- Sonderfälle

GUI
- elastisches Layout
- anpassbar und erweiterbar

Probleme und Verbesserungsansätze

ANNIKA SCHMIDT

ANFORDERUNG

Die einzelnen graphischen Elemente müssen nicht nur separat funktionieren (Würfelbutton gedrückt
-> Würfeln), sondern zusammen ein flüssiges Spiel ergeben (Würfelbutton gedrückt -> Würfeln -> Gewinn ermitteln -> ...).

UMSETZUNG

Ich habe das Spiel graphisch in drei Parts unterteilt. Tatsächlich ist aber nur einer von ihnen für den Großteil der Spielmechanik verantwortlich.

Setzen: 6 Eingabefelder, auf denen Zahlen >= 0 stehen können

Würfeln:

- Bucht Geld vom Konto ab (so viel wie gesetzt wurde)
- Würfelt drei Zufallszahlen
- Berechnet Gewinn
- Bucht den Gewinn auf das "kleine" Konto

Auszahlen: Bucht das "kleine" Konto auf das "richtige" Konto

(Unterteilung in "klein" und "richtig" gilt rein der Übersichtlichkeit)

SONDERFÄLLE UND ERGÄNZUNGEN

- Nicht genug Geld auf dem Konto (mehr gesetzt, als Geld vorhanden)

-> wird von Würfelbutton geprüft und ruft gegebenenfalls einen Fehler auf

- Spiel verloren

-> wird von Würfelbutton geprüft, ruft Info auf und startet dann ein neues Spiel

- Kontobeträge im 10er- oder 100er-Bereich

-> Schriftgröße muss angepasst werden

- Startkapital selbst bestimmen

-> Inputfenster am Spielstart

- Gewinnberechnung transparent machen

-> Würfelbutton kreiert neues Fenster mit graphischer Berechnung des Gewinns

ANNIKA SCHMIDT

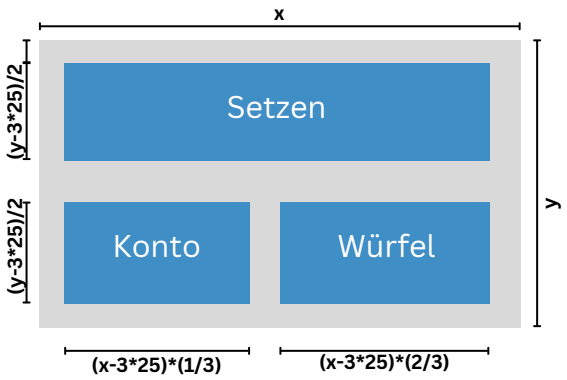
ANFORDERUNG

Die GUI soll ein elastisches Layout* haben.

Dafür müssen
- Widgetgröße
- Widgetanordnung im Raum

relativ zum übergeordneten Widget sein UND in Echtzeit aktualisiert werden.

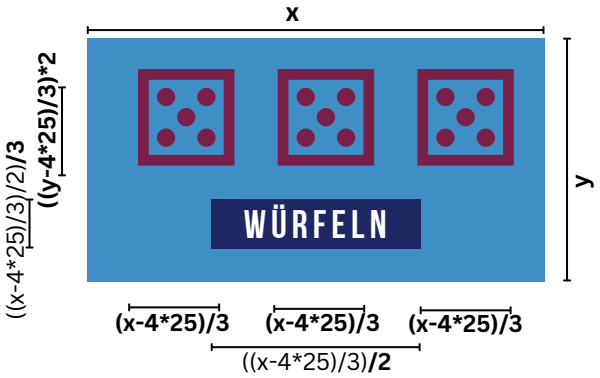
*https://de.wikipedia.org/wiki/Webdesign-Layouttyp#Elastisches_Layout



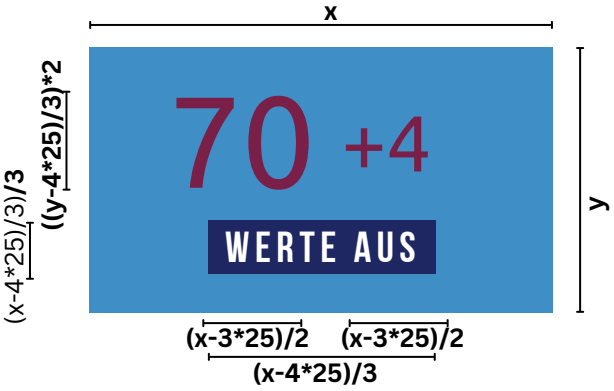
UMSETZUNG

Meine ersten Umsetzungsidee war es, die Größe des Hauptfensters abzurufen und relativ dazu die weiteren Widgetgrößen und Anordnungen zu berechnen. Nun fehlte allerdings noch die Aktualisierung in Echtzeit.

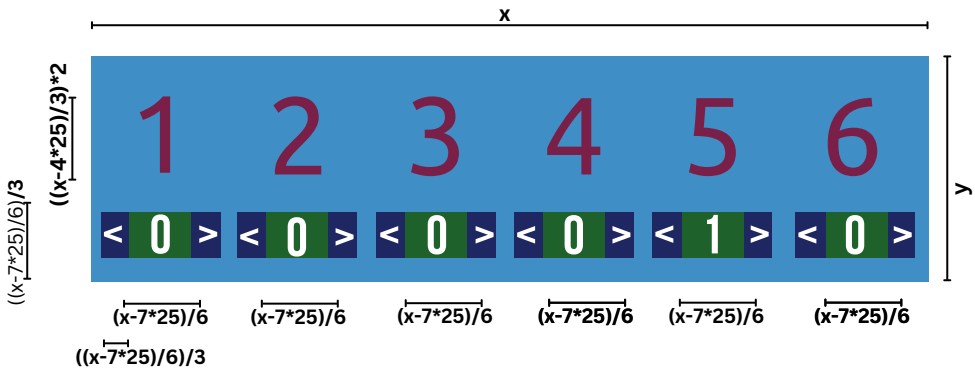
Zuerst wollte ich dazu die update()-Methode nutzen. Diese birgt allerdings Gefahren zu Endlosschleifen.
Dann stieß ich auf den relwidth-; relheight-; relx-; rely-Parameter, der eine Gleitkommazahl zwischen 0 und 1 nimmt (relx=0 wenn der linke Rand des übergeordneten Widgets, relx=1 der rechte Rand des übergeordneten Widgets gemeint ist). Ein Update in Echtzeit ist bereits inbegriffen. Die Gleitkommazahl wäre leicht zu berechnen: Das bereits Berechnete durch die übergeordnete Widgetslänge, bzw. Widgetbreite teilen.



Nun mussten allerdings die Maße des obersten Widgets (in meinem Fall das Hauptfenster) noch abgerufen werden. Dazu habe ich die bind()-Methode verwendet, mit der man eine Python-Funktionen an ein Ereignis binden kann:



widget.bind (Ereignis, Folge)
-> hauptfenster.bind(Größenveränderung, Variableupdaten)



ANNIKA SCHMIDT

ANFORDERUNG

Das Programm sollte möglichst einfach verstanden und abgeändert werden können (z.B. einen Würfel hinzufügen; einen Button entfernen; einen Abstand verändern), ohne, dass die GUI vollständig überarbeitet werden muss.

UMSETZUNG

Umsetzen lässt sich das, indem man möglichst wenig Konstanten im Quellcode verwendet. So braucht man später nur eine Konstante abändern und kann so einen Großteil des Layouts verändern. Wenn man z.B. die Variable der Würfelmenge von 3 auf 4 setzt, ist ohne großen Aufwand Platz für einen vierten Würfel im Layout.

VERBESSERUNGSANSATZ

Ab einem bestimmten Höhen-Seiten-Verhältnis werden Label abgeschnitten. Das liegt daran, dass ihre Einheit nicht in *px* sondern in *em* gemessen wird. Ich habe zwar eine Fallunterscheidung von Verhältnissen, bei denen die Höhe und solchen, bei denen die Breite größer ist, allerdings stößt das System irgendwann an seine Grenzen. Hier gäbe es also noch eine Verbesserungsmöglichkeit für einer besseren Umrechnung.