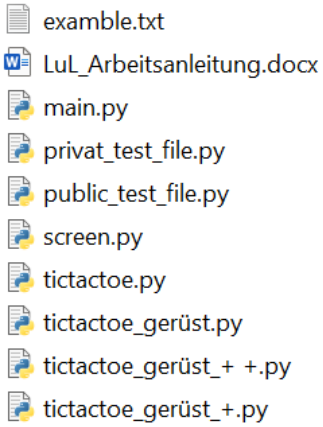
Nachdem der Pseudocode in ein blockbasiertes Framework eingearbeitet wurde, wird dieser im folgenden und letzten Schritt in Python implementiert. Dazu müssen die Schülerinnen und Schüler Laptops bzw. PCs zur Verfügung haben, auf denen zum Beispiel die Entwicklungsumgebung Visual Studio Code installiert ist. Die Unterrichtsmaterialien, die im folgenden Dokument präsentiert werden, sind an die Bedürfnisse der Schüler und Schülerinnen anpassbar.

Dieser Teil ist für die praktische Implementierung eines Alltagbeispiels gedacht. Er ersetzt nicht den Unterricht der theoretischen Grundlagen. Es wird vorausgesetzt, dass den Schülern und Schülerinnen diese Grundlagen bereits vermittelt wurden und sie mit der benutzten Entwicklungsumgebung bereits erste Erfahrungen gemacht haben.

Bevor den Schülern und Schülerinnen das vorgefertigte Gerüst ausgehändigt wird, ist es wichtig, das Gerüst, die globalen Variablen, die Konstanten und die einzelnen Funktionen zu besprechen. Im Vorfeld muss sich die Lehrperson bereits mit den zur Verfügung gestellten Materialien auseinandergesetzt haben.



Jeder Schüler bekommt einen Ordner, der die Dateien

- examble.txt
- public_test_file.py
- screen.py
- tictactoe_gerüst.py oder tictactoe_gerüst+.py oder tictactoe_gerüst++.py

enthält. Dieser wird in der Entwicklungsumgebung geöffnet und es kann los gehen.

Es müssen nicht alle Funktionen verstanden werden. Wichtige bzw. zu bearbeitende Funktionen sind mit Kommentaren und Todo‘s versehrt.

Es gibt drei Schwierigkeitsgrade tictactoe_gerüst.py, tictactoe_gerüst+.py und tictactoe_gerüst++.py. Die Lehrperson ist dafür verantwortlich, welcher / welche Lernende welches Gerüst bearbeiten soll. In diesen Python-files sind die genauen Arbeitsanweisungen beschrieben. Der Teil mit den zu bearbeitenden Funktionen wird abgegrenzt.

Die Datei screen.py zeigt den Schülern und Schülerinnen, wie Klassen erstellt werden können. Sie ist für den Anfangsunterricht noch nicht relevant, bietet jedoch für zukünftige Einheiten Weiterentwicklungspotential. Damit das Programm funktioniert, muss sie sich im Ordner befinden. Ist die Lehrperson der Meinung, dass für die Schüler und Schülerinnen eine weitere Datei für das Verständnis von Nachteil ist, kann diese Klasse auch in die Gerüst-Datei verschoben werden.

Da mit einigen globalen Variablen gearbeitet wird und diese kompliziert zu verstehen sein können, bietet die examble.txt ein vollständiges Beispiel, wie sich die Variablen im Laufe des Spiels verändern. Die Lehrperson soll zu Beginn gemeinsam mit den Schülern und Schülerinnen dieses Beispiel durcharbeiten.

```
#------------------------------------------------------------- global variables
#shows who is on turn
on_turn = PLAYER

#game is stopped while new box appeares
pause = 0

#computer or player can win or there is a draw. while there is no winner or draw one is in game
winner = IN_GAME

#after someone wins a field cannot be clicked. the game has to restart
game_stopped = 0

#counts how many fields are clicked
cross_count = 0

#after first field is clicked case is either middle, corner or edge. at start case is not defined
case = ND


#lists for the nine fields
#item in blocked_boxes is set to 1 if player or computer clicks field
blocked_boxes = [0, 0, 0, 0, 0, 0, 0, 0, 0]
#item in computer is set to 1 if computer clicks field
computer = [0, 0, 0, 0, 0, 0, 0, 0, 0]
#item in player is set to 1 if player clicks field
player = [0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Damit die Lehrperson nicht alle Programme kontrollieren muss, gibt es automatisierte Testcases für alle zu bearbeitenden Funktionen.

Aus diesem Grund ist es wichtig, dass nur die ToDo's in den Files implementiert werden. Entscheidet sich die Lehrperson, dass weitere Teile des Programmes bearbeitet werden sollen, müssen auch die Testfiles angepasst werden.

Den Schülern und Schülerinnen werden in der Datei public_test_file.py 10 Tests zur Verfügung gestellt. Damit können sie sich selbst überprüfen. Die Lehrperson hat die Möglichkeit mit der Datei private_test_file.py weitere 10 Testfälle zu überprüfen. Somit bekommt jeder Schüler und jede Schülerin ein Feedback für ihre Implementation.

```
class TestComputerDefends(unittest.TestCase):
    @patch('tictactoe.computer_clicks')
    def test_computer_defends_left_coloum_and_clicks_UL(self, mock_helper):
        tictactoe.player = [1,0,0,1,0,0,0,0,0]
        tictactoe.blocked_boxes = [1,0,0,1,0,0,0,0,0]

        tictactoe.computer_defends()

        mock_helper.assert_called_once_with(tictactoe.UL)
```

Weder Lehrer und Lehrerinnen noch Schüler und Schülerinnen müssen die Tests verstehen. Wird die Testdatei ausgeführt, erscheint das Ergebnis.

Damit die Tests funktionieren muss die Gerüstdatei in tictactoe.py umbenannt werden.

Des Weiteren hat die Lehrperson mit der Datei tictactoe.py eine Musterlösung bereitgestellt. Die main.py Datei enthält Erweiterungen für das Spiel. Die Lehrperson kann somit nach eigenem Ermessen das Unterrichtsmaterial erweitern und muss dazu nur einzelne Funktionen kopieren und anpassen.

Beispiel einer zu bearbeitenden Funktion:

Schwierigkeitsgrad 1:

```
#------------------------------------------------------------- not completed funcitons
#resets the whole game
def reset():
    global blocked_boxes, player, computer, winner, game_stopped, on_turn, cross_count, case

    # TODO: Weise der Variable 'blocked_boxes' eine Liste mit 9 Nullen zu.
    #blocked_boxes = [0, 0, 0, 0, 0, 0,
    # TODO: Weise den Variablen 'computer' und 'player' jeweils eine Liste mit 9 Nullen zu.
    #computer = [0, 0, 0, 0, 0,
    #player =
    # TODO: Setze 'on_turn' auf den Spieler, der das Spiel beginnt (PLAYER).
    #on_turn =
    # TODO: Setze 'cross_count' auf 0.
    #cross_count =
    # TODO: Setze 'case' auf den Anfangszustand (ND).
    #case =
    # TODO: Setze 'winner' auf den Wert, der anzeigt, dass das Spiel noch läuft (z.B. IN_GAME).
    #winner =
    # TODO: Setze 'game_stopped' auf 0, um anzuzeigen, dass das Spiel nicht gestoppt ist.
    #game_stopped =

    draw_boxes()
    draw_on_turn()
```

Schwierigkeitsgrad 2:

```
#------------------------------------------------------------- not completed funcitons
#resets the whole game
def reset():
    global blocked_boxes, player, computer, winner, game_stopped, on_turn, cross_count, case

    # TODO: Sieh dir an, wie die globalen Variablen vorweg definiert wurden. Nach einem Reset sollen sie genau die gleichen Werte besitzen.
    #       Resette die globalen Variablen blocked_boxes, player, computer, winner, game_stopped, on_turn, cross_count und case
    # blocked_boxes =
    # computer =
    # player =
    # on_turn =
    # cross_count =
    # case =
    # winner =
    # game_stopped =

    draw_boxes()
    draw_on_turn()
```

Schwierigkeitsgrad 3:

```
#------------------------------------------------------------- not completed funcitons
#resets the whole game
def reset():
    global blocked_boxes, player, computer, winner, game_stopped, on_turn, cross_count, case

    # TODO: Sieh dir an, wie die globalen Variablen vorweg definiert wurden. Nach einem Reset sollen sie genau die gleichen Werte besitzen.
    #       Resette die globalen Variablen blocked_boxes, player, computer, winner, game_stopped, on_turn, cross_count und case


    draw_boxes()
    draw_on_turn()
```