

TARTALOM JEGYZÉK

ELŐKÉSZÜLETEK: 2-4.oldal

PROGRAMOZÁS:

ALAPOK: 5. oldal

SPLASH SCREEN: 6. oldal

PIRACY SCREEN 7-9

MAIN WINDOW ALAPJAI: 10-12. oldal

MAIN WINDOW FUNKCIÓK: 13-20.oldal

FÜGGVÉNYEK:

SÖTÉT ÉS VILÁGOS MÓD: 21.oldal

FELHŐ ANIMÁCIÓK: 21-23.oldal

HANG ÉS ZENE: 23-24.oldal

HELP ÉS EASTER EGG: 24.oldal

KARAKTEREK: 24-27.oldal

START, CLOSE EVENT ÉS EXIT: 27-29.oldal

TEXT BOX: 29-30.oldal

SZÖVEG FRISSÍTÉS (LETTER BY LETTER ANIMÁCIÓ): 30-31.oldal

NYELVEK, UI FRISSÍTÉSE: 31-32.oldal

TALÁLATOK, IDŐ ÉS GOMBOK FUNKCIONÁLTATÁSA: 32-38.oldal

ÉGHAJLAT KEZELÉSE: 38-40.oldal

KEDVENCEK KÖZÖTTI VÁLTÁS: 40-41.oldal

SPLASH SCREEN, MAIN WINDOW FUTTATÁSA: 42.oldal

CLOUD SCY

Csapat név: **AmeKumo**

Csapat tagok: **Bayerle Bence, Lázár János**

Projekt név: Cloud SCY (Cloud Scan Control Yard)



Jelmagyarázat:

(...) – Az állandó ismételgetés elkerülése érdekében, gyakran stílus formázásra utal.

... - A parancsok ismétlése, valamint hasonló sorok leírásának rövidítésére.

Vastag – Kód részre utal.

Dőlt – A függvényekre utal.

Aláhúz – valami fontosabbat vagy kérdést jelöl.

Nagy – Címsorra utal.

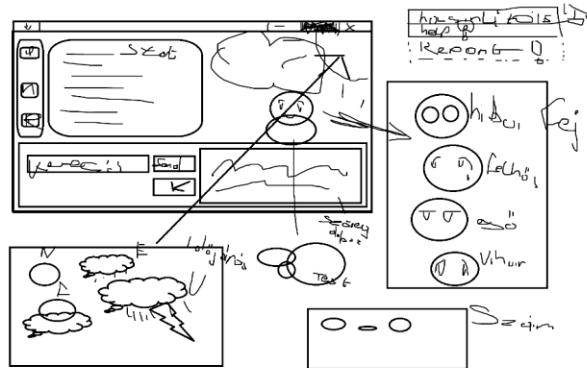
Előkészületek:

Egy vázlatfüzet segítségével felvázoltuk a lehetséges ötleteket, melyek belekerülhetnek az alkalmazásba a szoftver fejlesztése alatt, miket sokszor át gondoltunk és módosítottunk. Ezután elkészült egy táblázat melyben megszabtuk egymás munkáit és összegeztünk, hogy mennyi feladatot kell betölteni.

A létre hozott alappal elkezdtünk dolgozni, én a grafikai felülettel, Bence a programozással. Először Jani elkészítette az alkalmazás ikonját, mely tükrözi az alkalmazás jellemzőjét, felismerhető és egyben egyedi. Eme digitális rajzokhoz több alkalmazás is használatba vettünk, mint például az Ibis Paint X számítógépes verzióját, mely egy egyszerű és letisztult felülettel rendelkezik, viszont az ingyenes verziót naponta 1 óráig lehetett használni, a PaintTool-sai-t, mely szintén egy ingyenes digitális rajz szoftver, nem olyan letisztult és bő, mint a Ibis Paint, viszont nem vagy időkorláthoz kötve.

Az ikon terve egy egyszerű felhő, melyből egy villám cikáz ki és hullajt pár gyorsan mozgó eső cseppet. A felhő csillog, implikálva arra, hogy vizes, a háttér sötét, kékes szürke lett, hogy ne legyen túl egyszerű, ezen felül a felhőt a határain kívülre rajzolva, árnyékot vetve maga alá, azt az érzést kelti mintha a képen kívül lenne. Elkészült az első digitális tervrajz, mely kevésbé részletes, inkább vázlatszerűbb mint a papíron írt, viszont bőven elég ahhoz, hogy tudjuk mit kell csinálnunk. A szoftver grafikai felületéhez egy python bővítményt használtunk, a PyQt-t, mely segítségével elérhettük az általunk tervezett kinézetet.

A képen látható az előre megtervezett kinézet alapja, mely 5 dobozból állt volna, 1 kabalából, jelenlegi időjárás a jobb felső sarokban, valamint 5 darab nyomható gombból. Figyeltünk már az elején arra, hogy valószínűleg a teljes képernyőben elcsúszhatnak a grafikus objektumok, ezért ezt megakadályozva előre figyelmeztettem a csapat társam, hogy a nagyítás funkciót kapcsolja ki mielőst tudja, elkerülve ezzel a kellemetlenségeket.



Ahhoz, hogy futtatni tudjuk a programot, le kellett telepíteni a PyQt6-ot a Visual Studio Code termináljában a „pip3 install PyQt6” segítségével. Itt beleütközöttünk egy olyan problémába, hogy bármikor mikor beakartam importálni a PyQt-t, a rendszer nem találta meg a modult, ezért nem futtatta le a kódot. Egy kis kerestgélés után, rájöttünk, hogy nem a jó verzióban vagyunk, hanem egy „venw” verzióban, mely egy virtuális környezet olyan projektekhez, melyeket nem akarunk összekeverni másokkal. A venw mappát aztán kitöröltük, és már működött is a kód.

Ezután Jani visszament, és megrajzolta a hátteret a szoftvernek, mivel ez egy időjárás előrejelzés, ezért nem engedhettük meg, hogy egy üres tér legyen a helyszín, hanem mondjuk a fellegek. Egyből kettőt is rajzolt, egyet a nappal-hoz, a másikat az éjszakához. Terveink szerint úgy volt, hogy 06:00-tól 16:00-ig lett volna reggel, de az egyenlőtlenség miatt 18-ra emeltük, így 12 óráig van nappal, és 12 óráig este. Ezek után elkészítette az alap időjárási viszonyok

bemutatására szükséges éghajlati ábrákat a PaintTool-sai-ban, kezdve a nappal, a holddal, majd ezek felhővel takart verziójával. Utána persze beborult felhőt, és viharfelhőt készített. Ugyan akkor Bence elkezdte tanulni a PyQt6-ot, mivel egyikünk sem értett hozzá az elején. A különböfélé folyamatokat, kódokat a későbbiekbén részletezem.



Gondolkodva, hogy hogy nézzen ki a kész menü, elkészítette Jani a végleges tervrajzot, melyből az elején még nem tudtuk, hogy mennyi sikerül majd belőle. A karakter még kezdetleges fázisban volt, nem őt tervezte, mint kabalaként használni, hanem inkább egy lány kabalát, viszont a végső döntés rá fordult, mivel elég sok

projektjében szerepelt már, és mindig sikert hozott. A kabala neve Shinko Borakku, jelentése hit, valamint fekete. A fekete a japán burakku szóból ered melyben kicseréltem az első u-t, ezzel egy lágyabb, kevésbé durvább hangzást adva neki. A jobb sarokban egy dialógus dobozt tervezünk, ahol a kabala interaktálhat a használóval verbálisan, melyeket előre beprogramoztunk volna, továbbá a kabala arckifejezése az éghajlati viszonyok alapján változott volna. A bal oldali dobozban lett volna a kereső, ahol belehet írni egy várost, majd fellistázta volna a magyar találatokat, persze külföldi városokat is megtalál, viszont túl sok idő lett volna minden várost a rendszerbe átvinni, így csak a magyar települések mellett döntöttünk. Egy utógondolat volt, hogy egy kedvencek listát is csinálunk, így a felhasználó, ha egyszer kedvelt egy várost, pár kattintással megtudhatja a várható időjárást. A keresés után vagy egy hibaüzenetet kap, ha nem talált várost vagy nincs internet, vagy a képernyő közepén megjelenik a várható időjárás, 5 órára előre nézve. Egy nyíllal tovább lephetünk, és megnézhetjük a részleteket, pl. a további órákban milyen időjárás várható, páratartalom, szélirány (Legalábbis ez volt a terv, viszont ezek a funkciók csak a fizetős API verzióban elérhetők). Ezeket az adatokat ki lehet majd listázni egy külön ablakba, melyet szabadon lehet mozgatni az asztalon. A bal oldalon további gombok voltak egy oldal sávban, az egyik az összehasonlítás, a másik a jelentés, ha valami hiba van, és az utolsó a súgó, ha valamit nem értenének a használók.

Azután megrajzolta Jani a karakterek sprite-jait. A jó időjáráshoz egy boldog, a borúsra egy szomorú, a viharosra egy rémült, a hibákra egy meglepett, valamint egy bónusz arckifejezést, mely akkor jelenik meg ha nem a tiszta szoftvert használnád. Ezután kibővítette még 2 karakterrel, hogy a felhasználó kedve szerint válasszon közölik. Az egyik Tomoko Usagi a másik Awai Hiyori, melyeknek szintén megrajzolt minden arckifejezést.

PROGRAMOZÁS:

Könyvtárak importálása:

A kód elején több Python könyvtárat importáltunk, amelyek különböző funkciókat biztosítanak a program számára:

- **json** – JSON fájlok kezelésére. Egy külső raktár melybe az adatokat menti, pl: beállítások adatait, vagy a legutolsó karaktert.
- **datetime** – Időkezeléshez.
- **requests** – Külső API-khoz való kapcsolódáshoz.
- **PyQt6 modulok** – Grafikus felhasználói felület (GUI) létrehozásához.
 - **PyQt6.QtMultimedia** – A zenék kezeléséhez.
 - **PyQt6.QtCore** – Időzítők, dobozok és animációk kezeléséhez
 - **PyQt6.QtGui** – Betűk, képek, ikon, akciók, különböfle útvonalak kezelésére.
 - **PyQt6.QtWidgets** – Applikáció, Lista, keresősáv, szövegdoboz, MainWindow, gombok, menü, menübár, státuszbár és QsplashScreen inicializálásához.
 - **PyQt6.uic** – Saját UI (kezelőfelület) betöltése.
- **pyowm** – OpenWeatherMap API használatához.
- **time, threading, random** – Időkezeléshez, szálkezeléshez és véletlenszám-generáláshoz.
- **os, subprocess** – Rendszerfüggő műveletek végrehajtásához.

Alapvető változók és API inicializálás:

- **API_KEY** – Az OpenWeatherMap API kulcsát tárolja (Ezt a felhasználónak kell megadnia, amelyet érdemes titkosítani.).
- **owm és mgr** – Az OWM (OpenWeatherMap) objektum létrehozása az időjárási adatok lekérését, az mgr a memória kezelés szolgálja.

Az alkalmazás inicializálása:

- **app = QApplication(sys.argv)** – Egy Qt alkalmazás példányának létrehozása.
- **app.setWindowIcon(QIcon("Cloud.png"))** – Az alkalmazás ikonjának beállítása.

Töltőképernyő (SPLASH SCREEN) osztály létrehozása:

Konstruktor (`__init__` metódus):

- `super().__init__()` – Az ősosztály (QSplashScreen) inicializálása.
 - `self.setAttribute()` – Translucent háttér beállítása.
 - `self.setWindowFlags()` – Keretnélküli ablak beállítása.
 - `path = (QPainterPath)` – Az ablakot a PainterPath-el formázzuk.
 - `path.addRoundedRect()` – Lekerekíti az ablakot a szélesség és magasság alapján.
 - **Region** – Kitölti a splash területét.
 - `self.hatter = QLabel(self)` – Megadjuk a modulját ahogy viselkedjen és hogy magára vonatkozzon, egy self értéket.
 - `self.hatter.setGeometry(QRect(x,y,x,y))` – Megadjuk, hogy mettől meddig érjen a területe, és a többi az asztal szélessége és magassága alapján számítsa ki.
 - `self.hatter.setPixmap(QPixmap(„útvonal”))` – Beállítjuk aPixmap modult mellyel lehet képet beilleszteni, és megadjuk a kép útvonalát.
 - `self.hatter.setScaledContents(True/False)` – Engedélyezzük, hogy a tartalmat nyújtsa, ha nincs bekapcsolva a tartalmat meg sem jeleníti.
 - `self.hatter.setAlignment()` – A tartalom középre helyezése.
 - `self.hatter.stackUnder(self.progressBar)` – A progressBar alá helyezése.
-
- `screen.setGeometry` – az ablak középre helyezése a képernyő felbontásának megfelelően.

Progress Függvény:

- Ellenőrzi a betöltött adatokat és az alapján rendeli az értéket a taskbarhoz.

Kalózkodás ablak (PiracyScreen):

Konstruktor (`__init__` metódus):

- `super().__init__()` – Az ősosztály (QMainWindow) inicializálása.
- `self.Start()` – Üzenetet küld a konzolnak, betölti a konfigurációs szótárt (Json fájl), valamint a kedvencekből rákeres az egyik város adataira.
- `ora` – Lekéri a számítógép pontos óráját.
- `self.aktív_karakter` – Betölti a configból a karaktert a beállítások listából.
- `self.este = QLabel(self)` – Megadjuk a modulját ahogy viselkedjen és hogy magára vonatkozzon, egy self értéket.
- `self.este.setGeometry(QRect(x,y,x,y)` – Megadjuk, hogy mettől meddig érjen a területe. (pl: 0x és 0y-tól a 300x és 300y-tengelyig pixelekben számítva.)
- `self.este.setPixmap(QPixmap(„útvonal”))` – Beállítjuk aPixmap modult mellyel lehet képet beilleszteni, és megadjuk a kép útvonalát.
- `self.este.setScaledContents(True/False)` – Engedélyezzük, hogy a tartalmat nyújtsa, ha nincs bekapcsolva a tartalmat meg sem jeleníti.
- `setWindowIcon(QIcon(„útvonal”))` – Az alkalmazás ikonját beállítja.
- `setWindowTitle(„cím”)` – Az ablak címét adja meg.
- `setFixedSize(x, y)` – Fix méretű (1300x900) ablak létrehozása.
- `self.clouds = []` – Lista a megjelenített felhők tárolására.
- `self.max_clouds = 15` – Legfeljebb 15 felhő jelenhet meg egyszerre.
- `self.startCloudAnimation()` – A felhőanimáció elindítása. (192.oldal)
- **Ez a kódrész különböző dobozokat (QLabel) hoz létre:**
 - `self.Kulso`
 - `self.belso`
 - `self.dialogbox`
 - `self.belsoM`
 - `self.kulsoM`
- `self.Kulso.setGeometry`
- `self.Kulso.setStyleSheet("background-color:rgb(r, g, b)")` – Kiszínezi a dobozt egy megadott színre.

- **self.Kulso.setFrameShape(QLabel.Shape.Box)** – Ad egy külső keretet a doboz köré.
- **self.Kulso.setFrameShadow(QLabel.Shadow.Raised)** – A dobozra egy 3D-s árnyékot rak.
- **self.dialogbox.setLineWidth(szám)** – beállítja a QLabel szélességét.
- **self.dialog = QLabel(self)**
- **self.dialog.setGeometry(QRect(x, y, x, y))**
- **self.dialog.setFont(QFont("Eras Demi ITC", 18)** - Beállítja a szöveg betűtípusát és méretét.
- **self.dialog.setStyleSheet("font-size: px; color: szín ")** - CSS-szerű stílussal a szöveg mérete 30px, színe vörös.
- **self.dialog.setText(„szöveg”)** -Kezdő szöveget állít be a dialógus ablakban.
- **self.dialog.setAlignment(Qt.AlignmentFlag.AlignLeft|Qt.AlignmentFlag.AlignTop)** - A szöveg igazítása bal felső sarokhoz történik.
- **self.dialog.setWordWrap(True)** -A szöveg sortöréssel jelenik meg, hogy ne lógjon ki a dobozból.
- **self.License = QLabel(„szöveg”, self)** – A setText kihagyható, ha a QLabel-be írjuk be a szöveget. Ez megjeleníti a szöveget.
- **self.License.setStyleSheet("font-size: px; color: szín")**
- **self.License.setGeometry(QRect(x, y, x, y))**
- **self.verzio = QLabel(„szöveg”, self)**
- **self.verzio.setStyleSheet("font-size: px; color: szín")**
- **self.verzio.setGeometry(QRect(x, y, x, y))**
- **self.piracy_character = QLabel(self)**
- **self.piracy_character.setPixmap(QPixmap("..."))** – Egy képet rak be a Label helyére.
- **self.piracy_character.show()** – Megjeleníti azt.
- **self.piracy_massage = QLabel(self)**
- ...
- **self.pontosido = QLabel(self)**
- ...
- **self.borakku = QLabel(self)**
- ...

- **self.opacity_effect = QGraphicsOpacityEffect(self.borakku)** – Az opacity_efektusra tűzünk egy metódust, és ezt rátűzzük a borakkra.
- **self.borakku.setGraphicsEffect(self.opacity_effect)** – Az áttetszőséget az opacity_effect-el állíthassuk.
- **self.opacity_effect.setOpacity(0.0)** – Alapértelmezetten áttetsző.
- **self.fade_animation = QPropertyAnimation(self.opacity_effect, b"opacity")** – Ez egy animáció legyen, ami az áttetszőséget módosítja.
- **self.fade_animation.setDuration(100000)** - 100s alatt zajlódjon le.
- **self.fade_animation.setStartValue(0.0)** – A kezdő érték 0 (áttetsző)
- **self.fade_animation.setEndValue(1.0)** – A végső érték 1
- **self.animation_timer = QTimer(self)** - Időzítő
- **self.animation_timer.setSingleShot(True)** - Csak 1x fut le.
- **self.animation_timer.timeout.connect(self.fade_animation.start)** – Ha letelt az időzítő, elindítja az animációt.
- **self.animation_timer.start(10000)** – 10s legyen az időzítő.
- **if not os.path.exists("License"):** - Ha nem létezik a License fájl:
- **self.zene = QMediaPlayer()**
- **self.zeneOutput = QAudioOutput()**
- **self.zene.setAudioOutput(self.zeneOutput)**
- **self.zene.setSource(QUrl.fromLocalFile("..."))**
- **self.zene.setLoops(QMediaPlayer.Loops.Infinite)**
- **self.zene.play()** – Elindít egy zenét, folyamatosan ismétli.

A create és a remove cloud függvények a MainWindowban részletesen el van magyarázva, annyi különbség van, hogy ott egy felhő kép, itt egy szöveg megy végig a képernyőn, mely arabul azt jelenti: Kalózkodott!

A főablak (MainWindow) osztály alapjai:

Konstruktor (`__init__` metódus):

- `super().__init__()` – Az ősosztály (QMainWindow) inicializálása.
- `self.Start()` – Üzenetet küld a konzolnak, betölti a konfigurációs szótárt (Json fájl), valamint a kedvencekből rákeres az egyik város adataira.
- `ora` – Lekéri a számítógép pontos óráját.
- `self.aktív_karakter` – Betölti a configból a karaktert a beállítások listából.

Dinamikus háttér beállítása:

A háttérkép az aktuális időtől függően változik.

- **A háttérkép az idő alapján változik, például:**
 - **03:00 - 06:00** → Napfelkelte (SCY4.png)
 - **06:00 - 16:00** → Nappal (SCY1.png)
 - **16:00 - 18:00** → Napnyugta (SCY3.png)
 - **18:00 - 03:00** → Éjszaka (SCY2.png)
- **Kép beállítása (Ugyan az a séma mindegyik címkénél):**
 - **Különböző nevű címkék:**
 - `self.esté`
 - `self.Napfelkelte`
 - `self.Nappal`
 - `self.Napnyugta`
 - **`self.esté = QLabel(self)`** – Megadjuk a modulját ahogy viselkedjen és hogy magára vonatkozzon, egy self értéket.
 - **`self.esté.setGeometry(QRect(x,y,x,y)`** – Megadjuk, hogy mettől meddig érjen a területe. (pl: 0x és 0y-tól a 300x és 300y-tengelyig pixelekben számítva.)
 - **`self.esté.setPixmap(QPixmap(„útvonal”))`** – Beállítjuk aPixmap modult mellyel lehet képet beilleszteni, és megadjuk a kép útvonalát.
 - **`self.esté.setScaledContents(True/False)`** – Engedélyezzük, hogy a tartalmat nyújtsa, ha nincs bekapsolva a tartalmat meg sem jeleníti.

A MainWindow alapvető beállításai:

- **setWindowIcon(QIcon(„útvonal”))** – Az alkalmazás ikonját beállítja.
- **setWindowTitle(„cím”)** – Az ablak címét adja meg.
- **setFixedSize(x, y)** – Fix méretű (1300x900) ablak létrehozása.

Felhő értékei:

- **self.clouds = []** – Lista a megjelenített felhők tárolására.
- **self.max_clouds = 4** – Legfeljebb 4 felhő jelenhet meg egyszerre.
- **self.startCloudAnimation()** – A felhőanimáció elindítása. (192.oldal)

Felhasználói felület színes elemei:

- **Ez a kódrész különböző dobozokat (Qlabel) hoz létre:**
 - **self.Kulso**
 - **self.belso**
 - **self.dialogbox**
 - **self.belsoM**
 - **self.kulsoM**
- **Stílus beállítások:**
 - **self.Kulso.setGeometry**
 - **self.Kulso.setStyleSheet("background-color:rgb(r, g, b)")** – Kiszínezi a doboz egy megadott színre.
 - **self.Kulso.setFrameShape(QLabel.Shape.Box)** – Ad egy külső keretet a doboz köré.
 - **self.Kulso.setFrameShadow(QLabel.Shadow.Raised)** – A dobozra egy 3D-s árnyékot rak.
 - **self.dialogbox.setLineWidth(szám)** – beállítja a QLabel szélességét.

Szöveg doboz (Dialógus ablak):

- **self.dialog = QLabel(self)**
- **self.dialog.setGeometry(QRect(x, y, x, y))**
- **self.dialog.setFont(QFont("Eras Demi ITC", 18))** - Beállítja a szöveg betűtípusát és méretét.

- **self.dialog.setStyleSheet("font-size: px; color: szín ")** - CSS-szerű stílussal a szöveg mérete 30px, színe fehér.
- **self.dialog.setText(„szöveg”)** -Kezdő szöveget állít be a dialógus ablakban.
- **self.dialog.setAlignment(Qt.AlignmentFlag.AlignLeft|Qt.AlignmentFlag.AlignTop)** - A szöveg igazítása bal felső sarokhoz történik.
- **self.dialog.setWordWrap(True)** -A szöveg sortöréssel jelenik meg, hogy ne lógjon ki a dobozból.

Verziószám és Licenc információ:

- **self.License = QLabel(„szöveg”, self)** – A setText kihagyható, ha a QLabel-be írjuk be a szöveget. Ez megjeleníti a szöveget.
- **self.License.setStyleSheet("font-size: px; color: szín")**
- **self.License.setGeometry(QRect(x, y, x, y))**
- **self.verzio = QLabel(„szöveg”, self)**
- **self.verzio.setStyleSheet("font-size: px; color: szín")**
- **self.verzio.setGeometry(QRect(x, y, x, y))**

Menü és stáruszsor

- **self.centralwidget = QWidget(self)** – Létrehoz egy widgetet.
- **self.setCentralWidget(self.centralwidget)** - Központi widgeté alakítja, amelybe egyéb elemek helyezhetők.
- **self.menubar = QMenuBar(self)** – Egy menüsor hoz létre.
- **self.setMenuBar(self.menubar)** – Beállítja a menubar-t érvényes menüként.

Beállítások menü és téma:

- **self.menuSettings = QMenu("Settings", self)** – Létrehoz egy menü opciót.
- **self.menubar.addMenu(self.menuSettings)** – Hozzáadja a menühöz.
- **self.menuTheme = QMenu("Theme", self)** – Létrehoz egy theme almenüt.
- **self.actionDark = QAction("Dark", self)** – Létrehoz egy almenüt a theme-nek.
- **self.actionLight = QAction("Light", self)** - Létrehoz egy almenüt a theme-nek.

A főablak (MainWindow) funkciói:

Felhő megjelenítés kapcsoló:

- **self.actionCloudsBe = self.config.get("Beállítások").get("Clouds")** - Betölti a felhő beállításokat a konfigurációs fájlból (self.config).
- **self.actionClouds = QAction("Clouds", self, checkable=True)**
 - Létrehoz egy be/kikapcsolható menüpontot
- **self.actionClouds.setChecked(bool(self.config.get("Beállítások").get("Felhők")))**
 - Ellenőrzi, hogy legutóbb be volt-e kapcsolva, ha igen, bekapcsolja, ha nem, nem.
- **self.actionClouds.triggered.connect(self.be_ki_cloud)**
 - A felhő opció kiválasztásakor meghívódik a *self.be_ki_cloud()* metódus.

Felhő animáció engedélyezése:

- **if self.actionCloudsBe == "True":**
- **self.startCloudAnimation()**
 - Ellenőrzi a konfigurációs fájlból, hogy be van-e kapcsolva a "Felhők" funkció.
 - Ha igen ("True"), akkor elindítja a felhő animációt a *startCloudAnimation()* metódussal.

Téma beállítások:

- **self.menuTheme.addAction(self.actionClouds)**
- **self.menuTheme.addAction(self.actionDark)**
- **self.menuTheme.addAction(self.actionLight)**
 - **Clouds:** Felhők be/ki kapcsolása.
 - **Dark:** Sötét téma.
 - **Light:** Világos téma.

Nyelv választó almenü

- **self.menuLanguage = QMenu("Language", self)** - Létrehozza a "Language" (Nyelv) almenüt.
- **self.actionMagyar = QAction("Magyar", self)**
- **self.actionEnglish = QAction("English", self)**
- **self.actionDeutsch = QAction("Deutsch", self)**
- **self.actionEspanol = QAction("Español", self)**
- **self.actionJapanese = QAction("日本語", self)**
- **self.actionChinese = QAction("中文", self)**
 - Hat nyelvi opció létrehozása (magyar, angol, német, spanyol, japán, kínai).
- **self.menuLanguage.addAction([...])** - Hozzáadja az összes nyelvet a Language menűhöz.
- **self.actionMagyar.triggered.connect(lambda: self.Nyelvek("hu"))**
- **self.actionMagyar.triggered.connect(lambda: self.Nyelvek("en"))**
- ...
 - minden nyelvválasztó menüpont a Nyelvek() metódust hívja meg a megfelelő nyelvkóddal (ISO formátumban).

Hang beállítások:

- **self.menuSounds = QMenu("Sounds", self)**
- **self.menuMusic = QMenu("Music", self)**
 - Létrehozza a "Sounds" főmenüt és egy "Music" almenüt.
- **self.actionMusicOn = QAction("On", self, checkable=True, checked=False)**
 - Egy kapcsolható menüpont, alapértelmezetten nincs bejelölve.
- **if self.config.get("Beállítások").get("Zene") == "Be":**
- **self.actionMusicOn.setChecked(True)**
 - Ha a konfiguráció szerint a zene be van kapcsolva, akkor pipálva jelenik meg.
- **self.actionMusicOn.triggered.connect(self.media_Status_Changed)**
 - A menüpontra kattintva meghívja a media_Status_Changed() metódust.
- **self.menuMusic.addAction(self.actionMusicOn)**
 - Hozzáadja a zenét a "Music" almenühöz.

Hang (beszéd) beállítások:

- **self.jelenlegi_weather = "base"** - Ez egy alapértelmezett érték mely később lesz felhasználva.
- **self.menuVoice = QMenu("Voice", self)**
- **self.actionVoiceOn = QAction("On", self, checkable=True)**
- **self.menuVoice.addAction(self.actionVoiceOn)**
 - Létrehozza a "Voice" almenüt, benne egy pipálható kapcsolóval.
- **if self.config.get("Beállítások").get("Hang") == "Be":**
- **self.actionVoiceOn.setChecked(True)**
 - Beállítja alapértelmezetten aktívra, ha a konfigurációban engedélyezve van.
- **self.actionVoiceOn.triggered.connect(self.hangteszt)**
 - A kapcsoló aktiválásakor meghívja a *hangteszt()* metódust.

Rejtett hangteszt gomb:

- **self.waos = QPushButton(self)**
- **self.waos.setText("Waos")**
- **self.waos.clicked.connect(self.hangtest)**
- **self.waos.setFixedSize(x, y)**
- **self.waos.move(x, y)**
 - Egy „Waos” feliratú gomb, amely meghívja a *hangtest()* metódust.
 - Mivel nagyon messzire van mozgatva (-2235 az x tengelyen), nem látszik, így kvázi „rejtett” gombként funkcionál. Ezt a hang debuggoltatása közben használtuk a fejlesztés során.

Menü összeállítása:

- **self.mentsd_le = "Be"**
- **self.menuSounds.addMenu(self.menuMusic)**
- **self.menuSounds.addMenu(self.menuVoice)**
 - A menuSounds menübe beágyazza a menuMusic és menuVoice almenüket.
 - A mentsd_le = "Be" változó a hang beállítások mentéséhez lesz használva később.

Settings (Beállítások) menü: almenük és egyéb opciók hozzáadása:

- `self.menuSettings.addMenu(self.menuTheme)`
- `self.menuSettings.addMenu(self.menuLanguage)`
- `self.menuSettings.addMenu(self.menuSounds)`
 - A Settings fómenühöz hozzáadjuk a Theme, Language, és Sounds almenüket.
- `self.actionHistory = QAction("Delete history", self)`
- `self.actionHelp = QAction("Help", self)`
- `self.actionExit = QAction("Exit", self)`
- `self.menuSettings.addActions([self.actionHistory, self.actionHelp, self.actionExit])`
- Három további opción:
 - **Delete history:** Előzmények törlése
 - **Help:** Súgó
 - **Exit:** Kilépés az alkalmazásból

Téma (sötét/világos) mód kapcsolás:

- `self.actionDark.triggered.connect(self.DarkMode)`
- `self.actionLight.triggered.connect(self.LightMode)`
 - Ha a felhasználó a Dark vagy Light menüpontot választja, akkor átváltja a témát a megfelelő metódussal.

Karakter választó menü:

- `self.jelenlegi_karakter = None`
- `self.menuCharacters = QMenu("Characters", self)`
 - Létrehozza a Characters menüt és egy változót az aktuális karakter tárolására.
- `self.actionBorakku = QAction("Borakku Shinko", self)`
- `self.actionHiyori = QAction("Hiyori Awai", self)`
- `self.actionUsagi = QAction("Usagi Tomoko", self)`
 - Három karakter opción, japán stílusú nevekkel.
- `self.menuCharacters.addActions([...])`
- `self.menuBar.addMenu(self.menuCharacters)`
 - Hozzáadja az akciókat és a menüt a felső menüsorhoz.
- `self.actionBorakku.triggered.connect(lambda: self.Karakterek("Borakku"))`

- ...
 - minden karakter választó hívja a *Karakterek()* metódust a karakter nevével.

Súgó és kilépés:

- **self.actionHelp.triggered.connect(self.Help)**
- **self.actionExit.triggered.connect(self.close)**
 - A Help opció a *Help()* metódust hívja meg.
 - Az Exit egyszerűen bezárja az alkalmazást.

Easter Egg:

- **self.menuVoice.triggered.connect(self.FlappyBird)**
 - Ha valaki a Voice menüre kattint, titokban elindul egy *FlappyBird()* nevű funkció – ez egy elrejtett meglepetés (easter egg).

Stáruszor:

- **self.statusbar = QStatusBar(self)**
- **self.setStatusBar(self.statusbar)**
 - Alul elhelyezkedő stáruszor, információk megjelenítésére.

Keresés gomb:

- **self.kereses = QPushButton(self)**
- ...
 - Egy "Keresés" feliratú gomb, ami egy keresési műveletet indít.
- **self.kereses.clicked.connect(self.keresesClicked)**
 - Egy "Keresés" feliratú gomb, ami egy keresési műveletet indít.

Összehasonlítás gomb:

- **self.ossz1b = QPushButton(self)**
- ...
 - "Összehasonlítás" gomb, összehasonlítás indításához. A *self.ossz* változó követi az állapotát (aktív vagy sem).
- **self.ossz = False**
 - "Összehasonlítás" gomb, összehasonlítás indításához. A *self.ossz* változó követi az állapotát (aktív vagy sem).

Keresősáv:

- **self.input = QLineEdit(self)**
- ...
 - Nagy méretű keresősáv szövegbevitelhez.
- **self.input.setFont(myfont)**
 - Nagy méretű keresősáv szövegbevitelhez.

Keresősáv háttér:

- **self.keresoHatter = QLabel(self)**
- ...
- **self.keresoHatter.setPixmap(QPixmap("Background/listBG.png"))**
 - Keresősáv mögötti vizuális háttérkép, hogy jobban nézzen ki az UI.

Időjárás lista:

- **self.weather = QListWidget(self)**
- ...
- **self.weather.setFont(myfont)**
 - Egy QListWidget, ami az időjárási eredményeket jeleníti meg. Átlátszó stílusú, egyedi görgetőstílus nélkül.

Értékváltó gombok:

- **self.ertekValtas = QPushButton(self)**
- ...
- **self.ertekValtas.clicked.connect(self.ErtekValtas)**

Fok váltás:

- **self.fokValtas = QPushButton(self)**
- ...
- **self.fokValtas.clicked.connect(self.FokValtasClicked)**

Kedvenc:

- **self.kedvenc = QPushButton(self)**
- ...
- **self.kedvenc.clicked.connect(self.KedvencClicked)**

Összehasonlítás lista és törlés gomb:

- **self.ossz1 = QListWidget(self)**
- ...
- **self.ossz1DeleteButton.clicked.connect(self.ossz1DeleteButtonClicked)**
 - Rejtett QListWidget és egy törlés gomb, az összehasonlítások törlésére szolgál.

Pontos idő kijelzése:

- **self.pontosido = QLabel(self)**
- ...
- **self.pontosido()**
 - Egy címke a pontos idő kijelzésére, amit a *pontosido()* metódus frissít rendszeresen.

Zenelejátszó beállításai:

- **self.zene = QMediaPlayer()**
- **self.zeneOutput = QAudioOutput()**
- **self.zene.setAudioOutput(self.zeneOutput)**
- **self.zene.mediaStatusChanged.connect(self.media_Status_Changed)**
- **self.MusicOn = True**
 - Létrehoz egy QMediaPlayer objektumot a háttérzene lejátszásához.
 - Hozzárendel egy QAudioOutput kimenetet.
 - Összeköti a mediaStatusChanged eseményt egy saját metódussal a zenei állapot kezelésére.
 - **self.MusicOn = True**: alapértelmezés szerint a zene be van kapcsolva.

Zene fájlok betöltése (nappali és esti):

- **self.Nappali_zene_fájlok = self.ZeneBetolt("Musics/Day")**
- **self.Este_zene_fájlok = self.ZeneBetolt("Musics/Night")**
 - Külön könyvtárból (Musics/Day, Musics/Night) betölti a zenefájlokat.
 - A *ZeneBetolt()* visszaad egy fájllistát.

Nap és Hold ikon beállítása az aktuális idő alapján:

- **if ora > 6 and ora < 18:**
- **self.IJ.setPixmap(QPixmap("Weather conditions/Sun.png"))**
- ...
- **self.play_random_Nappali_zene()**
- **else:**
- ...
- **self.play_random_Esti_zene()**
 - Ha nappal van (6:00 – 18:00), akkor **Nap ikon** jelenik meg, és nappali zene indul.
 - Ellenkező esetben **Hold ikon** jelenik meg, és esti zene indul.

Kedvencek közötti léptetés (bal/jobb nyilak):

- **self.FavSwitchRightButton = QPushButton(self)**
- **self.FavSwitchRightButton.setText(">>")**
- ...
- **self.FavSwitchRightButton.clicked.connect(self.FavSwitchButtonRightClicked)**
- **self.FavSwitchLeftButton = QPushButton(self)**
- **self.FavSwitchLeftButton.setText("<<")**
- ...
- **self.FavSwitchLeftButton.clicked.connect(self.FavSwitchButtonLeftClicked)**
 - Két gomb a kedvenc városok vagy keresések közötti léptetéshez.
 - Átlátszó stílus, nagy méretű nyilakkal (>> és <<).

Fokozat kijelző frissítése:

- **self.fokValtas.setText(self.config.get("Beállítások").get("FokÉrték"))**
 - Beállítja a fokváltó gomb szövegét az elmentett értékre (°C vagy °F például).

Utoljára használt karakter beállítása:

- **utolso_karakter = self.config.get("Beállítások").get("Karakter", "Borakku")**
- **self.Karakterek(utolso_karakter)**
 - Az utoljára kiválasztott karaktert betölti az alkalmazás elindításakor.
 - Ha nincs elmentve, akkor a **Borakku** lesz az alapértelmezett karakter.

Hang (voice) lejátszó beállítása:

- **self.voice_player = QMediaPlayer()**
- **self.voice_output = QAudioOutput()**
- **self.voice_player.setAudioOutput(self.voice_output)**
- **self.voice_player.setLoops(QMediaPlayer.Loops.Infinite)**
 - Beállítja az zenelejátszót, az outputot a karakter hangokhoz és folyamatosan loopolja míg meg nincs szakítva.

Függvények (metódusok):

def DarkMode(self):

- **self.setStyleSheet(dark_mode_stylesheet)**
- **self.dialog.setStyleSheet("font-size: 30px; color: #ffffff")**
- ...
 - Aktiválja a sötét módot az alkalmazásban.
 - Beállítja a főablak sötét témaját a 'dark_mode_stylesheet' alapján.
 - Módosítja az összes főbb GUI-elem stílusát (pl. betűméret, szín).
 - Biztosítja a self.setStyleSheet utáni részben, hogy minden vezérlő átméretezve és jól olvasható legyen sötét háttéren.

def LightMode(self):

- Aktiválja a világos módot az alkalmazásban.
- Visszaállítja a főablak stílusát a 'light_mode_stylesheet' alapján.
- Ugyanazokat az elemeket kezeli, mint a DarkMode, de világos dizájn szerint.

def create_cloud(self):

- **self.actionCloudsBe = self.config.get("Beállítások").get("Clouds")**
- **if self.actionCloudsBe == "True":**
 - Csak akkor aktív, ha a 'Clouds' beállítás 'True'.
- **if len(self.clouds) >= self.max_clouds:** - ellenőrzi, hogy nem lépte-e túl a 4-et.
- **return**
- **self.felho = QLabel(self)**
- ...
- **self.felho_animacio = QPropertyAnimation(self.felho, b"geometry")**
- **self.felho_animacio.setDuration(20000)**
- **self.felho_animacio.setLoopCount(-1) – végtelen loop**
 - Egy új, animált felhőt hoz létre, amely az ablakon áthalad.
 - A felhő QLabel objektumként jön létre, és animált mozgással halad át a képernyőn balról jobbra.
- **self.felho_animacio.finished.connect(lambda: self.remove_cloud(self.felho))**
- **self.felho_animacio.start()**
- **self.clouds.append((self.felho, self.felho_animacio))**
- **self.startCloudAnimation()**
 - Az animáció végeztével automatikusan törlődik a *remove_cloud()* metódus által.
- **else:**
- **self.config.get("Beállítások")["Clouds"] = "False"**

- Ha a 'Clouds' beállítás False, akkor a biztonság kedvéért Falsera állítja.

```
def be_ki_cloud(self):
```

- **self.actionCloudsBe = self.config.get("Beállítások").get("Clouds")**
- **if self.actionCloudsBe == "False":**
- **self.config.get("Beállítások")["Clouds"] = "True"**
- **self.create_cloud()**
- **else:**
- **self.config.get("Beállítások")["Clouds"] = "False"**
- **self.stopCloudAnimation()**
 - Kapcsolatja a felhő animációkat ki/be.
 - Ha jelenleg ki van kapcsolva ('False'), bekapcsolja, és elindítja az animációt.
 - Ha be van kapcsolva ('True'), leállítja az összes animációt, és eltünteti a felhőket.

```
def remove_cloud(self, felho):
```

- **self.actionCloudsBe = self.config.get(...)**
- **self.felho.deleteLater()**
- **self.clouds = [(c, a) for c, a in self.clouds if c != self.felho]**
 - felho (QLabel): Az a felhő QLabel, amelyet el kell távolítani.
 - Eltávolítja az adott felhőt a képernyőről és frissíti a felhőlistát (-1 felhő jelent).

```
def stopCloudAnimation(self):
```

- **self.actionCloudsBe = self.config.get(...)**
- **for felho, animacio in self.clouds:**
 - **animacio.stop()**
 - **felho.deleteLater()**
 - **self.clouds.clear()**
 - Megállítja az összes aktív felhő animációt.
 - Leállítja a folyamatban lévő animációkat.
 - Eltávolítja a felhő QLabel-eket.
- **if hasattr(self, 'timer') and self.timer.isActive():**
- **self.timer.stop()**
- **print("QTimer leállítva")**
 - Leállítja a QTimer-t is, ha aktív.

- ```
def startCloudAnimation(self):
 • self.actionCloudsBe = self.config.get(...)
 • if self.actionCloudsBe == "True":
 ○ Elindítja a felhő animáció időzítőt, ha engedélyezve van a 'Clouds' opció.
 • if not hasattr(self, 'timer') or not self.timer.isActive():
 • self.timer = QTimer(self)
 ○ Csak akkor indul el, ha nincs már aktív időzítő.
 • self.timer.timeout.connect(self.create_cloud)
 • self.timer.start(6000)
 • print(" QTimer elindítva")
 ○ 6 másodpercenként új felhőt generál (QTimer timeout alapján).
 • else:
 • self.stopCloudAnimation()
 ○ Ha a Clouds beállítás nem True, meghívja a stopCloudAnimation() metódust.
```

#### **def ZeneBetolt(self, mappa):**

- **zene\_fájlok = []**
- **for file in os.listdir(mappa):**
  - A mappában lévő összes fájlon végigmegy melyet még az lapoknál beállítottunk.
- **if file.endswith(".mp3"):**
- **zene\_fájlok.append(os.path.join(mappa, file))**
  - Ha a fájl mp3, akkor hozzáadja a zene\_fájlok tömbhöz.
- **return zene\_fájlok**
  - Az értéket visszaadja annak ami meghívta.

#### **def play\_random\_Esti\_zene(self):**

- **aktív\_zene = random.choice(self.Este\_zene\_fájlok)**
- **self.zene.setSource(QUrl.fromLocalFile(aktív\_zene))**
  - Véletlenszerű esti zenét választ ki és beállítja lejátszásra.

#### **def play\_random\_Nappali\_zene(self):**

**Véletlenszerű nappali zenét választ ki és beállítja lejátszásra.**

```
def media_Status_Changed(self, status):
```

- **if status == QMediaPlayer.MediaStatus.EndOfMedia:**
- **self.zene.play()**
  - Ha a zenefájl állapota megváltozik vagy véget ér, eljátssza a zenét.
- **if not self.actionMusicOn.isChecked():**
- **self.zene.stop()**
- **self.config.get("Beállítások")["Zene"] = "Ki"**
  - Viszont ha nincs bekapcsolva, akkor megállítja azonnal és átírja a configban.
- **elif self.actionMusicOn.isChecked():**
- **self.zene.play()**
  - Máskülönben az ellentéte.
- **self.config.get("Beállítások")["Zene"] = "Be"**
  - Ha a felhasználó kikapcsolja a zenét (GUI-ból), leállítja a lejátszót.

```
def hangtest(self):
```

- **if self.actionVoiceOn.isChecked():**
- **self.config.get("Beállítások")["Hang"] = "Be"**
- **self.voice\_player.play()**
- **else:**
- **self.voice\_player.stop()**
- **self.config.get("Beállítások")["Hang"] = "Ki"**
  - Elindítja vagy leállítja a voice-over funkciót a felhasználó beállítása szerint.
  - Ha a 'Hang' beállítás 'Be', elindítja a voice\_player-t.
  - Ha 'Ki', leállítja azt.

```
def hangteszt(self, status):
```

- **if not self.actionVoiceOn.isChecked():**
- **self.config.get("Beállítások")["Zene"] = "Ki"**
- **self.mentsd\_le = "Ki"**
- **elif self.actionVoiceOn.isChecked():**
- **self.config.get("Beállítások")["Zene"] = "Be"**
- **self.mentsd\_le = "Be"**
  - Belső logikát kezeli a voice funkcióhoz kapcsolódóan.
  - A 'mentsd\_le' változó beállításával jelzi, hogy menteni kell-e a beállítást.
  - Megkülönbözteti a be- és kikapcsolt állapotokat.

```
def Help(self):
```

- **útvonal = "READ ME.txt"**
- **subprocess.Popen(["notepad.exe", útvonal])**
  - Ez a metódus megnyitja a READ ME.txt fájlt a Windows Notepad alkalmazással.
  - A felhasználó egy gombnyomással elérhesse a program dokumentációját vagy használati útmutatóját.
  - A fájlnak a program könyvtárában kell lennie.

```
def FlappyBird(self):
```

- **útvonal = "Easter Egg/Flappy Bunny.py"**
- **subprocess.Popen(["python", útvonal])**
  - Ez egy rejtett funkció, ami elindít egy minijátékot, a Flappy Bunny-t.
  - A játék szkriptje az Easter Egg mappában kell legyen.
  - A játék kicserélhető bármilyen .py fájrra!

```
def Karakterek(self, Karakter_név, weather_status=None):
```

- **Karakter\_név:** a karakter neve, pl. "Borakku".
- **weather\_status:** opcionális, az időjárás típusát adja meg (pl. "sunny", "rainy"). Ha nincs megadva, akkor az aktuális tárolt értéket próbálja használni.
- **self.timer = QTimer(self)**
  - Létrehoz egy időzítőt (QTimer), ami majd később az animált kéz eltüntetésére szolgál.
- **if weather\_status is None:**
- **weather\_status = self.jelenlegi\_weather**
- **if hasattr(self, 'jelenlegi\_weather') else "base"**
  - Ha nincs explicit megadva az időjárás (weather\_status), akkor az objektum jelenlegi\_weather attribútumát használja, ha létezik.
  - Ellenkező esetben "base" értéket állít be (alapértelmezett állapot).
- **print(f"Karakter kiválasztva: {Karakter\_név}, időjárás állapot: {weather\_status}")**
  - Naplózza, hogy milyen karaktert és milyen időjárási típust próbál betölteni.
- **if self.jelenlegi\_karakter:**
- **self.jelenlegi\_karakter.deleteLater()**
  - Ha már van megjelenített karakter (jelenlegi\_karakter), azt eltávolítja a GUI-ról.

- **self.jelenlegi\_karakter = QLabel(self)**
- **self.aktív\_karakter = self.config.get("Beállítások")["Karakter"]**
- **sprite\_found = False**
  - Létrehoz egy új QLabel objektumot a karakter megjelenítéséhez.
  - Az aktuálisan aktív karaktert a configban találhatóra elnevezi.
  - A sprite\_found változó azt jelzi majd, hogy sikeres volt-e érvényes sprite fájlt betölteni.
- **for i in range(1, 3):**
- **sprite\_path = f"Sprites/{Karakter\_név}"**  
**sprites/{Karakter\_név.lower()}\_{weather\_status}\_{i}.png"**
- **pixmap = QPixmap(sprite\_path)**
  - Végigmegy egy 1–2-es tartományon, hogy megtalálja az első elérhető sprite-ot.
  - A sprite útvonala például így néz ki:
    - Sprites/Borakku sprites/borakku\_sad\_1.png
- **if pixmap and not pixmap.isNull():**
- **sprite\_found = True**
- **self.jelenlegi\_karakter.setPixmap(pixmap)**
- **(...)**
- **self.jelenlegi\_karakter.show()**
- **self.aktív\_karakter = Karakter\_név**
- **break**
  - Ha sikeres volt érvényes képet betölteni:
    - Beállítja a karakter képét a QLabel-re.
    - Alapbeállításokat a képnek megadja.
    - Megjeleníti a karaktert.
    - Elmenti az aktív karakter nevét.
- **if not sprite\_found:**
- **print(f"Nem sikeres volt betölteni a képet: {Karakter\_név}\_{weather\_status}")**
  - Ha nem sikeres volt egyetlen sprite-ot sem betölteni, ezt naplózza.
- **error\_pixmap = QPixmap(f"Sprites/{Karakter\_név}"**  
**sprites/{Karakter\_név.lower()}\_error.png")**
- **if error\_pixmap and not error\_pixmap.isNull():**
- **self.jelenlegi\_karakter.setPixmap(error\_pixmap)**
- **...**
- **print(f"Nem található az error sprite: {Karakter\_név.lower()}\_error.png")**
  - Megpróbálja betölteni a \*\_error.png fájlt, ha a normál sprite nem található.
  - Ez egy hibakezelés, hogy ne maradjon üres a képernyő.
- **self.kez = QLabel(self)**
- **kez\_pixmap = QPixmap(f"Sprites/{Karakter\_név}"**  
**sprites/{Karakter\_név.lower()}\_arm.png")**

- Egy új QLabel-t hoz létre a karakter "kezének" sprite-jához.
- **Betölti a képet: pl. borakku\_arm.png.**
- **self.kez.setPixmap(kez\_pixmap)**
- **(...)**
  - Beállítja a kéz sprite megjelenítését és méretét.
- **if hasattr(self, "kez") and self.kez.isVisible():**
- **self.kez.deleteLater()**
  - Biztonsági okokból ellenőrzi, hogy korábbi "kéz" sprite látható-e, és ha igen, eltávolítja.
- **self.kez.show()**
- **self.timer.start(3000)**
- **QTimer.singleShot(4000, self.kez.deleteLater)**
  - Megjeleníti a kezet.
  - 4 másodperc múlva automatikusan eltávolítja.
- **self.config["Beállítások"]["Karakter"] = Karakter\_név**
  - Frissíti a konfigurációs állományban az aktív karakter nevét.

#### **Start(self):**

- Ez a metódus az alkalmazás elindulásakor fut le. Betölti a konfigurációs fájlt, vagy ha az hibás vagy nem létezik, akkor létrehoz egy alapértelmezettet.
- **print("A program elindult!")**
  - Naplózza a terminálba, hogy a program elindult.
- **try:**
- **with open("config1.json", "r", encoding="utf-8") as r:**
- **loaded\_data = json.load(r)**
- **self.config = loaded\_data**
  - Megpróbálja megnyitni és beolvasni a config1.json fájlt.
  - A beolvasott JSON adatot betölti az osztály self.config attribútumába.
- **except (json.JSONDecodeError, FileNotFoundError) as e:**
- **print(f"hiba a config1.json fájl betöltésekor: {e}")**
  - Ha a fájl nem található vagy hibás a formátuma (nem értelmezhető JSON), akkor:
    - kiírja a hibát a terminálra,
    - majd alapértelmezett beállításokat hoz létre.

```

 self.config = {
 "Beállítások": {
 "Otthon": "",
 "FokÉrték": "C°",
 "Karakter": "Borakku",
 "Zene": "Be",
 "Előz/Fav": {
 "Előzmények": [],
 "Kedvencek": []
 }
 }
 }

 if "Beállítások" not in self.config:
 self.config["Beállítások"] = [
 "Otthon": "",
 "FokÉrték": "C°",
 "Karakter": "Borakku",
 "Zene": "Be",
 "Előz/Fav": {
 "Előzmények": [],
 "Kedvencek": []
 }
]
 }

```

Ez az alapkonfiguráció. Ha nincs fájl, ezt használja.

### Főbb kulcsok:

Otthon: üres (pl. városnév)

FokÉrték: Celsius

Karakter: alapértelmezett karakter

Zene: be van kapcsolva

Előz/Fav: előzmények és kedvencek üresek

- **if "Beállítások" not in self.config:**
- **self.config["Beállítások"] = { ... }**
  - Biztonsági vizsgálat: ha a "Beállítások" kulcs valamiért hiányzik (pl. hibás szerkezetű fájl), akkor újra létrehozza.
- **if len(self.config.get("Előz/Fav").get("Kedvencek")) == 0:**
- **self.FavSwitchRightButton.setEnabled(False)**
- **self.FavSwitchLeftButton.setEnabled(False)**
  - Ha nincsenek kedvenc városok, letiltja a bal/jobb váltógombokat a felületen.

### closeEvent(self, event)

- **self.on\_exit()**
- **event.accept()**
  - Ez a metódus akkor fut le, amikor a felhasználó bezárja az alkalmazást. Ez biztosítja, hogy a kilépés előtt minden fontos mentés megtörténjen.

### def closeEvent(self, event):

- **self.on\_exit()**
- **event.accept()**
  - Meghívja az on\_exit() metódust, amely elmenti a konfigurációt.
  - Ezután elfogadja az ablak bezárását (event.accept()).

### on\_exit(self)

- `print("A program leáll!")`
  - Naplózza, hogy a program be fog záródni.
- `self.config.get("Beállítások")["Karakter"] = self.aktív_karakter`
- `self.config.get("Beállítások")["Hang"] = self.mentsd_le`
  - Frissíti a konfigurációt az utoljára használt karakterrel és hang beállítással.
- `if not self.actionMusicOn.isChecked():`
- `self.config.get("Beállítások")["Zene"] = "Ki"`
- `elif self.actionMusicOn.isChecked():`
- `self.config.get("Beállítások")["Zene"] = "Be"`
  - A menüből származó actionMusicOn érték alapján eldönti, hogy a zene be vagy ki van kapcsolva, és ezt menti a configba.
- `with open("config1.json", "w", encoding="utf-8") as w:`
- `json.dump(self.config, w, ensure_ascii=False, indent=4)`
  - Elmenti az aktuális konfigurációt a config1.json fájlba, szépen formázva (indent=4, UTF-8 kódolással, ékezeteket megtartva).

### Def TextBox(self)

- `txt = f"{self.weather.item(0).text()} településen"`
  - Ez kezdi a szöveget a kiválasztott város nevével.
- `for x in self.ertekek:`
- `if x[8:] not in a:`
- `a.append(x[8:])`
  - Kiszedi az ertekek listából a különböző időjárási állapotokat, majd hozzáadja őket a txt szöveghez, például: „napos, felhős, esős”.
- `for x in self.hoFokok:`
- `a.append(float(x[8:-2]))`
  - Itt újrahasználja a listát a hőmérsékleti értékek begyűjtésére és kiszámolja az átlagot, majd hozzáfűzi:
- `txt += f" továbbá {(sum(a) / len(a)).__round__(1)}{self.config.get("Beállítások").get("FokÉrték")} átlag hőmérséklet várható."`
- `self.dialogs = {`
- `"Usagi": {`
- `"eső": [...],`
- `...`
- `},`

- "Hiyori": {
- ...
- }
- }
- Térkép (dict) alapú megoldás, ahol karakter + időjárás kulcs alapján a megfelelő üzenet választható ki. Ezek nem jelennek meg automatikusan itt, de későbbi verzióban integrálhatók a szöveghez.
- **self.letter = QTimer(self)**
- **self.letter.setInterval(30)**
- **self.letter.timeout.connect(self.szoveg\_frissites)**
- **self.letter.start()**

#### Def szoveg\_frissites(self):

- **if self.index < self.változo:**
  - Amíg a szöveg vége nincs elérve (self.teljes), addig:
- **self.jelenleg += self.teljes[self.index]**
- **self.dialog.setText(self.jelenleg)**
- **self.index += 1**
  - A következő karaktert hozzáfüzi az épp megjelenített szöveghez (self.jelenleg), és beállítja a dialógus mezőbe.
- **self.adjustFontSizeToFit()**
  - Automatikusan csökkenti a szöveg betűméretét, ha túl hosszú, hogy beleférjen a QLabel méretébe.
- **self.kereses.setEnabled(False)**
- **self.fokValtas.setEnabled(False)**
- **self.FavSwitchLeftButton.setEnabled(False)**
- **self.FavSwitchRightButton.setEnabled(False)**
  - A gombokat letiltja, hogy a felhasználó ne szakíthassa meg az animációt.
- **if self.actionVoiceOn.isChecked():**
  - Ha be van kapcsolva a hang:
- **if self.voice\_player.playbackState() !=**  
**QMediaPlayer.PlaybackState.PlayingState:**
  - Ellenőrzi, hogy épp nem játszik le mászt a QMediaPlayer.
- **if self.aktív\_karakter == "Borakku":**
- **self.voice\_player.setSource(QUrl.fromLocalFile("Musics/..."))**
  - Ezután karakter szerint választja ki a hangfájlt:
- **if self.aktív\_karakter == "Hiyori":**
- ...
- **if self.aktív\_karakter == "Usagi":**
- ...
- **self.voice\_player.play()**

- **except Exception as hiba:**
- **print(f"hiba a szöveg frissítése közben: {hiba}")**
- **self.letter.stop()**
- **self.voice\_player.stop()**
  - Megállítja az animációt és a hangot, ha valami félremegy.
- **else:**
- **self.letter.stop()**
- **self.voice\_player.stop()**
  - Leállítja a karakterenkénti írást.
  - Újra engedélyezi az interakciókat a felhasználó számára:
- **self.kereses.setEnabled(True)**

**def adjustFontSizeToFit(self):**

- **font = self.dialog.font()**
- **metrics = QFontMetrics(font)**
  - Ez segít kiszámítani, hogy egy adott szöveg mekkora helyet foglal el az adott betűtípussal.
- **while metrics.boundingRect(self.dialog.text()).width() > self.dialog.width() and font.pointSize() > 1:**
  - Ha a szöveg szélesebb, mint a címke (QLabel) szélessége, akkor...
- **font.setPointSize(font.pointSize() - 1)**
- **self.dialog.setFont(font)**
- **metrics = QFontMetrics(font)**
  - ...mindaddig, amíg bele nem fér, vagy el nem éri az 1-es méretet.

**def Nyelvek(self, nyelv):**

- nyelv: egy rövid nyelvkód (pl. "hu", "en", "de", stb.)
- **if nyelv == "hu":**
- **print("Nyelv beállítva: Magyar")**
  - Ezt minden nyelvnél külön-külön ellenőrzi és kiírja a konzolra, hogy melyik nyelv lett kiválasztva.
- **self.config["Beállítások"]["Nyelv"] = nyelv**
  - Elmenti a választott nyelvet a beállítások közé, így az alkalmazás újraindítás után is emlékezni fog rá.
- **self.frissitsd\_azUlt()**
  - Meghívja a frissitsd\_azUlt metódust, amely a gombok és címkék feliratait frissíti az adott nyelvhez tartozó szövegekre.

```
def frissitsd_azUlt(self):
```

- **nyelv = self.config["Beállítások"]["Nyelv"]**
  - Lekéri az aktuális nyelvet a konfigurációból:
- **if nyelv == hu:**
- **self.kereses.setText("Keresés")**
- **self.ossz1b.setText("Összehasonlítás")**
- **elif nyelv == en:**
- **self.kereses.setText("Search")**
- **self.ossz1b.setText("Compare")**
- ...

```
def nincsTalalat(self): [SCRIPTED!]
```

- **txt = self.input.text()**
  - Végigmegy a varosok listán (ami minden bizonnal az elérhető városok listája).
- **for x in range(min(len(txt), len(city))):**
- **if txt[x].lower() == city[x].lower():**
- **pont += 1**
  - Pontozza a városokat az alapján, hogy hány egyező karakterük van a beírt szöveg elején:
  - Például
  - *txt = "bud" és city = "Budapest" → 3 pont*
  - *txt = "bud" és city = "Bukarest" → 2 pont*
- **javaslatok[city] = pont**
  - Egy javaslatok nevű szótárba menti a városokat és pontszámaikat.

```
def weatherClicked(self):
```

- **print(self.weather.get)**

```
def OsszFokValtas(self):
```

- **a = []**
- **if self.ossz:**
- **if self.fokValtas.text() == "C°":**
- **for x in self.osszlistFok:**
- **temp = ((5 / 9) \* (float(x[8:-2]) - 32)).\_\_round\_\_(1)**
  - minden érték string, pl.: "2025-04-25 15:00 68F°" → ezért a x[8:-2] szedi ki a számot.
- **append(f"{x[0:8]} {temp}C°")**
  - Átváltja az osszlistFok-ban szereplő értékeket a másik mértékegységre.

- **varos = self.ossz1.item(0).text()**
- **self.ossz1.clear()**
- **self.ossz1.addlItem(varos)**
- **self.ossz1.addlItem("")**
  - Az új értékeket újra beállítja a ossz1 listába.
  - Ezután váltja a hoFokok-at.
- **...**
  - Ugyanez történik a fő város időjárási listájával (self.weather), újraszámolja, törli a régit, és frissíti.

**def FokValtasClicked(self):**

- **fok = self.config.get("Beállítások").get("FokÉrték")**
- **if fok == "C°":**
- **fok = "F°"**
- **elif fok == "F°":**
- **fok = "C°"**
  - Frissíti a gomb feliratát.
- **self.fokValtas.setText(fok)**
  - Frissíti a configot is.
- **self.config.get("Beállítások")["FokÉrték"] = fok**
  - Átváltja a hőmérsékleti értékeket is az új rendszer szerint.
- **self.OsszFokValtas()**

**def Elozmeny(self):**

- **a = list(self.config.get("Előz/Fav").get("Előzmények"))**
  - Betölti az előzményeket a configból.
- **if self.input.text() not in a:**
  - **insert(0, self.input.text())**
    - Ha az aktuális keresés nincs az előzmények között, hozzáadja a lista elejére.
- **else:**
- **a.remove(self.input.text())**
- **a.insert(0, self.input.text())**
  - Ha már szerepel az aktuális keresés, akkor kiveszi a régi helyéről, és az elejére teszi.
- **if len(a) > 10:**
- **a.remove(a[10])**
  - Az előzmények hossza nem lehet több mint 10, így ha túllépné a 10-et, törli a legutolsó elemet.
- **self.config.get("Előz/Fav")["Előzmények"] = a**

```
def KedvencClicked(self):
```

- **a = list(self.config.get("Előz/Fav").get("Kedvencek"))**
- **if self.weather.item(0) != "":**
  - Megnézi, hogy a város kiválasztva van-e (ellenőrzi, hogy a város neve nem üres)
- **if self.weather.item(0).text() in a:**
- **a.remove(self.weather.item(0).text())**
- **self.kedvenc.setText("★")**
  - Ha a város már a kedvencek között van, eltávolítja a listából, és a csillag ikonját üres csillagként (★) módosítja
- **else:**
- **a.append(self.weather.item(0).text())**
- **self.kedvenc.setText("★")**
  - Ha a város nincs a kedvencek között, hozzáadja a listához, és a csillag ikont kitöltött csillagként (★) módosítja
- **self.config.get("Előz/Fav")["Kedvencek"] = a**

```
def pontosido(self):
```

- **now = datetime.now()**
  - A datetime.now() segítségével lekérdezi a pontos időt.
- **self.pontosido.setText(now.strftime("%H:%M"))**
  - A strftime formátumban a %H:%M az órát és percet jeleníti meg (például "15:30").
- **font = QFont()**
- **font.setBold(True)**
- **self.pontosido.setFont(font)**
  - A megjelenített időt félkövér betűtípussal rendezи el, hogy jobban kiemelkedjen.
- **QTimer.singleShot(10000, self.pontosido)**
  - A QTimer.singleShot(10000, self.pontosido) egy egyszeri időzítő, amely 10 másodperc után újra meghívja a pontosido metódust, ezzel folyamatosan frissítve az időt.

```
def OsszClicked(self):
```

- **self.ossz1.clear()**
- **self.osszlistFok = self.hoFokok.copy()**
- **self.osszlistido = self.ertekek.copy()**
- **self.ossz = True**
  - Először tisztítja az előzőleg hozzáadott elemeket az ossz1 listából.

- Ezután két listát, a hoFokok és ertekek listákat másolja az új listákba (oszlistFok, oszlistldo).
- A self.oszsz = True beállítja, hogy az összesített nézet aktív.
- **for x in range(len(self.weather)):**
- **self.oszsz1.addItem(self.weather.item(x).text())**
  - Az összes időjárási adatot hozzáadja az oszsz1 listához. A self.weather.item(x).text() minden elemének szövegét veszi.
- **self.oszsz1.setFixedSize(420, 400)**
- **self.oszsz1.move(570, 50)**
- **self.oszsz1.setStyleSheet(...')}**
  - A oszsz1 elemnek fix méretet és pozíciót ad.
  - A setStyleSheet segítségével beállítja a betűméretet, átlátszó háttért és eltávolítja a görgetősávot.
- **myfont = QFont()**
- **myfont.setBold(True)**
- **self.oszsz1.setFont(myfont)**
  - A lista szövegét félkövér betűtípussal jeleníti meg.
  - Háttérkép hozzáadása:
- **if not hasattr(self, 'oszkeresoHatter') or self.oszkeresoHatter is None or not self.oszkeresoHatter.isVisible():**
- **self.oszkeresoHatter = QLabel(self)**
- **...**
- **self.oszkeresoHatter.show()**
  - Ellenőrzi, hogy a háttérkép (oszkeresoHatter) már létezik-e és látható-e. Ha nem, akkor létrehozza azt, beállítja a háttérképet, és az alsó réteghez helyezi (stackUnder).
- **self.oszsz1DeleteButton.setFixedSize(50, 50)**
- **self.oszsz1DeleteButton.setText("X")**
- **...**
  - Hozzáad egy törlés gombot (X), amelynek szintén beállítja a stílusát (átlátszó háttér, nagy betűméret) és a pozícióját.

```
def oszsz1DeleteButtonClicked(self):
```

- **self.oszsz1.clear()**
- **self.oszlistFok.clear()**
- **self.oszlistldo.clear()**
  - Az oszsz1 lista (ahol az összesített adatokat jeleníti meg), valamint a oszlistFok és oszlistldo listák tartalmát törli.

- **self.ossz1.setFixedSize(0, 0)**
- **self.ossz1DeleteButton.setFixedSize(0, 0)**
  - A ossz1 listát és a törlés gombot eltünteti a képernyőről azáltal, hogy nullára állítja a méretüket.
- **try:**
- **self.osszkeresoHatter.deleteLater()**
- **except:**
- **print("HIBA: 1080.sor")**
  - A háttérképet, amelyet az előző OsszClicked metódusban adtunk hozzá, eltávolítja a deleteLater() hívással.
  - Ha hiba történik (pl. ha a háttérkép már nem létezik), egy hibaüzenetet ír ki.
- **self.ossz = False**
  - Az összesített nézetet letiltja a self.ossz = False beállítással.
- **x = 0**
- **while x < len(self.weather):**
- **self.ossz1.addItem(self.weather.item(x))**
- **x += 1**
  - A self.weather listából újra hozzáadja az adatokat az ossz1 listához. Ezzel visszaállítja a nézetet az alapállapotába, miután törölte az összesített adatokat.

### **def ErtekValtas(self):**

- **if self.weather.item(0).text().\_\_contains\_\_("HIBA") or self.weather.count() == 0:**
- **self.weather.clear()**
- **self.weather.addItem("HIBA: Nincs megadott város!")**
  - Az első elem szövegét ellenőrzi, hogy tartalmazza-e a "HIBA" szót, vagy ha a lista üres. Ha igen, törli az összes adatot, és megjelenít egy hibaüzenetet: "Nincs megadott város!".
- **elif self.weather.item(2).text().\_\_contains\_\_("o"):**
- **varos = self.weather.item(0).text()**
- **self.weather.clear()**
- **self.weather.addItem(varos)**
- **self.weather.addItem("")**
- **for x in self.ertekek:**
- **self.weather.addItem(x)**
- **if self.ossz:**
- **osszVaros = self.ossz1.item(0).text()**
- **self.ossz1.clear()**

- **self.ossz1.addItem(osszVaros)**
- **self.ossz1.addItem("")**
  
- **for x in self.osszlistido:**
  - Ha az időjárási lista 2. eleme tartalmaz "°" jelet (ez azt jelzi, hogy hőmérsékleti adatokat tartalmaz), akkor:
    - A város nevét és az időjárási adatokat frissíti a listában.
    - Ha az összesített nézet aktív (self.ossz), akkor az összesített adatokat is frissíti, hozzáadva az új időjárási adatokat (self.osszlistido).
    - Ha nem hőmérséklet:
- **else:**
- **varos = self.weather.item(0).text()**
- **self.weather.clear()**
- **self.weather.addItem(varos)**
- **self.weather.addItem("")**
- **for x in self.hoFokok:**
  - **self.weather.addItem(x)**
- **if self.ossz:**
  - **osszVaros = self.ossz1.item(0).text()**
  - **self.ossz1.clear()**
  - **self.ossz1.addItem(osszVaros)**
  - **self.ossz1.addItem("")**
  - **for x in self.osszlistFok:**
    - **self.ossz1.addItem(x)**
      - Ha az időjárási lista nem tartalmaz hőmérsékletet (nem található benne "°"), akkor a listát a megfelelő egyéb adatokkal tölti fel (pl. self.hoFokok), és az összesített nézetet is frissíti, ha aktív.

**def keresesClicked(self):**

- **self.weather.clear()**
- **self.ertekek.clear()**
- **self.hoFokok.clear()**
- **self.weather.addItem(self.input.text())**
- **self.weather.addItem("")**
  - Az előző keresés eredményeit törli a weather listából, az értékeket (ertekek és hoFokok) is törli, majd hozzáadja az új város nevét a keresett városként.

- **self.Elozmeny()**
  - Az Elozmeny metódus frissíti az előzményeket (pl. a keresett városokat), hogy nyomon követhesd a legutóbbi kereséseket.
- **self.Weather(self.input.text())**
  - A Weather metódus meghívása az új városra, hogy lekérje és megjelenítse a hozzá tartozó időjárási adatokat.
- **for x in self.hoFokok:**
- **self.weather.addItem(x)**
  - Az új városhoz kapcsolódó hőmérsékleti adatokat hozzáadja a listához (self.hoFokok).
- **if self.input.text() in self.config.get("Előz/Fav").get("Kedvencek"):**
- **self.kedvenc.setText("★")**
- **else:**
- **self.kedvenc.setText("☆")**
  - Ha az új város szerepel a kedvencek között, akkor a kedvenc gombot csillaggal (★) jelöli, különben üres csillaggal (☆).
- **self.TextBox()**
  - A TextBox metódus meghívása, ami a szöveges információt frissít az UI-ban.

```
def Weather(self, varos, hours=12):
```

- **url =**  
`f"http://api.openweathermap.org/data/2.5/forecast?q={varos}&appid={API_KEY}&units=metric&lang={self.config.get("Beállítások").get("Nyelv")}"`
- **response = requests.get(url)**
  - URL felépítése: Az URL-ben szereplő q={varos} paraméter a keresett várost adja meg. Az appid={API\_KEY} az API kulcsot, míg a units=metric azt jelenti, hogy a hőmérsékletet Celsius-ban kapjuk. A lang paraméter az időjárás leírásának nyelvét határozza meg, amelyet a beállításokban tárolunk.
- **if response.status\_code == 200:**
- **data = response.json()**
- **forecast\_list = data["list"]**
  - API válasz: Az API válasza egy JSON formátumú adat, amely többek között tartalmazza az előrejelzés adatait (pl. hőmérséklet, szélsebesség, időjárási leírás).
- **if forecast\_list:**
- **condition\_nyelv = self.config.get("Beállítások").get("Nyelv")**

- A válaszban található első előrejelzést használjuk arra, hogy meghatározzuk az aktuális időjárás állapotát, amely befolyásolja a karakterek hangulatát.
- **description = forecast\_list[0]["weather"][0]["description"].lower()**
  - Leírás kinyerése: Az description tartalmazza az időjárás típusát, mint például "clear sky", "clouds", "rain", stb. Ezt a leírást a kód a következő lépésben az időjárás típusának megfelelő karakterek hangulatának beállítására használja.
- **if condition\_nyelv == "hu":**
- **elif condition\_nyelv == "en":**
- **...**
  - Leírás nyelv szerint: Az időjárás leírását a beállított nyelv szerint kezeljük, és az állapotot ennek megfelelően frissítjük.
  - Az időjárás leírásától függően a karakterek hangulata változik. Ez a rész az időjárás leírásának elemzésén alapul.
  - Nyelv szerint specifikus állapotok:
  - A következő rész kódok felelnek az egyes időjárás típusoknak megfelelő karakterhangulatok beállításáért.
- **if condition\_nyelv == "hu":**
- **if "derült ég" in description or "felhő" in description:**
- **weather\_status = "base"**
- **elif "eső" in description or "hó" in description:**
- **weather\_status = "sad"**
- **elif "vihar" in description:**
- **weather\_status = "scared"**
- **...**
  - Ez az elágazás minden nyelven külön-külön kezeli az időjárás leírást, és meghatározza a karakterek hangulatát:
  - "base": Derült égbolt, felhős idő
  - "sad": Eső, hó, borongós idő
  - "scared": Vihar, köd, füst, tornádó, stb.
  - A nyelvi ellenőrzések az if elágazásokban találhatók, így például a magyar nyelvű leírások más szavakat használnak (pl. "derült ég" a "clear sky"-val szemben).
  - A metódus egy előrejelzést is lekér a következő 12 órára (default érték, amely a hours paraméterből származik).
- **for item in forecast\_list[:hours // 3]:**
- **dt = datetime.fromtimestamp(item["dt"]).strftime("%H:%M")**
- **temp = item["main"]["temp"]**
- **description = item["weather"][0]["description"]**

- Előrejelzés feldolgozása: Az API válaszában több időpont is szerepel (háromórás előrejelzés), és az első 12 órát (vagy a kívánt órát, ha más értéket adunk meg) feldolgozzuk. A ciklus minden egyes előrejelzéshez kiolvassa az időpontot (dt), a hőmérsékletet (temp), és az időjárás leírást.
- Formázás: Az időpontot olvasható formátumban (%H:%M) jelenítjük meg, és a hőmérsékletet Celsius vagy Fahrenheit egységben.
- A kód hibát kezel akkor, ha a város nem található vagy ha API probléma lép fel.
- **else:**
- **print(f"✖ Hiba: Város nem található vagy API probléma  
{response.status\_code}"")**
- **self.weather.clear()**
- **self.weather.addItem("A település nem található!")**
- **weather\_status = "error"**
  - API válasz hiba: Ha a válasz státuszkódja nem 200 (sikeres), akkor a felhasználó értesítést kap a problémáról.
  - Város nem található: Ha a keresett város nem található, akkor egy hibaüzenet jelenik meg, és a karakterek állapota is hibára változik.
- **except requests.exceptions.ConnectionError:**
- **print("✖ Hiba: Nincs internetkapcsolat!")**
- **self.weather.clear()**
- **self.weather.addItem("Nincs internetkapcsolat!")**
- **weather\_status = "error"**
- **self.dialog.setText("Nincs internetkapcsolat!")**
  - Nincs internetkapcsolat: Ha a kérést nem lehet végrehajtani (pl. a felhasználónak nincs internetkapcsolata), a program figyelmezteti őt, és a karakterek állapota ismét hibára változik.

### **def FavSwitchButtonRightClicked(self):**

- **self.FavSwitchLeftButton.setEnabled(True)**
  - Ez a sor engedélyezi (bekapcsolja) a bal oldali "Kedvenc" váltó gombot (FavSwitchLeftButton), amint a jobb oldali gombot (a FavSwitchRightButton) rákattintják.
- **if self.weather.item(0).text() not in  
self.config.get("Előz/Fav").get("Kedvencek"):**
- **self.input.setText(self.config.get("Előz/Fav").get("Kedvencek")[0])**
- **self.keresesClicked()**
  - Első feltétel: Ha az aktuálisan megjelenített város nincs benne a kedvencek listájában (self.weather.item(0).text()), akkor az első kedvenc város neve kerül be az input mezőbe (self.input.setText), majd

végrehajtódik a keresési funkció (self.keresesClicked), ami frissíti az időjárási adatokat a keresett városra.

- *Mi történik, ha nincs a város a kedvencek között?*: A kedvencek listájából az első elem kerül a keresett városként az input mezőbe, és új keresést indítunk.
- **else:**
- **a = self.config.get("Előz/Fav").get("Kedvencek")**
- **for x in range(len(a)):**
- **if a[x] == self.input.text() and x + 1 < len(a):**
- **self.input.setText(a[x+1])**
- **self.keresesClicked()**
- **break**
  - Második feltétel: Ha a város már szerepel a kedvencek között, akkor az aktuális város helyett a következő kedvenc város lesz beállítva. A for ciklus végigmegy a kedvencek listáján.
  - Ha a város megtalálható a kedvencek között ( $a[x] == self.input.text()$ ), és van következő város ( $x + 1 < len(a)$ ), akkor az input mezőbe a következő város neve kerül. Ezután új keresést indítunk a keresesClicked metódussal.
- **if self.weather.item(0).text() == a[-1]:**
- **self.FavSwitchRightButton.setEnabled(False)**
  - Harmadik feltétel: Ha az aktuális város már a kedvencek listájának utolsó eleme ( $a[-1]$ ), akkor a jobb oldali kedvenc váltó gomb le van tiltva (FavSwitchRightButton.setEnabled(False)), mert nincs több kedvenc, amit mutathatnánk.
- **print("FavSwitchButtonRightClicked called")**
- **print(f"Kedvencek: {self.config.get('Előz/Fav').get('Kedvencek')}")**
  - A metódus végén két print utasítással debug üzeneteket íratunk ki a konzolra:
  - "FavSwitchButtonRightClicked called": Ez jelzi, hogy a gomb megnyomásával a metódus futott le.
  - A második üzenet kiírja az aktuális kedvencek listáját, hogy lássuk, mi szerepel benne.

**def FavSwitchButtonLeftClicked(self):**

- Ugyan az, mint a FavSwitchButtonRightClicked, annyi különbséggel, hogy:
- Ha a város megtalálható a kedvencek között és van következő város ( $x - 1 > len(a)$ ), akkor az input mezőbe a következő város neve kerül.

## Alkalmazás futtatása:

- **splash = splashScreen()**
    - A splash screen egy Qsplash screen legyen.
  - **if not os.path.exists(„License”):** - Ha nem létezik a licensz:
  - **print("X License fájl nem található!")**
  - **piracy.show()** – A piracy screen jelenik meg.
  - **else:** - Ha meg van a licensz:
  - **splash.show()**
    - jelenjen meg a Splash
  - **app.processEvents()**
    - Töltsse be az alkalmazást egészben
  - **splash.setProgress()**
    - A folyamatát mutassa.
  - **splash.finish(None)**
    - Ha befejezte a splash a folyamatot, ne csináljon semmit.
  - **window = MainWindow()**
    - Az ablak egy QMainWindow
  - **window.setVisible(True)** – Megjeleníti az ablakot
- 
- **app.exec()**
    - Ha a folyamat megáll, fejezze be a futást a háttérben.