Memory słowne

Sprawozdanie z projektu w ramach kursu: Narzędzia Pracy Grupowej

Członkowie zespołu:
- Jakub Kołton
- Krzysztof Grund
- Stanisław Butryn

Link do repozytorium z projektem: https://github.com/krzgr/Memory-slowne
Link do tablicy Kanban projektu: https://github.com/krzgr/Memory-slowne/projects/1

1. Krótki opis organizacji pracy

Zdecydowaliśmy się na realizację tego projektu z dwóch powodów. Pierwszym z nich była chęć stworzenia programu o wysokiej interaktywności z użytkownikiem. Gra jest tutaj oczywistym wyborem z racji na wysoką ilość opcji do wybrania przed grą jak i w trakcie niej. Drugim była złożoność tematu od strony programistycznej. By zrealizować założenia projektowe w stu procentach musieliśmy stworzyć od podstaw funkcje zarządzające bazą słów, bazą graczy, wyświetlaniem elementów na ekran oraz przeprowadzające liczenie czasu rzeczywistego.

Jako język programowania wybraliśmy Python. Jeżeli chodzi o bibliotekę GUI, wybór padł na PyQt5. Następnie zostało utworzone repozytorium na platformie Github. Zaraz po utworzeniu i omówieniu struktury plików w projekcie, utworzone zostały dodatkowe gałęzie (devGUI oraz devGameLogic) na których rozwijane było oprogramowanie w zakresie interfejsu użytkownika oraz implementacji logiki gry. Na gałęzi master w trakcie trwania projektu stopniowo dodawane były kolejne wyrazy do naszej bazy słów. Platforma Github została przez nas wybrana, ponieważ jest ona darmowa (co najważniejsze) oraz dawała ona możliwość łatwego śledzenia zmian, nadzoru nad projektem oraz zgłaszania ewentualnych błędów i sugestii ulepszenia oprogramowania. W naszym projekcie miejscem zgłaszania błędów była m.in. zakładka Issues. Sam projekt był rozłożony w czasie, choć jego największy rozwój przypadł na ostatnie tygodnie kwietnia.

2. Dla kolejnych sprintów (newsletter) – role (SM), cel, kamień milowy, zadania, przydział, czasy realizacji

Naszym zadaniem projektowym było napisanie gry Memory słowne. Rozgrywka miała odbywać się przeciwko komputerowi z możliwością zmiany poziomu trudności. Dodatkowo w samej grze miała istnieć możliwość sprawdzenia statystyk (wygranych i błędów). Jako dodatkową funkcjonalność naszej aplikacji dodaliśmy możliwość zmiany nazwy gracza oraz zapis statystyk na dysku. Sposób podziału zadań był bardzo naturalny i sprowadzał się do pytania co kto umie. Tak więc osobą głównym programistą był Krzysztof

Grund, który zajmował się projektowaniem interfejsu użytkownika. Jakub Kołton zajmował się logiką gry, czyli m.in. kwestiami zapisu, czy odczytu danych z plików, obsługą błędów, itp. Natomiast Stanisław Butryn sprawował pieczę nad bazą słów, a w późniejszym etapie nad dodaniem możliwości gry na czas.

3. Szczegółowy BACKLOG produktu (PBL) - ewolucja, priorytety, porządkowanie

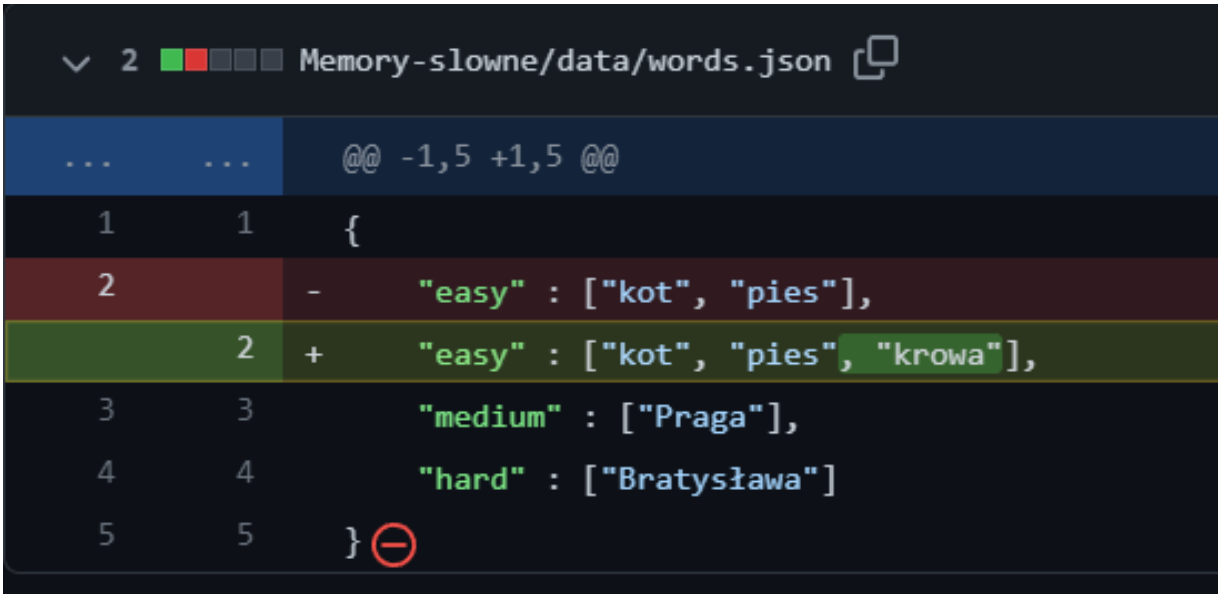
Od początku naszym priorytetem było poprawne działanie aplikacji. Jednym z pierwszych elementów programu, jaki został dodany, było działające okienko aplikacji. W międzyczasie implementowana zostawała logika gry, która w tym etapie skupiała się nad poprawnym odczytem słów z pliku. Dodatkowo niezależnie od reszty dodawane były kolejne wyrazy, z rozróżnieniem na kategorię trudności, do bazy słów. Następnym etapem było połączenie logiki gry z interfejsem użytkownika oraz sprawdzenie poprawności działania. Po wielu testach oraz poprawkach można było zagrać w grę, ale w trybie pozbawionym limitu czasowego. Na samym końcu dodany został tryb gry na czas, którym to zakończyliśmy nasze zmagania z projektem.

4. Wykonane zadania przez członków zespołu:

1. Baza haseł z poziomami łatwy, średni, trudny co najmniej 25 w każdym. - Stanisław Butryn (15.04.2022)
2. Tryb gry na czas i ilość - Stanisław Butryn, Krzysztof Grund (28.04.2022)
3. Implementacja zasad gry - Jakub Kołton (2.05.2022)
4. Statystyka wygranych - Jakub Kołton (15.04.2022)
5. Statystyka błędów - Jakub Kołton (15.04.2022)
6. Jedna zaproponowana przez grupę(opcja zapamiętywania graczy) - Jakub Kołton (22.04.2022)
7. GUI - Krzysztof Grund (28.04.2022)

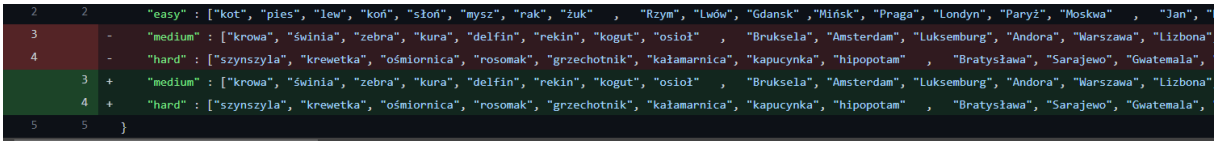
5. Zdjęcia z postępu pracy nad projektem

Ten screenshot prezentuje pierwotną wersję pliku words.json, który pełni rolę bazy słów dostępnych do wylosowania w grze. W tym konkretnym commicie dodano wyłącznie jedno słowo, powodem było testowanie zachowania się plików typu .json w programach w języku python.



Zawartość pliku była systematycznie poszerzana o nowe słowa.

Aż do swojej finalnej wersji umożliwiającej grę na najwyższym poziomie trudności pod względem ilości słów do zapamiętania. Powtarzanie się słów w kolejnych rozgrywkach jest nieuniknione, jednak ich ilość jest na tyle duża, by



Jedna z pierwszych wersji klasy logiki. Początkowe implementacje skupiały się głównie na utworzeniu poprawnego szkieletu klasy realizującej funkcje silnika gry.

```
1  1   import json
   2 + import random
2  3
3  4   class WordMemoryGame:
4  5       def __init__(self):
5  6           pass
6  7
7  8       def loadWordsFromFile(self):
8    -         return None
9    -
10   -     def loadPlayersDataFromFile(self):
   9 +         words_file = open("words.json", "r")
   10 +       self.words_dict = json.loads(words_file.read())    #wszystkie słowa wpisane z pliku
   11 +       self.words_list = self.words_dict[self.difficulty] #słowa wybrane zależnie od poziomnu trudności
11 12          return None
12 13
13   -     def setGameType(self):
   14 +     def setNewPlayer(self, player_nickname):
   15 +         self.players_dict[player_nickname] = [0, 0, 0, 0]
   16 +         self.savePlayerStatistics()
14 17          return None
15   -
16   -     def setDifficulty(self):
   18 +
   19 +     def loadPlayersDataFromFile(self, player_nickname):
   20 +         players_file = open("players.json", "r")
   21 +         self.players_dict = json.loads(players_file.read())
   22 +         if(player_nickname in self.players_dict):
   23 +             self.current_player = self.players_dict[player_nickname]
   24 +         else:
   25 +             self.setNewPlayer(player_nickname) # ta funkcja bedzie dopisywac nowego gracza, doda sie domyslne staty
```

Fragment kodu przedstawiający pierwotną wersję okna gry. Dodany został np gridLayout przycisków zarówno z menu jak i tych używanych w trakcie gry.

```
   @@ -5,11 +5,14 @@ class WordMemoryGUI(PyQt5.QtWidgets.QMainWindow):
5  5       def __init__(self):
6  6           super().__init__()
7  7
8    -         self.setGeometry(0, 0, 720, 480)
   8 +         self.setGeometry(0, 0, 1080, 720)
9  9           self.move(self.screen().geometry().center() - self.frameGeometry().center())
10 10          self.setWindowTitle("Memory słowne")
   11 +
   12 +       #self.setFixedSize(1080, 720)
11 13
12 14          self._addMenuBar()
   15 +       self._renderGame()
13 16
14 17      def _addMenuBar(self):
15 18          menubar = self.menuBar()
   @@ -26,8 +29,68 @@ def _addMenuBar(self):
26 29          self.statsMenu.addAction(self.allStatsAction)
27 30          menubar.addMenu(self.statsMenu)
28 31
29   -         self.helpMenu = PyQt5.QtWidgets.QMenu("&Pomoc", self)
30   -         menubar.addMenu(self.helpMenu)
   32 +         menubar.addAction(self.helpAction)
   33 +
   34 +
   35 +     def _renderGame(self):
   36 +         self.mainWidget = PyQt5.QtWidgets.QWidget()
```

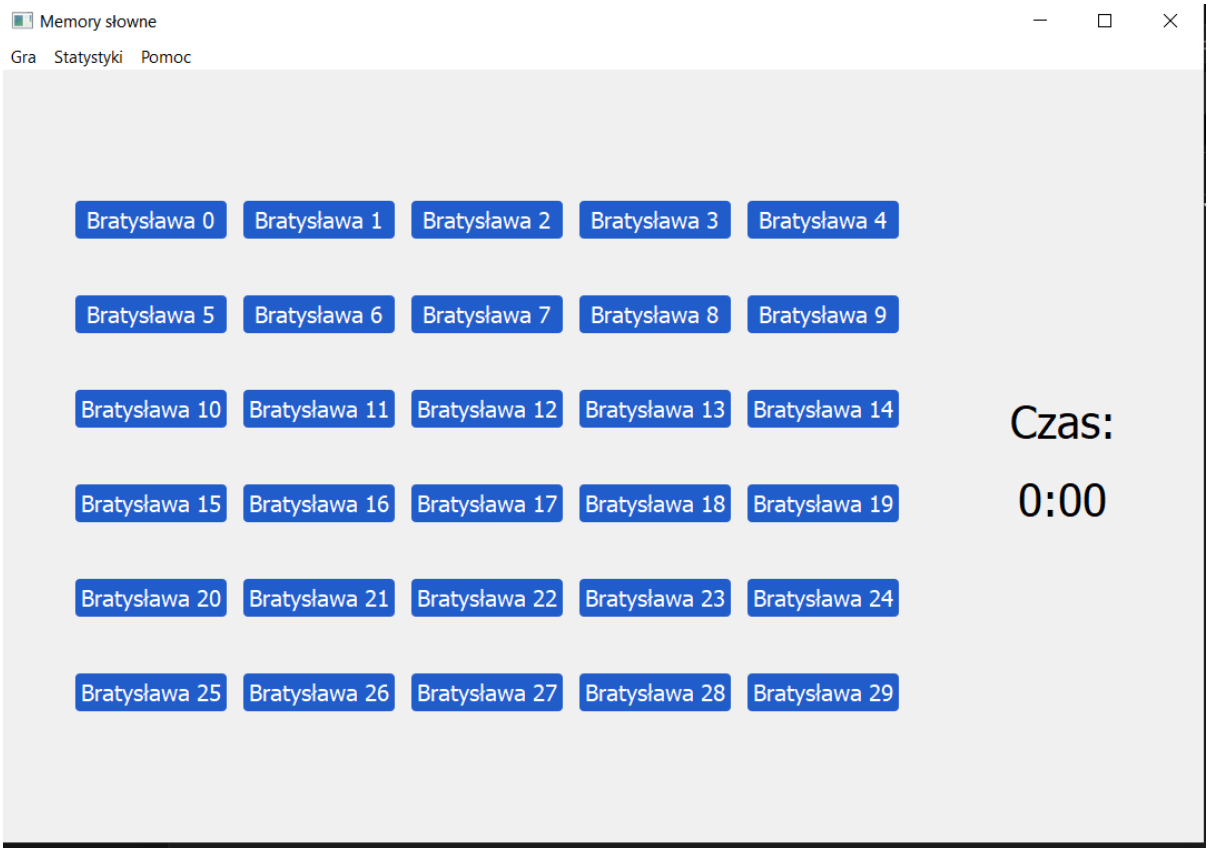
Przykład commita służącego w celu poprawy komunikacji między programistami. Wykorzystanie listy jest może mniej czytelne, ale inny złożony typ danych np. kolejny zagnieżdżony słownik zmniejszyłby czytelność samego kodu.

```
14  14      def setNewPlayer(self, player_nickname):
15  15          self.players_dict[player_nickname] = [0, 0, 0, 0]
    16  +       #[ilosc wygranych, ilosc gier, ilosc poprawnych odpowiedzi, ilosc wszystkich odpowiedzi]
16  17          self.savePlayerStatistics()
17  18          return None
```

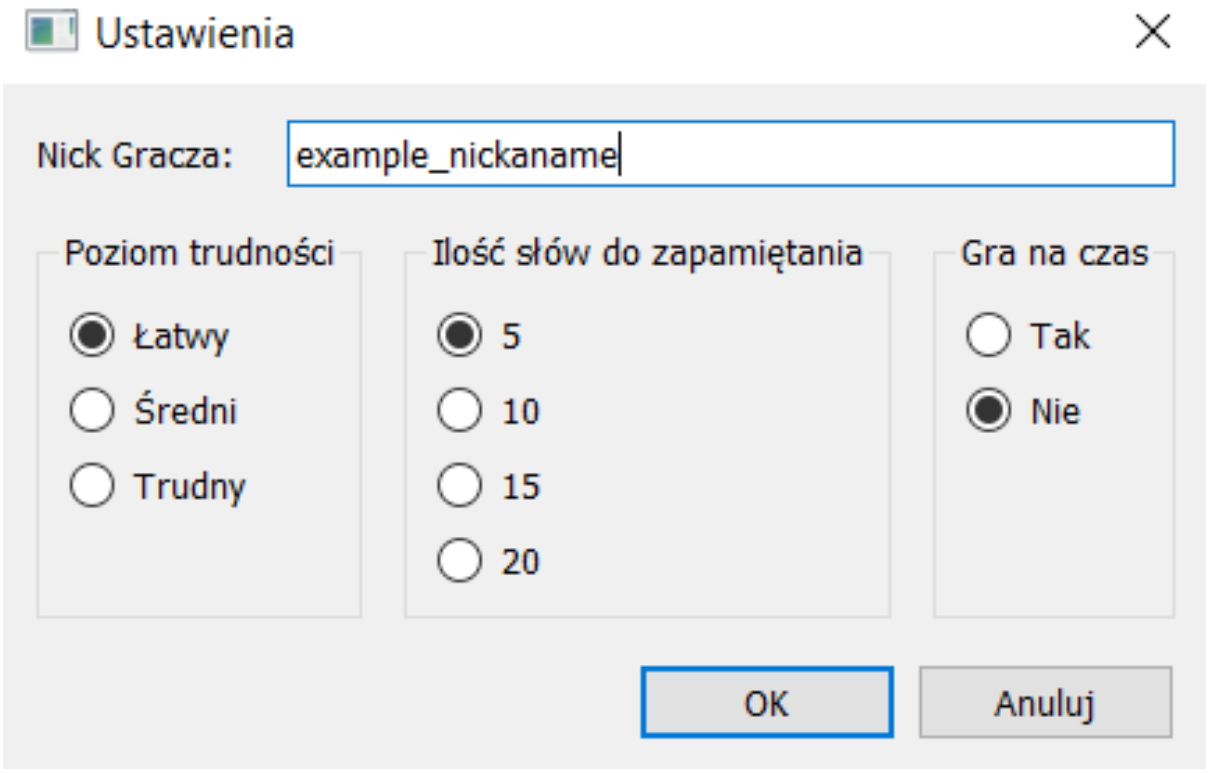
Równocześnie miał miejsce rozwój klasy WordMemoryGUI, w tym poprawianie struktury oraz czytelności kodu.

```
83
    -       spacerItem7 = PyQt5.QtWidgets.QSpacerItem(15, 20, PyQt5.QtWidgets.QSizePolicy.Minimum, PyQt5.QtWidgets.QSizePolicy.Expanding)
    -       self.mainGridLayout.addItem(spacerItem7, 1, 4)
84  +   def _addMenuBar(self):
85  +       menubar = self.menuBar()
86  +       self._createMenuBarActions()
87  +
88  +       self.gameMenu = PyQt5.QtWidgets.QMenu("&Gra", self)
89  +       self.gameMenu.addAction(self.newGameAction)
90  +       self.gameMenu.addAction(self.settingsAction)
91  +       self.gameMenu.addSeparator()
92  +       self.gameMenu.addAction(self.exitAction)
93  +       menubar.addMenu(self.gameMenu)
94  +
95  +       self.statsMenu = PyQt5.QtWidgets.QMenu("&Statystyki", self)
96  +       self.statsMenu.addAction(self.myStatsAction)
97  +       self.statsMenu.addAction(self.allStatsAction)
98  +       menubar.addMenu(self.statsMenu)
99
100 +       menubar.addAction(self.helpAction)
```

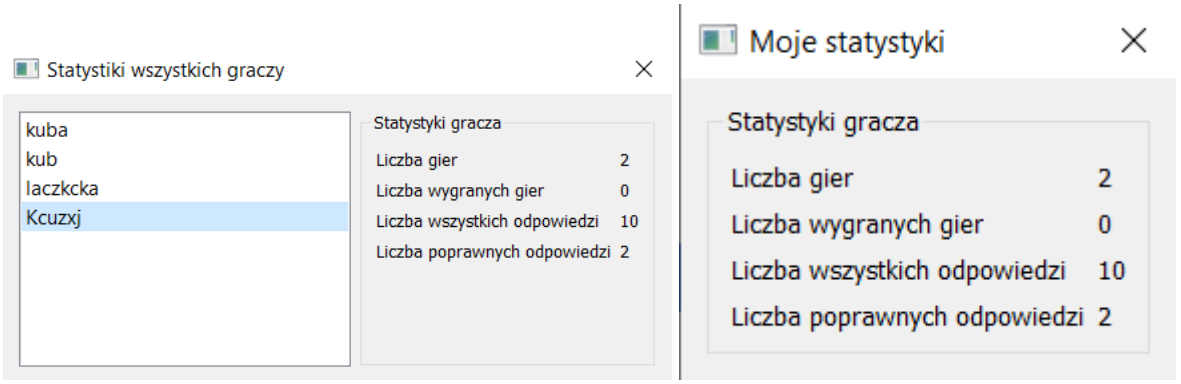
Poniższy screenshot przedstawia jeden z pierwszych commitów związanych z klasą GUI. Był to raczej koncept tego jak docelowo ma wyglądać nasza gra, aniżeli jej ostateczny wygląd. Jedyne co nie uległo zmianie to szata graficzna. Uznaliśmy, iż jest ona wystarczająco czytelna, a zarazem nie rażąca w oczy, aby wykorzystać ją w ostatecznej wersji projektu.



W około połowie czasu trwania projektu dodano okno z ustawieniami. Pozwala ono wybrać tryb gry, ilość słów, a także zaznaczyć opcję gry na czas. Następnie dodano możliwość wyświetlania statystyk gracza lub wszystkich graczy zapisanych w bazie danych. Poniżej fragment kodu oraz wygląd zaimplementowanych funkcjonalności.

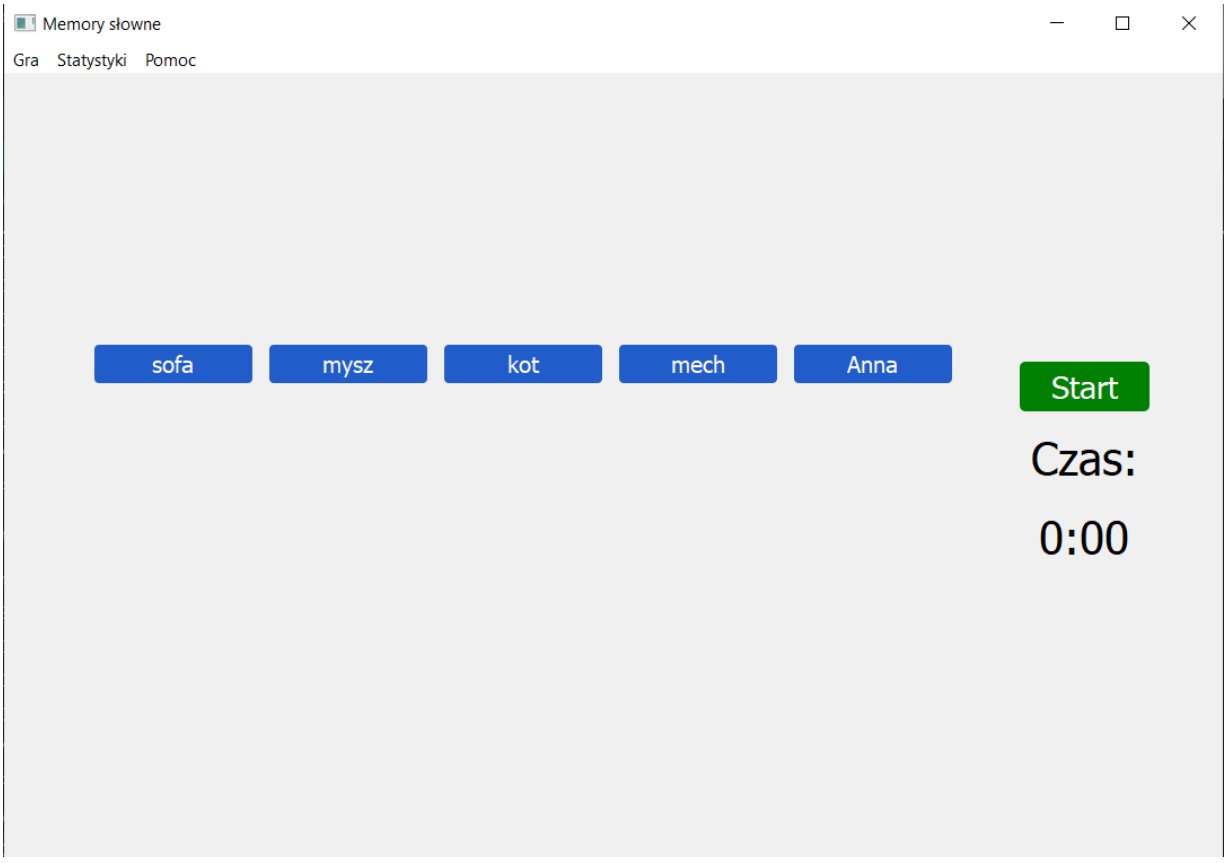


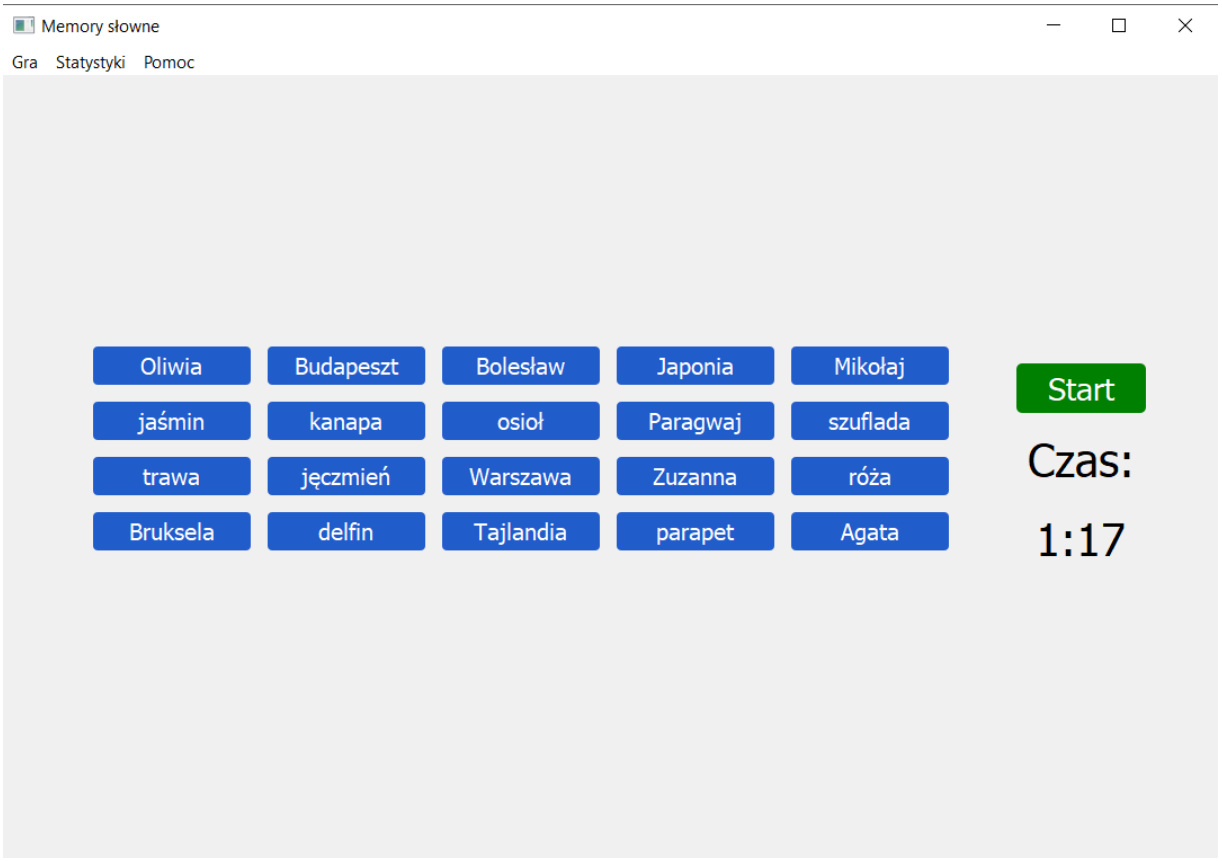

```
@@ -0,0 +1,27 @@
+ import PyQt5.QtWidgets
+ import PyQt5.QtCore
+
+ class StatisticsDialog(PyQt5.QtWidgets.QDialog):
+     def __init__(self):
+         super().__init__()
+
+         self._initUI()
+
+     def _initUI(self):
+         self.setWindowFlags(self.windowFlags() & ~PyQt5.QtCore.Qt.WindowContextHelpButtonHint)
+         self.setWindowTitle("Statystiki wszystkich graczy")
+
+         self.mainHorizontalLayout = PyQt5.QtWidgets.QHBoxLayout(self)
+
+         self.playersScrollArea = PyQt5.QtWidgets.QScrollArea()
+         self.scrollAreaGroupBox = PyQt5.QtWidgets.QGroupBox()
+         self.playersScrollArea.setWidget(self.scrollAreaGroupBox)
+         self.playersScrollArea.setWidgetResizable(True)
+         self.mainHorizontalLayout.addWidget(self.playersScrollArea)
+
+         self.playerStatisticsGroupBox = PyQt5.QtWidgets.QGroupBox("Statystyki gracza")
+         self.mainHorizontalLayout.addWidget(self.playerStatisticsGroupBox)
+
+         #test
+         self.scrollAreaLayout = PyQt5.QtWidgets.QVBoxLayout()
+         self.scrollAreaGroupBox.setLayout(self.scrollAreaLayout)
```



Poniższe screenshoty prezentują ostateczną wersję gry. Zaimplementowane zostały takie opcje jak:

- poziom trudności
- ilość słów do zapamiętania
- tryb gry na czas





6. Wnioski

Wnioski, które wyciągnęliśmy z przeprowadzenia tego projektu można wyszczególnić względem kilku różnych grup:

Praca w grupie

Bez podziału zadań oraz pracy zespołowej nikomu z nas nie udałoby się doprowadzić zadania do końca. Dobrym pomysłem było rozdzielenie konkretnych zadań już na początkowym etapie, zanim przeszliśmy do implementacji samego programu. Dzięki licznym spotkaniom ustaliliśmy między sobą w jakich zadaniach każdy z nas czuje się dobrze oraz jakie z nich chciałby wykonać.

Narzędzia wspomagające pracę

Dzięki poznanym na zajęciach narzędziom przeznaczonym do pracy nad projektami grupowymi nasz projekt mógł być na bieżąco aktualizowany. Nie potrzebne było wysyłanie sobie wzajemnie kodu w plikach tekstowych czy żaden inny sposób. Aktualna wersja programu zawsze była dostępna na serwerze, gotowa do zpullowania na lokalne repozytorium. Bardzo dobrym zabiegiem okazało się utworzenie dwóch różnych branchy specjalnie pod

oddzielne zadania, tj. utworzenie interfejsu graficznego oraz osobno silnika logicznego gry. Kolejnym nieocenionym atutem GitHub`a jest możliwość dodawania issues. Dzięki tej możliwości komunikacja między ludźmi pracującymi o różnych porach w różnych miejscach nie stanowi problemu. Wykorzystywaliśmy ją głównie w formie listy rzeczy do zrobienia lub wymagających poprawy. Kolejną przydatną funkcją okazała się tablica Kanban. Korzystaliśmy z niej w podobnej formie co z kalendarza Google. Pomagała nam ustalać kolejność wykonywanych zadań oraz dodawać kolejne pozostałe do zrobienia.

Internet

Bez tej kopalni wiedzy nie bylibyśmy w stanie ukończyć projektu z powodzeniem. Dla każdego z nas biblioteka PyQt5 była nowością oraz wyzwaniem. W internecie można znaleźć niezliczoną ilość wszelkich tutoriali, poradników, pomocy czy nawet gotowców w dziedzinie programowania. Bez niego nikt z nas nie byłby w stanie znaleźć wystarczającej ilości informacji dotyczącej potrzebnej nam biblioteki, jej funkcjonalności oraz możliwości, które oferuje ona użytkownikowi.

Wnioski ogólne

Praca grupowa jest obosieczną bronią. Jeśli zespół nie zostanie wyposażony w odpowiednie narzędzia do organizacji, podziału zadań, współdzielenia aktualnego stanu projektu czy nawet dróg komunikacji najprawdopodobniej nie osiągnie on sukcesu. Zarządzanie projektem jest zadaniem trudnym, nieraz cięższym niż samo powierzone zadanie. Jeśli jednak zadba się o wszystkie z powyższych można osiągnąć efekt nieporównywalny do tego będącego owocem pracy samodzielnej.