

Mensch-Maschine-Interface (MMI)

AIT (Prof. Georg Umlauf)

Sommersemester 2021

Project Kniffel

	Student (Vor- und Nachname)	Matrikel-Nummer
1	Leonard Kahdemann	296713
2	Luca Moye	294632

Einleitung:




Im Rahmen unseres MMI Projektes, in welchem wir eine Oberfläche und ein zugehöriges Spiel mit QT erstellen sollten, haben wir ein Kniffel basiertes Spiel entwickelt, welches wir im Folgenden vorstellen möchten. Als Programmiersprache haben wir uns für Python entschieden und die Oberfläche haben wir mit PyQt5 erstellt. Ziel des Projektes ist es sich mit den Regeln des GUI Designs vertraut zu machen und eine funktionierende Spieloberfläche zu generieren.



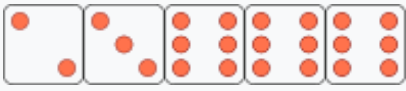
Spielregeln

Jeder Spieler erhält einen Eintrag auf dem Spielblock, in dem er seine Ergebnisse eintragen muss. Gewinner ist, wer am Ende die höchste Gesamtsumme auf seinem Zettel erzielen kann. Gespielt wird mit fünf Würfeln und einem Würfelbecher (Button „Würfeln“) Gewürfelt wird reihum gewürfelt. Jeder Wurf muss mit dem Becher geworfen werden. In jeder Runde darf jeder Spieler dreimal hintereinander würfeln. Dabei darf man „passende“ Würfel durch darauf klicken zur Seite legen (sichern) und mit den verbleibenden Würfeln weiter würfeln. Nach dem zweiten Wurf dürfen Würfel, die beim ersten Wurf behalten wurden, wieder aufgenommen werden. Spätestens nach dem dritten Wurf muss man sich für ein freies Feld auf dem Spielzettel entscheiden, welches nun mit dem Ergebnis dieses Wurfes bewertet wird – oder ein Feld streichen. Der Wurf muss mit dem Becher geworfen werden.

Oberer Block

Im oberen Block („drinnen“ oder Sammeln) werden jeweils gleiche Würfelwerte verzeichnet. Wenn man beim Sammeln in der Summe mindestens 63 Punkte (beispielsweise für jedes Feld drei Würfel) bekommen hat, gibt es einen Bonus von 35 Punkten (Jargon: man „kommt raus“, „man liegt im Soll“).





Kategorie	Beschreibung	Wertung	Beispiel
Einser	Jede Kombination	Die Summe der Augenzahlen der Einser	 zählt 3
Zweier	Jede Kombination	Die Summe der Augenzahlen der Zweier	 zählt 6
Dreier	Jede Kombination	Die Summe der Augenzahlen der Dreier	 zählt 12

Vierer	Jede Kombination	Die Summe der Augenzahlen der Vierer	 zählt 8
Fünfer	Jede Kombination	Die Summe der Augenzahlen der Fünfer	 zählt 0
Sechser	Jede Kombination	Die Summe der Augenzahlen der Sechser	 zählt 18

Unterer Block

Im unteren Block werden sogenannte Figuren gewürfelt und eingetragen. Wenn man ein Ergebnis in ein Feld einträgt, bei dem die Bedingung nicht erfüllt ist (zum Beispiel, wenn beim Feld Dreierpasch nicht drei Würfel gleich sind), dann wird das Feld gestrichen bzw. die Punktzahl „0“ eingetragen.

Kategorie	Beschreibung	Wertung	Beispiel
Dreierpasch	Mindestens drei gleiche Zahlen	Summe aller Augenzahlen	 zählt 17
Viererpasch	Mindestens vier gleiche Zahlen	Summe aller Augenzahlen	 zählt 24
Full House	Drei gleiche und zwei gleiche, andere Zahlen	25	 zählt 25

Kleine Straße	Vier aufeinanderfolgende Augenzahlen (1-2-3-4, 2-3-4-5 oder 3-4-5-6)	30	 zählt 30
Große Straße	Fünf aufeinanderfolgende Augenzahlen (1-2-3-4-5 oder 2-3-4-5-6)	40	 zählt 40
Kniffel	5 gleiche Zahlen	50	 zählt 50
Chance	Jede Kombination	Summe aller Augenzahlen	 zählt 13

Sonderregeln

Beim Kniffel sind für zusätzliche Kniffel-Würfe Sonderpunkte vorgesehen, die allerdings (noch) nicht implementiert wurden. Dafür werden nach offiziellen Regeln für jeden weiteren Kniffel auf der Rückseite des Spielblatts jeweils 50 Punkte vermerkt, zugleich wird bei den Zahlenwürfen im oberen Bereich die Summe der Würfel eingetragen – ein Kniffel aus fünf Zweien bringt dann also 10 Punkte im oberen Feld + 50 Punkte Bonus.

Spielende

Ist der Spielzettel voll, so ist das Spiel beendet und die Punkte vom Sammeln und den anderen Spielen und eventuell der Bonus werden zusammengezählt. Es gewinnt der Spieler mit den meisten Punkten.

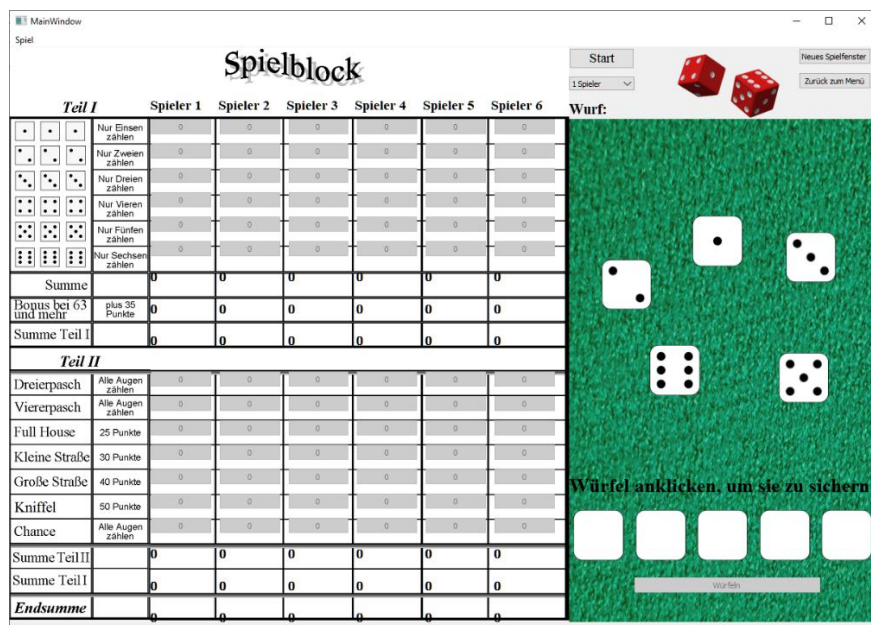
Die Oberfläche

Die Oberfläche des Spiels besteht aus einem zentralen Menü, wo von man auf alle wichtigen Spielflächen und Funktionen zugreifen kann. Diese ist einfach zu verstehen und übersichtlich aufgebaut, damit sich auch unerfahrene Spieler schnell zurechtfinden.

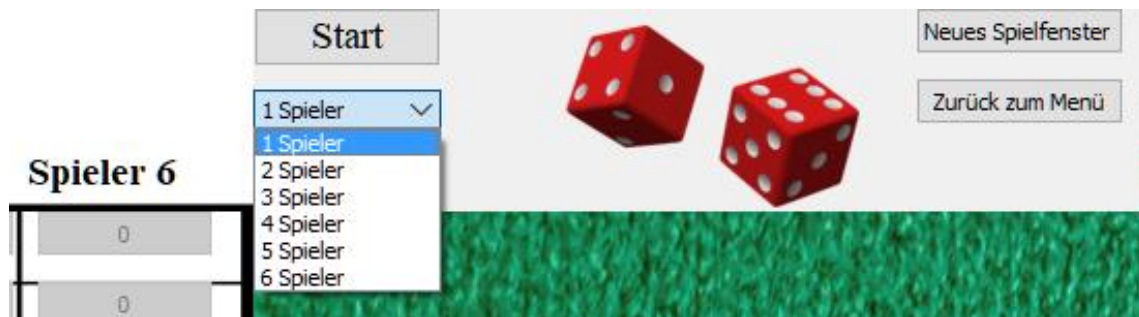
Dein Einstieg macht das Hauptmenü, über das man das Spiel starten, Hilfe zu den Funktionen erhalten oder die Spielregeln studieren kann. Zudem ist es möglich die Bestenliste einzusehen und das Spiel zu beenden. Letzteres ist ebenfalls jeder Zeit über das rote „X“-Symbol in der rechten oberen Ecke oder über einen ActionButton in der Menüleiste möglich. Für eine atmosphärische Stimmung sorgt ein Bar-Soundtrack, der im Hintergrund abgespielt wird.



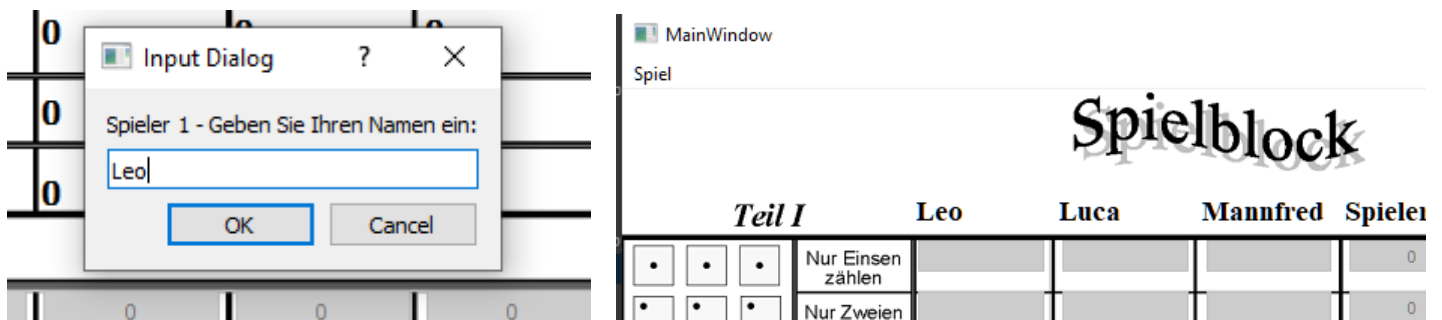
Wird **SPIELEN** gewählt gelangt man auf das nächste Fenster, welches das Spielfeld samt Kniffel-charakteristischem Spielblock und Würfelfeld darstellt. Dort wird gespielt.



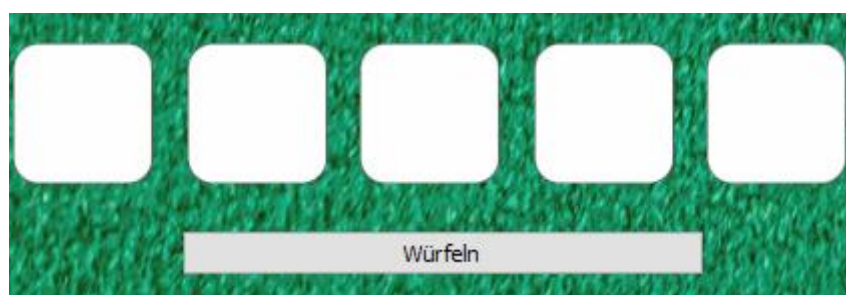
Im Spielblock können sich bis zu sechs Spieler gleichzeitig eintragen und gegen einander antreten. Zuerst sollte die Anzahl der Spieler per Dropdown-Menü festgelegt werden, da diese sich im späteren Spielverlauf nicht mehr ändern lässt.



Wird ein dann ein Spiel über den **START**-Button gestartet öffnet sich die Namenseingabe per MessageBox. Jeder Spieler hat nun die Möglichkeit nacheinander seinen Namen einzutragen.



Wurden die Namen aller Spieler eingetragen werden diese im Spielblock hinterlegt und der erste Spieler beginnt das Spiel mit dem Druck auf den **WÜRFELN**-Button.



Hat der Spieler den **WÜRFELN**-Button gedrückt erhält er sofort Feedback in Form eines Würfelbecher-Würfel-Sounds. Dann hat er die Möglichkeit seine Würfel für den nächsten Wurf zu sichern.

Sind zufriedenstellende Würfelbilde oder Augenzahlen vorhanden kann der Spieler seinen Wurf jeder Zeit durch Drücken des jeweiligen Buttons im Spielblock eintragen. Nach dem dritten Wurf muss eingetragen werden. Um Fehler zu vermeiden, hat jeder Spieler nur die Möglichkeit in seiner Spalte Punkte einzutragen, die Spalten der anderen Spieler wird währenddessen blockiert.

Spielblock

Start
Neues Spielfenster

3 Spieler
Zurück zum Menü


Teil I

	Leo	Luca	Mannfred
Nur Einsen zählen	0		
Nur Zweien zählen	2		
Nur Dreien zählen	0		
Nur Vieren zählen	4		
Nur Fünfen zählen	0	15	
Nur Sechsen zählen	24		
Summe	24	15	0
Bonus bei 63 und mehr	plus 35 Punkte	0	0
Summe Teil I	24	15	0


Teil II

	Leo	Luca	Mannfred
Dreierpasch	Alle Augen zählen	24	
Viererpasch	Alle Augen zählen	0	
Full House	25 Punkte	25	
Kleine Straße	30 Punkte	0	
Große Straße	40 Punkte	0	40
Kniffel	50 Punkte	0	
Chance	Alle Augen zählen	24	17
Summe Teil II	25	40	57
Summe Teil I	24	15	0
Endsumme	49	55	57

Wurf übrig: 0



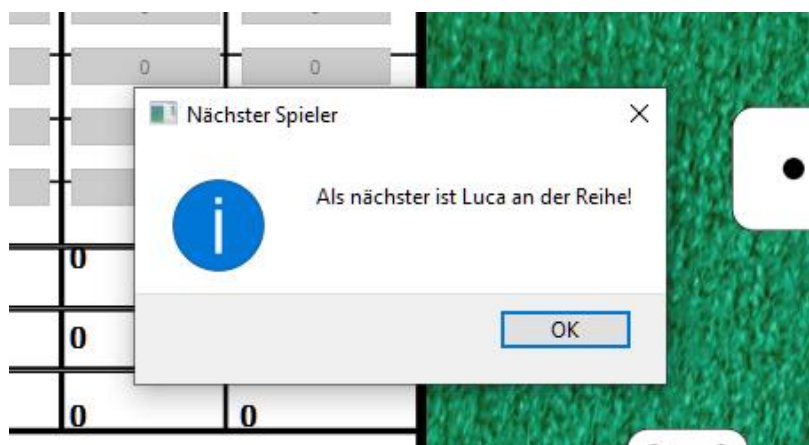
Würfel anklicken, um sie zu sichern



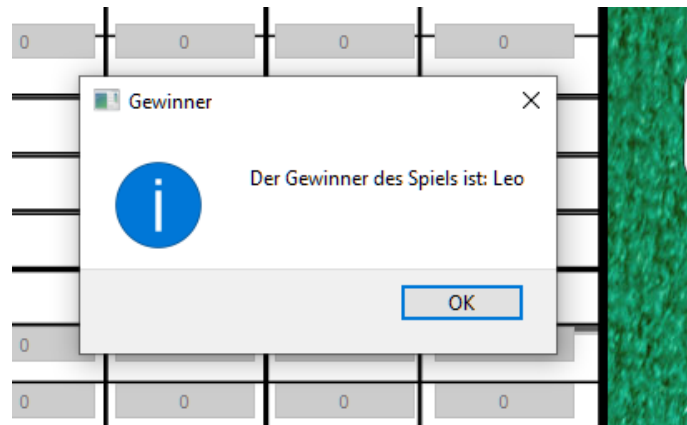
Würfel

Ist es nicht möglich eine passende Kategorie zu finden, muss eine ausgewählt werden, die gestrichen wird. Dies geschieht durch Drücken eines **Kategorie**-Buttons der als Wert die „0“ anzeigt.

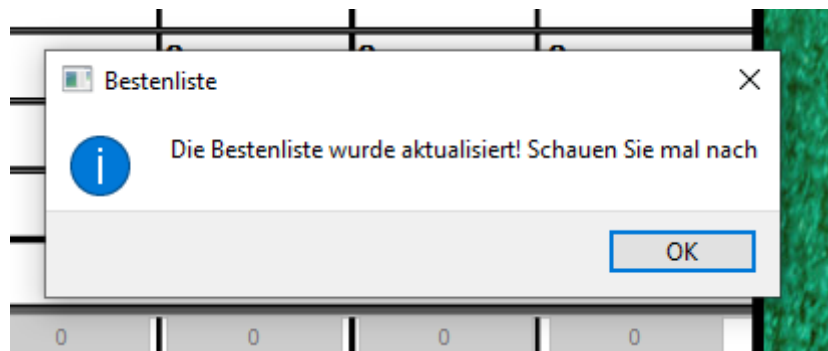
Dann ist der nächste Spieler an der Reihe, der die gleichen Züge durchläuft. Der nächste Spieler wird mittels MessageBox angezeigt.



Sind alle 14 Runden durchlaufen, wird ein Sieger gekürt. Dieser wird mittels MessageBox angezeigt und durch einen Gewinner-Sound beglückwünscht.



Die 10 Spieler mit den meisten Punkten jemals werden in der Bestenliste verwahrt.



Mit dem Button „Neues Spielfenster“ kann nach Beendigung des Spiels ein neues Spiel erstellt werden, um direkt weiterzuspielen.

Die 8 goldenen Regeln des GUI Design (nach Shneiderman)

Während der Erstellung unserer GUI haben wir uns an die 8 goldenen Regeln der GUI-Designs nach Shneiderman orientiert. Diese sind im Folgenden aufgeführt und auf unser Projekt übertragen:

1. Konsistenz anstreben

Jeder unseren Button oder Funktionen ist einer festen Aktion zugewiesen, die vom Benutzer vorhergesehen werden können. Durch die Anordnung im GridLayout, ist eine einfachere Funktionszuweisung der Buttons möglich, da Buttons mit derselben Funktion, von anderen Spielern direkt daneben liegen.

2. Berücksichtige unterschiedliche Erfahrungen

Auf Grund der einfachen und übersichtlichen Anordnung unserer Schaltflächen sollten Spieler jeder Erfahrungsstufe mit unserer GUI zurechtkommen. Zudem haben wir bei der Erstellung der Spieloberfläche darauf geachtet, dass diese möglichst nah an der realen Variante angelehnt ist

3. Rückmeldungen auf Aktionen des Benutzers

In den meisten Situationen wird der Benutzer über die Ausführung seiner Eingabe mittels MessageBox, Hinweis oder Spielaktion informiert.

4. Abgeschlossene Benutzerdialoge

Jeder Dialog, in dem der Benutzer eine Eingabe tätigen muss, ist fest zugeordnet, einmalig und bedarf keiner weiteren Eingabe von Daten.

5. Vermeide Fehler

Irren ist menschlich, denn noch war unser Ziel ein gutes Spielerlebnis zu gewährleisten. Deshalb wurden Verlinkungen und Aktionen mehrfach getestet und alle bekannten Fehler weitestgehend behoben.

6. Es muss immer ein Zurück geben

Jedes Fenster oder Dialog bietet eine Möglichkeit, um zum vorherigen Menü zurückzukehren. (**ZURÜCK**-Button, rotes „X“-Symbol)

7. Benutzer muss Aktionen initiieren

Der Benutzer hat die Kontrolle, das Programm wartet nach jeder Eingabe oder Aktion auf den nächsten Befehl des Benutzers.

8. Geringe Belastung des Kurzzeitgedächtnisses

Alle öfters auftretende Eingaben oder Dialoge sind aussagekräftig und beschreiben alle benötigten Parameter.

Spielimplementierung 1. Erstellung der Spielelogik im Terminal

Neben der Erstellung der Oberfläche, war es parallel das Ziel, das Spiel im Terminal spielen zu können. Hierfür 2 Dateien erstellt. Die Kniffelclass.py und die main.py. Die main.py wird im späteren Verlauf jedoch nicht mehr gebraucht, da die Aufrufe der Funktionen die Oberfläche übernehmen.

Jedoch kann die kniffelclass.py genutzt werden um viele Aktionen auszulagern.

Im Folgenden wird erklärt, wie die Kniffelclass.py erstellt wurde. Zunächst war die Idee, eine Klasse für jeden Spieler zu erstellen, die für den jeweiligen Spieler, sowohl Namen, Punktzahl als auch eine Liste mit eingetragenen Ereignissen speichert. Dies wurde mit den Variablen Punkte, Name und der Liste bisherig (Z. 7-9) realisiert. Zudem braucht die Klasse Funktionen, welche zusammen mit dem Spieler aufgerufen werden können. Diese Funktionen beinhalten eine Funktion zum Würfeln, eine save Funktion, um sich nach einem Wurf Würfel sichern zu können, die im nächsten Wurf nicht mitgewürfelt werden sollen bzw. falsch gespeicherte Würfel wieder in den Würfelbecher zurückgibt und dann abfragt, ob die Auswahl so korrekt ist, um den nächsten Wurf vorbereiten zu können. Außerdem benötigen wir eine Funktion *possiblePoints*, die uns für jedes Ereignis, welches eingetragen werden könnte, die richtigen Punkte angibt. Die Funktion *getPunkte* ist dafür da, um alle Punkte zu addieren. Nun werden die oben genannten Funktionen noch im Einzelnen erklärt.

Manche Teile des Codes in Kniffelclass.py haben für den aktuellen Stand des Programms keine Funktion mehr, da Sie Befehle im Terminal ausführen und dort Eingaben erwarten etc. Wir wollten Sie jedoch im Code behalten, um den Fortschritt unseres Codes zu veranschaulichen.

```
def wuerfeln(self,numwuerfel,wurf): (Zeile 11-16)
```

Um Würfeln zu können, benötigen wir zuerst am Anfang des Files „import random“ um Zufalls- bzw. Pseudozufallszahlen zu erzeugen. Die Funktion wird aufgerufen, mit der Anzahl an Würfeln und der Wurfnummer (Bei Wurf 1 automatisch mit 5 Würfeln bei Wurf 3 muss eingetragen werden und alles größer als 3 sollte nicht möglich sein. Um die Würfel abzuspeichern, wird die Liste „gewuerfeltewuerfel“ (Z.12) erstellt. Danach wird für die Anzahl an Würfeln, die übergeben werden sollen, in einer for Schleife ein Wert zwischen 1-6 pro „numwuerfel“ generiert und in „gewuerfeltewuerfel“ zurückgegeben.

```
def save(self,wuerfel,save,wurf): (Z.23-46, wird später nicht verwendet aufgrund von Terminaleingaben)
```

Übergabewerte: Saveliste, Würfel im Würfelbecher, Wurf

Hier wird zunächst unterschieden, ob wir uns in den ersten beiden, oder im letzten Wurf befinden, da es beim letzten Wurf keinen Unterschied macht, da man sowieso nicht mehr würfeln darf, werden einfach alle Würfel in save gespeichert. Wenn wir uns aber in den ersten beiden Würfeln befinden, wird eine Schleife durchlaufen in der so lange hintereinander abgefragt wird, welche Würfel gespeichert werden sollen und welche wieder in den Würfelbecher sollen, bis der Benutzer angibt,

dass er zufrieden mit seiner Eingabe ist. Wenn ein Würfel ausgewählt wurde, wird beim speichern die *Saveliste* um den Eintrag des Würfelbechers erweitert und der Eintrag im Würfelbecher gepopt.

Genauso in die andere Richtung, wenn ein Würfel in den Würfelbecher zurücksoll.

```
def possiblepoints(self, save, wuerfel, wurf): (Z.48-184)
```

Dies war vermutlich die umfangreichste Funktion, da sie alles enthält, für das man im Spiel Punkte bekommen kann und berechnet wie viele Punkte für jedes Ereignis eingetragen werden können.

Zunächst werden temp Listen von *save* und *bisherig* erstellt, um nicht auf den Originalen herum zu schreiben (Z.49-56). „*tempsave*“ hat zudem den Vorteil, dass garantiert immer alle Würfel berücksichtigt werden, egal ob Sie im Würfelbecher liegen oder unter den Gesicherten, dadurch dass sowohl *save* als auch Würfel appended werden (Z.51 + 53)

Danach werden die oberen Felder des Kniffelblocks, also die Ziffern 1-6 abgefragt, wie oft die jeweiligen Ziffern unter den Würfeln des Spielers sind. Diese werden in einer jeweiligen Variablen gespeichert. Dies hat den Vorteil, dass diese Variablen bei den unteren Ereignissen des Würfelblocks mitverwendet werden können.

Allgemein wird zuerst geprüft, ob in das Feld schon einmal etwas eingetragen wurde bzw. ob das Feld leer ist, um Doppeleintragungen zu verhindern.

Für die Felder 1 bis 6 wird dann die Anzahl der Würfel mit der Augenzahl multipliziert, um zu ermitteln, wie viele Punkte eingetragen werden können. Diese Punkte werden dann an der jeweiligen Stelle in „*tempbisherig*“ eingetragen. Die Reihenfolge bzw. von „*bisherig*“ und „*tempbisherig*“ setzen sich aus allen Punktevents von oben nach unten zusammen + die Felder, in die die Summen und der Bonus eingetragen werden zum Schluss (auch von oben nach unten).

Nachdem für die Zahlen 1-6 die Anzahlen festgestellt wurden, müssen nun die Sonderevents festgestellt werden.

Beginnend beim Dreier und Viererpass (Z.107 – 125) wird festgestellt, ob eine Zahl mindestens 3- bzw. 4-mal vorhanden ist. Falls dies der Fall ist, werden alle Augen zusammenaddiert und in „*tempbisherig*“ eingetragen.

Bei einem Full House (Z.129-134) benötigen wir einen Drilling und ein Paar. Zunächst wird geprüft, ob ein Drilling vorhanden ist. Ist dies der Fall wird zusätzlich geprüft ob ein Paar vorhanden ist. Ist dies beides der Fall werden 25 Punkte eingetragen. (Es gibt Spielvarianten, bei denen die Augenzahl addiert wird. Wir haben uns für die Variante mit festen 25 Punkten entschieden)

Bei den Straßen wird festgestellt, ob 4 bzw. 5 aufeinanderfolgende Zahlen mindestens 1x vorhanden sind. Straßen, die über 1 und 6 hinaus gehen zählen nicht. Für eine kleine Straße werden 30 und für eine große Straße 40 Punkte eingetragen.

Beim Kniffel gibt es neben dem Abprüfen ob 5 gleiche Würfelaugen vorhanden sind eine weitere Bonusregel. Für einen 2. Kniffel werden 50 Bonuspunkte eingetragen, solange man diesen auch bei einer der oberen Felder eintragen kann. Wird z.B. ein 2. Kniffel mit z.B. 3ern erreicht so werden bei dem 3er Feld anstatt 15, 65 Punkte eingetragen, wenn noch nicht bei 3 eingetragen wurde.

Bei der Chance werden nur alle Augen zusammengezählt.

Zurückgegeben wird dann *tempbisherig*, und nach einer Eingabe des Benutzers wird nur der Eintrag, den der Benutzer gewählt hat in *bisherig* eingetragen und *tempbisherig* gelöscht.

```
def getPunkte(self): (Z.187-198)
```

Hier wird die ganze Liste „*bisherig*“ durchlaufen und die Punkte werden addiert, um sie in die Ergebniszellen am Ende von „*bisherig*“ einzutragen. In einem if-cause wird zudem abgefragt, ob der Bonus im oberen Teil erreicht wurde oder nicht. Je nach dem werden dann 35 Punkte eingetragen oder eben nicht.

```
def eintragen(self,stelle,punkte): (Z.214-216)
```

Übergabeparameter: Die Stelle an der in „*bisherig*“ eingetragen werden soll, Wie viele Punkte

Diese Funktion ist er später entstanden und wird für die Terminal Benutzung nicht gebraucht. Über diese Funktion wird später erkannt, welches Ereignis auf der QT Oberfläche gedrückt wurde, um dann die Punkte an der richtigen Stelle einzutragen.

Spielimplementierung 2. Verknüpfung mit QT Spieloberfläche

Nachdem ein funktionierendes Spiel im Terminal erstellt wurde, ging es nun darum, dass die Input Eingabe Felder im Terminal und die prints auf die QT Oberfläche übertragen werden.

Hierfür wurde die Datei Main.py erstellt. In der zunächst alles an Funktionen importiert wurde, welche benötigt wurden, um die Funktionen des Spiels realisieren zu können.

Danach werden die Klassen „*hauptmenue*“ und „*spielmenue*“ erstellt, die später noch genauer erklärt werden.

Zunächst wird unten eine Application generiert (ab Z. 490) und 2 Main Windows generiert.

Dann wird für „*hauptmenue*“ und „*spielmenue*“ jeweils 1 Objekt erstellt und einem MainWindow zugewiesen. Da zunächst nur das Hauptmenü angezeigt werden soll, wird nur MainWindow1 angezeigt.

```
class hauptmenue(Ui_MainWindow): (Z.14-33)
```

Die Klasse "hauptmenue" erbt zunächst alle Eigenschaften von "Ui_MainWindow". Das ist der Name der Klasse in Hauptfenster.py. Aufgrund dieser Vererbung können wir alle Buttons verwenden, die wir im Designer erstellt haben.

```
def __init__(self, window):
```

In der init Funktion, wird das Fenster aufgebaut, hierfür wird auch die *super().__init__()* Funktion benötigt, um die Elterninstanz zu initialisieren und deren Funktionen zu benutzen, von der die Fensterklasse in der *Main.py* erbt.

Danach wird das Window mit „*setupUi(window)*“ generiert

Zudem weisen wir dem Fenster den Barsound.mp3 file in einer Loop zu um im Menü dauerhaft Musik zu haben. Um die Größe der Ressourcen zu begrenzen wurde der Sound file auf 1min zugeschnitten.

Zudem weisen wir noch unsere Buttons, den jeweiligen Funktionen zu.

Wenn Spielen geklickt wird, soll das Hauptmenü *hide()* ausführen und das Spielmenü soll *show()* ausführen um die Fenster zu wechseln. Zudem soll der Sound vom Hauptmenü aufhören, da es sonst evtl. nervig wird für den Spieler.

Falls auf den Beenden Button gedrückt wird, soll das Programm geschlossen werden.

```
class spielmenue(spielfenster): (Z.75-502)
```

In dieser Klasse steckt das Spiel, welches mit Hilfe von Kniffelclass das Spiel ablaufen lässt.

Die Initialisierung erfolgt ebenso wie beim Hauptmenü. Dann werden jegliche Variablen erstellt, die später benötigt werden und jegliche Buttons verknüpft, welche später eine Funktion haben sollen.

Hier eine Auflistung der wichtigsten Variablen: (Z.79-88)

```
self.numplayers = 0 – Anzahl der Spieler
```

self.Wurf = 0 – Welcher Wurf (1-3)
self.Runde = 0 – wie viele Runden wurden gespielt
self.played = 0 – Hilfsvariable für Menüwechsel, wenn neue Fensterinstanz erstellt wurde
self.dran = 0 - index für die Liste Spielerliste um zu wissen wer dran ist
self.Spielerliste = [] – Hilfsvariable für Menüwechsel, wenn neue Fensterinstanz erstellt wurde
self.wuerfelbecher = [] – Liste für Würfel die gewürfelt werden sollen
self.gesichert = [] – Liste für Würfel die gesichert werden sollen
self.wuerfelsound = AudioPlayer("wuerfelsound.mp3") – Sound beim Würfeln
self.winnersound = AudioPlayer("winnersound.mp3") # Sound beim Popup, wenn Gewinner angezeigt wird

self.buttonlist = [self.buttonsp1, self.buttonsp2, self.buttonsp3, self.buttonsp4, self.buttonsp5, self.buttonsp6]

- Liste für dynamischen Zugriff auf PushButtons im Grid Layout, um diese dynamisch anzusteuern

self.buttonsp1 ist Beispielsweise das Vertical Layout zu allen PushButtons von Spieler1

self.labellist = [self.labelS1, self.labelS2, self.labelS3, self.labelS4, self.labelS5, self.labelS6]

- In labellist werden alle Labels der jeweiligen Spieler gespeichert, um diese dynamisch anzusteuern

self.wuerfelist = [self.wuerfel1, self.wuerfel2, self.wuerfel3, self.wuerfel4, self.wuerfel5]

- In *wuerfelist* werden alle wuerfelobjekte gespeichert, die auf dem Grün angezeigt werden, um diese dynamisch anzusteuern

self.savelist = [self.save1, self.save2, self.save3, self.save4, self.save5]

- Liste in welcher alle save slots auf dem Grün gespeichert sind, um diese dynamisch anzusteuern

self.namelist = []

- Liste in der alle Namen von Spielern abgespeichert werden (für Popups)

Wenn das Fenster angezeigt wird, kann das Spiel gestartet werden, indem in der Combobox ausgewählt wird, wie viele Spieler spielen möchten. Danach kann auf „Start“ gedrückt werden.

Mit dem Button „Start“ wird die Funktion „*rungame*“ (Z.186) gestartet.

```
def rungame(self): (Z.227-267)
```

Zuerst wird die ComboBox mit der Spieleranzahl ausgelesen und in „*numplayers*“ gespeichert. Danach werden die PushButtons für „Start“ und „Restart“ deaktiviert, um einem Absturz durch doppeltes Drücken vorzubeugen. Die ComboBox wird ebenfalls deaktiviert, da sie nicht mehr gebraucht wird.

Ziel war es dem Benutzer möglichst wenige Aktionen zu geben, damit er sich einfacher zurechtfinden kann. Alles was gerade nicht gebraucht wird und Fehler verursachen könnte durch falsche Nutzereingaben wird zunächst deaktiviert, bis es gebraucht wird.

In der for Schleife (Z.233-235) werden alle PushButtons zum Eintragen von Punkten deaktiviert, da es zu einem Absturz geführt hat, wenn ein Benutzer einen Wert eintragen will, obwohl noch nicht gewürfelt wurde. Zudem werden vor dem ersten Wurf auch die Würfel versteckt.

Dann beginnt das Spiel in dem „Wurf“, „Runde“ und „dran“ auf 1 gesetzt werden.

In Z.243-244 werden in einer for Schleife alle *kniffelclass* objekte erzeugt und Spielerliste hinzugefügt. Ist z.B. *numplayers* = 2 wird folgendes ausgeführt:

```
globals() player1 = kn.Kniffel
```

```
globals() player2 = kn.Kniffel
```

In Zeile 257 – 259 werden alle Felder mit dem Inhalt von den Punkten des jeweiligen Spielers befüllt. Zum Startzeitpunkt ist also jedes *pushButton* Textfeld, der aktiven Spieler leer.

In Zeile 260 – 263 wird für jeden Spieler ein Name abgefragt und dieser wird in die Label über der jeweiligen Spalte geschrieben. Danach werden die Namen in die „*namelist*“ hinzugefügt)

In Zeile 264-266 werden die Würfel und Saveslots deaktiviert, um fehlerhafte Benutzereingaben zu vermeiden, diese werden ebenfalls erst freigegeben, wenn gewürfelt wird.

Wenn alle Spieler ihre Namen eingetragen haben, kann das Spiel beginnen. Jedes Mal, wenn ein anderer Spieler an der Reihe ist erscheint ein Popup window über die Funktion

```
def popup(self): (Zeile 424-430)
```

Das Window liest den Namen aus, der aktuell an der Reihe ist und gibt ihn als *InformationWindow* an die Spieler aus.

Ist Spieler 1 bereit kann er würfeln. Hierfür drückt er auf den „Würfeln“ *pushButton* und öffnet damit die Funktion.

```
def wuerfeln(self): (Z 269–336):
```

Zunächst werden alle *pushButtons*, des Spielers, der an der Reihe ist wieder freigeschalten.

Danach wird unterschieden welcher Wurf ausgeführt wird. Da der Würfelbecher am Anfang leer ist, gibt es ein Problem mit Zeile 248 beim ersten Wurf. Die Bedeutung von Z. 276-279 wird klarer, wenn die „*wuerfelsave*“ und die „*wuerfelzurueck*“ Funktion erklärt werden. Z.276-287 haben die Funktion

zu erkennen, ob der Würfelbecher aktuell leer ist, da alle Würfel in den Saveslots liegen. Wenn dies der Fall ist, wird ein Popup Window zur Information generiert und die Funktion wird abgebrochen. Der Wurf zählt nicht und kann wiederholt werden.

Wenn Würfel vorhanden sind, wird der Würfelsound 1x abgespielt, dann werden die Würfel angezeigt und Würfel und Savebuttons freigegeben (Z. 288 – 295)

Nun wird der Würfelbecher auf Einträge mit 0 überprüft, dies ist wichtig, da diese Würfel nicht gewürfelt werden dürfen. Sie sind Überbleibsel davon, wenn ein Würfel von „*wuerfelbecher*“ in „gesichert“ verschoben wird. Wenn die Zahl in Würfelbecher nach dem verschieben gepopt werden würde, würde sich der index der nachfolgenden Würfel verschieben und somit könnte man einen Fehler verursachen, wenn man Würfel 3 sichert und danach Würfel 5 sichern möchte, weil „*wuerfelbecher*“ nicht mehr die Länge 5 hat und sich daher der Wert von *wuerfelbecher*[4] auf *wuerfelbecher*[3] verschoben hat. Aus diesem Grund füllen wir alle verschobenen Würfel mit 0 auf. So wissen wir, dass diese Stelle ignoriert werden muss, jedoch behalten alle anderen Werte ihre Position.

Falls es der 1. Wurf ist, wird immer mit 5 Würfeln gewürfelt. Hierbei wird in Zeile 301 eine Funktion von *Kniffelclass* verwendet. Es wird mit *self.Spielerliste[self.dran-1]* der aktuelle Spieler ermittelt und dessen Würfelfunktion aufgerufen. Die Ergebnisse werden in „*wuerfelbecher*“ gespeichert und dann die möglichen Punkte für diesen Wurf aufgerufen um die Punkte zu aktualisieren. Außerdem wird der Wurf um 1 erhöht.

In den anderen Würfen werden nur die Würfel gewürfelt, welche ungleich 0 sind (Z. 310 + 317) und Wurf 3 unterscheidet sich zu Wurf 2 darin, dass der Würfelbutton deaktiviert wird, um zu verhindern, dass öfter als 3x gewürfelt wird, bzw. der Button öfter als 3x gedrückt werden kann.

Zuletzt werden die Werte von „*wuerfelbecher*“ einem Bild eines Würfels zugeordnet, welcher die passende Augenzahl hat und diese wird dann als Icon für den jeweiligen Würfel auf dem Grün angezeigt. (Z.323-336)

```
def wuerfelsave(self, saven): (Z.338 – 349):
```

„*wuerfelsave*“ wird dazu verwendet, die Würfel die im Würfelbecher angezeigt werden, durch einen Klick auf den entsprechenden Würfel in die Saveliste zu verschieben und unten anzuzeigen.

Das besondere hier am Aufruf ist, dass alle Würfel dieselbe Funktion aufrufen, jedoch eine andere Aktion ausgeführt wird. Hierfür muss das Programm möglichst dynamisch steuerbar sein.

Mit Zeile 204-205 können wir jedem Würfel eine eigene Action zuweisen, welche ein Index von 0 bis 4 ist. Daraus kann die „*wuerfelsave*“ Funktion anhand des Index „*saven*“ erkennen, welchen Würfel sie verschieben muss.

Hierfür wird zuerst der Wert an der Stelle „*saven*“ im Würfelbecher an die „gesichert“ Liste angehängt (Z.340). Danach muss der Würfel, der verschoben werden soll, versteckt werden (Z.343) und das Icon des 1. Leeren Felds auf der Savelist mit dem passenden Bild an Augenzahlen gefüllt werden (Z. 344).

Danach wird der Wert in „*wuerfelbecher*“ an der verschobenen Stelle 0 gesetzt (aufgrund der bereits in „*wuerfeln*“ genannten Verschiebung der Indizes)

Um zu gewährleisten, dass nur Savebuttons gedrückt werden können, auf denen auch Inhalte liegen, werden die Savefelder je nach aktueller Länge aktiviert und deaktiviert (Z.346-349)

Um Würfel wieder von der Savelist in den Würfelbecher zurück zu verschieben, benötigen wir die Funktion:

Hierbei ist wichtig, dass beim Entfernen eines Elements, die anderen Elemente nach vorne rücken und das letzte Element, welches jetzt wieder leer ist deaktiviert wird.

```
def wuerfelzurueck(self, back): (Z.351-378):
```

Zunächst werden wieder die Länge der Savelist und der Wert der zurück in den Würfelbecher soll gespeichert. Danach wird durchlaufen an welcher Stelle der wuerfelbecher eine 0 als Wert hat, um dort den richtigen Wert einzutragen (Z.357). Wenn dieser Platz gefunden wurde, wird an dieser Stelle der Wert eingetragen, der Würfel wieder gezeigt und das Icon aktualisiert (Z.358-362)

Danach wird die for-Schleife abgebrochen, um den Wert nicht mehrmals einzutragen.

Um den Wert richtig von den Savebuttons zu entfernen benötigt man die Schleife in Zeile 368-374

Hierbei wird ab der Stelle an der der Würfel in den Würfelbecher zurück soll (Stelle j), alles eine Stelle nach links verschoben. Heißt: $j=j+1$ $j+1=j+1$ usw.

In den Zeilen 346-348 wird dann die Stelle von „gesichert“ entfernt und der freigewordene Savebutton mit dem Standardicon gefüllt und deaktiviert.

Um die Punkte anzuzeigen benutzen wir die Funktion possiblePoints, welche ebenfalls die possiblePoints Funktion aus Kniffelclass, des aktuellen Spielers benutzt.

```
def possiblePoints(self): (Z.432-438)
```

Zunächst wird jeder Button des jeweiligen Spielers, der bereits einen Eintrag aus früheren Runden hat, deaktiviert. Dann wird in „tempbisherig“ der Output von *Player*“x“.possiblePoints(...) gespeichert (Z.436) und danach wird der Text jedes PushButtons des jeweiligen Spielers, der an der Reihe ist, mit den entsprechenden Punkten aktualisiert (Z.437-438)

```
def eintragen(self, action): (Z.440-448)
```

Um die Punkte einzutragen wird der entsprechende *pushButton* mit den Punkten darauf gedrückt und ähnlich wie bei der „wuerfelsave“ Funktion hat hier jeder Button wieder einen anderen index um zu unterscheiden, an welche Stelle von *Player*“x“.bisherig[...] eingetragen werden soll. Dies wird in Zeile 199-200 definiert. In „betrag“ wird der Text des Buttons, also die Punkte gespeichert und dann wird mit der eintragen Funktion von Kniffelclass der „betrag“ an der Stelle „action“ in *bisherig[action]* eingetragen. Nachdem eingetragen wurde, kommt automatisch der nächste Spieler dran, deswegen dienen Z.445-448 als Vorbereitung für den nächsten Spieler indem z.B. die PushButtons zum Eintragen wieder deaktiviert werden und die Funktion „naechster“ aufgerufen wird:

def naechster(self): (Z.380-422):

Nachdem 3 Würfe ausgeführt wurden und/oder das Ergebnis eingetragen wurde kommt der nächste Spieler an die Reihe. Zunächst werden aber noch die eingetragenen Punkte in die Ergebnislabels des letzten Spielers eingetragen (Z. 383-387). Danach wird zunächst unterschieden, ob es einen Wechsel normalen Wechsel zum nächsten Spieler gibt oder ob der letzte Spieler an der Reihe war und es wieder bei Spieler1 weitergeht und die Runde um 1 erhöht wird. Bei einem normalen Wechsel wird „*dran*“ um 1 erhöht und „Wurf“ wieder auf 1 gesetzt. Zudem wird jeder Knopf des Vorgängers deaktiviert, um falsche Eingaben zu vermeiden (Z.393-397). Der *else* cause (Z.398-406) unterscheidet sich nur darin, dass „*dran*“ auf 1 gesetzt wird und alle Buttons von „*numplayers-1*“ (Z.403) anstatt „*dran-2*“ deaktiviert werden.

Danach wird der Würfeln Knopf für den nächsten Spieler wieder aktiviert und ein Popup Window verkündet den nächsten Spieler, der nun an der Reihe ist, solange die Runden ≤ 13 sind, denn ab Runde 14 sind alle Felder eingetragen und der Gewinner muss ermittelt werden (Z. 410-412). Ab hier kann nun auch ein neues Spiel mit dem Knopf „Neues Spielfenster“ gestartet werden

Zum Schluss der Funktion wird die Würfecke wieder vorbereitet (Z.413-422) in dem alle Würfel standardmäßig auf 6 gesetzt werden und die *Saveliste* geleert wird. Zudem werden Würfel und *Saveliste* wieder deaktiviert, um Fehler vor dem 1. Wurf zu vermeiden

In der „*gewinner*“ Funktion wird der Spieler mit den höchsten Punkten ermittelt und dann wird der Gewinner in einer Popup Meldung verkündet.

def gewinner(self): (Z.421-431):

Zunächst wird in einer Variable „*max*“ der *index* des Spielers in der Spielerliste gespeichert, der die meisten Punkte hat. Für den Vergleich wird eine Schleife durchlaufen, die jeden Spieler mit dem Nächsten vergleicht. Danach wird ein Gewinnsound abgespielt und ein Popup Window erscheint um den Namen des Gewinners zu verkünden.

Ist *player1.bisherig[18] < player2.bisherig[18]* --> *i = 1* (Spieler2 ist in *Spielerlist[1]*)

...*bisherig[18]* enthält die Gesamtsumme an Punkten Teil 1 + Teil 2 + Bonuspunkte

Mit *str(self.namelist[max])* (Z. 458) wird der Name des Spielers ausgegeben der gewonnen hat.

Um nach dem Spiel ein neues Spiel zu starten, gibt es die Funktion „*restart*“ die mit dem Knopf „Neues Spielfenster“ verknüpft ist.

Am Ende der Funktion wird mit der Funktion *bestenliste()* abgefragt ob die Punkte der Spieler hoch genug sind um in die Bestenliste eingetragen zu werden. (genaue Funktion siehe unten)

def restart(self): (Z. 214-217):

Zunächst wird das aktuelle *MainWindow2* zerstört, welches das Spielfenster beinhaltet, dann wird ein neues *MainWindow2* erstellt und ein neues Objekt von *spielmenue* wird den *MainWindow2* zugewiesen.

def zurueck(self): (Z.219-225):

Mit dieser Funktion kann durch den Knopf „Zurück zum Menü“ das Fenster zum Hauptmenü zurück gewechselt werden. Der Aufruf ändert sich minimal, wenn die 2. Instanz an Fenstern erstellt wurde, deswegen die if/else Abfrage mit „played“. Solange noch kein neues Fenster erstellt wurde, soll der Fensterwechsel normal laufen und sobald ein neues Fenster erstellt wurde benutzen wir den else Weg. Die Besonderheit ist, dass während dem Spiel ins Hauptmenü gewechselt werden kann, um z.B. die Regeln nochmal nachzulesen o.Ä. und danach das Spiel mit Spielen an derselben Stelle fortgesetzt werden kann, da wir die Fenster nicht schließen, sondern nur einen *hide()* ausführen.

def bestenliste(self): (Z.463-498)

In dieser Funktion wird überprüft, ob die Scores der Spieler hoch genug sind, um in die Top10 Bestenliste eingetragen zu werden. Diese Bestenliste ist in einer .txt Datei gespeichert und beinhaltet zu Beginn/bei erster Ausführung einige Ergebnisse.

Diese Textdatei wird in den Zeilen 464-472 ausgelesen und die Zeilen werden aufgespalten in Namen und Punkte der Platzierung. Nun wird in einer äußeren for Schleife jeder Spieler abgefragt und in einer inneren Schleife überprüft auf welche Platzierung er es in der Bestenliste schafft (bei Platz 10 anfangen). Nun wird unter folgenden Fällen unterschieden: (Z.473-497)

1. Der neue Score ist höher als alle bisherigen und muss auf der 1 eingetragen werden.
2. Der Score ist zwar höher als der aktuelle, es muss aber überprüft werden, ob der Score auch größer ist, als der Nächste.
3. Der neue Score ist sogar kleiner als Platz 10. Die Liste muss also nicht aktualisiert werden.
4. Es wurde eine Stelle gefunden, die auf der Bestenliste ist, jedoch nicht die größte. Somit muss sie hinter dieser größeren Zahl eingetragen werden

Wenn eine Neue Punktzahl eingetragen wird, wird der Name des jeweiligen Spielers aus dem Label ausgelesen, in das der Spieler zuvor seinen Namen eingetragen hat und zusammen mit den Punkten an die richtige Platzierung geschrieben. Zuvor müssen aber alle anderen Plätze eine Position nach unten rutschen. Dies wird in Zeile 476-477 und in Zeile 490-491

Zuletzt wird im Falle einer Aktualisierung ausgegeben, dass sich die Bestenliste aktualisiert hat, und die txt Datei wird mit den neuen Platzierungen beschrieben (Z.499-502).

Um die Bestenliste anzuzeigen wird die Klasse

class bestenliste(besten): (Z.51-72)

Benötigt um im Hauptmenü ein Fenster mit der Bestenliste aufzurufen.

Die Klasse beinhaltet lediglich ein Label, welches auf die .txt Datei zugreift und sie im generierten MainWindow4 darstellt. Für einen dynamischen Zugriff wird wieder jeweils eine Liste aller Namenlabels und eine Liste aller Punktelabels der Platzierungen zu einer Liste zusammengefasst um diese in einer Schleife beschreiben zu können.

```
def aktualisieren(self): (Z.60-72)
```

Um die Veränderungen in der .txt Datei sichtbar zu machen, muss das Label beim Drücken des “Bestenliste” buttons aktualisiert werden. Hierfür wird die Bestenliste.txt aufgerufen und ausgelesen, die Inhalte der Zeilen, werden gesplittet, um Namen und Punkte separat verwenden zu können (Z.66-69) und danach werden sie auf die jeweiligen Labels im Grid Layout geschrieben (Z.70-72).

Um die Hilfe/Regeln Button eine Funktion zu geben, wurde eine Klasse

```
class help(Ui_Form): (Z.46-49)
```

erstellt. In dieser wird nur das Window initialisiert, in welchem das Hilfe-Fenster angezeigt wird, welches wir im Designer erstellt haben. Durch die Klasse `def help(self):` wird das Window dem Benutzer angezeigt. Hierfür werden fast keine Befehle benötigt, da die Arbeit bereits im Designer erledigt wurde.

Fazit:

Wir sind froh, dass wir unser Projekt in Python realisiert haben und uns somit das Hantieren mit Pointern in C++ gespart haben. Es war sehr interessant ein Spiel von 0 auf zu entwerfen und nach und nach zu sehen wie das Programm immer mehr Funktionen bekommt, welche dann auch mit der Oberfläche verknüpft werden können. QT bietet viele Möglichkeiten eine Oberfläche im Baukastenprinzip zusammenzubauen und deren Knöpfe einfach mit Funktionen zu verbinden. Jedoch hatten wir mit QT so unsere Schwierigkeiten. Im Designer war die Anzeige des Objektmanagers sehr unübersichtlich und teilweise waren nicht alle Funktionen verfügbar, die über direkte Befehle möglich wären oder sie sind so versteckt, dass man sie als Neueinsteiger nicht findet. Dies erfordert oft eine unnötig lange Recherche im Internet für ein kleines Problem.

Quellen (Medien und Bilder)

Würfelsound(uncut):

<https://www.youtube.com/watch?v=6l21rx54VSg>

Barsound(uncut):

<https://www.youtube.com/watch?v=ZSrVznkaMEM&t=672s>

Gewinnersound(uncut):

https://www.youtube.com/watch?v=xP1b_uRx5x4

Kneipenbild:

https://barguide.mixology.eu/wpcontent/uploads/sites/7/2019/03/190304_Daktari_essen-web.jpg

Billiard-Tisch:

https://www.dynamic-billard.de/media/image/07/eb/18/80-860-60-1_191_gelb_gruen.jpg

Kniffelbild:

<https://b-interaktive.com/wp/wp-content/uploads/2020/06/Dice-Clubs-Kniffel.png>

Würfel(rot)

https://images.clipartlogo.com/files/images/41/415345/red-two-recreation-cartoon-dice-free-games-game-dices_p.jpg

Würfelbecher:

https://shop.koenig.de/out/pictures/master/product/1/68485_kp_wuerfelbecher_1000x1000px.png

Würfel(weiß):

<http://www.electronicplanet.ch/catan/wuerfel/wuerfelaugen-1-6.png>

Spielblock:

[https://www.picclickimg.com/00/s/MTYwMFgxMTM1/z/5AwAAOSwXedeuue/\\$_57.PNG?set_id=8800005007](https://www.picclickimg.com/00/s/MTYwMFgxMTM1/z/5AwAAOSwXedeuue/$_57.PNG?set_id=8800005007)