

Záródolgozat

Rebman Fanni Viktória

Budapest

2020

Budapesti Komplex Szakképzési Centrum
Weiss Manfréd Szakgimnáziuma,
Szakközépiskolája és Kollégiuma

Thesis Quest

Témavezető:

Englert Ervin

Készítette:

Rebman Fanni Viktória
Szoftverfejlesztő OKJ –
esti tagozat

Budapest

2020

TARTALOMJEGYZÉK

1. BEVEZETÉS	5
1.1. Köszönetnyilvánítás.....	5
1.2. Témaválasztás	5
1.3. Témaválasztás indoklása	5
2. FELHASZNÁLÓI DOKUMENTÁCIÓ	6
2.1. Rendszerkövetelmények.....	6
2.2. A program elindítása.....	6
2.2.1. Program indítása exe fájlal	6
2.2.2. Program elindítása fejlesztőkörnyezetből.....	6
2.3. Főmenü.....	7
2.3.1. Start gomb	8
2.3.2. High Scores gomb.....	9
2.3.3. Exit gomb	10
2.4. A játék	10
2.4.1. Játéktér	10
2.4.2 Szövegdoboz	11
2.4.3. Irányítás	12
2.4.4. Küldetések és Nem Játszható Karakterek	12
2.4.5. Vége képernyő	12
2.5. Dicsőségtábla	13
3. FEJLESZTŐI DOKUMENTÁCIÓ	14
3.1. Fejlesztői környezet.....	14
3.2. Fájlok, osztályok, főprogramok, alprogramok, változók.....	14
3.2.1. run.py	15
3.2.2. main_menu.py	16
3.2.3. name.py	17
3.2.4. game.py	17
3.2.5. player_class.py	19
3.2.6. npc1.py, npc2.py, npc3.py, npc4.py, npc5.py	20
3.2.7. hitbox.py	21
3.2.8. gg.py	22
3.2.9. high_score.py	22
3.3. Fontosabb kódrészletek.....	23
3.3.1. Gombok.....	23
3.3.2. Felhasználónév input.....	23
3.3.3. Karakter mozgása, animációja	24
3.3.4. Küldetések és NJK-k.....	28

3.3.5.	Dicsőséglista	30
3.4.	Grafikus elemek és képek.....	31
3.4.1.	Főmenü háttérkép.....	31
3.4.2.	Játéktér kialakítása.....	31
3.4.3.	Szövegdoboz	32
3.4.4.	Játékos karakter	32
3.4.5.	Nem Játékos Karakterek	32
3.5.	Betűtípusok.....	33
3.5.1.	Menüben felhasznált betűtípus	33
3.9.2.	Játékban felhasznált betűtípus	33
3.10.	Teszt dokumentáció.....	34
4.	FEJLESZTÉSI LEHETŐSÉGEK.....	34
5.	IRODALOMJEGYZÉK	34

1. Bevezetés

1.1. Köszönetnyilvánítás

Szeretném megköszönni Englert Ervin tanár úrnak, hogy segített a program elkészítésében. Az ő segítsége nélkül nem jöhetett volna létre a játék.

Továbbá szeretném megköszönni Kiss Viktornak, aki szintén végig támogatott az egész OKJ képzés alatt.

1.2. Témaválasztás

A záródolgozatom témájának választása egy pixelgrafikus kalandjátékra esett, melyben a játékos egy sor küldetés elvégzésével „nyerheti meg” a játékot.

A játék története szerint a karakter elveszítette a záródolgozatát, melynek lapjai egy – egy küldetés elvégzésével szerezhetőek vissza. Ha sikerült az összes küldetést megcsinálnia, a záródolgozatát leadhatja, ezzel megnyerve a játékot.

1.3. Témaválasztás indoklása

A pixelgrafikus játékok mindig is fontos helyet töltöttek be a szívemben, mivel gyerekkoromat idézik. Sajnos mára a kereslet az ilyesfajta játékok iránt csökkent, ezért szerettem volna ezzel a játékkal megalapozni a későbbiekben tervezett szintén pixelgrafikus játékaimat. A játékom továbbá egy kedves, nyugodt hangulatú program, ami kikapcsolódást nyújt, nincs benne a mai agresszív játékok alapvető eleme: a harc, az embernek csupán figyelnie kell, és gondolkodnia a küldetések megoldásához. További indok, hogy szeretnék később hasonló játékokat fejleszteni a barátaim számára is.

2. Felhasználói dokumentáció

2.1. Rendszerkövetelmények

Minimális rendszerkövetelmény:

- **CPU:** Intel Core i3 processzor
- **RAM:** 4 GB
- **HDD:** 1 GB
- **VGA:** integrált videokártya 64 MB
- **Operációs rendszer:** Windows 7

Optimális rendszerkövetelmény:

- **CPU:** Intel Core i5 processzor vagy jobb
- **RAM:** 8 GB
- **HDD:** 2 GB
- **VGA:** integrált videokártya 4 GB
- **Operációs rendszer:** Windows 10

2.2. A program elindítása

A programot a <https://drive.google.com/open?id=1jqPG-KUnxn8r1OZCprWFC0x90430u5W> linkre kattintva érheti el. Amennyiben a programot exe fájlal szeretné elindítani, akkor a *ThesisQuest_exe.zip* fájlt töltsse le, csomagolja ki, ellenkező esetben a *ThesisQuest.zip* fájllal járjon el hasonló módon.

2.2.1. Program indítása exe fájlal

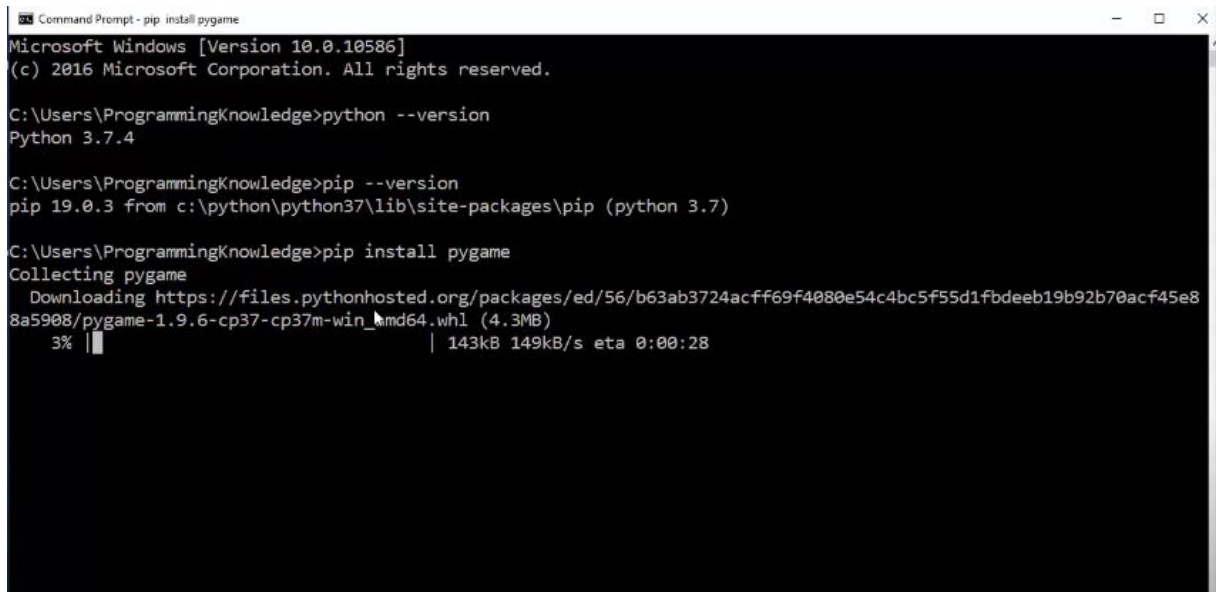
A program elindításához a ThesisQuest nevű mappában található run.exe fájlra az egér bal gombjával duplán kattintson.

2.2.2. Program elindítása fejlesztőkörnyezetből

A programot egy python fejlesztőkörnyezetben is futtathatja. Ez a Linux, továbbá az IOS operációs rendszerrel rendelkezők számára is lehetővé teszi a program futtatását.

Amennyiben fejlesztőkörnyezet által szeretné futtatni a programot, a Thonny fejlesztőkörnyezet ajánlott a játék futtatására. A Thonny-t letöltheti a <https://thonny.org/> weboldáról, majd az oldalon található telepítési útmutatókkal, telepítheti a szoftvert. Ezek

után a <https://www.pygame.org/wiki/GettingStarted> oldalon található útmutatóval installálja a Pygame multimédiás könyvtárat.



```
Command Prompt - pip install pygame
Microsoft Windows [Version 10.0.10586]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\ProgrammingKnowledge>python --version
Python 3.7.4

C:\Users\ProgrammingKnowledge>pip --version
pip 19.0.3 from c:\python\python37\lib\site-packages\pip (python 3.7)

C:\Users\ProgrammingKnowledge>pip install pygame
Collecting pygame
  Downloading https://files.pythonhosted.org/packages/ed/56/b63ab3724acff69f4080e54c4bc5f55d1fbdeeb19b92b70acf45e88a5908/pygame-1.9.6-cp37-cp37m-win_amd64.whl (4.3MB)
    3% | 143kB 149kB/s eta 0:00:28
```

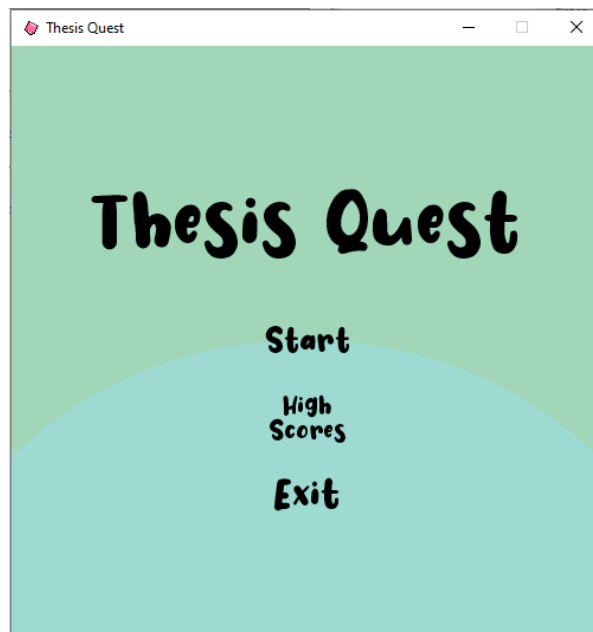
Windows10 operációs rendszer esetében a Pygame installálása

A telepítést követően, ha a Thonny-t megnyitja, majd abból megnyitja a ThesisQuest mappa *run.py* fájlját, majd az *F5* billentyűt lenyomja, a program elindul.

2.3. Főmenü

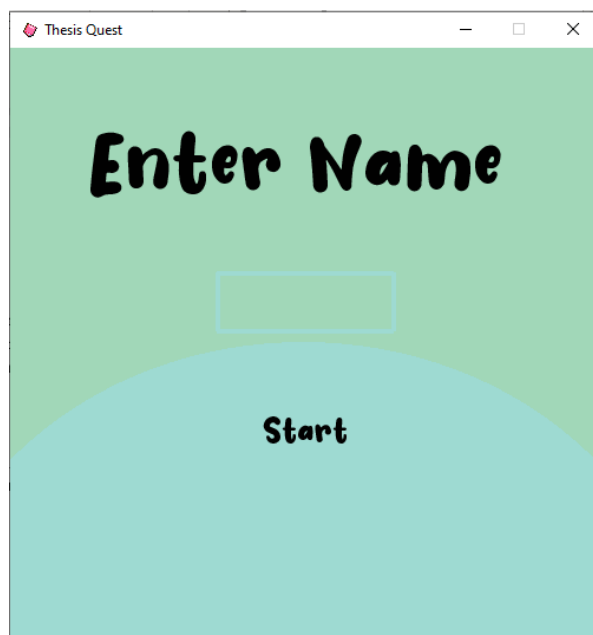
A program indításakor megnyílik egy ablak, amelyben rögtön a játék főmenüjét láthatjuk. Felül látható a játék ikonja, valamint mellette a címsorban a program neve. Alul a főmenüben található ismét a játék címe, valamint három gomb, melyekre az egérrel kattintva lesznek elérhetők az adott funkciók, illetve menüpontok:

- Start
- High Scores
- Exit

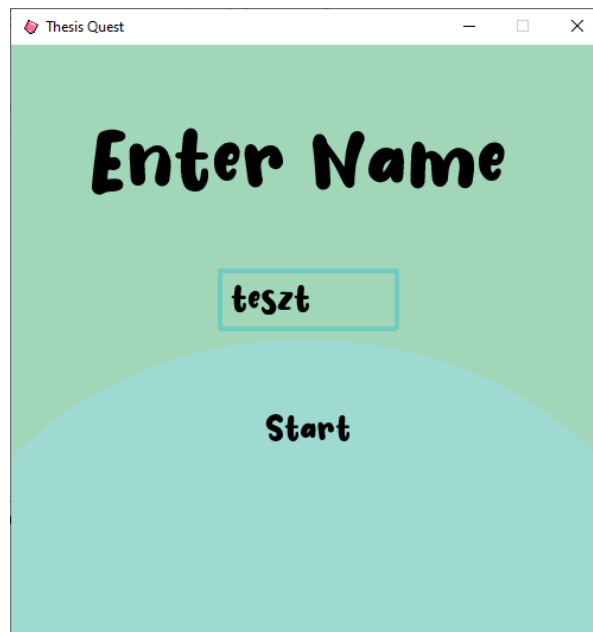


2.3.1. Start gomb

A start gombra való kattintással a program a játék indítását fogja végrehajtani, de ezt megelőzően először a felhasználónak egy maximum 8 karakter hosszúságú nevet kell megadnia, mellyel felkerülhet a dicsőséglistára, a megoldott küldetéseinek számával együtt.



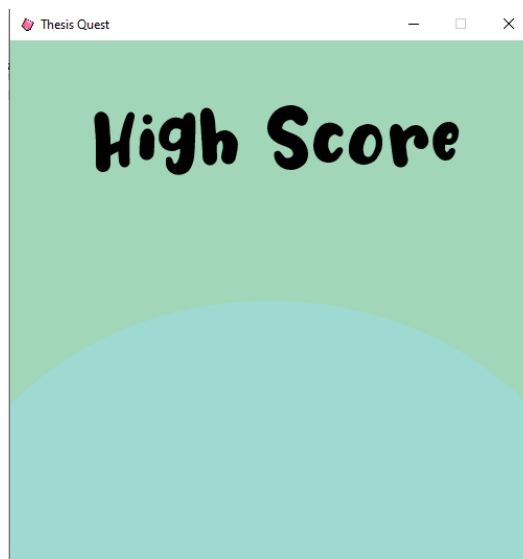
A játékosnév megadásához először bele kell kattintani a név megadására szolgáló négyzetbe az egérrel. Akkor tud bele írni, ha a négyzet kerete világosról sötét színűre változik



A felhasználónév megadása után ismét egy START nevezetű gombra kattintva, a játék immár ténylegesen elindul.

2.3.2. High Scores gomb

A HIGH SCORE gombra kattintva megtekintheti a játékkal játszottak dicsőséglistáját. A lista tartalmazza a játék kezdete előtt megadott felhasználóneveket, illetve a játék során elvégzett küldetések számát.

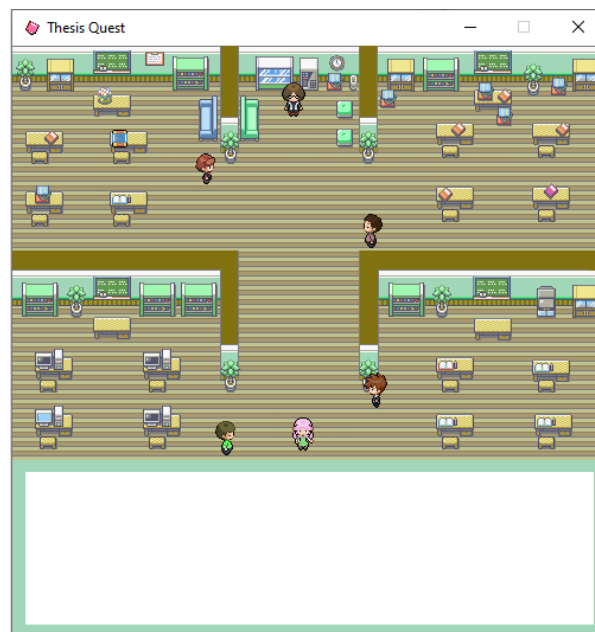


2.3.3. Exit gomb

Az EXIT gombra kattintva kiléphet a programból.

2.4. A játék

A játék elindulásakor az ablak felső részén kap helyet maga a játéktér, alul pedig egy általam „szövegdoboz” néven futó rész, mely a játék során kiírja a játékban helyet kapó párbeszédek és egyéb szövegeket.



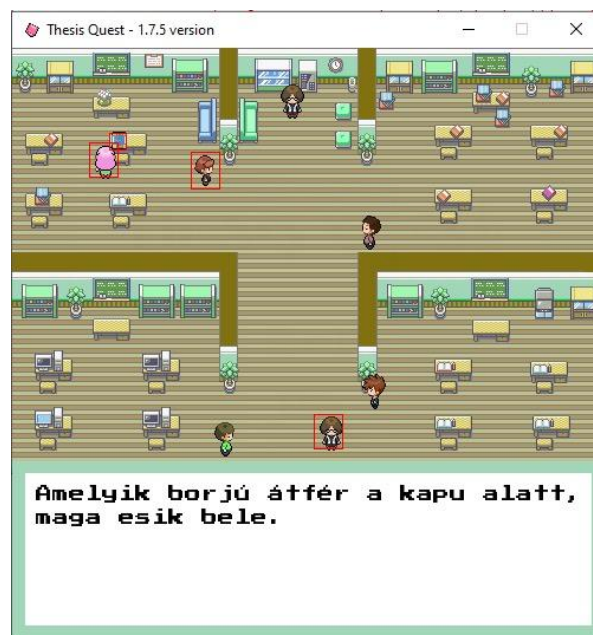
2.4.1. Játéktér

A játéktér egy iskolának ad otthont, melyben különböző helyiségek vannak, mindegyikhez egy - egy NJK (nem játszható karakter). Mindegyik NJK a maga helyiségénél áll, ezzel reprezentálva azt is, hogy hol kell megcsinálni az adott küldetéseket.



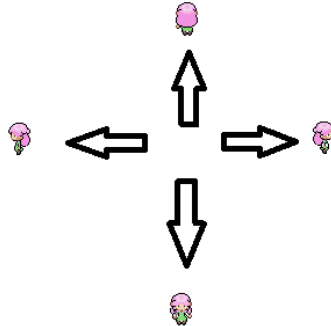
2.4.2 Szövegdoboz

A szövegdobozban jelenik meg minden, amit az NJK-ek mondanak, illetve a küldetésekhez tartozó tárgyak szövegei, ha a játékos elég közel van az NJK-hoz, illetve a tárgyhoz



2.4.3. Irányítás

A játékost a billentyűzeten található nyíl gombokkal lehet irányítani. A karakter az adott billentyű lenyomásával, az ahhoz megfelelő irányba fog elmozdulni. A játék menete alatt más gombok használatára vagy az egérre nincs szükség. A karakter csak egyetlen egy tempóban tud haladni.



2.4.4. Küldetések és Nem Játoszható Karakterek

A játékban öt küldetés van, ezeket csak egy adott sorrendben lehet elvégezni, ezáltal csak akkor válik elérhetővé a következő küldetés, ha az előzőt már teljesítette a játékos. A küldetések alatt az NJK-k elmondják a feladatot, melyet a karakternek teljesítenie kell, hogy leadhatja a záródolgozatát. A játék során négy normális küldetés van, az ötödik, egyben utolsó küldetés akkor válik aktívvá, ha a játékosnak sikerül az összes küldetést megcsinálnia. Itt a záródolgozatot kell leadni.

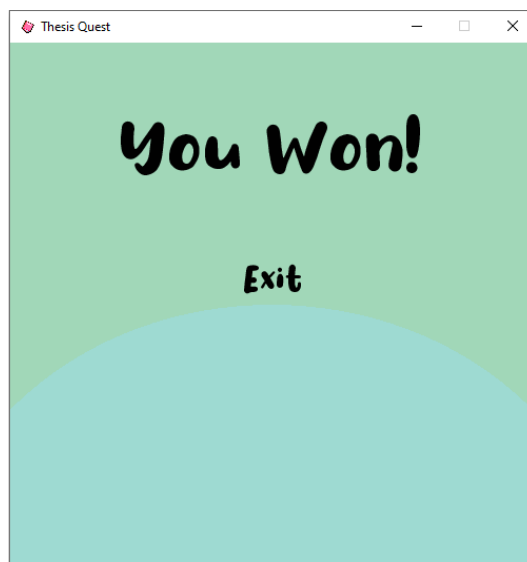
A küldetések elvégzéséhez a karakternek megfelelő távolságra kell lennie az NJK-tól, valamint a tárgytól, hogy haladjon a küldetésekkel, és megjelenítse a hozzájuk tartozó szöveget. Azok az NJK-k, akiknek a küldetése nem aktív, vagy már elkészült, jelezni fogják, hogy a jelenleg aktív küldetést csinálja.

A küldetések működése rendkívül egyszerű. A karakter a soron következő NJK-val kell beszélnie, aktiválja a küldetést, majd, ha megfelelő tárgy is megjeleníti a szövegét, akkor már csak vissza kell menni az adott NJK-hoz, hogy befejezze a küldetést.



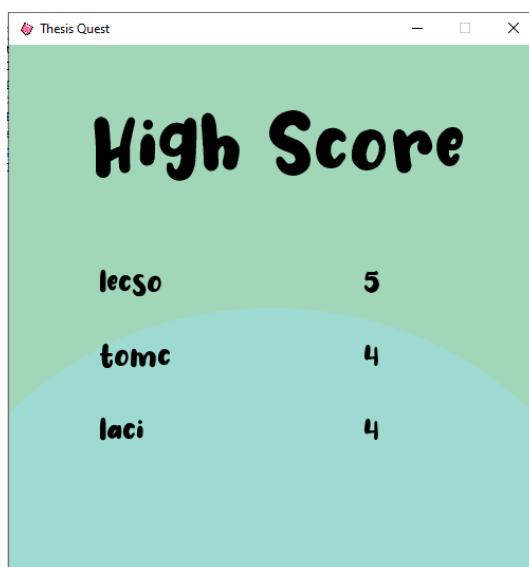
2.4.5. Vége képernyő

Ha sikerül az összes küldetés, akkor a játéknak vége, a felhasználó nyert, majd megjelenik a vége képernyő. Itt található egy felirat, mely szerint a játékos nyert, valamint egy EXIT feliratú gomb, ami a főmenüben lévőhöz hasonlóan működik, tehát ha az egérrel rákattint a felhasználó, akkor kilép a programból.



2.5. Dicsőségtábla

A dicsőségtáblát a főmenüben a HIGH SCORES gombra kattintva válik elérhetővé. Az ablak tetején helyezkedik el a cím szöveg, alatt pedig maga a dicsőségtábla, amely tartalmazza azoknak a felhasználóknak a nevét, valamint teljesített küldetéseinek számát rangsorolva. A dicsőségtábla a három legjobb játékos adatait tartalmazza, amennyiben ennél kevesebb játékos játszott, a legjobb kettő, egy játékos eredményeit mutatja, vagy egyet se, ha még egy játékos se ért el eredményt.



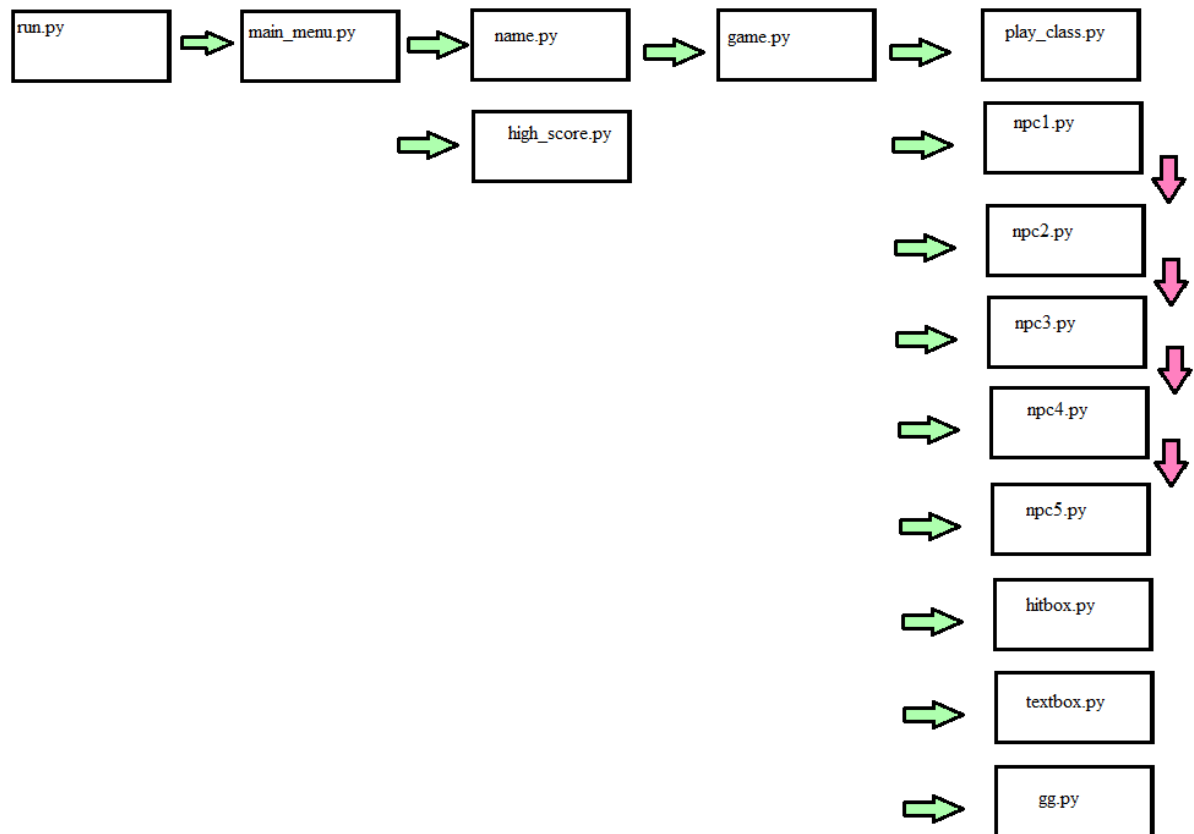
3. Fejlesztői dokumentáció

3.1. Fejlesztői környezet

Python 3.8.0	A programomat Python nyelven írtam
Pygame	Egy multimédiás könyvtár a Pythonhoz.
IDLE	A fejlesztői környezet, amiben írtam a programot
Paint	Egy ingyenes képszerkesztő szoftver, amit a programomban lévő képek és egyéb grafikus elemek szerkesztéséhez, rajzolásához használtam
Adobe Photoshop 2020	Egy képszerkesztő szoftver, amit szintén a képek és egyéb grafikus elemek szerkesztéséhez, rajzolásához használtam

3.2. Fájlok, osztályok, főprogramok, alprogramok, változók

A játék több fájl által működik. A fájlokban lévő osztályok, fő- és alprogramok egymásba importálása lehetővé teszi a játék zavartalan működését, miközben elszeparálódnak az egyes részek.



3.2.1. run.py

A run fájl használatát és annak ötletét egy internetes tutorialvideo-ból vettem, és a játék GitHubra felöltött változatát használtam fel, és azt alakítottam kicsit.

<https://github.com/techwithtim/Tower-Defense-Game/blob/master/run.py>

A *run.py* a játék alapköve, amelyben inicializálódik a *screen* nevű változó, azaz az ablak, amelyet az összes többi osztályba fel lesz használva, továbbá itt állítódik be a program ikonképe, és az ablakban megjelenő cím, így a továbbiakban ezeket sem kell beállítani.

Az alapvető inicializálások után a *menu* almappából a *main_menu.py* fájlból beimportálja a *MainMenu* osztályt, azt *main_menu* néven inicializálja, oda adja neki a *screen* változót, végül futtatja az osztály *run* nevű programját.

```

import pygame

if __name__ == "__main__":
    pygame.init()
    screen = pygame.display.set_mode((500, 500))

    pygame.display.set_caption("Thesis Quest")

    iconpic = pygame.image.load('icon.png')
    pygame.display.set_icon(iconpic)

    from menu.main_menu import MainMenu
    main_menu = MainMenu(screen)
    main_menu.run()
  
```

3.2.2. main_menu.py

A *run.py* futtatása meghívja a *main_menu.py* fájlt, amelyben a játék főmenüjének részei találhatóak: egy *MainMenu* nevű osztály, amiben inicializálódnak a *MainMenuban* használt változók. Ezek a változók az ablak szélessége (*self.width*), magassága (*self.height*), a képernyőn megjelenő háttérkép (*self.bg*), valamint cím (*self.title*), a gombokhoz tartozó képek (*self.start_button*, *self.hs_button*, *self.exit_button*), négyzetek (*self.button_s*, *self.button_hs*, *self.button_e*), végül maga a képernyő (*screen*), amely a *run.py* fájlban lett korábban inicializálva, de a *MainMenu* meghívásakor ezt tovább adta.

```
class MainMenu:
    def __init__(self, screen):

        self.width = 500
        self.height = 500
        self.title = pygame.image.load("menu/title.png")
        self.start_button = pygame.image.load("menu/button_start.png")
        self.hs_button = pygame.image.load("menu/button_hs.png")
        self.exit_button = pygame.image.load("menu/button_exit.png")
        self.b_width = 100 # b = button
        self.b_height = 50
        self.bg = pygame.image.load("menu/main_menu_bg.png")
        self.screen = screen
        # x = ablak szélességének fele - a go szélességének fele (középen) y = szintén közép, w, h 100x50
        self.button_s = pygame.Rect(self.width/2 - self.b_width/2, self.height/2 - self.b_height/2, self.b_width, self.b_height)
        self.button_hs = pygame.Rect(self.width/2 - self.b_width/2, self.button_s[1] + 65, self.b_width, self.b_height)
        self.button_e = pygame.Rect(self.width/2 - self.b_width/2, self.button_hs[1] + 65, self.b_width, self.b_height)
```

A *run* a főmenühöz tartozó főprogram, amelyben egy végtelen ciklus van, amely addig megy, amíg a felhasználó ki nem lép a programból, akkor a *while run*, *run* nevű logikai változója HAMIS értéket vesz fel, így nem fut tovább a ciklus.

```
def run(self):
    run = True

    while run:

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
```

A *Draw* egy olyan alprogram, amelyben felsorolódnak a képernyőre kirajzolandó képek, szövegek, egyéb síkidomok, mint például a gombokhoz használatos négyzetek. Ennek végén található a *pygame.display.update* nevű parancs, mely frissíti a képernyőt, ezzel megjelenítve a képeket. Ezeket a *run* programban meghívva lehet látni.

```
def Draw(self):
    self.screen.blit(self.bg, (0,0))
    self.screen.blit(self.title, (self.width / 2 - 400 / 2, 100))
    self.screen.blit(self.start_button, (self.button_s[0],self.button_s[1]))
    self.screen.blit(self.hs_button, (self.button_hs[0],self.button_hs[1]))
    self.screen.blit(self.exit_button, (self.button_e[0],self.button_e[1]))

    pygame.display.update()
```


3.2.3. name.py

A *MainMenu* osztály *run* programjában van egy eventhez kötött egérlenyomás, mely meghívja a *game* almappa *name.py* fájl *Main* nevű osztályát és annak összes eljárását.

A *Main* osztályban inicializálódik a képernyő(*screen*), annak szélessége(*width*), magassága(*height*), az inputboxhoz tartozó színek, egy, amikor aktív az inputbox(*color_active*) és egy, amikor inaktív(*color_inactive*). A képernyő háttere(*bg*), a cím(*title*), egy START gomb képe(*start_button*), és a hozzá tartozó négyzet(*button_s*), és végül egy betűtípus(*font*).

```
class Main:
    def __init__(self, screen):
        self.screen = screen
        self.width = 500
        self.height = 500
        self.color_inactive = (150, 210, 210)
        self.color_active = (115, 202, 191)
        self.font = pygame.font.Font('font/wash_your_hand/Wash Your Hand.ttf', 32)
        self.bg = pygame.image.load("menu/main_menu_bg.png")
        self.title = pygame.image.load("game/name.png")
        self.start_button = pygame.image.load("menu/button_start.png")
        self.button_s = pygame.Rect(self.width/2 - 100/2, self.height/2 + 50, 100, 50)

        self.input_box = InputBox(self.width/2 - 150/2, self.height/2 - 60, 150, 50, self.font, self.color_active, self.color_inactive, self.screen)
```

A *Main* osztály mellett helyet kapott még egy osztály, aminek a neve *InputBox*, ebben inicializálódik, és határozódik meg az az inputbox, amibe a felhasználó megadja a nevét. Itt meg lett határozva egy négyzet(*textbox*), a képernyő(*screen*), az aktív- és inaktív színek, a betűtípus, egy *text* nevű üres string, és egy *txt_surface*, amely egy lerenderelt szöveg az adott betűtípussal, és egy *active* logikai változó, ami meghatározza, hogy az inputbox aktív-e, tehát a felhasználó tud bele írni, vagy sem.

```
class InputBox:

    def __init__(self, x, y, w, h, font, color_active, color_inactive, screen):
        self.textbox = pygame.Rect(x, y, w, h)
        self.font = font
        self.color_active = color_active
        self.color_inactive = color_inactive
        self.screen = screen
        self.color = self.color_inactive
        self.text = ""
        self.txt_surface = self.font.render(self.text, True, (0,0,0))
```

A *Handle_event* alprogram kezeli az inputboxba a szövegbevitelt. A *Draw* alprogram rajzolja ki az inputboxot és a rerenderelt szöveget.

A *Main run* programjában van egy végtelen ciklus, ahol a főmenühöz hasonlóan kezeli a gomb működését, de csak akkor nyomhat rá a felhasználó, ha a felhasználónevet tartalmazó *text* változó nem üres. Ekkor a *game.py* fájl *Game* nevű osztályát inicializálja, és megadja neki a *screen*, és az *InputBox text* változóját, azaz a játékos nevét. Továbbá itt hívódik meg a *Main Draw* programja, és az *InputBox* alprogramjai.

3.2.4. game.py

A *game.py* fájlban van a *Game* osztály, ahol inicializálódik a játékhoz szükséges változók és a többi osztály, amelyek szükségesek a játék működéséhez. A képernyő(*screen*), annak

szélessége(*width*), magassága(*height*), a játékon belül használt betűtípus(*font*), az óra(*clock*), egy logikai változó, mely a karakter mozgásával és fizikájában kap szerepet(*collide*), egy int változó, ami a teljesített küldetéseket számolja(*score*), és egy olyan logikai változó, ami azt vizsgálja, hogy a játékos megnyerte e a játékot vagy sem(*win*). Ezek alatt jönnek a beimportált osztályok inicializálása, hogy fel lehessen őket használni, a Player(*player*), a HitBox(*hitbox*), a Textbox(*textbox*), NPC1(*npc1*), NPC2(*npc2*), NPC3(*npc3*), NPC4(*npc4*), NPC5(*npc5*).

```
class Game:
    def __init__(self, screen, text):
        self.width = 500
        self.height = 500
        self.bg = pygame.image.load("game/background.png")
        self.screen = screen
        self.font = pygame.font.Font('font/press_start_2p/PressStart2P.ttf', 14)
        self.clock = pygame.time.Clock()
        self.collide = False

        self.score = 0
        self.name = text

        self.win = False

    # class inicializálás
    self.player = Player(self.width/2 - 15, 350 - 32 - 5, 32, 32) # x y width height
    self.hitbox = HitBox(self.player, self.screen, self.collide)
    self.textbox = Textbox(0, 350, 500, 150, self.screen)
    self.npc1 = NPC1(168, 316, 25, 32, self.font, self.player, self.screen)
    self.npc2 = NPC2(293, 275, 25, 32, self.font, self.player, self.screen, self.npc1)
    self.npc3 = NPC3(293, 141, 25, 32, self.font, self.player, self.screen, self.npc2)
    self.npc4 = NPC4(151, 88, 25, 32, self.font, self.player, self.screen, self.npc3)
    self.npc5 = NPC5(223, 31, 25, 32, self.font, self.player, self.screen, self.npc3)
```

A *run* program meghívja az összes olyan programot, amit felhasznál a játék, a képernyőre rajzolás, a küldetések, a teljesített küldetésének számának számolása, a karakter mozgása és fizikai hatása a környezetere, valamint annak eldöntése, hogy a játékos megnyerte e a játékot, vagy sem.

A *Draw*-ban van felsorolva minden, amit ki kell rajzolni a képernyőre, a háttérkép, a karakter, a textbox a képernyő aljára, az NJK-khoz tartozó kiírandó szövegek, végül frissíti a képernyőt, hogy ezeket látni lehessen a program futtatásakor.

```
def Draw(self):
    self.screen.blit(self.bg, (0, 0))
    self.player.Draw(self.screen)
    self.hitbox.Draw(self.screen)
    self.textbox.Draw(self.screen)

    self.npc1.Draw(self.screen)
    self.npc2.Draw(self.screen)
    self.npc3.Draw(self.screen)
    self.npc4.Draw(self.screen)
    self.npc5.Draw(self.screen)

    pygame.display.update()
```

A *Score* alprogram számolj a teljesített küldetések számát. Ha egy küldetés elkészül, akkor a *score* értéke változik, vagyis eggyel növekszik. Ha az utolsó küldetést is teljesítette a játékos, akkor a *win* változó IGAZ értéket vesz fel.

```

def Score(self): # a self.score += 1 re kvázi örökre pörgeti a számokat
    if self.npc1.done == True:
        self.score = 1
        #print(self.score)
    if self.npc2.done == True:
        self.score = 2
        #print(self.score)
    if self.npc3.done == True:
        self.score = 3
        #print(self.score)
    if self.npc4.done == True:
        self.score = 4
        #print(self.score)
    if self.npc5.done == True:
        self.score = 5
        self.win = True
        # print(self.score)

```

A *Win* alprogram hívja meg a játék vége képernyőjét. Ha a *win* IGAZ, akkor a *game* almappából a *gg.py* nevű fájl *GG* osztályát beimportálja, inicialja azt *gg* néven, majd annak főprogramját, a *run*-t, futtatja, végül frissíti a képernyőt.

```

def Win(self):
    if self.win == True:

        self.Write()

        from game.gg import GG
        gg = GG(self.screen, self.score, self.name)
        gg.run()

        pygame.display.update

    return self.score

```

A *Write*-ban fájlkezelés történik. A *game* almappába, ha nem létezik kreál, ha létezik hozzáfűz a *scoreboard.txt* nevű fájlhoz, melyet *f* néven használ a program a későbbiekben. A játékosnevet, valamint a teljesített küldetések számát írja bele a fájlba. A fájlba írás csak a program bezárásakor, pontosabban közvetlenül azelőtt történik, különben, ha a többi alprogrammal együtt lenne meghívva, akkor minden egyes lefutáskor/frissítéskor belekerülne a játékos neve.

```

def Write(self):
    with open ('game/scoreboard.txt', 'a') as f:
        f.write(self.name + ';' + str(self.score) + "\n")

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            self.Write()
            run = False

```

3.2.5. player_class.py

A *Player* osztályban vannak a karakterhez tartozó változók, az *x*, *y*, szélesség(*width*), magasság(*height*), a játékos mozgási sebessége(*vel*), irányokat jelölő logikai változók (*left*,

right, up, down), egy *int* típusú változó, ami az animációkat számolja (*walking*), egy olyan logikai változó, ami azt nézni, hogy a karakter egyhelyben áll e, vagy mozog(*standing*), a karakterhez tartozó *hitbox*, ami egy négyzet, és végül négy lista, amelyek a négy irány szerint különíti el a karakter animációihoz tartozó képeket.

A *Draw* alprogram kezeli a játékos animációját, amely a játékon belül megjelenik.

```
class Player(object):
    def __init__(self, x, y, width, height):
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.vel = 2
        self.left = False
        self.right = False
        self.up = False
        self.down = False
        self.walking = 0
        self.standing = True
        self.hitbox = pygame.Rect(self.x + 5, self.y + 20, 15, 10) # x y width height (a hitbox direkt a lehető legkisebb)

        self.WalkLeft = [pygame.image.load('player/L_STAND.png'), pygame.image.load('player/L_WALK1.png'), pygame.image.load('player/L_WALK2.png')]
        self.WalkRight = [pygame.image.load('player/R_STAND.png'), pygame.image.load('player/R_WALK1.png'), pygame.image.load('player/R_WALK2.png')]
        self.WalkUp = [pygame.image.load('player/U_STAND.png'), pygame.image.load('player/U_WALK1.png'), pygame.image.load('player/U_WALK2.png')]
        self.WalkDown = [pygame.image.load('player/D_STAND.png'), pygame.image.load('player/D_WALK1.png'), pygame.image.load('player/D_WALK2.png')]
```

3.2.6. npc1.py, npc2.py, npc3.py, npc4.py, npc5.py

Az összes NJK-nek külön fájlban van személyre szabottan osztály, NPC plusz a hozzájuk rendelt sorszám néven(*NPC1, NPC2, NPC3, NPC4, NPC5*). Az első négynél azonosak a változók is, csak értékekben térhetnek el, az ötödik picit eltérő, hiszen az akkor lesz aktív, ha az összes korábbi küldetés kész van, így vele elég egyszer beszélni, nem kell az előzőkhez hasonlóan egy tárgyat megkeresni, csupán leadni az elvégzett küldetést.

Mindegyikben van egy *x*, *y*, egy *screen*, egy *x* és *y* pont a textboxban(*tb_x, tb_y*), egy szélesség(*width*), egy magasság(*height*), a betűtípus(*font*), egy *activeline* nevű üres string változó, egy *npc_count* nevű integer, a karakter hitboxa(*hitbox*), egy *lines* nevű listában az adott karakter szövegei, a küldetésekhez tartozó tárgy hitboxa(*item*), a tárgy szövege(*itemline*), a küldetés aktívságát jelző logikai változó(*active*), az tárgy aktívságát - (*itemactive*), a küldetés elkészültségét(*done*) jelző logikai változók, valamint egy, ami azt jelzi, hogy a tárgy meg van e(*itemfound*). Az *NPC1* osztályt kivéve, van még egy változó a többi osztályban. Az előtte levő NJK osztályát beimportálja, a *done* változó miatt.

```
class NPC1(object): # Lecso
    def __init__(self, x, y, width, height, font, player, screen):
        self.x = x
        self.y = y
        self.screen = screen
        self.tb_x = 20 # textbox x
        self.tb_y = 370 # textbox y
        self.width = width
        self.height = height
        self.font = font
        self.activeline = ""
        self.player = player
        self.npc_count = 0
        self.hitbox = pygame.Rect(self.x, self.y, self.width, self.height) # hitbox: x y w h
        self.lines = [self.font.render('Lecso quest line', True, (0, 0, 0)),
                      self.font.render('Lecso quest done line', True, (0, 0, 0))]
        self.item = pygame.Rect(18, 306, 24, 18) # hitbox: x y w h
        self.itemline = self.font.render("Lecso item line", True, (0, 0, 0))
        self.active = True
        self.itemactive = False # akkor lesz true ha a npc_count >=1
        self.done = False # nincs kész a küldetés
        self.itemfound = False # meg van e találva az item
```

Az ötödik NJK kicsit másképp működik, mint a többi. Itt nincs szükség küldetés béli tárgyhöz, se ahhoz tartozó egyéb változókra, a szövege is kevesebb.

```

class NPC5(object): # Teacher
    def __init__(self, x, y, width, height, font , player, screen, npc4):
        self.x = x
        self.y = y
        self.screen = screen
        self.tb_x = 20 # textbox x
        self.tb_y = 370 # textbox y
        self.width = width
        self.height = height
        self.font = font
        self.activeline = ""
        self.player = player
        self.npc_count = 0
        self.hitbox = pygame.Rect(self.x, self.y, self.width, self.height) # hitbox: x y w h
        self.lines = [self.font.render('Teacher quest done line', True, (0, 0, 0)),
                      self.font.render('Teacher complete all quest line', True, (0, 0, 0))]
        self.active = False
        self.done = False # nincs kész a küldetés
        self.npc4 = npc4

```

3.2.7. hitbox.py

A hitbox.py fájlban van a HitBox osztály. A hitboxok a játékban fontos helyet foglalnak el, mivel ezek által lesz fizika a játékban. Az osztály használj a Player osztályt(*player*), a *collide* változót a *Game* osztályból, valamint a *screen*. Van egy *hitboxes* nevű lista, ami tartalmazza a játéktérben található összes hitboxot.

A *Collide* program a karakter hitboxa és a *hitboxes* lista elemeit felhasználva eldönti, hogy a *collide* mikor IGAZ, és ezt az IGAZ, vagy éppen HAMIS értéket felhasználva történik meg a játékos mozgása a *Moving* programban.

```

class HitBox: # was npc class

    def __init__(self, player, screen, collide): # playeren + itemen kívül minden más hitbox

        self.player = player
        self.collide = collide
        self.screen = screen
        #self.hitbox = pygame.Rect(x, y, width, height)
        self.hitboxes = [pygame.Rect(82, 72, 32, 14), #0
                          pygame.Rect(10, 72, 32, 14), #1
                          pygame.Rect(10, 122, 32, 14), #2
                          pygame.Rect(82, 122, 32, 14), #3
                          pygame.Rect(357, 65, 32, 14), #4
                          pygame.Rect(357, 118, 32, 14), #5
                          pygame.Rect(437, 118, 32, 14), #6
                          pygame.Rect(437, 65, 32, 14), #7
                          pygame.Rect(67, 42, 32, 14), #8
                          pygame.Rect(388, 35, 32, 14), #9
                          pygame.Rect(18, 258, 32, 14), #10
                          pygame.Rect(18, 307, 32, 14), #11
                          pygame.Rect(110, 258, 32, 14), #12
                          pygame.Rect(110, 307, 32, 14), #13
                          pygame.Rect(390, 228, 32, 14), #14
                          pygame.Rect(355, 263, 32, 14), #15
                          pygame.Rect(355, 311, 32, 14), #16
                          pygame.Rect(437, 263, 32, 14), #17
                          pygame.Rect(437, 311, 32, 14), #18
                          pygame.Rect(67, 229, 32, 14), #19
                          pygame.Rect(175, 0, 15, 100), #20
                          pygame.Rect(292, 0, 15, 100), #21
                          pygame.Rect(175, 172, 15, 120), #22
                          pygame.Rect(292, 172, 15, 120), #23
                          pygame.Rect(0, 172, 190, 60), #24
                          pygame.Rect(292, 172, 210, 60), #25
                          pygame.Rect(0, 30, 500, 10), #26
                          pygame.Rect(407, 50, 15, 18), #27
                          pygame.Rect(308, 37, 15, 18)] #28

```

3.2.8. gg.py

A *GG* osztályban az alapvető adatok, hasonlóan a korábbiakhoz lesznek megadva, képernyő(*screen*), magasság(*height*), szélesség(*width*), egy exit gomb képe(*button*) és a hozzá tartozó négyzet(*b_hitbox*), egy háttérkép(*bg*), és végül egy cím képe (*title*)

A *Draw* alprogram kirajzoltatja a háttérképet, a címet, a gomb képét és négyzetét, amit a *run* meghív, és a gombot használhatóvá teszi.

```
class GG():

    def __init__(self, screen):

        self.screen = screen
        self.width = 500
        self.height = 500
        self.title = pygame.image.load('game/won.png')
        self.button = pygame.image.load("menu/button_exit.png")
        self.b_hitbox = pygame.Rect(self.width/2 - 100/2, 400, 100, 50)

        self.bg = pygame.image.load("menu/main_menu_bg.png")
```

3.2.9. high_score.py

A *high_score.py* fájlban van a *HighScore* osztály, amely tartalmazza a *screen* változót, a képernyő magasságát(*height*), szélességét(*width*), a betűtípust(*font*), a háttérképet(*bg*), egy *top* nevű üres listát. Továbbá hat darab üres string változó, a top három játékos nevének és pontszámának, továbbá mindegyiknek egy képernyőre kiíratható változataik.

A *run* alprogram meghívja a *Draw* és a *Read* alprogramokat. A *Draw* alprogram kirajzoltat minden a képernyőre, A *Read* alprogramban történik a fájlból olvasás, és a dicsőséglista működése.

```
class HighScore():
    def __init__(self, screen):
        self.screen = screen
        self.width = 500
        self.height = 500
        self.font = pygame.font.Font('font/wash_your_hand/Wash Your Hand.ttf', 32)
        self.title = pygame.image.load("menu/score.png")
        self.bg = pygame.image.load("menu/main_menu_bg.png")
        self.top = []

        self.else_pont = ''
        self.e_pont = self.font.render(self.else_pont, True, (0,0,0))
        self.else_nev = ''
        self.e_nev = self.font.render(self.else_nev, True, (0,0,0))

        self.masodik_pont = ''
        self.m_pont = self.font.render(self.masodik_pont, True, (0,0,0))
        self.masodik_nev = ''
        self.m_nev = self.font.render(self.masodik_nev, True, (0,0,0))

        self.harmadik_pont = ''
        self.h_pont = self.font.render(self.harmadik_pont, True, (0,0,0))
        self.harmadik_nev = ''
        self.h_nev = self.font.render(self.harmadik_nev, True, (0,0,0))
```

3.3. Fontosabb kódrészletek

3.3.1. Gombok

A gombok működést a következő linken található oldalról vettem, és felhasználtam.

<https://stackoverflow.com/questions/46390231/how-to-create-a-text-input-box-with-pygame>

A programon belül több gomb is helyet kapott, melyekre az egérrel kattintva elérhetőek lesznek az adott funkciók. Ezek működése megegyezik. Az egér bal gombjának lenyomásakor a kurzor x és y koordinátája az adott gomb x, y koordinátájára, ugyanakkora szélességűre és magasságúra rajzolt négyzetben van, akkor a megadott esemény történik: például kilép a program, láthatjuk a dicsőséglistát, vagy elindíthatjuk a játékot.

```
while run:

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False

    if event.type == pygame.MOUSEBUTTONDOWN:
        if self.button_s.collidepoint(event.pos):
            name = Main(self.screen)
            name.run()

        elif self.button_hs.collidepoint(event.pos):
            hs = HighScore(self.screen)
            hs.run()

        elif self.button_e.collidepoint(event.pos):
            run = False
```

3.3.2. Felhasználónév input

A felhasználónév inputboxának működését a következő linken találtam, és azt alkalmaztam.

<https://stackoverflow.com/questions/46390231/how-to-create-a-text-input-box-with-pygame>

A főmenüben a START gombra kattintás meghívja a *name.py* fájl főprogramját. A felhasználónév megadásához egy külön osztályt használtam, azon belül is kettő eljárást. Az elsőben, ha az egérrel az előre megrajzolt inputboxba kattintunk, akkor az aktív állapotba kerül, ezt a szegély színének megváltozása jelzi a felhasználónak. Alaphelyzetben az inputbox nem aktív ezért, ha belekattintunk aktív lesz, viszont következő kattintásra ismét inaktív lesz. Amennyiben az inputbox aktív állapotban van, akkor a felhasználó alfanumerikus, valamint numerikus karaktereket vihet be, a megfelelő billentyű lenyomásával. A *text* nevű változó kezdetben üres string típusú változó, amelyhez a lenyomott billentyű karaktere hozzáadódik. A BACKSPACE/ TÖRLÉS billentyűvel a *text*

változóban lévő karakterek törlődnek a végeről kezdődően. Amennyiben a szöveg hosszabb 8 karakternél, a legutolsó karakter törlődik. A *txt_surface* újrendereli a *text*-et, amit a második eljárásban kiíródik grafikusán a képernyőre, hogy a felhasználó láthassa a begépet felhasználónevet és annak jelenlegi állapotát, és az inputboxot is megrajzolja.

```
def Handle_event(self, event):
    if event.type == pygame.MOUSEBUTTONDOWN:

        if self.textbox.collidepoint(event.pos):

            self.active = not self.active
        else:
            self.active = False

        self.color = self.color_active if self.active else self.color_inactive
    if event.type == pygame.KEYDOWN:
        if self.active:
            if event.key == pygame.K_BACKSPACE:
                self.text = self.text[:-1]
            else:
                self.text += event.unicode
            if len(self.text) > 8:
                self.text = self.text[:-1]

        self.txt_surface = self.font.render(self.text, True, (0, 0, 0))

def Draw(self, screen):

    self.screen.blit(self.txt_surface, (self.textbox.x+10, self.textbox.y+10))
    pygame.draw.rect(self.screen, self.color, self.textbox, 4)
```

3.3.3. Karakter mozgása, animációja

A karakter animációjának kialakítását, és annak a mozgásban lévő működését a következő linken található oldalról vettem, és felhasználtam.

<https://techwithtim.net/tutorials/game-development-with-python/pygame-tutorial/pygame-animation/>

A karakter mozgásának animációja a *Player* osztályhoz tartozó *Draw* alprogramban megy végre. Az osztály inicializálásakor a *WalkLeft*, *WalkRight*, *WalkUp*, *WalkDown* listákban betöltődnek az adott irányhoz tartozó képek az álló, egyik láb, másik láb sorrendben.

Ha a *walking* + 1 nagyobb egyenlő kilencnél, akkor a *walking* nullára állítódik, ezzel egyfajta felső határértéket szabva. a kilences számot elosztva az egyirányban felhasználható képek számával, ami három, hármat kapunk, ennyivel kell majd elosztani a *walkingot*, így ennyiszer fog mutatni képet 1 animáció alatt, lényegében ez a változó számolja az animációk számát.

Ezalatt következik a karakter igazi „animálása”. Ha a karakter nem áll, azaz mozgásban van, és az adott irányhoz tartozó logikai változó értéke IGAZ, akkor az azonos irányhoz tartozó lista elemét rajzolja ki, a fentebb említett módszerrel, vagyis a *walking* hárommal való osztásával biztosítva lesz, hogy egy kép háromszor látható minden animációkor.

Ha a karakter nem mozog, vagyis áll, hogy abba az irányba álljon arccal előre, ahogy megállt, ismét az irányokat megadó logikai változók határozzák meg a lehetőségeket, és minden irány animációjának legelső képét mutatják, mivel az az álló kép.

Ezek mellett szükséges volt egy ötödik állóképre, mivel enélkül a játék indításakor a karakter nem látszódna a legelső mozgásig, mivel csak akkor mutatna képet, ha bármelyik irány IGAZ. A játék indításakor a karakter nem mozog, így az álló képek között kell lennie, ha semelyik irányba se állt meg, mivel nem is ment azelőtt sehova. Ilyenkor a karakter lefele néző, álló képe jelenik meg.

Mivel a karakterrel együtt mozog a hitboxa, ezért minden egyes elmozduláskor újra rajzolja a *hitboxot*.

```
self.WalkLeft = [pygame.image.load('player/L_STAND.png'), pygame.image.load('player/L_WALK1.png'), pygame.image.load('player/L_WALK2.png')]
self.WalkRight = [pygame.image.load('player/R_STAND.png'), pygame.image.load('player/R_WALK1.png'), pygame.image.load('player/R_WALK2.png')]
self.WalkUp = [pygame.image.load('player/U_STAND.png'), pygame.image.load('player/U_WALK1.png'), pygame.image.load('player/U_WALK2.png')]
self.WalkDown = [pygame.image.load('player/D_STAND.png'), pygame.image.load('player/D_WALK1.png'), pygame.image.load('player/D_WALK2.png')]

def Draw(self, screen):
    if self.walking + 1 >= 9:
        self.walking = 0

    if not(self.standing): # ha nincs állásban a karakter (megy)
        if self.left: # bal
            screen.blit(self.WalkLeft[self.walking // 3], (self.x, self.y)) # leanimálva megy a képeket egymásutánrakva x és y on
            self.walking += 1
        elif self.right: # jobb
            screen.blit(self.WalkRight[self.walking // 3], (self.x, self.y)) # leanimálva megy a képeket egymásutánrakva x és y on
            self.walking += 1
        elif self.up: # fel
            screen.blit(self.WalkUp[self.walking // 3], (self.x, self.y)) # leanimálva megy a képeket egymásutánrakva x és y on
            self.walking += 1
        elif self.down: # le
            screen.blit(self.WalkDown[self.walking // 3], (self.x, self.y)) # leanimálva megy a képeket egymásutánrakva x és y on
            self.walking += 1
    else: # áll - x y on a bal jobb fel le irányokba az adott irányú Walk'stb' listán belül a [0]s kép az egyhelyben állás, azt fogja mutatni, abba
        if self.left:
            screen.blit(self.WalkLeft[0], (self.x, self.y))
        elif self.right:
            screen.blit(self.WalkRight[0], (self.x, self.y))
        elif self.up:
            screen.blit(self.WalkUp[0], (self.x, self.y))
        elif self.down:
            screen.blit(self.WalkDown[0], (self.x, self.y))
        else:
            screen.blit(self.WalkDown[0], (self.x, self.y)) # saját, hogyha egyik fele se van akkor szembe !!! enélkül csak akkor jelenne meg a kar

    self.hitbox = pygame.Rect(self.x + 5, self.y + 20, 15, 10) #!
```

A karakter valódi mozgása a *HitBox* osztály *Moving* alprogramjában történik. Mivel a pályatér tele van „bútorokkal” és „falakkal”, ezért nem lenne jó, ha a karakter ezeken a fizika törvényeit meghazudtolva, átsétálna. Az osztály változóinak inicializálásakor meghatározott *hitboxes* nevű listában lett felsorolva a pályán található összes elem hitboxa, amelyeken nem kéne átsétálnia a karakternek. A *Collide* alprogramban lefut egy ciklus, ami a *hitboxes* lista elemein megy végig. A *hitbox* alapvetően egy négyzet, amelyet az adott entitás köré rajzolnak. A *colliderect* parancs két négyzet átlapolását vizsgálja, jelen esetben, ha a karakter hitboxa beleütközik a játéktéren elhelyezett bármelyik másik hitboxba, akkor a *collide* változó IGAZ értéket vesz fel.

```
def Collide(self):
    for item in self.hitboxes:
        if self.player.hitbox.colliderect(item):
            self.collide = True
```

A *Moving* alprogramban a *keys* lesz a lenyomott gomb. Ha a *collide* HAMIS értékű, akkor az adott irányt jelző gomb (bal, jobb, föl, le) van lenyomva, és az x és y tengely helyzetében a maximum távolság nagyobb, mint a játékos helyzete(x, y), plusz a sebessége (*vel*), akkor ha balra akar menni, a játékos x értéke egyenlő lesz az x érték mínusz sebességgel. A

jobboldal esetében ez plusz lesz, mivel pythonban az x és y tengely nullás értéke a képernyő bal felső sarka, így balra és felfele csökken, lefele és jobbra nő az érték. Ennek tudatában a föl és le irányok ugyanígy működnek, csak az y tengelyen. A karakter animációja az ahhoz tartozó irányokról elnevezett logikai változók állapotától függően mennek végbe, ilyenkor az adott irányba IGAZ értéket vesznek fel, a többi HAMIS, valamint az egyhelyben állás (*standing*). Ha áll a karakter, akkor ennek fordítottja történik, valamint a *walking* értéke 0 lesz újra, hogy megfelelően mutassa az animációkat, ha újra elindul a karakter.

```

else:
    #self.collide = False
    print(self.collide)

    if keys[pygame.K_LEFT] and self.player.x > self.player.vel :
        #if keys[pygame.K_LEFT] and self.collide == False:
            self.player.x -= self.player.vel

        self.player.left = True
        self.player.right = False
        self.player.up = False
        self.player.down = False
        self.player.standing = False

    elif keys[pygame.K_RIGHT] and self.player.x < (500 - self.player.width) - self.player.vel: #w_width
        self.player.x += self.player.vel

        self.player.left = False
        self.player.right = True
        self.player.up = False
        self.player.down = False
        self.player.standing = False

    elif keys[pygame.K_UP] and self.player.y > self.player.vel :
        self.player.y -= self.player.vel

        self.player.left = False
        self.player.right = False
        self.player.up = True
        self.player.down = False
        self.player.standing = False

    elif keys[pygame.K_DOWN] and self.player.y < (350 - self.player.height) - self.player.vel: #w_height
        self.player.y += self.player.vel

        self.player.left = False
        self.player.right = False
        self.player.up = False
        self.player.down = True
        self.player.standing = False

    else:
        #self.player.left = False
        #self.player.right = False
        #self.player.up = False
        #self.player.down = False
        self.player.standing = True
        self.player.walking = 0

```

(A kódrészletet két képrészletben tudtam itt megjeleníteni, ezért az elágazásokban csúszások lehetnek, egyes részeket akár kétszer is szerepelhetnek)

Ha a *collide* IGAZ, vagyis a játékos belemegy valamibe, akkor az adott irányhoz képest ellentétes irányba fog elmozdulni, így a karakter nem tud belesétálni a másik hitboxba, hanem mindig visszamegy oda, ahol összeért a másik hitboxal.

```

keys = pygame.key.get_pressed()

if self.collide == True:
    print(self.collide)

    if keys[pygame.K_LEFT] and self.player.x > self.player.vel :
        self.player.x = self.player.x + self.player.vel

        self.player.left = True
        self.player.right = False
        self.player.up = False
        self.player.down = False
        self.player.standing = False

        self.collide = False

    elif keys[pygame.K_RIGHT] and self.player.x < (500 - self.player.width) - self.player.vel: #w_width
        self.player.x = self.player.x - self.player.vel

        self.player.left = False
        self.player.right = True
        self.player.up = False
        self.player.down = False
        self.player.standing = False

        self.collide = False

    elif keys[pygame.K_UP] and self.player.y > self.player.vel :
        self.player.y = self.player.y + self.player.vel

        self.player.left = False
        self.player.right = False
        self.player.up = True
        self.player.down = False
        self.player.standing = False

    elif keys[pygame.K_DOWN] and self.player.y > self.player.vel :
        self.player.y = self.player.y + self.player.vel

        self.player.left = False
        self.player.right = False
        self.player.up = True
        self.player.down = False
        self.player.standing = False

        self.collide = False

    elif keys[pygame.K_DOWN] and self.player.y < (350 - self.player.height) - self.player.vel: #w_height
        self.player.y = self.player.y - self.player.vel

        self.player.left = False
        self.player.right = False
        self.player.up = False
        self.player.down = True
        self.player.standing = False

        self.collide = False

    else:
        self.collide = False
        self.player.standing = True
        self.player.walking = 0

```

(A kódrészletet két képrészletben tudtam itt megjeleníteni, ezért az elágazásokban csúszások lehetnek, egyes részletek akár kétszer is szerepelhetnek)

3.3.4. Küldetések és NJK-k

Minden Nem Játsható Karakternek külön osztály és küldetés lett meghatározva, lényegében hasonló, mégis egy kicsit eltérő módon. A legelső és a legutolsó küldetés lett a leginkább eltérő tartalmilag.

Alapvetően az egyes küldetés aktív, így amikor a karakter odamegy hozzá, és a hitboxaik összeérnek, akkor az *activeline* alaphoz a nullás indexű szöveg, ami a küldetést mondja el. Ha a karakter hitboxa összeér az NJK hitboxával, akkor kiírja az *activeline*-t, és az *npc_count* értéke nő eggyel. Ha már egynél többször beszélt vele a karakter, akkor az *itemactive* IGAZ értékre változik, tehát a küldetés tárgy elérhetővé válik a karakter számára. Ha ez megtörténik, akkor, ha a játékos hitboxa összeér a tárgyával, akkor az *itemline* kiíródik a képernyőre, és az *itemfound* IGAZ értéket vesz fel, tehát a karakter megtalálta a keresett tárgyat. Ha megtalálta, akkor az *activeline* vált az egyes indexű szövegre, ami a küldetés végét jelzi, az *npc_count* értéke ismét nulla lesz. Ha ezt követően beszél a játékos az NJK-val, akkor az *activeline* új szövegét írja ki, Ha beszélt vele, akkor a *done* változó IGAZ értéket vesz fel, tehát a küldetés kész van, az *npc_count* nő eggyel, valamint az *active* FALSE értéket vesz fel, hiszen már vége a küldetésnek.

Ha a küldetés nem aktív, és a játékos mégis beszélne az NJK-val, akkor a *lines* lista második indexén lévő szöveg lesz az *activeline*, amiben arra kéri a játékost, hogy csináljon egy másik küldetést.

```
def Quest(self):
    if self.active == True: # ha a quest aktív toggled
        self.activeline = self.lines[0] # akkor az activeline a quest line
        if self.player.hitbox.colliderect(self.hitbox): # ha collide ol az npc vel
            self.screen.blit(self.activeline, (self.tb_x, self.tb_y)) # activelinet kiírja
            self.npc_count += 1 # számolja hányszor beszélt az npc vel
            if self.npc_count >= 1: # ha legalább egyszer beszélt vele
                self.itemactive = True # az item megtalálható a questhez
            if self.itemactive == True: # ha megtalálható az item
                if self.player.hitbox.colliderect(self.item): # és ha collideol az itemmel
                    self.screen.blit(self.itemline, (self.tb_x, self.tb_y)) # kiírj az itemlinet
                    #self.itemcount += 1 # növeli egyel hányszor találta meg az itemet
                    self.itemfound = True # vagy alaphoz ez egynek számít már...
            if self.itemfound == True: # ha megvan az item
                self.activeline = self.lines[1] # az aktivline a megvan az item line lesz
                self.npc_count = 0
                if self.player.hitbox.colliderect(self.hitbox): # és ha megvan az item és collideol az npc vel
                    self.screen.blit(self.activeline, (self.tb_x, self.tb_y))
                    self.done = True
                    self.npc_count += 1
                    self.active = False
    else:
        self.activeline = self.lines[1]
        if self.player.hitbox.colliderect(self.hitbox): # ha collide ol az npc vel
            self.screen.blit(self.activeline, (self.tb_x, self.tb_y))
```

A második, harmadik, negyedik küldetésnél annyi eltérés van, hogy csak akkor lesz aktív, ha az előző küldetés *done* változója IGAZ.

```

def Quest(self):
    if self.npc2.done == True:
        self.active = True
        if self.active == True:
            self.activeline = self.lines[0]
            if self.player.hitbox.colliderect(self.hitbox):
                self.npc_count += 1
            if self.npc_count >= 1:
                self.itemactive = True
            if self.itemactive == True:
                if self.player.hitbox.colliderect(self.item):
                    self.screen.blit(self.itemline, (self.tb_x, self.tb_y))
                    self.itemfound = True
            if self.itemfound == True:
                self.activeline = self.lines[1]
                self.npc_count = 0
                if self.player.hitbox.colliderect(self.hitbox):
                    self.done = True
                    self.npc_count += 1
                    self.active = False

        else:
            self.activeline = self.lines[2]

    else:
        self.activeline = self.lines[2]

pygame.display.update()

def Draw(self, screen):
    #pygame.draw.rect(self.screen, (255,0,0), self.item, 1)
    #pygame.draw.rect(self.screen, (255,0,0), self.hitbox, 1)
    if self.player.hitbox.colliderect(self.hitbox):
        self.screen.blit(self.activeline, (self.tb_x, self.tb_y))

```

Az ötödik küldetésnél egyszerűbb a helyzet. Ha kész a negyedik küldetés, akkor egyrészt az összes küldetés kész van, tehát az ő küldetése is kész van, ami az, hogy minden küldetést végezzen el. Ha aktív a küldetése, akkor a nullás indexű szöveg egyben a küldetés leadó szöveg. Amint a játékos karakter odamegy beszélni vele, az *activeline*t kiírja, a küldetés elkészül és többé nem aktív. Amíg nem lesz aktív az ő küldetése, addig az összes küldetés elvégzésére bíztatja a játékost.

```

def Quest(self):
    if self.npc4.done == True:
        self.active = True
        if self.active == True:
            self.activeline = self.lines[0]
            if self.player.hitbox.colliderect(self.hitbox):
                self.done = True
                self.active = False
        else:
            self.activeline = self.lines[1]

pygame.display.update()

```

Minden NJK osztályához tartozik továbbá

3.3.5. Dicsőséglista

A dicsőséglista egy fájlban tárolódik el, amit a *Game* osztály *Write* alprogramja kezel. Ezt a *HighScore* osztály *Read* alprogramja olvassa. Egy *matrix* nevű kétdimenziós listába a fájl elemeit soronként beleírjuk. Így a *matrix* nevű listában lesznek listák, melyek egy karakternevet és egy pontszámot tartalmaznak. Ezeket a *strip*-el megszabadítja a sorvégi *\n*-től, és a sorok elemeit a pontosvessző karakter mentén választja szét, ezzel kétdimenziód listát, azaz mátrixot hozva létre. Ezek után a *top* nevű listában rendeződnek az adatok pontszám szerint, de mivel alapból növekvő sorrendbe rakná őket a *sorted*, ezért ezt meg kell fordítani, így lesznek a legmagasabb pontszámok a lista elején. Ebből adódóan, a top 3 játékos kiírásához az első három listán belüli lista adatira van csak szükség.

A hibák elkerülése végett, ha nem volt még 3 név, akkor annyival kevesebb játékost fog megjeleníteni a dicsőséglista, amennyiben még egyszer sem játszott a felhasználó a játékkal, akkor a dicsőséglista üres.

```
def Read(self):
    with open('game/scoreboard.txt', 'r', encoding = 'utf-8-sig') as f:
        matrix = [sor.strip('\n').split(';') for sor in f]

    self.top = sorted(matrix, reverse=True)

    if len(matrix) >= 3:
        self.első_pont = self.top[0][0]
        self.első_nev = self.top[0][1]

        self.masodik_pont = self.top[1][0]
        self.masodik_nev = self.top[1][1]

        self.harmadik_pont = self.top[2][0]
        self.harmadik_nev = self.top[2][1]

    elif len(matrix) >= 2:
        self.első_pont = self.top[0][0]
        self.első_nev = self.top[0][1]

        self.masodik_pont = self.top[1][0]
        self.masodik_nev = self.top[1][1]

        self.harmadik_pont = ''
        self.harmadik_nev = ''

    elif len(matrix) >= 1:
        self.első_pont = self.top[0][0]
        self.első_nev = self.top[0][1]

        self.masodik_pont = ''
        self.masodik_nev = ''

        self.harmadik_pont = ''
        self.harmadik_nev = ''

    else:
        self.első_pont = ''
        self.első_nev = ''

        self.masodik_pont = ''

        self.masodik_nev = ''

        self.harmadik_pont = ''
        self.harmadik_nev = ''

    self.e_pont = self.font.render(self.első_pont, True, (0,0,0))
    self.e_nev = self.font.render(self.első_nev, True, (0,0,0))
    self.m_pont = self.font.render(self.masodik_pont, True, (0,0,0))
    self.m_nev = self.font.render(self.masodik_nev, True, (0,0,0))
    self.h_pont = self.font.render(self.harmadik_pont, True, (0,0,0))
    self.h_nev = self.font.render(self.harmadik_nev, True, (0,0,0))
```

(A kódrészletet két képrészletben tudtam itt megjeleníteni, ezért az elágazásokban csúszások lehetnek, egyes részletek akár kétszer is szerepelhetnek)

3.4. Grafikus elemek és képek

A programomban felhasznált képek egy része nem az én tulajdonomat képezik, én csupán kivágtam darabokat, majd képszerkesztésre alkalmas szoftverrel/szoftverekkel átszíneztem, esetleg kisebb átalakításokat vittem véghez.

3.4.1. Főmenü háttérkép

A főmenü háttérképéhez a játéktér falának színeit használtam fel, mivel a zöld kellemes és vidám szín, továbbá a karakter ruhája is hasonló árnyalatú. Miután elkészült, csináltam néhány tervet, hogy fog kinézni a menü véglegesen is. A színek a játéktér falán lévő két zöld árnyalatot használja. Az alapja egy egyszínű háttér, amire a másik színnel egy teli félkört helyeztem el.



Menü legelső tervezet



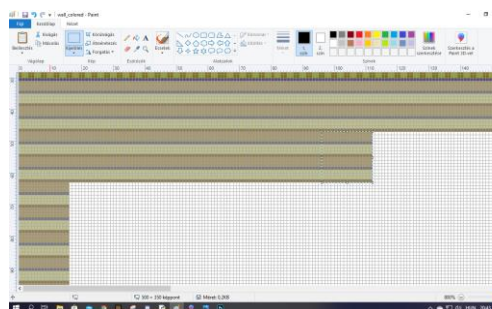
Menü tervezet

Scoreboard		
Player	Score	Rank
Attila	5	1
Tome	5	2
Leeco	4	3

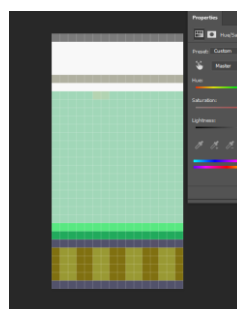
Dicsőséglista tervezet

3.4.2. Játéktér kialakítása

A játéktér kialakítása igényelte a rajzolt elemek közül a legtöbb időt. Minden faldarab, padlódarab, NJK, és berendezés egyesével lett lerakva, a játéktérre, mely sok átalakításon esett át. A helyiségek úgy lettek kialakítva, hogy mindegyikhez tartozzon egy NJK, és viszonylag könnyen tudjon mozogni a karakter. A játéktér alapjait a klasszikus Paintel csináltam, a színezéseket valamint a bútorokat Photoshoppal helyeztem el.



Padló lerakása



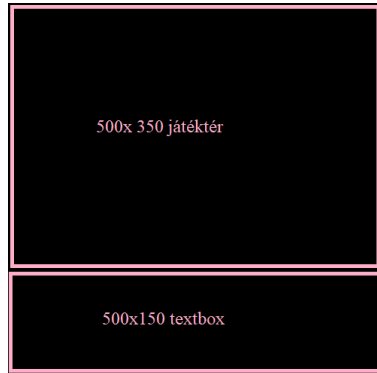
Fal kinézete



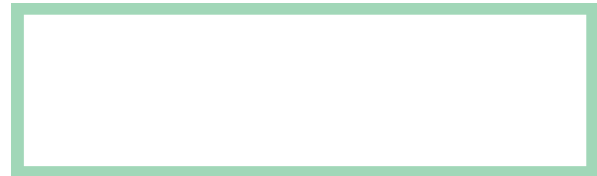
A játéktér kialakításának legelső fázisa

3.4.3. Szövegdoboz

A szövegdoboz ötlete régi pixelgrafikus játékok alapján jutott eszembe, ezért szerettem volna én is hasonló módszert alkalmazni.



Játéktér és szövegdoboz tervek



Szövegdoboz

3.4.4. Játékos karakter

A játékos karaktert egy több karaktert tartalmazó, internetről letöltött képből vágtam ki, színeztem ki, majd a mozgási állapotokat kivágtam. Photoshoppal eltávolítottam a kép háttérét, majd kisebb átdolgozásokat követően a színeket is megváltoztattam. Végül az animáció darabjait egyesével kivágtam 32x32-es méretre.



Eredeti képrészlet



Átrajzolt karakter



Kivágási segédvonalak

3.4.5. Nem Játékos Karakterek

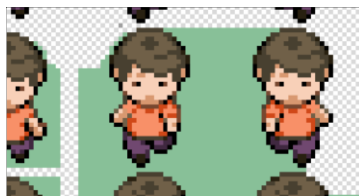
A NJK-t a játékos karakterhez hasonlóan, a korábban már említett képről vágtam ki, változtattam a kinézetükön, és daraboltam szét, melyekből csak a jobbra vagy balra néző képeket használtam fel, az ötödik NJK estében szemből látható a karakter. Ezeket a játékoskarakteréhez hasonlóan szerkesztettem át.



4.NJK alapozása



Az 1. NJK teljes mozgása



Az 1. NJK rajzolás közben



Az 5. NJK
Átrajzolás,
tisztítás előtt

3.5. Betűtípusok

A játékban megjelenő szövegek betűtípusait a <https://www.dafont.com/> linken elérhető weboldalról töltöttem le. Ezek a betűtípusok ingyenesen elérhetőek, valamint felhasználhatóak.

3.5.1. Menüben felhasznált betűtípus

A játékon kívül mindenhol, például a menüben is ugyanazt a betűtípust használtam, amit a következő linken lehet elérni:

<https://www.dafont.com/wash-your-hand.font?psize=s&text=Exit>

A betűtípus a következőképpen néz ki:

Program

3.5.2 Játékban felhasznált betűtípus

A játékon belülre egy olyan betűtípust szerettem volna használni, amely illik a pixelgrafikus környezetbe. A betűtípus linkje:

<https://www.dafont.com/press-start-2p.font>

A betűtípus kinézete:

Amelyik borjú átfér a kapu alatt, maga esik bele.

Amelyik borjú átfér a kapu alatt, maga esik bele.

3.9. Tesztdokumentáció

A program szinte tökéletesen működik, hibaüzenetekre alaptól nincs rá szükség, mivel a gombokra kattintva válthat az ablakok között. A karakter mozgása néhol nem működik tökéletesen a fizikai határokkal, hogyha *collide* közben a játékos a vele ellentétes irányba mozogna úgy, hogy két ellentétes irányú billentyűt nyom le egyszerre és az egyiket hirtelen felengedi.

A játék tesztelve lett több, informatikához szakmailag nem értő személyekkel, akiknek a játék tetszett, ám nehézségeik akadtak egy- egy ponton a játék fizikájával. Ezen felül pozitív visszajelzéseket kaptam a programomról.

4. Fejlesztési lehetőségek

A játékomba több fejlesztési lehetőség is eszembe jutott:

- A játékosnév megadásakor lehetne különböző kinézetű karakterek közül is választani
- Nagyobb játéktér, több pálya, helyiség
- Több NJK, valamint több küldetés
- Többfajta küldetés
- Alájátszási zenét lehetne rakni a játékba
- A játék fizikájának szofisztikáltabb megvalósítása

5. Irodalomjegyzék

- <https://www.pygame.org/docs/>
- <https://stackoverflow.com/>
- <https://www.dafont.com/>
- <https://pypi.org/project/auto-py-to-exe/> - exe konverter
- https://www.google.com/search?q=pok%C3%A9mon+tilet&safe=strict&source=lnms&tbm=isch&sa=X&ved=2ahUKEwiGksSRwqLpAhV05aYKHX90ClSQ_AUoAXoECAsQAw&biw=1280&bih=586#imgsrc=rKcOqlZhFS_gQM&imgdii=qdURQfY87oihIM – a játéktér kialakításában használt elemeket tartalmazó kép
- <https://techwithtim.net/tutorials/game-development-with-python/pygame-tutorial/pygame-animation/> - az animáció és annak mozgásba bevitele
- <https://github.com/techwithtim/Tower-Defense-Game/blob/master/run.py> - a run fájl használata
- https://www.sprisers-resource.com/ds_dsi/pokemonblack2white2/sheet/48049/ - az összes karakter alapját tartalmazó kép
- <https://stackoverflow.com/questions/46390231/how-to-create-a-text-input-box-with-pygame> - felhasználónév inputboxa és a gomb működése

6. Elérhetőségek

Amennyiben problémája, esetleg javaslata lenne a programmal kapcsolatban, kérem a rebmanfanny@gmail.com email címen keressen meg.