

# Big Driver 2



Fejlesztette

Fekete Krisztián | Török Gábor

Kovács Jázmin-Mónika | Koncz Nimród-Kálmán

2024

# 1. Témaindoklás

Dolgozatunk témáját onnan merítettük, hogy a tavalyi év során (2023-ban) a mi osztályunk rendezhette meg A Nagy Fizika Hű-Hót. Ez alkalomból az osztályunk sokat foglalkozott a fizikával, így sokunk számára érdekesebbé vált.

Amikor meglett a csapatunk, sokat gondolkodtunk, hogy milyen témában dolgozzunk. Több ötlet felmerült, amik közül végül egyiket sem választottuk. Ekkor jött az ötlet, hogy ha már ennyire benne vagyunk a fizikában, akkor ez lenne a jó megoldás. Inspirációt kerestünk az interneten és megkértük a fizika tanárunkat, hogy adjon pár ötletet, így jutottunk a dinamika választására.

Felosztottuk négy részre a projektet azzal a céllal, hogy amennyire tőlünk telik a legjobbat tudjuk kihozni belőle, még ha nem is a leghízelgesebb.

## 2. Rendszer követelmények

### 2.1. Hardver igény

A program kb. 330 MB memóriát igényel

A teszteléshez Intel(R) Core(TM) i5-1235U processzorokat használtunk, melyeken kiváló eredménnyel működött minden.

### 2.2. Szoftver igény

Windows operációs rendszer szükséges a szoftver futtatásához, mi Windows 11-en futtattuk



Az alábbi képen látható a program adathasználata

Processzor	Memória	Lemez
0,4%	504,5 MB	0,1 MB/s

### 3. Felhasználói kézikönyv

#### Program elindítása és a kezdő oldal:

A játékot a "vegso\_osszeallitas" nevű fájl megnyitásával lehet elindítani. Indításkor a játék főmenüjébe lépünk be, felül látható a játék és az oktató rész témája: „DINAMIKA”, továbbá három lehetőségünk van:


- Az elmélet tanulása:  gomb
- A játék elindítása:  gomb
- Kilépés gomb

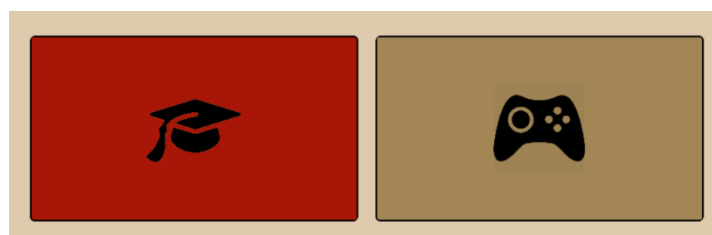
Az oldal alsó felen jelennek meg a programozói csapat nevei.

Az „X” gomb megnyomásával, amely a jobb felső sarokban található, kilépünk a programból.



#### Oktató oldal:

A  gomb megnyomásával érjük el az „Oktató” című oldalt. Ennek megnyitása után látható négy lehetőség: „Sebesség”, „Gyorsulás”, „Gravitáció” és „Súrlódás” gombok, melyekre ha rá kattintunk, elérjük a megfelelő témához kapcsolódó információkat.



A bal felső sarokban található „<-” gombbal tudunk vissza lépni a kezdő



oldalra.



Rá kattintva a „**Sebeség**” nevű gombra érjük el az ehhez kapcsolódó információkat, hasonlóan a „**Gyorsulás**”, a „**Gravitáció**” és „**Súrlódás**” gombokkal.



Ha most nyomjuk meg a vissza-gombot, akkor az „Oktató” nevű oldalra jutunk újra. Innen elérjük a többi dinamikához kapcsolatos anyagot.



A „Surlodas” nevű oldalon, a hely hiánya és az anyag nagysága miatt, ha rá visszuk a kurzort az anyagreszre, a gorgo segítségével tudjuk elérni a további információkat.

A súrlódási erő nyugalomban tartja a testet, kiegyenlítve a súlyerő párhuzamos összetevőjét. A súrlódási erő növekedésének viszont felső határa van.

Ha a súlyerő párhuzamos összetevője meghaladja a súrlódási erő felső határát, akkor a test lecsúszik a lejtőn  
 $F_s = \mu_s \cdot N$

Ha a test már mozgásban van, akkor csúszó súrlódásról beszélünk.  
 $F_s = \mu \cdot N$

## 2. Gördülési súrlódás

Az együttható a száraz betonon 0.01 - 0.05

A lassulás a következőképpen számítható:

$$F_s = \mu \cdot N \quad N = G_{\text{mer}} \quad G_{\text{mer}} = m \cdot g \quad F_s = m \cdot a$$

$$a = F_s / m \rightarrow a = \mu \cdot N / m \rightarrow a = \mu \cdot G_{\text{mer}} / m \rightarrow a = \mu \cdot m \cdot g / m \rightarrow a = \mu \cdot g$$

### Gyorsulás lejtőn felfele:

$F_s = \mu \cdot N$

## 2. Gördülési súrlódás

Az együttható a száraz betonon 0.01 - 0.05

A lassulás a következőképpen számítható:

$$F_s = \mu \cdot N \quad N = G_{\text{mer}} \quad G_{\text{mer}} = m \cdot g \quad F_s = m \cdot a$$

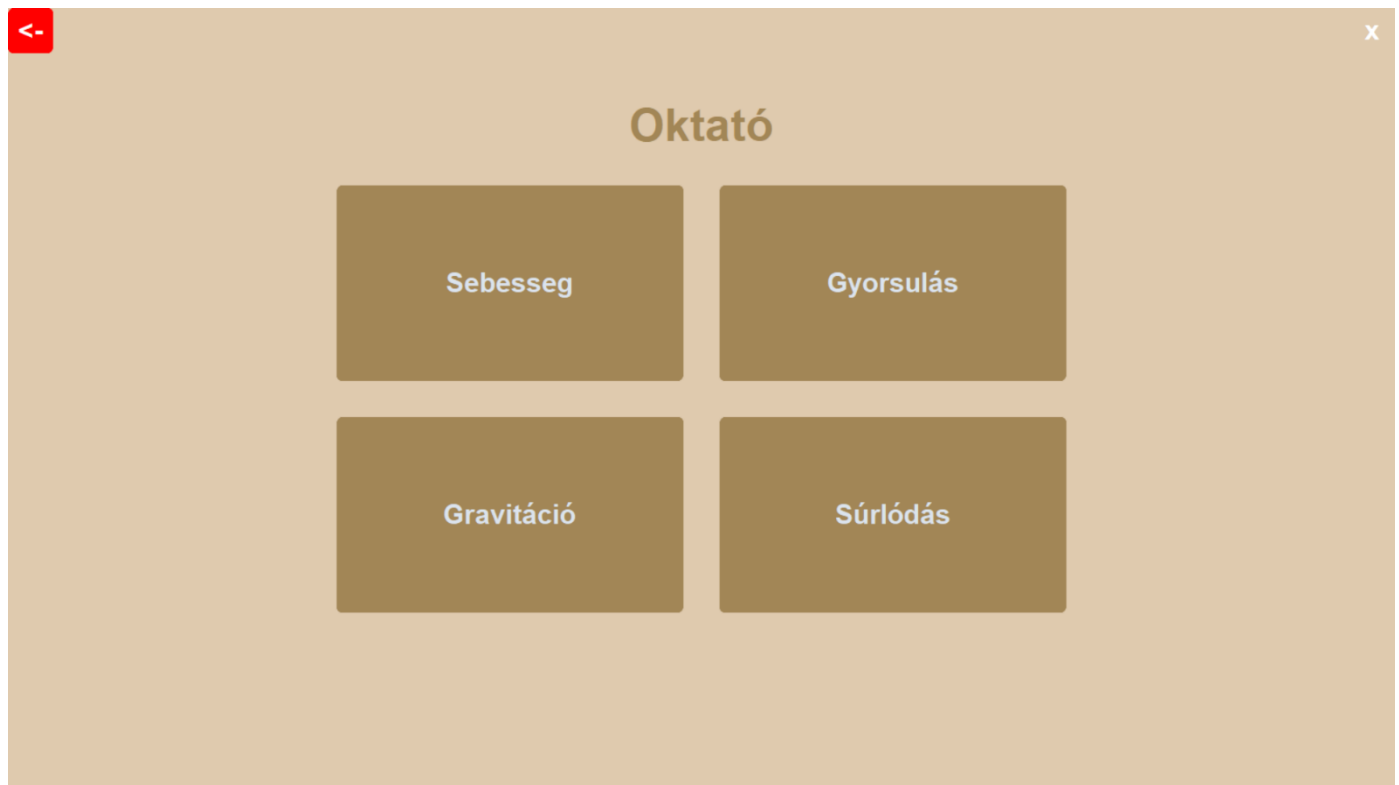
$$a = F_s / m \rightarrow a = \mu \cdot N / m \rightarrow a = \mu \cdot G_{\text{mer}} / m \rightarrow a = \mu \cdot m \cdot g / m \rightarrow a = \mu \cdot g$$

### Gyorsulás lejtőn felfele:

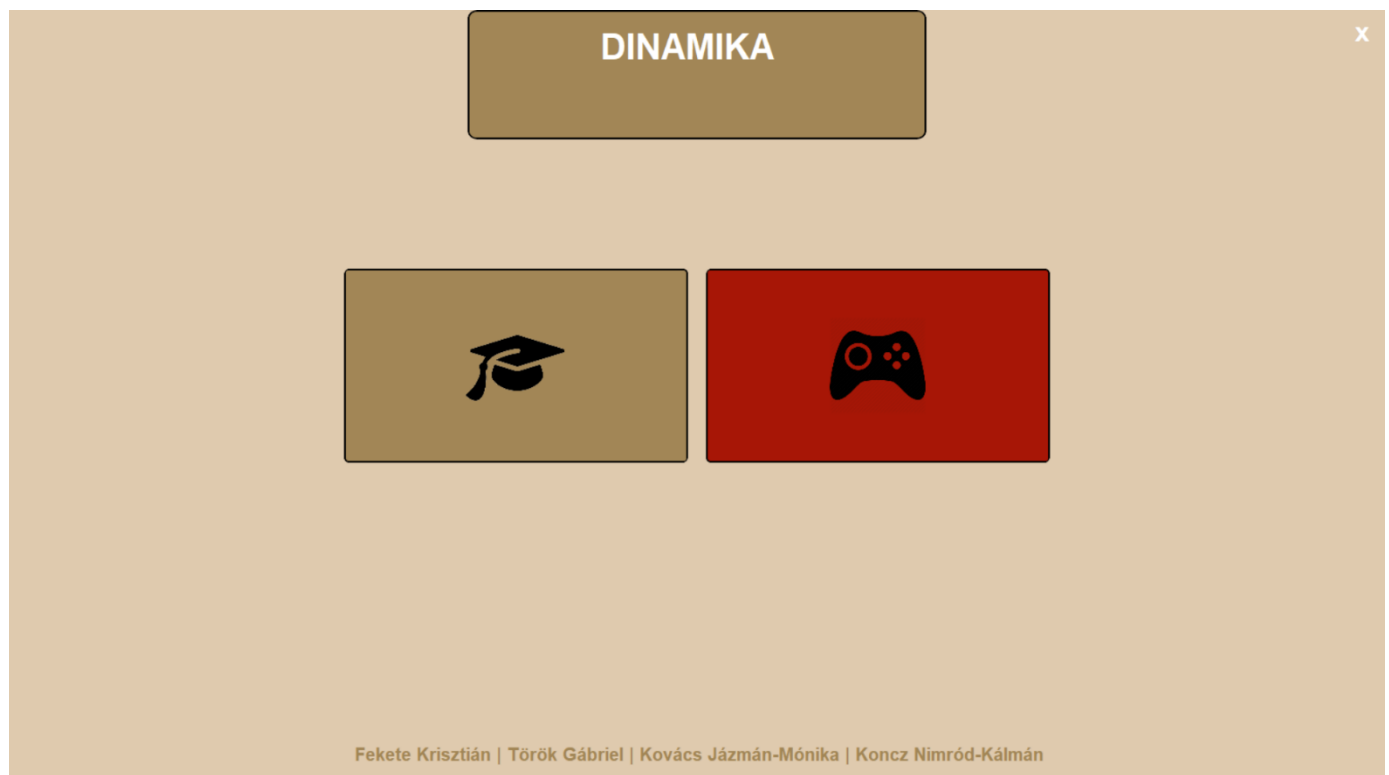
$F_1 = G \cdot \cos \alpha$ 
 $F_2 = G \cdot \sin \alpha$ 
 $T = F_e$


$T = F_1 - F_s$   
 $F_e = F_1 - F_2 \cdot \mu$   
 $m \cdot a = G \cdot \sin \alpha - G \cdot \cos \alpha \cdot \mu$   
 $m \cdot a = m \cdot g \cdot \sin \alpha - m \cdot g \cdot \cos \alpha \cdot \mu$   
 $a = g(\sin \alpha - \cos \alpha \cdot \mu)$

Ha újra megnyomjuk a vissza-gombot, akkor a Kezdo oldalra jutunk vissza.

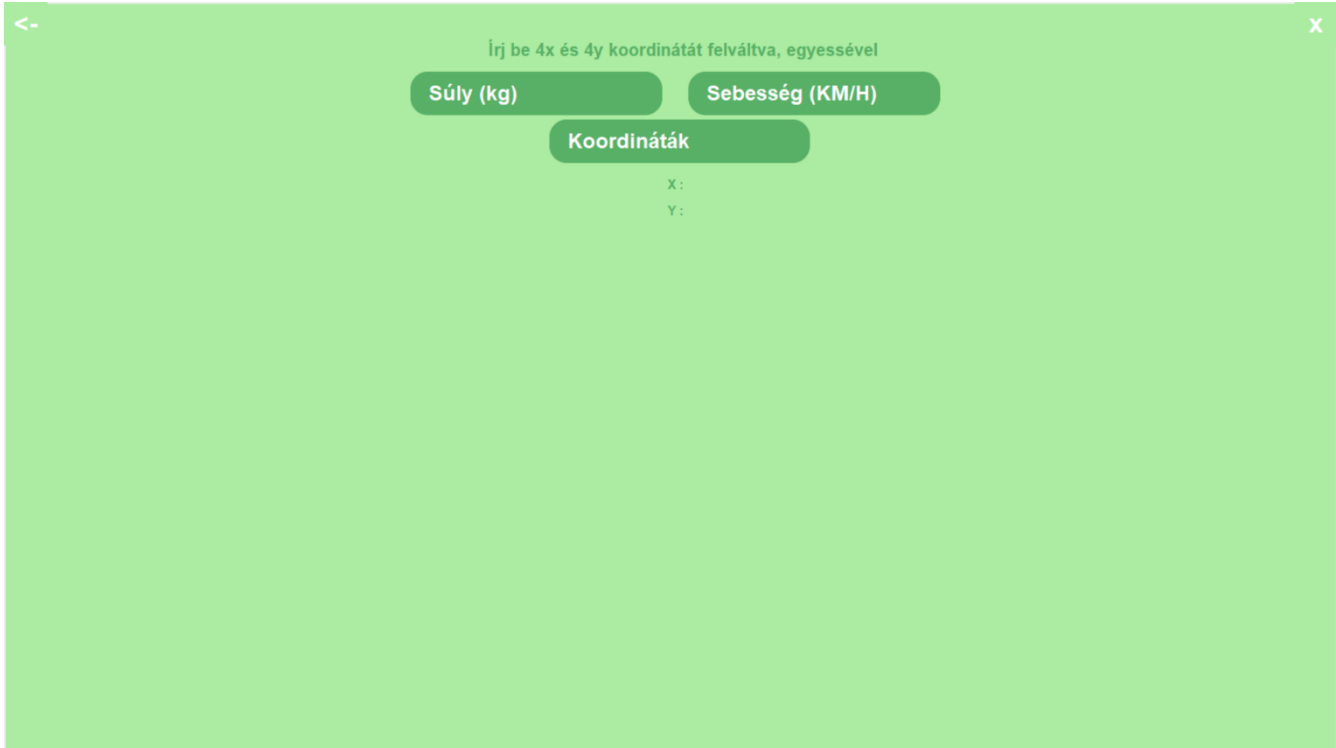


Vissza erve a kezdo oldalra, miutan kaptunk kello ismeretet az anyagreszekrol, kezdodhet az interaktiv resze a programunknak.



Megnyomva a  gombot, erjuk el az interaktiv reszet a programnak.

Az oldal megnyilasa utan van harom lehetosegunk (a kilepes es a vissza-gombon kivul).



The screenshot shows a light green application window with a title bar containing a back arrow and a close 'X' button. The main content area has a light green background. At the top, there is a instruction text: "Írj be 4x és 4y koordinátát felváltva, egyessével". Below this, there are three input fields: "Súly (kg)", "Sebesség (KM/H)", and "Koordináták". The "Koordináták" field is currently active, showing "X:" and "Y:" labels.

Eloszor is irjuk be az elso lehetoseghez az auto tetszoleges sulyat. Ezutan nyomjuk meg az ENTER-t. Ha nem megfelelo a beolvasasunk, akkor az „ERROR” szo fog megjelenni.

Súly (kg)

62.0 kg

Ebben az esetben ujra rakattintva a textboxra tudunk szamot beolvasni.

62.0 kg|

Sebesség (KM/H)

Mikor ez megvan, a kovetkezo textboxba irhatjuk be a kezdosebesseget a jarmunek.

150.0 km/h|

Ezután az utolsó lehetőséghez érve elolvassuk a szöveget, amely legfelül található.

**Írj be 4x és 4y koordinátát felváltva, egyessével**

Figyelembe véve a felhívást, írjunk be 8 tetszőleges számot, ha ez nem lenne a pálya felületén, akkor nem fogadja el, hanem ilyenkor új számot kell választanunk.

**Koordináták**

X:

Y:

A beolvasott számok meg fognak jelenni a textbox alatt a megfelelő helyen. Ezeket szintén az ENTER megnyomásával megvalósíthatóak.

Az összes adat helyes beolvasása után megjelenik a következő ablak. Itt egy lehetőségünk van, hogy megnyomjuk a START gombot. A gomb alatt láthatóak a koordináták amelyeket megadtunk.

<- x

**START**

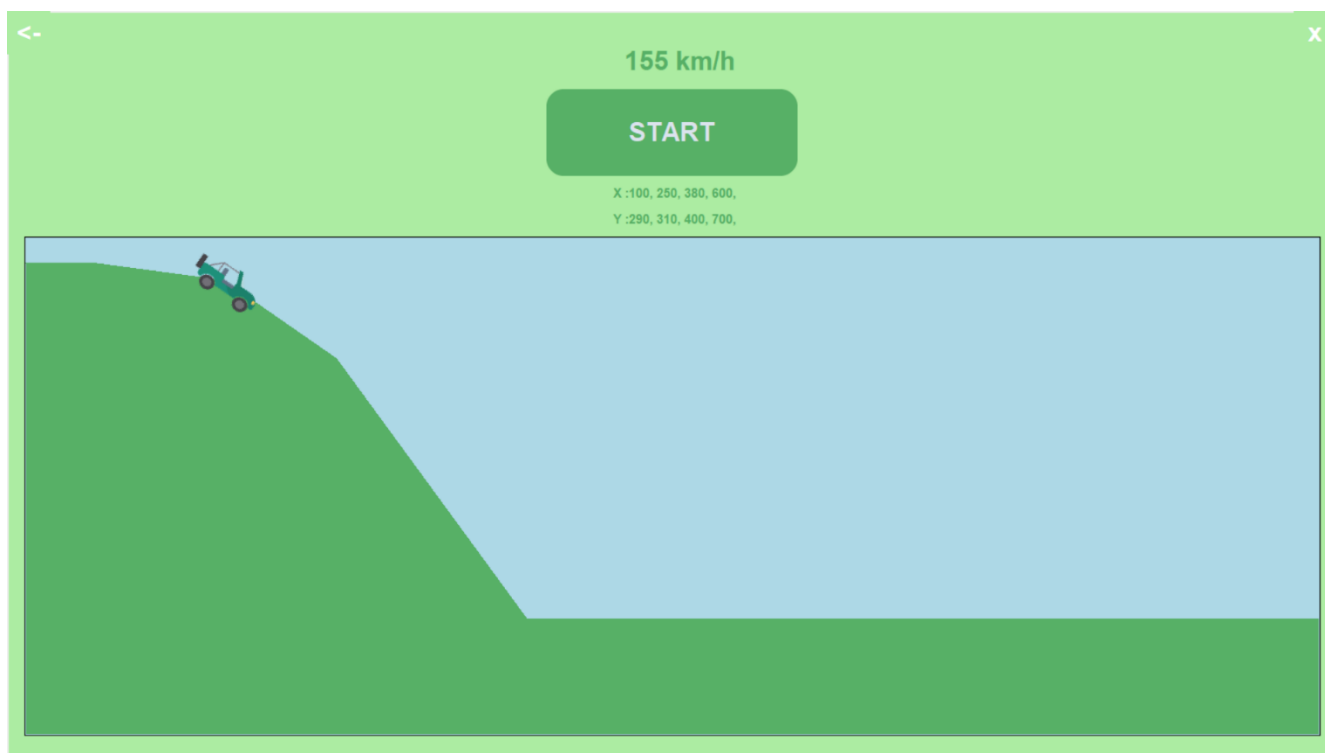
X :100, 250, 380, 600,  
Y :290, 310, 400, 700,



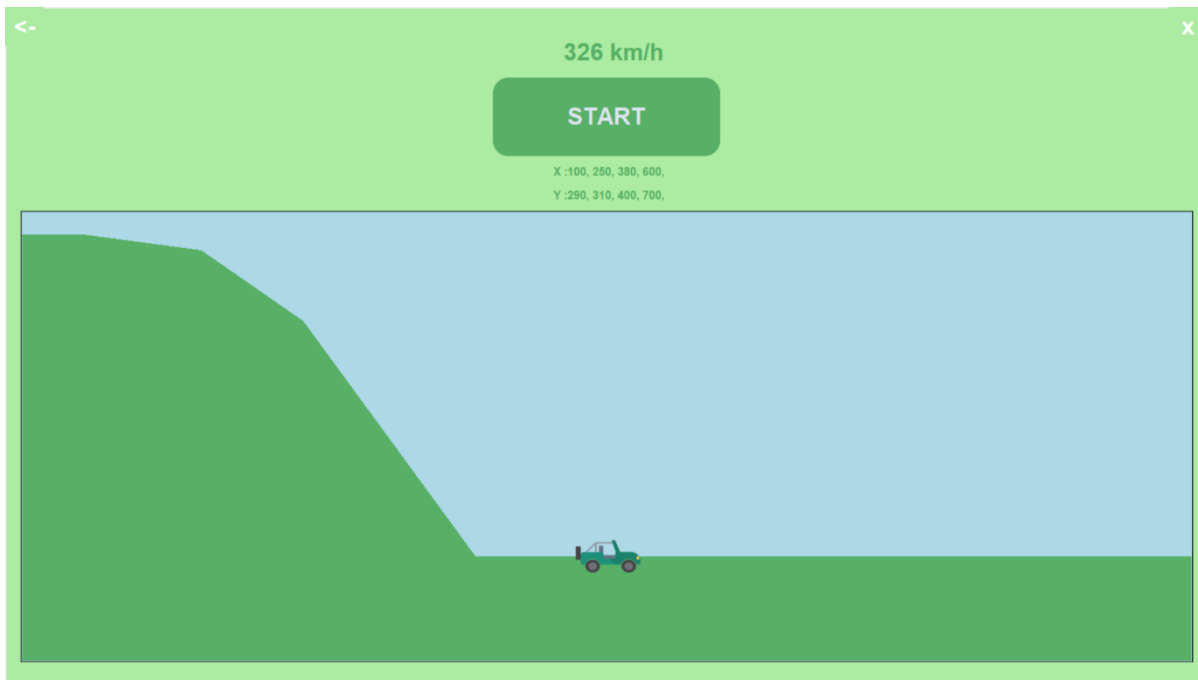




A START gomb megnyomása következtében a pálya fog megjelenni, az autoval, amelyet a koordinatak segítségével hoztunk létre.

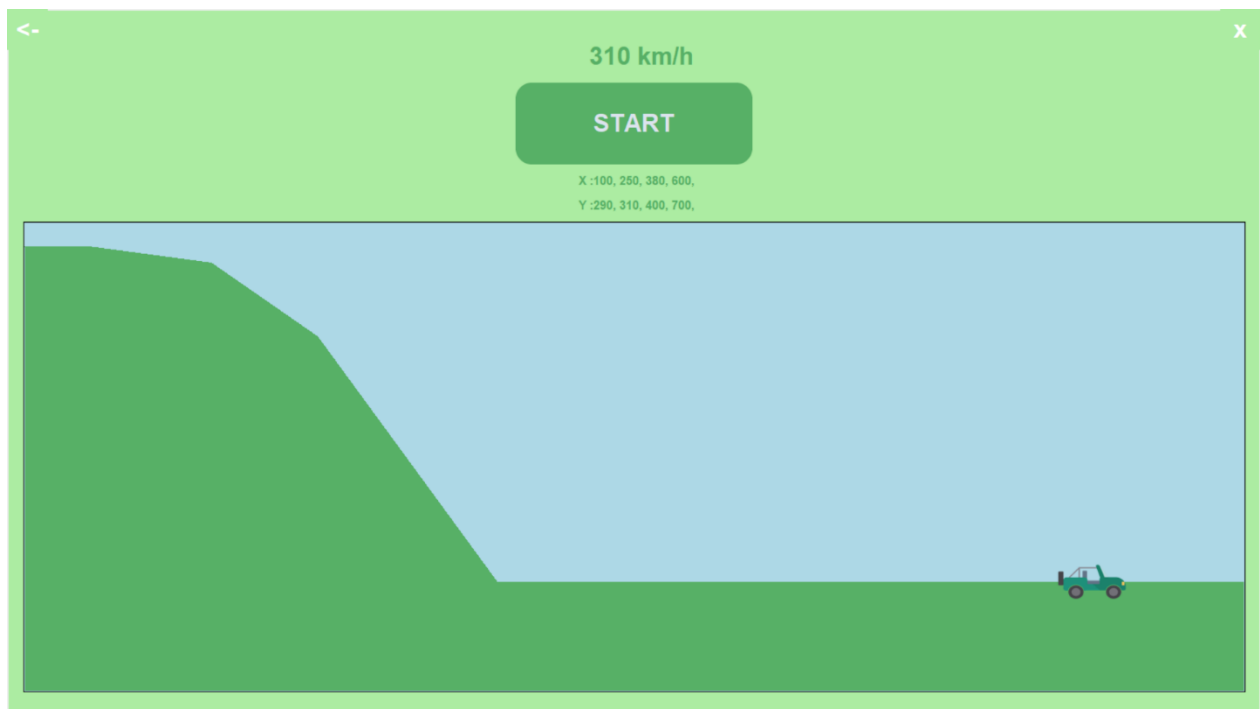


Ahogy az auto lejtőn megy lefele, vagy ahogy emelkedőre megy fel, változik a sebessége, ez megfigyelhető a START gomb felett és az auto látható sebességén is.



A lejtőn leérve megfigyelhető, hogy jelentősen gyorsabban megy, mint amikor elindult.

Kis idő elteltével lassulni fog, mivel nincs ami gyorsítsa.



Amikor vége, ha megnyomva a vissza-gombot, vissza jutunk a Kezdo oldalra, ha megnyomjuk az „X” gombot, akkor kilépünk a programból.

## 4. Programozói kézikönyv

### Általános információk

A "Big Driver 2"-nek elnevezett programunk oktató és interaktív részből áll. A programot teljes mértékben Python programozási nyelvben írtuk, ami tartalmazza a működést és a megjelenítést is.

A megjelenítéshez "tkinter"-t használtunk, amivel ablakokat készítettünk. Ezekre "Label"-t, "Frame"-et és más hasonló elemekkel személyre szabtuk az oldalakat. Az elemeket részletesebben változtattuk "customtkinter" segítségével. A projektben "import"-olt és bárhol felhasznált összes könyvtárak a következők: "threading", "time", "math", "functools", "partial", "tkinter", "tkinter.filedialog", "asksaveasfile", "tkinter.font", "customtkinter", "PIL".

### Felépítés

Az egész program alap ablaka a "root". Ezt beállítottuk "fullscreen"-re. Így esztétikusabb lett a végeredmény. Ezt a "root.attributes("-fullscreen", True)"-val értük el.

Az alkalmazás 2 fő oldalból áll:

#### 1. Oktató

- Az oktató részben a dinamika 4 részét tárgyaljuk. (sebesség, gyorsulás, gravitáció, súrlódás)

#### 2. Interaktív

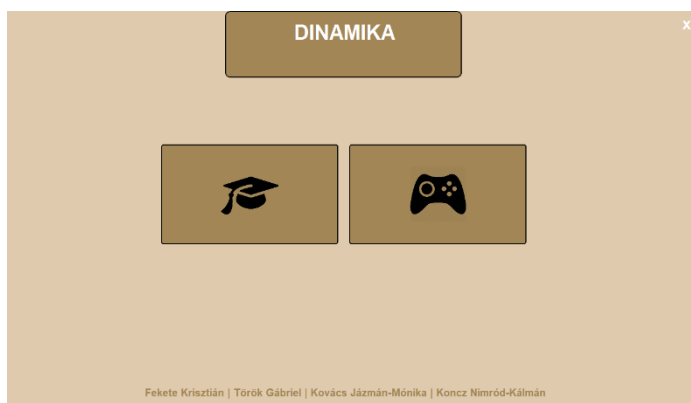
- Az interaktív részben a projekt oktató részében tanultakat használhatjuk fel. Egy autónak az útját követhetjük le, ami egy a felhasználó által pontosan meghatározott utat jár be. A felhasználó tetszőleges tömeget és kezdősebességet adhat meg. Az autó miközben halad az úton csökken a sebessége az út szögének és súrlódásának függvényében.

## Kezdő oldal (Home)

Az program futásával a "home " függvényt hívjuk meg.

### Az oldal tartalma:

1. Cím
2. Lábjegyzék
  - a. A programozói  
csapatok nevei
3. Gombok
  - a. Oktató
  - b. Interaktív
  - c. Bezárás



### 1. Cím

A cím az oldal tetején jelenik meg középen. Elneveztük "main\_frame"-nek azt a "frame"-et amit használtunk egy doboz létrehozásához, amiben szerepel a "Dinamika" szöveg, mint cím. A "main\_frame" megkapja a "home\_box\_cim" függvény visszatérítési értékét. Ez lesz a függvényen belül, "customtkinter"-rel létrehozott "frame" amit a képernyő szélességi és hosszúsági méreteihez igazítottuk. (Az igazítás miatt a "frame" minden a programot futtató eszközön ugyanakkora %-ban jelenik meg a képernyőn. Ez a tulajdonság igaz minden képernyő méreteihez igazításra). A "frame"-nél a "#a28656" és "#dfcaae" színek kódokat használtunk. Ezek a színek jelennek meg a projekt oktató felében is. Adtunk egy fekete körvonalat és a sarkokat lekerekítettük.

A "Dinamika" szöveg egy "Textbox"-ban lett létrehozva, aminél a testre szabáshoz ugyancsak "customtkinter"-t használtunk. A szöveget a "Textbox" létrehozása után tudtuk hozzáadni "insert"-et használva. Ezután a "Textbox"

állapotát "DISABLED"-re állítottuk. Ez azért volt szükséges mivel a "Textbox" segítségével a program futása közben írhatunk és változtathatjuk a megadott szöveget. A "DISABLED" miatt már nem lehet változtatni a szöveget vagy plusz szöveget beírni.

## 2. Lábjegyzék

A lábjegyzék a "labjegyzek" függvény meghívásával készül el. Ez egy "Label" segítségével készült, amelyhez a szövegnek a programozói csapattagok nevei vanna megadva egy-egy "l" jellel köztük.

## 3. Gombok

A "gombok" függvényt hívjuk meg az "Oktató" és "Interaktív" gombok létrehozásához. Ez után a meghívás után hívjuk meg az "x\_gomb" függvényt, ami a bezárás gombot hozza létre.

### Bezárás gomb

Az "x\_gomb" meghívásakor 2 paraméterátadás történik. Az első a "root", ami az az ablak, amin meg szeretnénk jeleníteni a gombot. Ez jelen esetben a fő ablakunk.

A második paraméter a "bg\_color\_here", ami a gomb háttérszínéért felel. Így akármilyen háttérű oldalnál megjeleníthető (és megjelenítsük) a bezárás gombot. A bezárás gomb a projekt minden oldalán megjelenik és mindenhol ezt a függvényt hívjuk meg.

```
def x_gomb(root,bg_color_here):
    end_button=customtkinter.CTkButton(
        root,
        width=50,
        height=50,
        text= "x",
        text_color="white",
        command=lambda:root.destroy(),
        font=("Helvetica",30,"bold"),
        fg_color=bg_color_here,
        bg_color=bg_color_here,
        hover_color="red",
    )
    end_button.place(x=root.winfo_screenwidth()-50)
```

A cél az volt, hogy bármilyen háttérszínű oldalnál a gombnak egy átlátszó háttérű érzetet adjunk. A gomb működéséhez tartozik, hogy "hover\_color" pirosra állítva mikor az egér "cursor"-t a gombra húzzuk a háttere pirosra vált.

## Oktató és Interaktív gomb

A "gombok" függvényt megnyitásával két gombot hozunk létre. Ezek a középben megjelenő "Interaktív" és "Oktató" gombok.

### A. Interaktív

Az "Interaktív" gombot az "oktato" függvényben a "button" hozza létre. Ezt is "customtkinter" segítségével testre szabtuk részletese a kép alapján. Az egér gombra vitelével a kurzor egy kattintásra mutató kurzorrá változik és a háttér a "#A71606" színre változik, ami a vörös egy árnyalata.

```
button = customtkinter.CTkButton(  
    root,  
    text=None,  
    bg_color="#dfcaae",  
    image=photoimage2,  
    command=interaktiv,  
    width=screen_width/4,  
    height=screen_height/4,  
    fg_color="#a28656",  
    font=("Helvetica", 30, "bold"),  
    compound="bottom",  
    cursor="hand2",  
    border_width=2,  
    border_color="black",  
    hover_color="#A71606"  
)  
  
button.place(x=screen_width/2+10, y=screen_height/3)
```

Kattintáskor az "interaktiv" függvényt hívja meg.

### B. Oktató

Az "Oktató" gombot az "oktato" függvényben az "oktato\_button" hozza létre. Ezt is "customtkinter" segítségével testre szabtuk részletese a kép alapján. Az egér gombra vitelével a kurzor egy kattintásra mutató kurzorrá változik és a háttér a "#A71606" színre változik ami a vörös egy

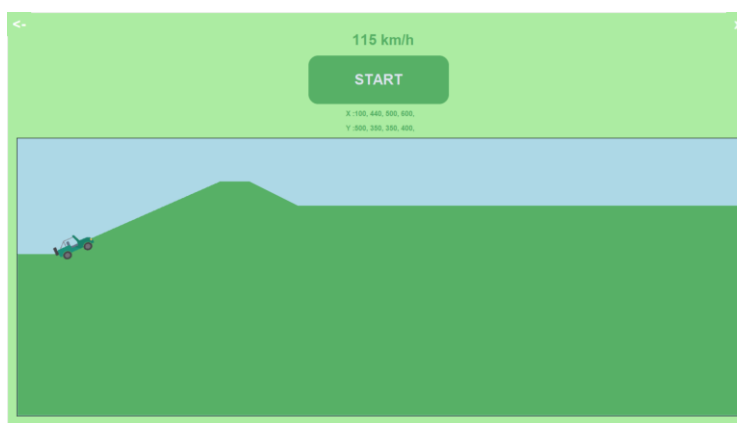
árnyalata. Kattintáskor az "oktato\_home" függvényt hívja meg.

```
oktato_button = customtkinter.CTkButton(  
    root,  
    text=None,  
    bg_color="#dfcaae",  
    command=oktato_home,  
    width=screen_width/4,  
    height=screen_height/4,  
    fg_color="#a28656",  
    font=("Helvetica", 30, "bold"),  
    image=photoimage1,  
    compound="bottom",  
    cursor="hand2",  
    border_width=2,  
    border_color="black",  
    hover_color="#A71606"  
)  
  
oktato_button.place(x=screen_width/4-10, y=screen_height/3)
```

## Interaktív oldal (működés)

### Tartalmaz:

1. Vissza gomb
2. Bezárás gomb
3. Adatokat lekérő szövegdobozok
  - a. Súly
  - b. Kezdősebesség
  - c. Koordináták
4. A koordinátákra vonatkozó szabályokat leíró szöveg
5. Megadott koordináták kiírása
6. Km óra
7. Játékfelület
8. Start gomb
- 9.



## Általános

### clear window

Az "interaktív" függvény meghívásával indul el a programunk Interaktív része. Kezdetben a "clear\_window" segítségével törölünk minden felhasznált és látható elemet az ablakunkról, ezáltal ezek nem fognak továbbra is látszódni az interaktív résznél és ugyanazt az ablakot ("root"-ot) használhatjuk, nem kell teljesen újat létrehozni.

```
def clear_window():
    # Destroy all widgets in the window
    for widget in root.winfo_children():
        widget.destroy()
```

Az elemek törlése a képen látható módon, egy "for" ciklussal történik meg. A "root.winfo\_children" tárolja a "root" oldalon lévő összes elemet felsorolva.

A "for" segítségével "widget" végigmegy ezeken az elemeken és a "destroy" törli azokat.

## Látható elemek

### 1. vissza gomb

Az "interaktiv" függvény meghívja a "vissza\_gomb" függvényt. Ezt a függvényt minden oldalon használtuk kivéve az első "home" oldalon.



Ezek a gombok a bal felső sarokban jelennek meg minden esetben.

Mindig az azelőtti oldalra visznek vissza, oda ahonnan a jelenlegihez értünk. (Ebben az esetben a "home" oldalra.)

A "vissza\_gomb" függvényt mivel sok különböző helyen és esetben kell meghívjuk ezért modularizáltuk, a pontos és változó paramétereket így paraméterátadással oldottuk meg. A függvény kap egy "root" egy "bg\_color" és egy "tohere" paramétert.

A "root" az előzőkhöz hasonlóan az az ablak, amiben meg szeretnénk jeleníteni a vissza gombot. A következő "bg\_color\_here" az a szín, amit a gomb nem aktív háttérének szeretnénk megadni, a mi esetünkben minden oldalnál az oldal háttérszíne

az átlátszó háttér hattás eléréséért.

```
def vissza_gomb(root,bg_color_here,tohere):
    vissza_button=customtkinter.CTkButton(
        root,
        width=50,
        height=50,
        text="<-",
        text_color="white",
        command=tohere,
        font=("Arial",30,"bold"),
        fg_color=bg_color_here,
        bg_color=bg_color_here,
        hover_color="red",
    )
    vissza_button.place(x=0,y=0)
```



A harmadik paraméter a "tohere" az a függvény neve, amit a visszagombbal meg szeretnénk hívni.

Az "interaktív" függvény elkészíti a start gombot, ami később lesz fontos. Jelenleg eltüntetve, deaktiválva készítsük el, szöveg nélkül, hiszen azt akarjuk, hogy csak később feltételek mellett jelenjen meg és akkor tudjuk majd használni.

```
def interaktiv():
    clear_window()
    canvas = tk.Canvas(root, width=width, height=height, bg="#aceca2")
    canvas.pack()
    #start button
    start_button=customtkinter.CTkButton(
        root,
        width=290,
        height=100,
        text= "",
        command=lambda:start_car(canvas,root),
        state=tk.DISABLED,
        fg_color="#aceca2",
        corner_radius=20,
        bg_color="#aceca2",
        font=("Helvetica", 30, "bold"),
    )
    start_button.place(x=center-145, y=90)
#end button
vissza_gomb(root,"#aceca2",home)
x_gomb(root,"#aceca2")
add_widgets_to_window(root,canvas,center,height,start_button)
```

## Látható és használható elemek

Az "add\_widget\_to\_window" függvény meghívásával készítjük el az összes látható elemet:

### 2. Adatokat lekérő szövegdozok

- a) Súly
- b) Kezdősebesség
- c) Koordináták

Mindhárom lekérő szövegdobozt "customtkinter" segítségével és "Entry"-kel készítettük el.

"insert"-et használtunk az "Entry" dobozoknak, az alapszövegének a megadásához.

Mindhárom dobozra állítottunk két eseményt. Az első azt figyel, hogy mikor kattint rá valaki a dobozra, a másik hogy mikor küld be értéket az "Enter" lenyomásával.

Ezeket "bind" segítségével készítettük el.

```
#Bind of the buttons
text_box2.bind("<FocusIn>", add_kg)
text_box2.bind("<Return>", check_weight)

text_box.bind("<FocusIn>", add_km_per_hour)
text_box.bind("<Return>", check_entry)

text_road.bind("<FocusIn>", add_cords)
text_road.bind("<Return>", check_cords)
```

### **A,B. Súly, Sebesség**

A súlyt (sebességet) lekérő "Entry" dobozra kattintva a "bind" és azon belül a "<FocusIn>" segítségével az "add\_kg" ("add\_kg\_per\_hour") függvényt hívja meg.

```
def add_kg(event):
    if " kg" or "Error" in text_box2.get():
        text_box2.delete(0, tk.END)
```

Ez a függvény az előre beírt szöveget tünteti el a "delete" segítségével. A lényeg az, hogy ha egy adott szöveg szerepel a dobozban, akkor azt eltünteti. Ezért itt a "kg" ("KM/H") és "Error" szövegeket nézi. Az alapból megadott szöveg tartalmazza a "kg" ("KM/H") szöveget, és minden, a program futtatója által helytelenül beírt szöveg után a szövegdobozban megjelenik az "Error" szöveg, így

mindkét esetben kattintás után eltűnik a szöveg és megürül a hely az adatok beírásához.

Egy érték beküldésekor a "check\_weight" ("check\_entry") függvényt hívja meg.

```
def check_weight(event=None):
    global car_weight, ok_entry
    try:
        weight = float(text_box2.get())
        car_weight = weight
        text_box2.delete(0, tk.END)
        text_box2.insert(tk.END, f"{weight} kg")
        ok_entry[0]=1
        if ok_entry[0]+ok_entry[1]+ok_entry[2]==3:
            start_appear(window, canvas, start_button)
    except ValueError:
        text_box2.delete(0, tk.END)
        text_box2.insert(0, "Error")
        ok_entry[0]=0
```

Itt, ha nem kap "Error"-t, akkor lementi a beküldött értéket a ".get" segítségével és elmenti a "string"-ből "float"-ba átalakított értéket a "car\_weight" ("car\_speed")-be. Ezután az "insert" segítségével a beírt érték után "kg" ("km/h") -ot ír.

Ha "Error"-t kap akkor az "except" részre ugorva helyettesíti a beírt értéket az "Error" szöveggel

### C. Koordináták (4,5)

A koordinátáknál is hasonlóképpen eltüntetjük a benne írt szöveget mikor a program futtatója rákattint az "Entry" dobozra.

A különbség a beérkezett válaszoknál jelentős.

A koordináták beírásának szabályát a "display\_rules" "Label" jeleníti meg.(4)

A beküldött válasznál meghívjuk a "check\_cords" függvényt.

A "cords" változóban tároljuk a beküldött érték "int" alakját.

A pályát és az autót egy külön felületen (téglalapban) jelenítjük meg.

Ennek a koordinátáit a képernyő méreteire szabjuk figyelembe véve, hogy jobb-, baloldalon és alól ne érjen el a képernyő széléig és felül ne érjen el a kiírt adatokig.

A beadott koordinátákat leellenőrizzük, hogy beleesnek ebbe a téglalapba és csak akkor fogadjuk el, ha igen. Ezeket beérkezés pillanatában a "display\_x" és "display\_y" (5) "Label"-ekbe írjuk ki és mentjük a "path\_points" kétdimenziós listába "insert" használva. Az elfogadott koordináták számát a "road\_cords\_number" változóba mentjük, mivel 0-tól indítjuk a változót, így tudjuk, hogy minden páros értékénél egy X és minden páratlan értékénél egy Y koordináta érkezett be. Emellett ha a változó értéke 8, tudjuk, hogy az összes koordináta beérkezett és ezután nem fogad el egy beírt koordinátát sem.

```
def check_cords(event=None):
    global ramp, road_x, road_y, road_cords_number, path_points, path_points_x, road_cords_ok, ok_entry
    try:
        cords = int(text_road.get())
        if (
            (road_cords_number%2==0 and cords>50 and cords<window.winfo_screenwidth()-50)
            or
            (road_cords_number%2==1 and cords>250 and cords<window.winfo_screenheight()-50) and road_cords_ok==1:
            text_road.delete(0, tk.END)

            #display the road
            if road_cords_number%2==0:
                path_points_x=cords

                road_x+=str(cords)+", "
            else:
                path_points.insert(road_cords_number, (path_points_x, cords))
                road_y+=str(cords)+", "

            road_cords_number+=1
            if road_cords_number==8:
                road_cords_ok=0
                ok_entry[2]=1
                if ok_entry[0]+ok_entry[1]+ok_entry[2]==3:
                    start_appear(window, canvas, start_button)

                display_x.configure(text=road_x)
                display_y.configure(text=road_y)
            |
        except ValueError:
            text_road.delete(0, tk.END)
            text_road.insert(0, "Error")
```

Ha a program futtatója eleget tesz mindhárom érték adás feltételeinek és elég értéket ad meg, és a hármat akármilyen sorrendbe végzi el, akkor változik az oldal a szimulációhoz.

## 6.Km óra

A szimuláció elkezdésekor meghívjuk a "start\_appear" függvényt, amivel az "Entry" dobozok és a koordináták beírásának szabályait leíró szöveg eltűnnek. És "place"-eljük a km órát. (6)

```
text_box.destroy()
text_box2.destroy()
text_road.destroy()
display_rules.destroy()
display_speed.place(x=root.winfo_screenwidth()/2-55, y=40)
```

## 7.Játékfelület

Létrehozzuk a kék háttérű felületet. Ebben fog a jármű közlekedni. Az út koordinátái ennek a téglalap belsejében kellett legyenek.

```
def start_appear(root,canvas,start_button):
    #DESTROY
    text_box.destroy()
    text_box2.destroy()
    text_road.destroy()
    display_rules.destroy()
    display_speed.place(x=root.winfo_screenwidth()/2-55, y=40)

    car_window=canvas.create_rectangle(20,260,root.winfo_screenwidth()-20,root.winfo_screenheight()-30, fill="lightblue")
    path_points.pop(0)
    start_button.configure(
        state=tk.NORMAL,
        text= "START",
        fg_color="#57b066",
        font=("Helvetica", 30, "bold"),
    )
```

A "path\_points" első elemének a törlésére ("pop") azért volt szükség, mivel 2 elemű lista deklarálásakor meg kell adni az első elemet, de nem akartuk, hogy az általunk megadott elem befolyásolja az utat, így a koordináták után töröltük azt, amit mi adtunk meg. A "start\_button"-t változtattuk, hogy megjelenjen az.

## 8. Start gombkattintás

Kattintáskor a "start\_car" függvényt hívjuk meg.

```
def start_car(canvas,root):
    #Path follower
    x,y=path_points[0]
    x_end,y_end=path_points[-1]
    path_points.insert(road_cords_number,(root.wininfo_screenwidth()-71,y_end))
    x,y=path_points[0]
    ground=[(root.wininfo_screenwidth()-21,y_end),(root.wininfo_screenwidth()-21,root.wininfo_screenheight()-31),
            (22,root.wininfo_screenheight()-31),(21,y)]+path_points

    canvas.create_polygon(ground,fill='#57b066')
    path_follower = PathFollower(canvas, path_points)
    path_follower.draw_car(x,y)
    path_follower.draw_path()
    path_follower.move_car_along_path(root)
    #path_follower.draw_car(x_end,y_end)

    root.mainloop()
```

Lementjük a megadott pontoknak az első és utolsó x és y koordinátáját, azért hogy tudjuk, honnan kell induljon az autó és hogy megrajzoljuk a talajt ("create\_polygon"). A "ground" koordinátái azok amik által rajzoljuk meg a földet és a "path\_points" az ahol az autó közlekedik. Az "insert"-elt elem azért kellett, hogy az utolsó pontnál ne álljon meg az autó, hanem menjen ki a kép leg baloldali pontjáig. Az volt a cél hogy a talaj vízszintesen legyen a kezdő pontig, utána a megadott pontok szerint összekötve bármilyen út (alakzat) készüljön, és ezután az utolsó ponttól megint vízszintesen a "játékfelület" leg jobboldalibbig vízszintesen. Ez alatt mindenhol kitöltve a zöld színnel, ami miatt kellett a "ground" pontjaiba megadni a két alsó sarkot is.

## A szimuláció

Általános :

```
path_follower = PathFollower(canvas, path_points)
```

A "PathFollower" tartalmazza az összes, a szimulációhoz szükséges változóka és függvényeket.

## Tartalmaz:

1. Az autó megjelenítése
2. Az út megrajzolása
3. Az animáció (az autó végighalad az úton)
  - I. Az út felosztása
    - A. Távolság
    - B. Szög
    - C. Gyorsulás
    - D. Időzítő
    - E. Kép elforgatása
    - F. Lépésekre osszuk
      - a. Következő lépésre lép
      - b. Sebesség változás, kiírás
      - c. Biztonság (kilépések, nincs negatív sebesség)

## 1. Az autó megjelenítése

”PathFollower.draw\_car” függvénnyel hívható meg.

```
def draw_car(self, x, y):  
    # Correct way to load the image using Pillow (PIL)  
    self.image_path = r"Kepek\kek_car.png"  
    # Use a raw string to handle backslashes  
    self.image = Image.open(self.image_path)  
  
    # Resize the image to fit within the 200x300 rectangle  
    self.image = self.image.resize((90, 80))  
    self.rotated_image = self.image.rotate(0)  
    self.photo = ImageTk.PhotoImage(self.rotated_image)  
  
    # Place the image inside the rectangle  
    self.car=self.canvas.create_image(x, y, image=self.photo)
```

- Megívjuk a képet a link segítségével
- Preferált méretre vágjuk
- 0 fokos szögbe fordítjuk el
- Létrehozzuk a kezdőkoordinátán az autót

## 2. Az út megrajzolása

```
def draw_path(self):  
    self.path = self.canvas.create_line(self.path_points, fill="#57b066")
```

- A megadott "path\_points"-ban lévő koordináták alapján rajzolja ki az utat

## 3. Az animáció

Az animáció a "PathFollower.move\_car\_along\_path" függvény meghívásával kezdődik el.

### I. Az út felosztása

Az egész, hosszú utat egyenes, két pont közötti szakaszokra osszuk, azokkal fogunk dolgozni.

```
for i in range(len(self.path_points) - 1):  
    start_x, start_y = self.path_points[i]  
    end_x, end_y = self.path_points[i + 1]
```

#### A. Távolság

Püthagorasz tétellel kiszámítható

```
distance = math.sqrt((end_x - start_x) ** 2 + (end_y - start_y) ** 2)
```

#### B. Szög

Minden egyenesnél az x tengellyel bezárt szöget számolunk ki, amit a gyorsulás kiszámítására használunk fel.

Mivel tudunk két oldalt, ezt szögfüggvények segítségével egyszerűen meghatározható

```
degree = -math.degrees(math.atan2(end_y - start_y, end_x - start_x))
```



### C. Gyorsulás

A gyorsulás kiszámításához a "PathFollower.calculate" függvényt kell meghívni.

```
def calculate_acceleration(self, theta_deg):  
  
    g=9.81  
    mu=0.05  
    theta_rad=math.radians(theta_deg)  
    acceleration=g*(math.sin(theta_rad)+math.cos(theta_rad)*mu)  
    return acceleration
```

A függvényhez a "theta\_deg" változóban a szög mértékét adjuk át, amit az előzetesen kiszámoltunk. A "g" a gravitációs állandó és a "mu" a megadott guruló súrlódási állandó, aminél most egy körülbelüli értéket vettünk a száraz aszfalt esetén.

Ezeket és az " $a=g*(\sin\alpha + \cos\alpha * \mu)$ " képletet használva ki tudtuk számolni a gyorsulást bármilyen szögű emelkedő vagy lejtő esetében.

Ezt " $\text{m/s}^2$ "-ben kapjuk meg, így a sebességet is átalakítva használjuk.

```
self.speed = car_speed*0.27778
```

(A számításba figyelembe vettük a guruló súrlódást is.)

### D. Időzítő

Időzítőt indítunk a "TimerThread.start" függvény meghívásával.

Ez az időzítő, a másodpercenkénti sebességváltozásba segít majd.

```
#Timer start  
second=0  
timer_thread = TimerThread()  
timer_thread.start()
```

A "TimerThread" meghívásával létrehozunk egy olyan másodperc számlálót, amely szinkronban tud futni, a "threading.Thread" segítségével, a program további soraival.

```
class TimerThread(threading.Thread):
    def __init__(self):
        super().__init__()
        self.seconds = 0
        self.running = True

    def run(self):
        while self.running:
            time.sleep(1)
            self.seconds += 1

    def stop(self):
        self.running = False
```

A "start" meghívásával a "run" fut le. Itt a "time.sleep" segítségével 1 másodpercenként jut a program a "self.seconds +=1" részhez. Ezt addig csinálja, ameddig meg nem hívjuk a "TimerThreading.stop" függvényt.

### E. Kép elforgatása

A képet a meghatározott szögbe forgatjuk el.

```
#Rotate image
self.rotated_image = self.image.rotate(degree)
```

### F. Lépésekre osszuk

```
steps = int(distance)
```

"steps"-be két pont közti távolság lesz lementve. Ebbe 1 lépés 1 pixel lesz és lépni az ezt követő forral fogunk.

```
for step in range(steps):
```

#### a. Következő lépésre lép

- Az "interp\_x" és "interp\_y" adja meg a következő lépés koordinátáit

```
for step in range(steps):
    interp_x = start_x + (end_x - start_x) * (step / steps)
    interp_y = start_y + (end_y - start_y) * (step / steps)
```

- Erre a koordinátákra készíti el a képet az autóról és frissíti a megjelenő képet és helyzetét az újra

```
self.photo = ImageTk.PhotoImage(self.rotated_image)
self.car=self.canvas.create_image(interp_x, interp_y, image=self.photo)
self.canvas.update()
```

## b. Sebesség változás, kiírás

- Ha az időzítő elindítása után 0ra állított "second" kisebb, mint a "timer\_thread.second", ami másodpercenként nő egyel, akkor változik a sebesség is. Így értjük el azt, hogy másodpercenként változzon a sebesség, a meghatározott gyorsulással.

```
if timer_thread.seconds > second:
    self.speed = self.speed + acceleration

    second=timer_thread.seconds
```

- "delay"-t használva érjük el azt, hogy az autó animáció szerűen mozogjon. A "delay" leteltével fut tovább a program, így mutatjuk be a sebességváltozást. A "delay"-elt idő a sebességtől függ szóval, ha gyorsabban halad az autó, akkor kevesebb a "delay", és ahogy lassul annál nagyobb.

```
delay = int(100 / self.speed)
```

```
self.canvas.after(delay)
```

- A képernyőre kiírjuk a jelenlegi sebességet a biztonsági feltételek leellenőrzése után.

```
display_speed.configure(text=f"{str(int(self.speed/0.27778))} km/h")
```

A "self.speed" m/s be van és mivel mi km/h-be szeretnénk kiírni, ezért van szükség az átalakítására, így osszuk 0.27778-el. Hogy ne legyen tizedes érték ezért alakítjuk "int" alakba.

### c. Biztonság

```
if self.speed <= 1:  
    ok = 1  
    break
```

```
if ok == 1:  
    break
```

Ha a "self.speed" lemegy 1 alá, akkor azt már elhanyagolhatónak vesszük és kilépünk az összes ciklusból. Ez azt jelenti, hogy megállt az autó.

Ha ez az ellenőrzés előtt változtatjuk a kiírt jelenlegi sebességet, akkor megeshet, hogy negatív sebességet kéne kiíratni. Ha utána, akkor, ha kilép a ciklusból, mert megállt akkor nem lesz frissítve 0-ra a sebesség és megáll egy nagyobb sebességen a mutató.

Ezeket a problémákat egy utolsó ellenőrzéssel küszöböljük ki.

```
if self.speed < 0 or ok == 1:  
    self.speed = 0  
display_speed.configure(text=f"{str(int(self.speed/0.27778))} km/h")
```

## **5. Fejlesztési lehetőségek**

### **Interaktív rész**

- Valóságghűbbé tenni a kocsi gyorsulását és lassulását úgy, hogy a sebessége egyenletesen változik, a mostanival ellentétben, amikor másodpercenként változik a sebessége, vagyis, ha 20 km/h -val megy akkor egy mp. múlva egyből 15 km/h lesz a sebessége
- A súrlódás befolyásolása:
  - a kerek változtatása (pl. hold, mocsár)
  - a pálya/terep változtatása
- Egy vonalzót betenni arra a területre, ahol megy a kocsi, hogy könnyebben és átláthatóbban tudják a koordinátát az útnak állítani (a terület tetejénél lenne a 0,0 koordináta)
- Akadályok hozzáadása a pályához, amiken áttör a kocsi, ha elég gyorsan megy
- Szöktető hozzáadása, adott esetben a kocsi ki is tudna repülni
- Boosterek hozzáadása a kocsához (pl. rakéta, hogy gyorsabban menjen)

### **Oktató rész**

Ebből a szinte végtelen témából kibővíteni ezt a részt, újabb gombok, oldalak hozzáadásával (pl. rugalmasság, rugók, a mechanika 3 alaptörvénye)

## 6. Szakirodalom

[ChatGPT](#)

<https://kozmatamas.netlify.app/>

<https://www.w3schools.com/python/>

<https://www.youtube.com/TkinterPython>

## Tartalomjegyzék

1. Témaindoklás .....	2
2. Rendszer követelmények.....	2
2.1. Hardver igény .....	2
2.2. Szoftver igény .....	2
3. Felhasználói kézikönyv.....	3
4. Programozói kézikönyv.....	11
5. Fejlesztési lehetőségek .....	29
6. Szakirodalom.....	30