GAMIFIKACIJA PROCESA NUKLEARNE FUZIJE

Lassinger, Igor

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ODJEL ZA FIZIKU

IGOR LASSINGER

GAMIFIKACIJA PROCESA NUKLEARNE FUZIJE

Završni rad

Osijek, 2023.

„Ovaj završni rad je izrađen u Osijeku pod vodstvom doc.dr.sc. Ivana Vazlera u sklopu Sveučilišnog preddiplomskog studija fizike na Odjelu za fiziku Sveučilišta Josipa Jurja Strossmayera u Osijeku“.

Sveučilište Josipa Jurja Strossmayera u Osijeku
Odjel za fiziku

Završni rad

GAMIFIKACIJA PROCESA NUKLEARNE FUZIJE

IGOR LASSINGER

SAŽETAK

Nuklearna fuzija trenutno je najefikasniji proces dobivanja energije, ali ljudima trenutno najmanje dostupan. Razumijevanjem procesa fuzije pomaže razumijevanju života zvijezda, njihovoj prošlosti, a tako i budućnosti, koja može poboljšati razumijevanje energije u svakom aspektu znanost. Interaktivnim sadržajem o procesu fuzije možemo razumjet sam proces na osnovnoj bazi.

Ključne riječi: fuzija / zvijezde / interaktivni sadržaj / Python

University Josip Juraj Strossmayer of Osijek Bachelor Thesis

Department of Physics

GAMIFICATION OF THE NUCLEAR FUSION PROCESS

IGOR LASSINGER

SUMMARY

Nuclear fusion is currently the most efficient process of obtaining energy, but it is currently the least accessible to people. Understanding of the process of fusion helps to understand the life of stars, their past, and thus their future, which can improve the understanding of energy in every aspect of science. With interactive content simulating the fusion process, we can understand the process itself effortlessly.

Key words: fusion / stars / interactive content / Python

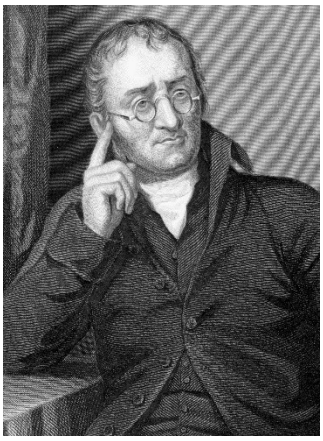
SADRŽAJ

1. UVOD

U ovom završnom radu govorit ćemo o računalnoj igri napravljenoj pomoću programskog jezika Python s temom nuklearne fuzije. Da se objasni način na koji je igra programirana, potrebno je objasniti neke pojmove poput nuklearnog raspada, fuzije kao i programskih petlji i modula Pythona. Razmotriti ćemo uloge različitih djelova programa te međusobno povezivanje i slanje informacija unutar samog programa. Gamifikacija je proces koji događajima ili procesima koje nisu direktno povezani s igrama, implementira princip igara [1]. Popularizacija video igara značajno utječe na dodjeljivanje resursa, što novčanih, što računalnih, u unaprijeđenje i inovaciju računalnih i multimedijskih sektora znanosti. Gamifikacijom prirodnih događaja poput nuklearne fuzije, možemo poticati nove ili skrivene vještine korisnika u razvijanje i informatičko tehnološkom i prirodoslovno znanstvenom spektru.

2. ATOM

Atom je najmanja sastavna jedinica materije koja čini bilo koji kemijski element. Do osnutka teorije atoma vjerovalo se da je cijeli svijet napravljen od četiri elementa: vatra, voda, zemlja i zrak. U 5. st. pr. Kr. grčki filozof Demokrit daje ideju o atomu. Grč. *atomos* ili „nedjeljiv“ postaje glavna teorija o nastanku svijeta koja biva zaboravljena početkom srednjeg vijeka. Kasnije u 17. st. postaje opet aktivna tema. Početkom 19. st. John Dalton (Slika 1) dokazuje postojanje atoma objašnjavajući kako elementi uvijek reagiraju u određenim omjerima. Kasnije je otkriveno da atom nije nedjeljiv kao što je Demokrit tvrdio. Otkriće subatomskih čestica, a kasnije i otkriće kvarkova dokazalo je da atom nije najmanja sastavna jedinica materije.



Slika 1 - John Dalton

Jezgra atoma

Slika 2 - Model atoma

Atomska jezgra je malo, gusto područje koje se sastoji od protona i neutrona u središtu atoma. 1909. Hans Geiger i Ernest Marsden, pod vodstvom Ernesta Rutherforda, bombardirali su metalnu foliju α-česticama kako bi promatrali kako se raspršuju. Očekivali su da će sve alfa čestice proći ravno kroz malo odstupanje, jer je Thomsonov model rekao da su naboji u atomu toliko difuzni da njihova električna polja ne bi mogla utjecati na α-čestice. Međutim, Geiger i Marsden uočili su nekolicinu alfa čestica koje se odbijaju pod kutom većim od 90°, što je prema Thomson-ovom modelu trebalo biti nemoguće. Kako bi to objasnio, Rutherford je predložio da se pozitivni naboj atoma koncentrira u sićušnoj jezgri u središtu atoma (Slika 2). To je opravdalo ideju atoma s gustim središtem pozitivnog naboja i mase.
Jezgra atoma sastoji se od neutrona i protona, što je manifestacija više elementarnih čestica, nazvanih kvarkovima, koje jaka nuklearna sila drži spojene. Svojstva samih atoma, te i njihovo međudjelovanje, ovisi ponajviše o broju protona što omogućuje razlikovanje pojedinih elemenata s različitim fizičkim i kemijskim svojstvima [2].
Svakako treba napomenuti da pojedini elementi mogu imati različiti broj neutrona u jezgri tako da atome s različitim brojem neutrona nazivamo **izotopima** pojedinih elemenata tipično

označeni s brojem pored slova elementa npr.: ^{13}C za ugljik s 6 protona i 7 neurona.

Radioaktivni raspad

Radioaktivni raspad je spontani raspad **nestabilne jezgre** atoma što rezultira oslobađanjem energije i materije iz jezgre. Neki izotopi imaju nestabilne jezgre koje nemaju dovoljno visoku energiju vezanja da bi jezgru držala na okupu. Zbog velikog broja protona u jezgri koji se međusobno odbijaju električnom silom, potreban je isti ili veći broj neutrona koji će držati protone na okupu te ih vezati u stabilnu jezgru [2][3]. Nestabilni izotopi uvijek teže prema većoj stabilnosti jezgre pa se neprestano mijenjaju, odnosno raspadaju kako bi se pokušali stabilizirati. U procesu će izotopi oslobađati energiju i materiju iz svojih jezgara nakon čega se često pretvaraju u novi element. Radioaktivne raspade možemo podijeliti po vrsti čestice nastale nakon raspada, tako da postoje:

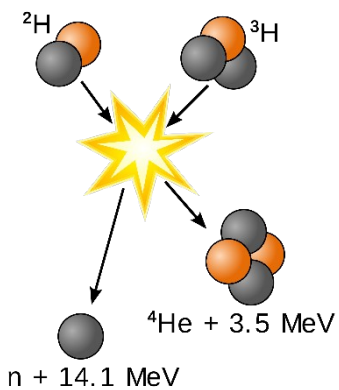
- α - raspad
    - Raspad koji je veoma čest za teže elemente (svi elementi koji imaju više od 67 protona imaju barem jedan izotop koji se raspada α - raspadom). α - raspadom iz jezgre izlazi α - čestica načinjana od 2 protona i 2 neutrona
- β^+ - raspad
    - Raspad samog protona u jezgri koji dolazi zbog visokog broja protona bez dovoljno neutrona da stabilizira jezgru te se taj proton raspada na pozitron (e^+), neutron (n^0) i elektronski neutrino (ν_e)
- β^- - raspad
    - Raspad neutrona u jezgri koji dolazi zbog visokog broja neutrona u jezgri te se isti neutron raspada na proton (p^+), elektron (e^-) i anti neutrino ($\overline{\nu_e}$)
- EC (eng. *Electrone capture*) elektronski uhvat
    - Proces u kojemu jezgra atoma "hvata" jedan elektron iz unutrašnje ljuske, koji se eventualno sudara s protonom nakon čega nastaje neutron (n^0).
    - Slobodan prostor nastao u unutarnjoj ljusci atoma dopušta spuštanje elektrona iz vanjske ljuske nakon čega dolazi do emisije x-zraka odnosno rendgenskog elektromagnetnog zračenja.
- p^+ - emisija
    - Rijetka pojava u kojoj proton direktno izlazi iz jezgre kako bi se ista stabilizirala.
    - Najčešće se događa kod laganih elemenata npr.: ^{5}Li koji izbacuje jedan proton da nastane stabilna jezgra ^{4}He

- n^0 - emisija
    - Proces u kojem jezgra izotopa s velikim brojem neutrona izbacuje neutron kao posljedica Paulijevog principa, tj. zbog određene količine energije koja je dozvoljena u jezgri, viškom neutrona remeti se stabilnost jezgre te se jezgra atoma raspada ili se emitira neutron radi stabilizacije jezgre
- Spontana fisija
    - Raspad koji "cijepa" jezgru atoma bez poticaja vanjske čestice kao posljedica velike količine odbojnih sila unutar jezgre
    - Spontana fisija u prirodi se događa jako rijetko, najčešće kod umjetno nastalih elemenata kod kojih zbog velikog radijusa jezgre i velike količine protona, neutroni ne mogu u potpunosti sačuvati stabilnost jezgre
- γ - raspad
    - Raspad koji nastaje kao posljedica viška energije nakon jednog od gore navedenih raspada, najčešće nakon β ili α raspada.
    - Kako bi novonastala jezgra sačuvala stabilnost iako s visokom energiju nakon raspada, jezgra mora odbaciti višak energije te emitira visoko energetske elektromagnetske valove tj. γ - valove ili γ - zrake.

Uz navedene raspade postoje još i brojni drugi raspadi koji nisu toliko česti [3].

Nuklearna fuzija

Nuklearna fuzija (Slika 3) je reakcija u kojoj se dvije ili više atomskih jezgara kombiniraju kako bi stvorile jednu ili više različitih atomskih jezgara i subatomskih čestica (neutrona ili protona). Razlika u masi između reaktanata i produkata očituje se ili kao oslobađanje ili apsorpcija energije. Ova razlika u masi nastaje zbog razlike atomske energije vezanja između atomskih jezgara prije i nakon reakcije. Kako bi došlo do fuzije elemenata, potrebna je visoka energija pojedinih čestica koje sudjeluju u fuziji. Zbog same veličine jezgri atoma ($\approx 10^{-16}$ m) potrebna je velika količina čestica i velika brzina kako bi uopće došlo do sudara [4]. Velika količina čestica odgovara visokom tlaku, a velika brzina visokoj temperaturi, tako da najpogodniji uvjeti za nuklearnu fuziju su jezgre zvijezda koje mogu imati temperaturu do nekoliko milijuna °C.



Slika 3 - Nuklearna fuzija

3. PYTHON

Programski jezik Python je fleksibilan, visoko namjenski programski jezik korišten najčešće u znanstveno istraživačke svrhe. Jednostavan za korištenje i lagan za razumjeti. Python se razlikuje od drugih programskih jezika poput C++ ili C jezika u smislu da radi na bazi "linija po linija" odnosno kod napisan u Pythonu se prevodi i izvršava redom naredba po naredba kako je napisan za razliku od navedenih jezika koji cijeli kod u komadu kompajliraju (*eng. compile = sažeti, skupiti*) u strojni jezik koji se tada može izvršavati više puta bez ponovnog kompajliranja. Po načinu kako Python izvršava svoj kod on spada u interpretativne programske jezike (*eng. interpret = prevesti, protumačiti*). Takvi jezici generalno su fleksibilni te jako modularani (više o tome u poglavlju Python moduli), no prevođenje koda liniju po liniju može usporiti izvršavanje programa [5].

Programske varijable i petlje

Varijable su vrijednosti koje se tijekom kalkulacije mogu mijenjati, odnosno neka veličina promjenjive vrijednosti. Varijable mogu poprimati vrijednosti različitih tipova podataka poput brojeva (cijelih, decimalnih ili kompleksnih), skupa znakova, logičkih uvjeta, listi, skupova i drugih. Promjena vrijednosti varijable se generalno u programu ne događa samo jednom, a svaka promjena, ako je zapisana dugačkim formulama, može znatno utjecati na veličinu, čitljivost i brzinu izvođenja programa [6]. Zbog tog razloga postoje programske petlje koje će izvršiti isti kod više puta i tako pojednostaviti postupak pisanja i povećati efikasnost izvođenja programa. Npr.: ova suma (1) može biti napisana pomoću petlji {1}.

$$S = \sum_{i=1}^{10} A + iB \tag{1}$$

{1}

```
S = 0
for i in range(1,11):
    S += (A + i * B)
```

Funkcije, klase i metode

U Python-u, funkcije su jedni od najvažnijih djelova programa. Neki programski jezici specifično razlikuju funkcije od podrutina u smislu da funkcije uvijek vraćaju neku vrijednost dok podrutine obavljaju neki zadatak [7]. U Python-u to nije tako te da bi funkcije mogle samo obavljati zadatke ne moraju biti naznačene kao podrutine, kao što će biti vidljivo u nastavku. Funkcije definiramo pomoću ključne riječi `def` nakon koje slijedi ime funkcije te, u zagradama, parametri koje funkcija zahtjeva za korištenje. Nakon zagrada dolazi znak dvotočke ":" koji označuje početak uvučenog bloka naredbi koje poziv funkcije izvršava. U Python-u je svaka varijabla, vrijednost ili funkcija zasebni objekt koji sadrži zasebna svojstva i metode kojima program može pristupiti pomoću notacije "." npr.:
`ID = lista.id(vrijednost)`

Ponekad standardni objekti u Pythonu-u nisu dovoljni pa se može definirati nove klase objekata s proizvoljnim atributima i metodama pomoću ključne riječi `class` {2}. Nova klasa objekata može imati svoje atribute i metode. U primjeru {2} je korištena `__init__` metoda koja je dio Python programa namijenjena za inicijalizaciju objekata, tj. postavljanje početnih vrijednosti atributa objekata.

{2}

```
class Drvo:
    def __init__(self, vrsta, visina, starost):
        self.vrsta = vrsta
        self.visina = visina
        self.starost = starost

D_001 = Drvo("Hrast", 7.8, 40 )
D_002 = Drvo("Breza", 4.0, 15)

print("Breza je visoka: ", D_002.visina, " m ")


>>> Breza je visoka 4.0 m
```

Python moduli

Unutar Python programa je ugrađena standardna knjižnica koja sadrži funkcije, klase i druge objekte koji se češće koriste kako ne bi smo morali uobičajene dijelove programa definirati svaki puta iznova. Kako bi pristupili tim naredbama i funkcijama potrebno ih je uvesti u program. Takve skripte imaju zajednički naziv moduli. Moduli {3} su unaprijed napisane skripte koje su napravljene kako bi samo programiranje bilo brže i jednostavnije [8]. Takvi moduli mogu biti i vlastoručno napravljeni. Instalirane module uvozimo u program pomoću ključne riječi `import` koja se obično nalazi na početku programa.

Slika 4 - Python logo

{3}

```
import random

lista = ["jabuka", "kruška", "breskva", "banana"]
košara = random.choice(lista)

print(košara)

>>> kruška
```

4. UČITAVANJE PODATAKA

Osnove baze podataka

Baza podataka je centralno mjesto informacijskog sustava. Pohranjeni podaci u bazi podataka opisuju. Svrha baze je prikupljanje, obrada, pohranjivanje i distribucija informacija, koje su potrebne za praćenje rada i upravljanje organizacijskim sustavom ili nekim njegovim podsustavom [9].

Baza podataka igre (tablica izotopa)

Program igre pristupa tablici izotopa pomoću modula `openpyxl` koji u ovom slučaju komunicira s tablicom iz datoteke u .xlsx formatu kako bi podatke iz tablice učitali u program`{4}`.

`{4}`

```
...
ID_chart_workbook = openpyxl.load_workbook(ID_chart_path)
ID_chart_sheet = ID_chart_workbook.active
ID_chart_isotopes = []
ID_chart_list = []
decay_list = []

for i in range(2, 3186):
    ID_chart_row = []
    for j in range(1, 19):
        ID_chart_cell = ID_chart_sheet.cell(i, j)
        ID_chart_row.append(ID_chart_cell)
    ID_chart_isotopes.append(ID_chart_row)
    loading_screen(i / 32)
...
```

Tablica izotopa i pridruženih nuklearnih raspada

Nekolicina izvučenih primjera iz baze podataka:

ID	Name	Z	A	Stability	Fusion stability	Alpha	Beta+	Beta-	Proton emission	Neutron emission	Electron capture	Spontaneous fission	Delayed Alfa	Delayed proton emission	Delayed neutron emission	Star energy	Star core
0	Neutron	0	1	1	1											1	
1	Hydrogen – 1	1	1	1	1											1	
2	Hydrogen – 2	1	2	1	1											1	
3	Hydrogen – 3	1	3		1			9								1	
4	Hydrogen – 4	1	4		1					2						1	
5	Hydrogen – 5	1	5							3						1	
6	Hydrogen – 6	1	6							4						1	
7	Hydrogen – 7	1	7							5						1	
8	Helium – 2	2	2				2									1	
9	Helium – 3	2	3	1	1											1	
10	Helium – 4	2	4	1	1											1	
11	Helium – 5	2	5							10						1	
12	Helium – 6	2	6					20								1	
13	Helium – 7	2	7							12						1	
14	Helium – 8	2	8		1			22								1	
15	Helium – 9	2	9							14						1	
16	Helium – 10	2	10							15						1	
17	Lithium – 3	3	3						8							1	

Svakom elementu i njegovom izotopu pridruženi su protonski i atomski broj, stabilnost jezgre, mogućnost ulaska u fuziju, odgovarajući raspadi te produkti istih raspada, zvijezda potrebna za fuziju te izotopi potrebni za progresiju. Podaci iz tablice prikupljeni su i kompajlirani iz raznih izvora [3][10][11][12][13][14].

5. PRAKTIČNI RAD - PROGRAMIRANJE IGRE

Zadatak ovoga rada je napraviti računalnu igru u Python programskom jeziku. Igra se zasniva na pojednostavljenoj prirodi nuklearne fuzije te je glavni cilj igre otkriti što više različitih izotopa. Nove izotope otkrivamo tako da, pomoću fuzije, lakše elemente spojimo u teže te da kroz lanac nuklearnih raspada otkrijemo novonastale stabilnije izotope raznih elemenata. Kako bi otkrili nove izotope moramo pratiti nastajanje novih elemenata prirodnim putem. Ne nastaju svi elementi u malim zvijezdama, za teže elemente potrebne su veće i toplije zvijezde. Različite zvijezde mogu biti odabrane u izborniku zvijezda. Za otključavanje određene zvijezde potrebno je otkriti bar jedan stabilni izotopi određenog elementa. Igra nema praćenje bodova niti statistiku nego dopušta slobodno otkrivanje raznih kombinacija potrebnih za nuklearnu fuziju. Svi novootkriveni izotopi mogu se spremiti za buduća pokretanja igre. Igra ima dva različita načina igranja. Glavni način igranja je Exploration u kojem se nuklearnom fuzijom otkrivaju novi elementi. Uz njega i postoji Creative način igranja u kojem su od početka otključani svi elementi i zvijezde te se u njemu može slobodno eksperimentirati s elementima i izotopima.

Alati i moduli

Pozadine su izrađene pomoću besplatnog alata za grafičko uređivanje GIMP. Za izradu programa bili su potrebni slijedeći Python moduli:

- pygame
  - Glavni modul korišten za izradu ove igre
  - Kao što je u imenu naznačeno, glavni cilj modula je stvaranje igre u Python programskom jeziku koji daju mogućnost programu najvažniju stvar kod računalnih igara: iscrtavanje kadrova ovisno o unosu korisnika
- sys
  - Modul koji komunicira sa sustavom za interpretaciju koda Python-a
  - Koristi se kako bi se zaustavio proces programa povodom naredbe za izlaženje
- pymunk
  - Modul koji daje fizička i kinetička svojstva objektima kako bi detaljno simulirali stvarne događaje
  - Donosi prostor za odvijanje simulacije koji je potreban za sudare između čestica

- random
  - Modul koji donosi mogućnosti nasumičnog odabira vrijednosti varijabli koji je uveden kako bi simulacija sudara izgledala realnija pomoću nasumičnih vrijednosti
- openpyxl
  - Modul koji daje mogućnost Python-u da čita iz i zapisuje u .xlsx datoteke koje u ovom slučaju sadržavaju tablicu izotopa s njihovim svojstvima potrebnu za igru
- numpy
  - Matematički modul namijenjen za linearnu algebru, no kao i za komunikaciju s .dat datotekama koje ova igra koristi za pohranjivanje napretka u igri

6. RAZRADA PROGRAMA

Klase i metode u igri

Čestice

Za naše potrebe u igri, morali smo stvoriti četiri nove klase objekata od kojih je klasa `Particle` {5} glavna značajka igre

{5}

```
class Particle(pygame.sprite.Sprite):
  def __init__(self, space, radius, pos_x, pos_y, mass,
               inertia, color, velocity, collision_type, id):
        super().__init__()
        self.body = pymunk.Body(mass,inertia,body_type =
pymunk.Body.DYNAMIC)
        self.body.position = (pos_x, pos_y)
        self.body.velocity = velocity
   ...
  def draw(self):
        pos_x = int(self.body.position.x)
        pos_y = int(self.body.position.y)
        color = self.color
        radius = self.radius
        pygame.draw.circle(screen, color, (pos_x, pos_y), radius)
   ...
```

Unutar klase `Particle` {5} navedene su najvažnije značajke posebice `pygame.sprite.Sprite`. Sprite-ovi su posebna klasa koja je dio modula pygame koja je specifično napravljena za prikazivanje. Metodom `pygame.sprite.Group()` sprite-ovi mogu biti stavljeni u jednu grupu te jednom naredbom svi biti prikazani u prozoru. Kako bi čestice mogle simulirati sudare u programu, moraju imati fizičko tijelo `self.body = pymunk.Body(mass, inertia, body_type = DYNAMIC)` (masu, inerciju i dinamičnost tijela, točnije naznaku da je tijelo u pokretu, a ne statični objekt). Modul pymunk ne može sam nacrtati neki objekt u prozoru, iako je definiran u prostoru te za to koristimo modul pygame.

U pygame-u svaki crtež ili kadar ima svoju zasebnu površinu definiranu položajem, visinom i širinom `self.rect = pygame.Rect(pos_x, pos_y, 0, 0)`. Naredba `Rect` stvara samo plohu na koju se slaže slika. Za crtanje slike, u ovom slučaju kruga, koristimo `self.shape = pymunk.Circle(self.body, radius).`

Korisnički ulazi (tipke)

Veliku važnost ima i interakcija korisnika s programom koja je u ovom programu izvedena pomoću `Button {6}` objekta. Kako bi korisnik mogao slobodno navigirati različitim izbornicima potrebne su tipke za njihov pristup.

{6}

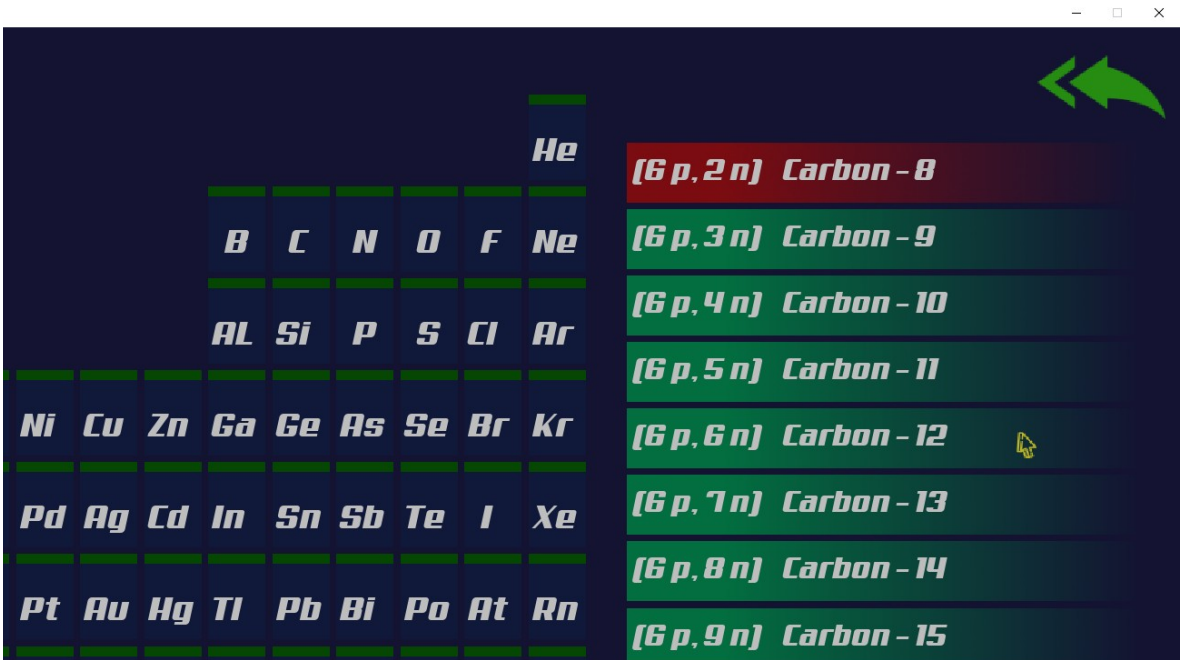
```
class Button():
    def __init__(self, pos_x, pos_y, image, display_group, scale = 1.0):
        super().__init__()
        self.image_load = pygame.image.load(image).convert_alpha()
        self.image = pygame.transform.scale_by(self.image_load,
global_scale)
        self.rect = self.image.get_rect()
  ...
  def draw(self):
        screen.blit(self.image, (self.rect.x, self.rect.y))

    def arrange(self):
        if self.display_group == "main_menu" and display_type !=
"main_menu":
            self.rect.move_ip(3000, 0)
        elif self.display_group == "main_menu" and display_type ==
"main_menu":
            self.rect.move_ip(-3000, 0)
   ...
```

Tipke, za razliku od čestica, nisu sprite-ovi. Razlog tome je što je zbog velike količine tipki u programu gotove nemoguće odrediti koja je tipka stisnuta što je pogotovo važno kod tipki za izotope kada program uzima i vraća svojstva izotopa nazad u program. Sljedećom naredbom program jednostavno provjerava što se nalazilo ispod pokazivača kada je tipka miša pritisnuta:

`if elements_button.rect.collidepoint(mouse_pos) == True:`

Ovako napravljene tipke `{6}` prikazuju se na zaslonu s funkcijom `screen.blit(self.image,...` koja povezanu sliku `self.image = pygame.image.load(image)` prikazuje na glavnom prozoru objekta `screen`.

Kako bi se jednostavnije pristupilo glavnim tipkama, nepotrebne tipke su metodom `arrange(self)` pomaknute van prozora programa. Tako i metoda `scroll_arrange(self, mouse_scroll)` pomiče tipke izotopa ovisno o unosu kotačića miša (Slika 5).



Slika 5 - Izbornik za izotope koji koristi unos kolutića miša za pomicanje tipki

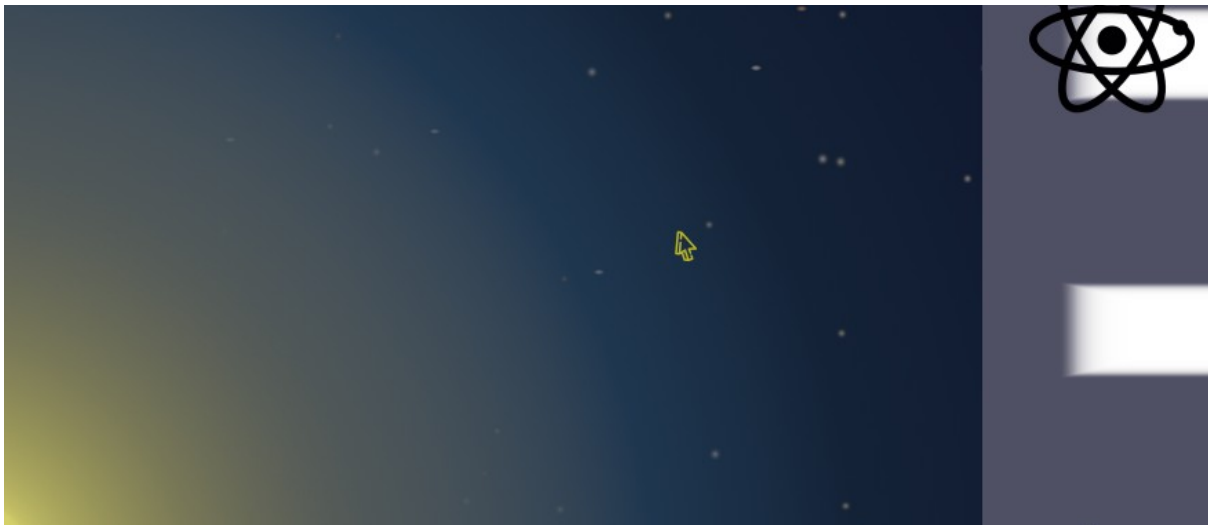
Ostale klase

Uz gore navedene klase postoje još i klasa za postavljanje zidova `class Wall(pygame.sprite.Sprite)`. Klasa `Wall` {7} će napraviti zidove kako bi se stvorila nevidljiva kutija za čestice da one ne bi izašle van prozora.

{7}

```
class Wall(pygame.sprite.Sprite):
    def __init__(self, space, wall_number,collision_type):
        super().__init__()
        self.body = pymunk.Body(body_type = pymunk.Body.STATIC)
        self.radius = 8
        self.wall_number = wall_number
...
```

Isto tako postoji i klasa za pokazivač {8} koja zamjenjuje standardni prikaz pokazivača miša (Slika 6).



Slika 6 - Pokazivač tokom pokretanja igre

{8}

```
class Cursor(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image_load =
pygame.image.load(cursor_path).convert_alpha()
        self.image = pygame.transform.scale_by(self.image_load,
global_scale)
        self.rect = self.image.get_rect()

    def update(self):
        self.rect.center = pygame.mouse.get_pos()
```

. . .

Funkcije programa i njihove značajke

U programu se nalazi niz funkcija koje imaju važnu ulogu u provođenju igre. Ove funkcije možemo podijeliti na dvije grupe: one koje upravljaju pozadinom i one koje upravljaju česticama.

Funkcije koje upravljaju pozadinom

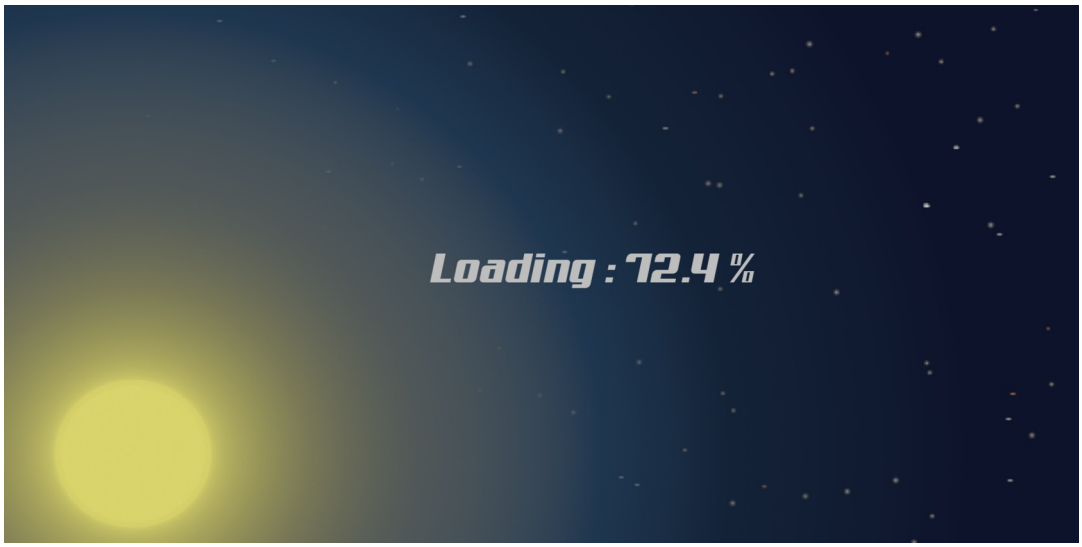
Funkcija `display_UI` glavna je funkcija za iscrtavanje objekta. Funkcija iz podatka o trenutno aktivnom izbornik određuje koji objekti moraju biti iscrtani. Kada je aktivan glavni izbornik `display_type = "main_menu"`, funkcija provjerava ima li čestica koje su predviđene za sudar te ako ima prikazuje simulaciju za označene čestice. Unutar nje se nalaz i `collision_handler_1` koji je objekt modula pymunk koji služi kako bi se mogao definirati događaj u slučaju da se dva objekta tipa `Body` u programu preklope, tj. kada se dogodi sudar. Klasa `Particle {5}` tada koristi tu informaciju za sudar kako bi se pronašle čestice koje su ušle u fuziju te da bi se izračunao konačni produkt nastale fuzije (Slika 7).

Uz funkciju `display_UI` postoji također i funkcija `star_transition_event {9}` koja će prilikom promjene aktivne zvijezde animirati pozadinu glavnog izbornika na izabranu zvijezdu. Također se u programu nalazi funkcija `loading_screen` koja prati učitavanje programa kako bi se prikazao trenutni postotak učitavanja (Slika 8).

Slika 7 - Čestice neposredno prije sudara

{9}

```
def star_transition_event(image_1, image_2):

    for i in range (255, 0, -5):
        screen.fill((0,0,0))
        image_1_load = pygame.image.load(image_1)
        image_1_draw = pygame.transform.scale_by(image_1_load, global_scale)
        image_1_draw.set_alpha(i)
        screen.blit(image_1_draw, screen_org_cord)
        pygame.display.update()
        time.sleep(0.001)

    time.sleep(1.5)

    for i in range (0, 255, 5):
        screen.fill((0,0,0))
        image_2_load = pygame.image.load(image_2)
        image_2_draw = pygame.transform.scale_by(image_2_load, global_scale)
        image_2_draw.set_alpha(i)
        screen.blit(image_2_draw, screen_org_cord)
        pygame.display.update()
        time.sleep(0.001)
```



Slika 8 - Prikaz postotka učitavanja prije početka igre

Funkcije koje upravljaju česticama

Funkcije `collision_prep` i `collision` namijenjene su specifično za sudare. Funkcija `{10}` priprema čestice dobivene iz izbornika za izotope, pridodaje ih u klasu čestica i zapisuje u arhivu za progresiju igre. Kako bi se čestice uopće vidjele, postavljen je minimalni radijus od 3 piksela, jer za bilo koji manji radijus, čestice jednostavno neće biti vidljive.

`{10}`

```
def collision_prep(ID_particle_A, ID_particle_B):

    particle_velocity_A = (random.randint(500, 700), 0)
    particle_radius_A = round(ID_chart_isotopes[ID_particle_A][3].value
** (0.67))
    particle_color_A = (250 - ID_chart_isotopes[ID_particle_A][2].value
* 2, random.randint(0, 255), ID_chart_isotopes[ID_particle_A][2].value)

    particle_A_pos_x = 100
    particle_A_pos_y = (screen_height / 2) + random.randint(-
round(particle_radius_A / 3, 0), round(particle_radius_A / 3, 0))

    particle_velocity_B = (random.randint(-700, -500), 0)
    particle_radius_B = round(ID_chart_isotopes[ID_particle_B][3].value
** (0.67))
    particle_color_B = (ID_chart_isotopes[ID_particle_A][2].value,
random.randint(0, 150), 250 - ID_chart_isotopes[ID_particle_A][2].value
* 2)

    particle_B_pos_x = particle_space - 100
    particle_B_pos_y = (screen_height / 2) + random.randint(-
round(particle_radius_B / 3, 0), round(particle_radius_B / 3, 0))

    if particle_radius_A < 3:
        particle_radius_A = 3
    if particle_radius_B < 3:
        particle_radius_B = 3

...

    particle_group.add(particle_A)
    particle_group.add(particle_B)
    particle_list.append(particle_A)
    particle_list.append(particle_B)
    particle_archive.append(particle_A)
    particle_archive.append(particle_B)
    progression_record_cache.append(particle_A.id)
    progression_record_cache.append(particle_B.id)
```

Funkcija `collision`, slično funkciji `collision_prep`, stvara novu česticu, ali tek nakon postignute fuzije. Funkcija prilikom sudara uzima atomske i protonske brojeve od svih čestica koje sudjeluju u sudaru te ih zbraja kako bi pronašla podatke o novoj čestici. Ako podaci, točnije protonski i atomski broj, novonastale čestice ne odgovaraju podacima u tablici izotopa, tada nema stvaranja nove čestice, nego se čestice koje su trebale ući u fuziju jednostavne odbiju jedna od drugu. Kada je fuzija valjana, odnosno podaci o novoj čestici se nalaze u tablici, funkcija miče početne čestice iz simulacije pomoću funkcije `{11}` te stvara novu česticu čiji podaci odgovaraju onima iz kalkulacije.

`{11}`

```
def kill_command(particle):
    particle.kill()
    space.remove(particle.body, particle.shape)
```

Da se ostvari glavni cilj igre (otkivanje novih izotopa), potrebna je dodatna funkcija koja će mijenjati česticu nastalu fuzijom ovisno o njenoj prirodi, točnije stabilnosti jezgre. Funkcija `decay` uzima identifikaciju nastale čestice te pretražuje u tablici izotopa stabilnost, odnosno nestabilnost iste čestice. U slučaju da je stabilna, čestica nastavlja svojim putem nepromjenjena te nema raspada. U slučaju nestabilnosti, provjerava se koji se raspad događa te ovisno o podacima iz tablice izotopa, stvaraju se nove čestice nakon raspada. Za svaki izotop postoji nekoliko oblika raspada koji se mogu dogoditi [13], ali zbog jednostavnosti je u igri definirano da bude samo jedan, onaj najčešći raspad. Nakon raspada, nastalim česticama daje se nasumična brzina i boja da se mogu razlikovati, upisuju se u arhivu te vraćaju u glavni izbornik gdje će biti u kontaktu s drugim česticama. U slučaju da novonastale čestice nisu stabilne, proces raspada se ponavlja dok se u glavnom izborniku ne nalaze samo stabilne čestice.

Glavne postavke programa i inicijalizacija

Unutar glavnih postavki prostora igre nalazi se veliki broj varijabli, lista i drugih postavki kako bi ostatak programa imao te postavke dostupne na jednom mjestu. U ovom segmentu programa se nalaze boje za pozadine izbornika, putanje do mjesta slika za iscrtavanje tipki i pozadina kao i putanje do tablice izotopai progresiju igre.

```
# *** Postavke prostora igre *** #

pygame.init()
pygame.display.set_caption("Game")
space = pymunk.Space()
space.gravity = (0,0)
options = pymunk.SpaceDebugDrawOptions()
space.debug_draw(options)


# *** Postavke pozadine *** #

display_size = pygame.display.Info().current_w,
pygame.display.Info().current_h - 60
screen_width = pygame.display.Info().current_w
screen_height = pygame.display.Info().current_h - 60
screen = pygame.display.set_mode((screen_width, screen_height))
screen_center_cord = (screen_width / 2, screen_height / 2)
screen_org_cord = (0, 0)

global_scale_x = round(screen_width / 1366, 2)
global_scale_y = round(screen_height / 768, 2)
global_scale = (global_scale_x, global_scale_y)

particle_space = screen_width - scale_for_x(400)
...
```

7. PETLJA IGRE (*eng.* GAME LOOP)

Ovaj segment programa je beskonačna petlja koja stalno provjerava korisničke ulaze te reagira na njih i iscrtava sadržaj prozora. Beskonačna petlja se prekida ukoliko je glavni prozor igre zatvoren. Tokom pokretanja igre, korisnik interakcijom s tipkama mijenja vrijednosti programa kako bi došlo do progresije igre, tako da se u ovom segmentu nalaze provjere za pokazivač i aktivnost igre{13}

{13}

```
...
while True:
     for event in pygame.event.get():
          if event.type == pygame.QUIT:
               pygame.quit()
               sys.exit()

          if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
               mouse_pos = pygame.mouse.get_pos()

...
```

U glavnim postavkama prije beskonačne petlje definiran je i sat {14}.

{14}

```
...
clock = pygame.time.Clock()
FPS = 60
...
```

Program koristi gore definirani sat u beskonačnoj petlji kako bi glatko bila prikazana simulacija čestica i interakcija s korisnikom tokom rada programa{15}. Nakon svakog otkucaja sata iscrtava se novi kadar prozora te se simulacije fizičkih tijela čestica pomiče za jedan korak

{15}

```
...
pygame.display.update()
     clock.tick(FPS)
     space.step(1/FPS)
```

8. ZAKLJUČAK

Proces fuzije glavni je proces oslobađanja energije u zvijezdama. Za naš planet, Sunce je glavni izvor energije za sve biljke, a time i životinje. Proučavanjem fuzije može se znatno poboljšati i unaprijediti naše spoznaje o energiji, njenoj potrošnji pa tako i proizvodnji, a jednostavnim interaktivnim programima, poput programa razvijenog u ovom radu, olakšavamo objašnjavanje glavnih principa procesa nuklearne fuzije. Spoznaja o kompleksnim događajima uvijek polazi od razumijevanja najosnovnijih.

9. LITERATURA

[1] https://www.arbona.hr/hr/sto-je-gamifikacija/1683 (pristupljeno 27. 09. 2023.)

[2] Goeppert Mayer, M. and Jenson, J. H. D. Elementary Theory of Nuclear Shell Structure, John Wiley & Sons, Inc., New York, 1955

[3] M.Wang, G.Audi, A.H.Wapstra, F.G.Kondev, M.MacCormick, X.Xu, B.Pfeiffer. Chin.Phys.C 36, 1603 (2012) The AME2012 atomic mass evaluation (II)

[4] Morse, E. (2018). Nuclear Fusion. Graduate Texts in Physics

[5] Shell, Scott (2014). "An introduction to Python for scientific computing"

[6] https://sites.engineering.ucsb.edu/~shell/che210d/python.pdf (pristupljeno 23. 09. 2023.)

[7] https://www.educba.com/python-vs-c-plus-plus/?

[8] https://docs.python.org/3/ (pristupljeno 23. 09. 2023.)

[9] https://medium.com/omarelgabrys-blog/database-introduction-part-1-4844fada1fb0 (pristupljeno 27. 09. 2023.)

[10] https://www-nds.iaea.org (pristupljeno 01. 08. 2023.)

[11] https://www.nndc.bnl.gov/ensdf (pristupljeno 01. 08. 2023.)

[12] https://www.nndc.bnl.gov/endf (pristupljeno 01. 08. 2023.)

[13] https://www.nndc.bnl.gov(pristupljeno 01. 08. 2023.)

[14] Dayah, M. (1997, October 1). Periodic Table - Ptable. Ptable. https://ptable.com (pristupljeno 01. 08. 2023.)

[15] https://icons8.com/icons (pristupljeno 01. 08. 2023.)

10. POPIS SLIKA

11. ŽIVOTOPIS

Igor Lassinger rođen je 11. prosinca 1996. godine u Vinkovcima, u Republici Hrvatskoj. Pohađao je Osnovnu školu A. G. Matoš u Vinkovcima. Po završetku osnovne škole upisuje Gimnaziju M. A. Reljkovića u Vinkovcima koju završava 2015. godine kada upisuje preddiplomski studiji fizike i informatike na Odjelu za Fiziku, Sveučilište Josipa Jurja Strossmayera Osijek.

requirements.txt:

numpy==1.26.0
openpyxl==3.1.2
pygame==2.5.2
pymunk==6.5.1

Game-v1.03.py:

```
import pygame, sys, pymunk, random, openpyxl, time, numpy

class Particle(pygame.sprite.Sprite):
    def __init__(self, space, radius, pos_x, pos_y, mass,
                 inertia, color, velocity, collision_type, id):
        super().__init__()
        self.body = pymunk.Body(mass,inertia,body_type = pymunk.Body.DYNAMIC)
        self.body.position = (pos_x, pos_y)
        self.body.velocity = velocity
        self.shape = pymunk.Circle(self.body, radius)
        self.shape.elasticity = 0.5
        self.shape.density = 1
        self.shape.collision_type = collision_type
        self.rect = pygame.Rect(pos_x, pos_y, 0, 0)
        self.image_load = pygame.Surface((0, 0))
        self.image = pygame.transform.scale_by(self.image_load, global_scale)
        self.color = color
        self.radius = radius
        self.id = id
        self.proton_number = ID_chart_isotopes[self.id][2].value
        self.atomic_number = ID_chart_isotopes[self.id][3].value
        self.stability = ID_chart_isotopes[self.id][4].value
        space.add(self.body, self.shape)

    def draw(self):
        pos_x = int(self.body.position.x)
        pos_y = int(self.body.position.y)
        color = self.color
        radius = self.radius
        pygame.draw.circle(screen, color, (pos_x, pos_y), radius)

    def change_drag(self, arbiter, space, data):
        space.damping = 0.8
        return True

    def collision_merge(self, arbiter, space, data):
        collision(particle_list)
        return True

class Cursor(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image_load = pygame.image.load(cursor_path).convert_alpha()
        self.image = pygame.transform.scale_by(self.image_load, global_scale)
        self.rect = self.image.get_rect()

    def update(self):
        self.rect.center = pygame.mouse.get_pos()

class Wall(pygame.sprite.Sprite):
    def __init__(self, space, wall_number,collision_type):
        super().__init__()
        self.body = pymunk.Body(body_type = pymunk.Body.STATIC)
        self.radius = 8
```

```
        self.wall_number = wall_number

        if self.wall_number == 0:
            self.shape = pymunk.Segment(self.body, (particle_space, screen_height),
(particle_space, 0), self.radius)
            self.rect = pygame.Rect(particle_space, 0, self.radius, screen_height)

        if self.wall_number == 1:
            self.shape = pymunk.Segment(self.body, (particle_space, 0), (0, 0),
self.radius)
            self.rect = pygame.Rect(0, screen_height, particle_space, self.radius)

        if self.wall_number == 2:
            self.shape = pymunk.Segment(self.body, (0, 0), (0, screen_height),
self.radius)
            self.rect = pygame.Rect(0, 0, self.radius, screen_height)

        if self.wall_number == 3:
            self.shape = pymunk.Segment(self.body, (0, screen_height),
(particle_space, screen_height), self.radius)
            self.rect = pygame.Rect(0, 0, particle_space, self.radius)

        self.shape.elasticity = 1
        self.shape.collision_type = collision_type
        space.add(self.body, self.shape)

    def draw(self):
        if self.wall_number == 0:
            pygame.draw.line(screen, (0, 0, 0), (particle_space, 0),
(particle_space, screen_height), width = self.radius)
        if self.wall_number == 1:
            pygame.draw.line(screen, (0, 0, 0), (particle_space, screen_height),
(0, screen_height), width = self.radius)
        if self.wall_number == 2:
            pygame.draw.line(screen, (0, 0, 0), (0, screen_height), (0, 0), width =
self.radius)
        if self.wall_number == 3:
            pygame.draw.line(screen, (0, 0, 0), (0, 0), (particle_space, 0), width
= self.radius)

class Button():
    def __init__(self, pos_x, pos_y, image, display_group, scale = 1.0, icon =
False):
        super().__init__()
        if icon == True:
            self.g_scale = (global_scale_x, global_scale_x)
        else:
            self.g_scale = (global_scale_x, global_scale_y)
        self.image_load = pygame.image.load(image).convert_alpha()
        self.image = pygame.transform.scale_by(self.image_load, self.g_scale)
        self.rect = self.image.get_rect()
        self.rect.center = [pos_x, pos_y]
        self.pos_x = pos_x
        self.pos_y = pos_y
        self.display_group = display_group
        if scale != 1.0:
            self.image = pygame.transform.scale_by(self.image, scale)

    def draw(self):
        screen.blit(self.image, (self.rect.x, self.rect.y))

    def arrange(self):
        if self.display_group == "main_menu" and display_type != "main_menu":
            self.rect.move_ip(6000, 0)
        elif self.display_group == "main_menu" and display_type == "main_menu":
            self.rect.move_ip(-6000, 0)

        if self.display_group == "all_menu" and display_type == "main_menu":
```

```
            self.rect.move_ip(6000, 0)
        elif self.display_group == "all_menu" and display_type != "main_menu":
            self.rect.move_ip(-6000, 0)

        if self.display_group == "elements_menu" and display_type !=
"elements_menu":
            self.rect.move_ip(6000, 0)
        elif self.display_group == "elements_menu" and display_type ==
"elements_menu":
            self.rect.move_ip(-6000, 0)

        if self.display_group == "star_menu" and display_type != "star_menu":
            self.rect.move_ip(6000, 0)
        elif self.display_group == "star_menu" and display_type == "star_menu":
            self.rect.move_ip(-6000, 0)

        if self.display_group == "options_menu" and display_type != "options_menu":
            self.rect.move_ip(6000, 0)
        elif self.display_group == "options_menu" and display_type ==
"options_menu":
            self.rect.move_ip(-6000, 0)

        if self.display_group == "launcher_menu" and display_type !=
"launcher_menu":
            self.rect.move_ip(6000, 0)
        elif self.display_group == "launcher_menu" and display_type ==
"launcher_menu":
            self.rect.move_ip(-6000, 0)

    def scroll_arrange(self, mouse_scroll):
        if mouse_scroll == 1:
            self.rect.move_ip(0, -dropmenu_scroll)
        else:
            self.rect.move_ip(0, dropmenu_scroll)

#-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

def display_UI(display_type, image):
    if display_type == "main_menu":
        screen.fill(main_menu_color)

        display_bg_load = pygame.image.load(image).convert_alpha()
        display_bg = pygame.transform.scale_by(display_bg_load, global_scale)
        display_bg_rect = display_bg.get_rect()
        screen.blit(display_bg, (display_bg_rect.x, display_bg_rect.y))
        pygame.draw.rect(screen, main_menu_color , main_menu_surface, screen_width-
particle_space)

        for button_select in button_list:
            button_select.draw()

        if len(particle_group) >= 0:
            i = 0

            for particle in particle_archive:
                particle_tracker_rect = pygame.Rect(particle_space -
scale_for_x(150), scale_for_y(3 + i * 14), 260, 20)
                particle_tracker_position = (particle_tracker_rect[0],
particle_tracker_rect[1] + 7)
                particle_position = (particle.body.position.x,
particle.body.position.y)
                if particle in particle_group:
                    pygame.draw.line(screen, (150, 150, 150,), particle_position,
particle_tracker_position)

                particle_text_surface =
text_font_05_particle.render(ID_chart_isotopes[particle_archive[i].id][1].value,
```

```
True, "Gray")
                screen.blit(particle_text_surface, (particle_tracker_position[0],
particle_tracker_position[1] - 7))

                for j in range(len(decay_chain_list)):
                    if particle == decay_chain_list[j][0]:
                        decay_particle_text = decay_chain_list[j][1]
                        decay_particle_text_surface =
text_font_05_particle.render(decay_particle_text, True, "Gray")
                        screen.blit(decay_particle_text_surface,
(particle_tracker_position[0] - 10 * len(decay_particle_text) - 20,
particle_tracker_position[1] - 7))
                i += 1

            for particle in particle_group:
                handler_1.begin = particle.change_drag
                handler_1.post_solve = particle.collision_merge

                particle.draw()

    if display_type == "elements_menu":
        screen.fill(elements_menu_color)

        for element_select in element_button_list:
            element_select.draw()

            if len(element_name_list[element_button_list.index(element_select)]) ==
1:
                element_cord_select =
(element_cord_list[element_button_list.index(element_select)][0] - scale_for_x(10),

element_cord_list[element_button_list.index(element_select)][1])
            else:
                element_cord_select =
(element_cord_list[element_button_list.index(element_select)][0] - scale_for_x(20),

element_cord_list[element_button_list.index(element_select)][1])

            element_text_surface =
text_font_02_elements.render(element_name_list[element_button_list.index(element_se
lect)], True, "Gray")
            screen.blit(element_text_surface, element_cord_select)

        pygame.draw.rect(screen, elements_menu_color_2 , elements_menu_border,
scale_for_x(400))
        for button_select in button_list:
            button_select.draw()

    if display_type == "elements_menu_isotopes":
        screen.fill(elements_menu_color)

        for element_select in element_button_list:
            element_select.draw()

            if len(element_name_list[element_button_list.index(element_select)]) ==
1:
                element_cord_select =
(element_cord_list[element_button_list.index(element_select)][0] - scale_for_x(10),

element_cord_list[element_button_list.index(element_select)][1])
            else:
                element_cord_select =
(element_cord_list[element_button_list.index(element_select)][0] - scale_for_x(20),

element_cord_list[element_button_list.index(element_select)][1])

            element_text_surface =
text_font_02_elements.render(element_name_list[element_button_list.index(element_se
```

```
lect)], True, "Gray")
            screen.blit(element_text_surface, element_cord_select)

        for isotope_select in isotope_button_list:
            isotope_select.draw()

            isotope_dysplay_name =
ID_chart_isotopes[isotope_ID_list[isotope_button_list.index(isotope_select)]]
[1].value
            isotope_dysplay_proton_number =
ID_chart_isotopes[isotope_ID_list[isotope_button_list.index(isotope_select)]]
[2].value
            isotope_dysplay_neutron_number =
ID_chart_isotopes[isotope_ID_list[isotope_button_list.index(isotope_select)]]
[3].value - isotope_dysplay_proton_number
            isotope_dysplay = "(" + str(isotope_dysplay_proton_number) + " p, " +
str(isotope_dysplay_neutron_number) + " n)    " + str(isotope_dysplay_name)
            isotope_text_surface = text_font_01_isotopes.render(isotope_dysplay,
True, "Gray")
            screen.blit(isotope_text_surface, (isotope_select.rect[0] + 5,
isotope_select.rect[1] + 18))

        pygame.draw.rect(screen, elements_menu_color_2 , elements_menu_border,
scale_for_x(400))

        for button_select in button_list:
            button_select.draw()

        if len(collision_bin) > 0:
            collision_bin_text_surface =
text_font_03_bin.render(ID_chart_isotopes[collision_bin[0]][1].value, True, "Gray")
            screen.blit(collision_bin_text_surface, (200, 200))

    if display_type == "star_menu":
        screen.fill(star_menu_color)

        for button_select in button_list:
            button_select.draw()

        star_text_surface_1 = text_font_04_star.render(" - Young yellow star",
True, "Gray")
        star_text_surface_2 = text_font_04_star.render(" - Orange main sequence
star", True, "Gray")
        star_text_surface_3 = text_font_04_star.render(" - Red giant star", True,
"Gray")
        star_text_surface_4 = text_font_04_star.render(" - Supernova", True,
"Gray")
        star_text_surface_5 = text_font_04_star.render(" - Artificial star", True,
"Gray")
        screen.blit(star_text_surface_1, (scale_for_x(400), scale_for_y(590)))
        screen.blit(star_text_surface_2, (scale_for_x(500), scale_for_y(490)))
        screen.blit(star_text_surface_3, (scale_for_x(600), scale_for_y(390)))
        screen.blit(star_text_surface_4, (scale_for_x(700), scale_for_y(290)))
        screen.blit(star_text_surface_5, (scale_for_x(800), scale_for_y(190)))

        screen.blit(star_icon_2, star_core_list[current_star[1]-1][1])

        if progression <= 4:
            screen.blit(lock_icon, (scale_for_x(540), scale_for_y(110)))
        if progression <= 3:
            screen.blit(lock_icon, (scale_for_x(440), scale_for_y(210)))
        if progression <= 2:
            screen.blit(lock_icon, (scale_for_x(340), scale_for_y(310)))
        if progression <= 1:
            screen.blit(lock_icon, (scale_for_x(240), scale_for_y(410)))

    if display_type == "options_menu":
        screen.fill(options_menu_color)
```

```
        for button_select in button_list:
            button_select.draw()

        screen.blit(decay_speed_text_surface_1, (scale_for_x(70),
scale_for_y(120)))
        screen.blit(decay_speed_text_surface_2, (scale_for_x(100),
scale_for_y(220)))
        screen.blit(decay_speed_text_surface_3, (scale_for_x(100),
scale_for_y(320)))
        screen.blit(decay_speed_text_surface_4, (scale_for_x(100),
scale_for_y(420)))

        screen.blit(decay_speed_text_surface_pointer, (scale_for_x(60),
scale_for_y(delay_max + 120)))

    if display_type == "launcher_menu":
        launcher_image_load = pygame.image.load(image).convert_alpha()
        launcher_image = pygame.transform.scale_by(launcher_image_load,
global_scale)
        launcher_screen.blit(launcher_image, (0, 0))

        for button_select in launcher_button_list:
            button_select.draw()

        screen.blit(launcher_text_surface_1, (launcher_button_list[0].pos_x -
scale_for_x(90), launcher_button_list[0].pos_y - scale_for_y(5)))
        screen.blit(launcher_text_surface_2, (launcher_button_list[1].pos_x -
scale_for_x(50), launcher_button_list[1].pos_y - scale_for_y(5)))
        screen.blit(launcher_text_surface_3, (launcher_button_list[2].pos_x -
scale_for_x(90), launcher_button_list[2].pos_y - scale_for_y(5)))

    if display_type == "launcher_instructions":
        launcher_image_load = pygame.image.load(image).convert_alpha()
        launcher_image = pygame.transform.scale_by(launcher_image_load,
global_scale)
        launcher_screen.blit(launcher_image, (0, 0))


        for button_select in launcher_button_list:
            button_select.draw()

        screen.blit(launcher_instruction_surface_00, (launcher_screen_width / 2 -
scale_for_x(150), scale_for_y(60)))
        screen.blit(launcher_instruction_surface_01, (launcher_screen_width / 2 -
scale_for_x(400), scale_for_y(140)))
        screen.blit(launcher_instruction_surface_02, (launcher_screen_width / 2 -
scale_for_x(400), scale_for_y(180)))
        screen.blit(launcher_instruction_surface_03, (launcher_screen_width / 2 -
scale_for_x(400), scale_for_y(220)))
        screen.blit(launcher_instruction_surface_04, (launcher_screen_width / 2 -
scale_for_x(400), scale_for_y(260)))
        screen.blit(launcher_instruction_surface_05, (launcher_screen_width / 2 -
scale_for_x(400), scale_for_y(300)))
        screen.blit(launcher_instruction_surface_06, (launcher_screen_width / 2 -
scale_for_x(400), scale_for_y(340)))
        screen.blit(launcher_instruction_surface_07, (launcher_screen_width / 2 -
scale_for_x(400), scale_for_y(380)))
        screen.blit(launcher_instruction_surface_08, (launcher_screen_width / 2 -
scale_for_x(400), scale_for_y(420)))
        screen.blit(launcher_instruction_surface_09, (launcher_screen_width / 2 -
scale_for_x(400), scale_for_y(460)))
        screen.blit(launcher_instruction_surface_10, (launcher_screen_width / 2 -
scale_for_x(400), scale_for_y(500)))
        screen.blit(launcher_instruction_surface_11, (launcher_screen_width / 2 -
scale_for_x(400), scale_for_y(540)))
        screen.blit(launcher_instruction_surface_12, (launcher_screen_width / 2 -
scale_for_x(400), scale_for_y(580)))
```

```
        screen.blit(launcher_instruction_surface_13, (launcher_screen_width / 2 -
scale_for_x(400), scale_for_y(620)))
        screen.blit(launcher_instruction_surface_14, (launcher_screen_width / 2 -
scale_for_x(400), scale_for_y(660)))

def star_transition_event(image_1, image_2):

    for i in range (255, 0, -5):
        screen.fill((0,0,0))
        image_1_load = pygame.image.load(image_1)
        image_1_draw = pygame.transform.scale_by(image_1_load, global_scale)
        image_1_draw.set_alpha(i)
        screen.blit(image_1_draw, screen_org_cord)
        pygame.display.update()
        time.sleep(0.001)

    time.sleep(1.5)

    for i in range (0, 255, 5):
        screen.fill((0,0,0))
        image_2_load = pygame.image.load(image_2)
        image_2_draw = pygame.transform.scale_by(image_2_load, global_scale)
        image_2_draw.set_alpha(i)
        screen.blit(image_2_draw, screen_org_cord)
        pygame.display.update()
        time.sleep(0.001)

def collision_prep(ID_particle_A, ID_particle_B):

    particle_velocity_A = (random.randint(500, 700), 0)
    particle_radius_A = round(ID_chart_isotopes[ID_particle_A][3].value ** (0.67))
    particle_color_A = (250 - ID_chart_isotopes[ID_particle_A][2].value * 2,
random.randint(0, 255), ID_chart_isotopes[ID_particle_A][2].value)
    particle_A_pos_x = 100
    particle_A_pos_y = (screen_height / 2) + random.randint(-
round(particle_radius_A / 3, 0), round(particle_radius_A / 3, 0))

    particle_velocity_B = (random.randint(-700, -500), 0)
    particle_radius_B = round(ID_chart_isotopes[ID_particle_B][3].value ** (0.67))
    particle_color_B = (ID_chart_isotopes[ID_particle_A][2].value,
random.randint(0, 150), 250 - ID_chart_isotopes[ID_particle_A][2].value * 2)
    particle_B_pos_x = particle_space - 100
    particle_B_pos_y = (screen_height / 2) + random.randint(-
round(particle_radius_B / 3, 0), round(particle_radius_B / 3, 0))

    if particle_radius_A < 3:
        particle_radius_A = 3
    if particle_radius_B < 3:
        particle_radius_B = 3

    particle_A = Particle(space,
                          particle_radius_A,
                          particle_A_pos_x,
                          particle_A_pos_y,
                          ID_chart_isotopes[ID_particle_A][3].value,
                          1,
                          particle_color_A,
                          particle_velocity_A,
                          1,
                          ID_particle_A)

    particle_B = Particle(space,
                          particle_radius_B,
                          particle_B_pos_x,
                          particle_B_pos_y,
                          ID_chart_isotopes[ID_particle_B][3].value,
                          1,
                          particle_color_B,
```

```
                            particle_velocity_B,
                            1,
                            ID_particle_B)

    particle_group.add(particle_A)
    particle_group.add(particle_B)
    particle_list.append(particle_A)
    particle_list.append(particle_B)
    particle_archive.append(particle_A)
    particle_archive.append(particle_B)
    progression_record_cache.append(particle_A.id)
    progression_record_cache.append(particle_B.id)

def kill_command(particle):
    particle.kill()
    space.remove(particle.body, particle.shape)

def collision(particle_list):
    particle_proton_number = particle_list[0].proton_number +
particle_list[1].proton_number
    particle_atomic_number = particle_list[0].atomic_number +
particle_list[1].atomic_number
    decay_chain_list = []

    if particle_proton_number <= 118:
        for i in range(1, len(ID_chart_list) + 1):
            if particle_proton_number == i:
                for j in range(0, len(ID_chart_list[i - 1])):
                    if particle_atomic_number == ID_chart_list[i - 1][j][3].value:
                        ID_particle_C = ID_chart_list[i - 1][j][0].value
                        particle_velocity_C = (random.randint(-300, 300),
random.randint(-300, 300))
                        particle_radius_C = round(particle_atomic_number ** (0.67),
0)
                        particle_color_C = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                        particle_C_pos_x = particle_list[0].body.position.x
                        particle_C_pos_y = particle_list[0].body.position.y
                        if particle_radius_C < 3:
                            particle_radius_C = 3

                        for particle in particle_group:
                            kill_command(particle)

                        collision_bin = []
                        particle_list = []

                        particle_C = Particle(space,
                                              particle_radius_C,
                                              particle_C_pos_x,
                                              particle_C_pos_y,
                                              particle_atomic_number,
                                              2,
                                              particle_color_C,
                                              particle_velocity_C,
                                              2,
                                              ID_particle_C )

                        particle_group.add(particle_C)
                        particle_list.append(particle_C)
                        particle_archive.append(particle_C)
                        progression_record_cache.append(particle_C.id)
                        break

def decay(particle):
    if particle.stability == 1:
        particle.draw()
```

```
    else:
        for i in range(6, 16):
            if ID_chart_isotopes[particle.id][i].value != None:

                if i == 6:  # --- Alfa
------------------------------------------------------------------------------
--------------------------------------
                    new_particle_A_id = ID_chart_isotopes[particle.id][i].value
                    new_particle_A_atomic_number =
ID_chart_isotopes[new_particle_A_id][3].value
                    new_particle_velocity_A = (random.randint(-100, 100),
random.randint(-100, 100))
                    new_particle_radius_A = round(new_particle_A_atomic_number **
(0.67), 0)
                    new_particle_color_A = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_A_pos_x = particle.body.position.x
                    new_particle_A_pos_y = particle.body.position.y

                    new_particle_B_id = 10
                    new_particle_B_atomic_number = 4
                    new_particle_velocity_B = (random.randint(-400, 400),
random.randint(-400, 400))
                    new_particle_radius_B = 3
                    new_particle_color_B = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_B_pos_x = particle.body.position.x
                    new_particle_B_pos_y = particle.body.position.y

                    for particle in particle_group:
                        if particle.stability != 1:
                            kill_command(particle)

                    new_particle_A = Particle(space,
                                              new_particle_radius_A,
                                              new_particle_A_pos_x,
                                              new_particle_A_pos_y,
                                              new_particle_A_atomic_number,
                                              2,
                                              new_particle_color_A,
                                              new_particle_velocity_A,
                                              2,
                                              new_particle_A_id)

                    new_particle_B = Particle(space,
                                              new_particle_radius_B,
                                              new_particle_B_pos_x,
                                              new_particle_B_pos_y,
                                              new_particle_B_atomic_number,
                                              2,
                                              new_particle_color_B,
                                              new_particle_velocity_B,
                                              2,
                                              new_particle_B_id)

                    particle_group.add(new_particle_A)
                    particle_list.append(new_particle_A)
                    particle_archive.append(new_particle_A)
                    progression_record_cache.append(new_particle_A.id)
                    particle_group.add(new_particle_B)
                    particle_list.append(new_particle_B)
                    particle_archive.append(new_particle_B)
                    progression_record_cache.append(new_particle_B.id)

                    decay_chain_list.append((new_particle_A, decay_list[i]))

                    break
```

```
                if i == 7:  # --- Beta +
------------------------------------------------------------------------------------
-----------------------------------
                    new_particle_A_id = ID_chart_isotopes[particle.id][i].value
                    new_particle_A_atomic_number =
ID_chart_isotopes[new_particle_A_id][3].value
                    new_particle_velocity_A = (random.randint(-100, 100),
random.randint(-100, 100))
                    new_particle_radius_A = round(new_particle_A_atomic_number **
(0.67), 0)
                    new_particle_color_A = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_A_pos_x = particle.body.position.x
                    new_particle_A_pos_y = particle.body.position.y

                    new_particle_B_id = 3183
                    new_particle_B_atomic_number = 1
                    new_particle_velocity_B = (random.randint(-400, 400),
random.randint(-400, 400))
                    new_particle_radius_B = 2
                    new_particle_color_B = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_B_pos_x = particle.body.position.x
                    new_particle_B_pos_y = particle.body.position.y

                    for particle in particle_group:
                        if particle.stability != 1:
                            kill_command(particle)

                    new_particle_A = Particle(space,
                                              new_particle_radius_A,
                                              new_particle_A_pos_x,
                                              new_particle_A_pos_y,
                                              new_particle_A_atomic_number,
                                              2,
                                              new_particle_color_A,
                                              new_particle_velocity_A,
                                              2,
                                              new_particle_A_id)

                    new_particle_B = Particle(space,
                                              new_particle_radius_B,
                                              new_particle_B_pos_x,
                                              new_particle_B_pos_y,
                                              new_particle_B_atomic_number,
                                              2,
                                              new_particle_color_B,
                                              new_particle_velocity_B,
                                              2,
                                              new_particle_B_id)

                    particle_group.add(new_particle_A)
                    particle_list.append(new_particle_A)
                    particle_archive.append(new_particle_A)
                    progression_record_cache.append(new_particle_A.id)
                    particle_group.add(new_particle_B)
                    particle_list.append(new_particle_B)
                    particle_archive.append(new_particle_B)
                    progression_record_cache.append(new_particle_B.id)

                    decay_chain_list.append((new_particle_A, decay_list[i]))


                    break

                if i == 8:  # --- Beta -
------------------------------------------------------------------------------------
-----------------------------------
```

```
                    new_particle_A_id = ID_chart_isotopes[particle.id][i].value
                    new_particle_A_atomic_number =
ID_chart_isotopes[new_particle_A_id][3].value
                    new_particle_velocity_A = (random.randint(-100, 100),
random.randint(-100, 100))
                    new_particle_radius_A = round(new_particle_A_atomic_number **
(0.67), 0)
                    new_particle_color_A = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_A_pos_x = particle.body.position.x
                    new_particle_A_pos_y = particle.body.position.y

                    new_particle_B_id = 3182
                    new_particle_B_atomic_number = 1
                    new_particle_velocity_B = (random.randint(-400, 400),
random.randint(-400, 400))
                    new_particle_radius_B = 2
                    new_particle_color_B = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_B_pos_x = particle.body.position.x
                    new_particle_B_pos_y = particle.body.position.y

                    for particle in particle_group:
                        if particle.stability != 1:
                            kill_command(particle)

                    new_particle_A = Particle(space,
                                              new_particle_radius_A,
                                              new_particle_A_pos_x,
                                              new_particle_A_pos_y,
                                              new_particle_A_atomic_number,
                                              2,
                                              new_particle_color_A,
                                              new_particle_velocity_A,
                                              2,
                                              new_particle_A_id)

                    new_particle_B = Particle(space,
                                              new_particle_radius_B,
                                              new_particle_B_pos_x,
                                              new_particle_B_pos_y,
                                              new_particle_B_atomic_number,
                                              2,
                                              new_particle_color_B,
                                              new_particle_velocity_B,
                                              2,
                                              new_particle_B_id)

                    particle_group.add(new_particle_A)
                    particle_list.append(new_particle_A)
                    particle_archive.append(new_particle_A)
                    progression_record_cache.append(new_particle_A.id)
                    particle_group.add(new_particle_B)
                    particle_list.append(new_particle_B)
                    particle_archive.append(new_particle_B)
                    progression_record_cache.append(new_particle_B.id)

                    decay_chain_list.append((new_particle_A, decay_list[i]))


                    break

                if i == 9:  # --- Proton emission
--------------------------------------------------------------------------------
----------------------------
                    new_particle_A_id = ID_chart_isotopes[particle.id][i].value
                    new_particle_A_atomic_number =
ID_chart_isotopes[new_particle_A_id][3].value
```

```
                    new_particle_velocity_A = (random.randint(-100, 100),
random.randint(-100, 100))
                    new_particle_radius_A = round(new_particle_A_atomic_number **
(0.67), 0)
                    new_particle_color_A = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_A_pos_x = particle.body.position.x
                    new_particle_A_pos_y = particle.body.position.y

                    new_particle_B_id = 1
                    new_particle_B_atomic_number = 1
                    new_particle_velocity_B = (random.randint(-200, 200),
random.randint(-200, 200))
                    new_particle_radius_B = 3
                    new_particle_color_B = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_B_pos_x = particle.body.position.x
                    new_particle_B_pos_y = particle.body.position.y

                    for particle in particle_group:
                        if particle.stability != 1:
                            kill_command(particle)

                    new_particle_A = Particle(space,
                                              new_particle_radius_A,
                                              new_particle_A_pos_x,
                                              new_particle_A_pos_y,
                                              new_particle_A_atomic_number,
                                              2,
                                              new_particle_color_A,
                                              new_particle_velocity_A,
                                              2,
                                              new_particle_A_id)

                    new_particle_B = Particle(space,
                                              new_particle_radius_B,
                                              new_particle_B_pos_x,
                                              new_particle_B_pos_y,
                                              new_particle_B_atomic_number,
                                              2,
                                              new_particle_color_B,
                                              new_particle_velocity_B,
                                              2,
                                              new_particle_B_id)

                    particle_group.add(new_particle_A)
                    particle_list.append(new_particle_A)
                    particle_archive.append(new_particle_A)
                    progression_record_cache.append(new_particle_A.id)
                    particle_group.add(new_particle_B)
                    particle_list.append(new_particle_B)
                    particle_archive.append(new_particle_B)
                    progression_record_cache.append(new_particle_B.id)

                    decay_chain_list.append((new_particle_A, decay_list[i]))


                    break

                if i == 10:  # --- Neutron emission
--------------------------------------------------------------------------------
-------------------------
                    new_particle_A_id = ID_chart_isotopes[particle.id][i].value
                    new_particle_A_atomic_number =
ID_chart_isotopes[new_particle_A_id][3].value
                    new_particle_velocity_A = (random.randint(-100, 100),
random.randint(-100, 100))
                    new_particle_radius_A = round(new_particle_A_atomic_number **
```

```
(0.67), 0)
                    new_particle_color_A = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_A_pos_x = particle.body.position.x
                    new_particle_A_pos_y = particle.body.position.y

                    new_particle_B_id = 0
                    new_particle_B_atomic_number = 1
                    new_particle_velocity_B = (random.randint(-200, 200),
random.randint(-200, 200))
                    new_particle_radius_B = 3
                    new_particle_color_B = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_B_pos_x = particle.body.position.x
                    new_particle_B_pos_y = particle.body.position.y

                    for particle in particle_group:
                        if particle.stability != 1:
                            kill_command(particle)

                    new_particle_A = Particle(space,
                                              new_particle_radius_A,
                                              new_particle_A_pos_x,
                                              new_particle_A_pos_y,
                                              new_particle_A_atomic_number,
                                              2,
                                              new_particle_color_A,
                                              new_particle_velocity_A,
                                              2,
                                              new_particle_A_id)

                    new_particle_B = Particle(space,
                                              new_particle_radius_B,
                                              new_particle_B_pos_x,
                                              new_particle_B_pos_y,
                                              new_particle_B_atomic_number,
                                              2,
                                              new_particle_color_B,
                                              new_particle_velocity_B,
                                              2,
                                              new_particle_B_id)

                    particle_group.add(new_particle_A)
                    particle_list.append(new_particle_A)
                    particle_archive.append(new_particle_A)
                    progression_record_cache.append(new_particle_A.id)
                    particle_group.add(new_particle_B)
                    particle_list.append(new_particle_B)
                    particle_archive.append(new_particle_B)
                    progression_record_cache.append(new_particle_B.id)

                    decay_chain_list.append((new_particle_A, decay_list[i]))


                    break

                if i == 11:  # --- Electron capture
--------------------------------------------------------------------------------
-------------------------
                    new_particle_A_id = ID_chart_isotopes[particle.id][i].value
                    new_particle_A_atomic_number =
ID_chart_isotopes[new_particle_A_id][3].value
                    new_particle_velocity_A = (random.randint(-100, 100),
random.randint(-100, 100))
                    new_particle_radius_A = round(new_particle_A_atomic_number **
(0.67), 0)
                    new_particle_color_A = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
```

```
                    new_particle_A_pos_x = particle.body.position.x
                    new_particle_A_pos_y = particle.body.position.y

                    for particle in particle_group:
                        if particle.stability != 1:
                            kill_command(particle)

                    new_particle_A = Particle(space,
                                              new_particle_radius_A,
                                              new_particle_A_pos_x,
                                              new_particle_A_pos_y,
                                              new_particle_A_atomic_number,
                                              2,
                                              new_particle_color_A,
                                              new_particle_velocity_A,
                                              2,
                                              new_particle_A_id)

                    particle_group.add(new_particle_A)
                    particle_list.append(new_particle_A)
                    particle_archive.append(new_particle_A)
                    progression_record_cache.append(new_particle_A.id)

                    decay_chain_list.append((new_particle_A, decay_list[i]))


                    break

                if i == 12:  # --- Spontaneous fission
--------------------------------------------------------------------------------
----------------------
                    new_particle_A_proton_number = int(round(particle.proton_number
* (0.5 + random.uniform(-0.15, 0)), 0))
                    new_particle_B_proton_number = particle.proton_number -
new_particle_A_proton_number

                    new_particle_A_atomic_number =
ID_chart_list[new_particle_A_proton_number][-random.randint(1, 9)][3].value
                    new_particle_B_atomic_number = particle.atomic_number -
new_particle_A_atomic_number

                    for j in
range(len(ID_chart_list[new_particle_A_proton_number])):
                        if new_particle_A_atomic_number ==
ID_chart_list[new_particle_A_proton_number][j][3].value:
                            new_particle_A_id =
ID_chart_list[new_particle_A_proton_number][j][0].value
                            break
                    for j in
range(len(ID_chart_list[new_particle_B_proton_number])):
                        if new_particle_B_atomic_number ==
ID_chart_list[new_particle_B_proton_number][j][3].value:
                            new_particle_B_id =
ID_chart_list[new_particle_B_proton_number][j][0].value
                            break

                    new_particle_velocity_A = (random.randint(-100, 100),
random.randint(-100, 100))
                    new_particle_radius_A = round(new_particle_A_atomic_number **
(0.67), 0)
                    new_particle_color_A = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_A_pos_x = particle.body.position.x
                    new_particle_A_pos_y = particle.body.position.y

                    new_particle_velocity_B = (random.randint(-200, 200),
random.randint(-200, 200))
                    new_particle_radius_B = round(new_particle_B_atomic_number **
```

```
(0.67), 0)
                    new_particle_color_B = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_B_pos_x = particle.body.position.x
                    new_particle_B_pos_y = particle.body.position.y

                    for particle in particle_group:
                        if particle.stability != 1:
                            kill_command(particle)

                    new_particle_A = Particle(space,
                                              new_particle_radius_A,
                                              new_particle_A_pos_x,
                                              new_particle_A_pos_y,
                                              new_particle_A_atomic_number,
                                              2,
                                              new_particle_color_A,
                                              new_particle_velocity_A,
                                              2,
                                              new_particle_A_id)

                    new_particle_B = Particle(space,
                                              new_particle_radius_B,
                                              new_particle_B_pos_x,
                                              new_particle_B_pos_y,
                                              new_particle_B_atomic_number,
                                              2,
                                              new_particle_color_B,
                                              new_particle_velocity_B,
                                              2,
                                              new_particle_B_id)

                    particle_group.add(new_particle_A)
                    particle_list.append(new_particle_A)
                    particle_archive.append(new_particle_A)
                    progression_record_cache.append(new_particle_A.id)
                    particle_group.add(new_particle_B)
                    particle_list.append(new_particle_B)
                    particle_archive.append(new_particle_B)
                    progression_record_cache.append(new_particle_B.id)

                    decay_chain_list.append((new_particle_A, decay_list[i]))
                    decay_chain_list.append((new_particle_B, decay_list[i]))

                    break

                if i == 13:  # --- Beta delayed alfa decay
--------------------------------------------------------------------------------
------------------
                    new_particle_A_id = ID_chart_isotopes[particle.id][i].value
                    new_particle_A_atomic_number =
ID_chart_isotopes[new_particle_A_id][3].value
                    new_particle_velocity_A = (random.randint(-100, 100),
random.randint(-100, 100))
                    new_particle_radius_A = round(new_particle_A_atomic_number **
(0.67), 0)
                    new_particle_color_A = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_A_pos_x = particle.body.position.x
                    new_particle_A_pos_y = particle.body.position.y

                    new_particle_B_id = 10
                    new_particle_B_atomic_number = 4
                    new_particle_velocity_B = (random.randint(-400, 400),
random.randint(-400, 400))
                    new_particle_radius_B = 3
                    new_particle_color_B = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
```

```
                    new_particle_B_pos_x = particle.body.position.x
                    new_particle_B_pos_y = particle.body.position.y

                    new_particle_C_id = 3182
                    new_particle_C_atomic_number = 1
                    new_particle_velocity_C = (random.randint(-400, 400),
random.randint(-400, 400))
                    new_particle_radius_C = 2
                    new_particle_color_C = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_C_pos_x = particle.body.position.x
                    new_particle_C_pos_y = particle.body.position.y

                    for particle in particle_group:
                        if particle.stability != 1:
                            kill_command(particle)

                    new_particle_A = Particle(space,
                                              new_particle_radius_A,
                                              new_particle_A_pos_x,
                                              new_particle_A_pos_y,
                                              new_particle_A_atomic_number,
                                              2,
                                              new_particle_color_A,
                                              new_particle_velocity_A,
                                              2,
                                              new_particle_A_id)

                    new_particle_B = Particle(space,
                                              new_particle_radius_B,
                                              new_particle_B_pos_x,
                                              new_particle_B_pos_y,
                                              new_particle_B_atomic_number,
                                              2,
                                              new_particle_color_B,
                                              new_particle_velocity_B,
                                              2,
                                              new_particle_B_id)

                    new_particle_C = Particle(space,
                                              new_particle_radius_C,
                                              new_particle_C_pos_x,
                                              new_particle_C_pos_y,
                                              new_particle_C_atomic_number,
                                              2,
                                              new_particle_color_C,
                                              new_particle_velocity_C,
                                              2,
                                              new_particle_C_id)

                    particle_group.add(new_particle_A)
                    particle_list.append(new_particle_A)
                    particle_archive.append(new_particle_A)
                    progression_record_cache.append(new_particle_A.id)
                    particle_group.add(new_particle_B)
                    particle_list.append(new_particle_B)
                    particle_archive.append(new_particle_B)
                    progression_record_cache.append(new_particle_B.id)
                    particle_group.add(new_particle_C)
                    particle_list.append(new_particle_C)
                    particle_archive.append(new_particle_C)
                    progression_record_cache.append(new_particle_C.id)

                    decay_chain_list.append((new_particle_A, decay_list[i]))

                    break

                if i == 14:  # --- Beta delayed proton emission
```

```
------------------------------------------------------------------------------
------------
                    new_particle_A_id = ID_chart_isotopes[particle.id][i].value
                    new_particle_A_atomic_number =
ID_chart_isotopes[new_particle_A_id][3].value
                    new_particle_velocity_A = (random.randint(-100, 100),
random.randint(-100, 100))
                    new_particle_radius_A = round(new_particle_A_atomic_number **
(0.67), 0)
                    new_particle_color_A = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_A_pos_x = particle.body.position.x
                    new_particle_A_pos_y = particle.body.position.y

                    new_particle_B_id = 1
                    new_particle_B_atomic_number = 1
                    new_particle_velocity_B = (random.randint(-400, 400),
random.randint(-400, 400))
                    new_particle_radius_B = 3
                    new_particle_color_B = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_B_pos_x = particle.body.position.x
                    new_particle_B_pos_y = particle.body.position.y

                    new_particle_B_id = 3182
                    new_particle_B_atomic_number = 1
                    new_particle_velocity_B = (random.randint(-400, 400),
random.randint(-400, 400))
                    new_particle_radius_B = 2
                    new_particle_color_B = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_B_pos_x = particle.body.position.x
                    new_particle_B_pos_y = particle.body.position.y

                    for particle in particle_group:
                        if particle.stability != 1:
                            kill_command(particle)

                    new_particle_A = Particle(space,
                                              new_particle_radius_A,
                                              new_particle_A_pos_x,
                                              new_particle_A_pos_y,
                                              new_particle_A_atomic_number,
                                              2,
                                              new_particle_color_A,
                                              new_particle_velocity_A,
                                              2,
                                              new_particle_A_id)

                    new_particle_B = Particle(space,
                                              new_particle_radius_B,
                                              new_particle_B_pos_x,
                                              new_particle_B_pos_y,
                                              new_particle_B_atomic_number,
                                              2,
                                              new_particle_color_B,
                                              new_particle_velocity_B,
                                              2,
                                              new_particle_B_id)

                    new_particle_C = Particle(space,
                                              new_particle_radius_C,
                                              new_particle_C_pos_x,
                                              new_particle_C_pos_y,
                                              new_particle_C_atomic_number,
                                              2,
                                              new_particle_color_C,
                                              new_particle_velocity_C,
```

```
                                        2,
                                        new_particle_C_id)

                    particle_group.add(new_particle_A)
                    particle_list.append(new_particle_A)
                    particle_archive.append(new_particle_A)
                    progression_record_cache.append(new_particle_A.id)
                    particle_group.add(new_particle_B)
                    particle_list.append(new_particle_B)
                    particle_archive.append(new_particle_B)
                    progression_record_cache.append(new_particle_B.id)
                    particle_group.add(new_particle_C)
                    particle_list.append(new_particle_C)
                    particle_archive.append(new_particle_C)
                    progression_record_cache.append(new_particle_C.id)

                    decay_chain_list.append((new_particle_A, decay_list[i]))

                    break

                if i == 15:  # --- Beta delayed neutron emission
--------------------------------------------------------------------------------
-----------
                    new_particle_A_id = ID_chart_isotopes[particle.id][i].value
                    new_particle_A_atomic_number =
ID_chart_isotopes[new_particle_A_id][3].value
                    new_particle_velocity_A = (random.randint(-100, 100),
random.randint(-100, 100))
                    new_particle_radius_A = round(new_particle_A_atomic_number **
(0.67), 0)
                    new_particle_color_A = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_A_pos_x = particle.body.position.x
                    new_particle_A_pos_y = particle.body.position.y

                    new_particle_B_id = 0
                    new_particle_B_atomic_number = 1
                    new_particle_velocity_B = (random.randint(-400, 400),
random.randint(-400, 400))
                    new_particle_radius_B = 3
                    new_particle_color_B = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_B_pos_x = particle.body.position.x
                    new_particle_B_pos_y = particle.body.position.y

                    new_particle_B_id = 3183
                    new_particle_B_atomic_number = 1
                    new_particle_velocity_B = (random.randint(-400, 400),
random.randint(-400, 400))
                    new_particle_radius_B = 2
                    new_particle_color_B = (random.randint(50, 200),
random.randint(50, 200), random.randint(50, 200))
                    new_particle_B_pos_x = particle.body.position.x
                    new_particle_B_pos_y = particle.body.position.y

                    for particle in particle_group:
                        if particle.stability != 1:
                            kill_command(particle)

                    new_particle_A = Particle(space,
                                        new_particle_radius_A,
                                        new_particle_A_pos_x,
                                        new_particle_A_pos_y,
                                        new_particle_A_atomic_number,
                                        2,
                                        new_particle_color_A,
                                        new_particle_velocity_A,
                                        2,
```

```
                                    new_particle_A_id)

                    new_particle_B = Particle(space,
                                    new_particle_radius_B,
                                    new_particle_B_pos_x,
                                    new_particle_B_pos_y,
                                    new_particle_B_atomic_number,
                                    2,
                                    new_particle_color_B,
                                    new_particle_velocity_B,
                                    2,
                                    new_particle_B_id)

                    new_particle_C = Particle(space,
                                    new_particle_radius_C,
                                    new_particle_C_pos_x,
                                    new_particle_C_pos_y,
                                    new_particle_C_atomic_number,
                                    2,
                                    new_particle_color_C,
                                    new_particle_velocity_C,
                                    2,
                                    new_particle_C_id)

                    particle_group.add(new_particle_A)
                    particle_list.append(new_particle_A)
                    particle_archive.append(new_particle_A)
                    progression_record_cache.append(new_particle_A.id)
                    particle_group.add(new_particle_B)
                    particle_list.append(new_particle_B)
                    particle_archive.append(new_particle_B)
                    progression_record_cache.append(new_particle_B.id)
                    particle_group.add(new_particle_C)
                    particle_list.append(new_particle_C)
                    particle_archive.append(new_particle_C)
                    progression_record_cache.append(new_particle_C.id)

                    decay_chain_list.append((new_particle_A, decay_list[i]))

                    break

def scale_for_x(scale):
    return int(round(scale * global_scale_x, 0))

def scale_for_y(scale):
    return int(round(scale * global_scale_y, 0))

def loading_screen(load_value):

    screen.blit(loading_display_bg, (0, 0))
    loader_text = "Loading : " + str(round(load_value, 1)) + " %"
    loader_text_surface_1 = text_font_08_loader.render(loader_text, True, "Gray")
    screen.blit(loader_text_surface_1, (launcher_screen_width / 2 -
scale_for_x(100), launcher_screen_height / 2))
    pygame.display.update()
    #clock.tick(FPS)

#-------------------------------------------------------------------------------
-------------------------------------------------------------------------
#-------------------------------------------------------------------------------
-------------------------------------------------------------------------
#-------------------------------------------------------------------------------
-------------------------------------------------------------------------
# --- Putevi za bazu i assete jer IDLE editor (1) i moj Visual Code Studio (0)
nerade jednako s putevima --- #

editor_type = 1
```

```
if editor_type == 1:
    text_font_path = "assets/font/Aero.ttf"
    button_surface_path = "assets/button_surface.png"
    arrow_button_path = "assets/arrow_button.png"
    atoms_icon_path = "assets/atom_icon.png"
    star_icon_path = "assets/star_icon.png"
    star_icon_2_path = "assets/star_icon2.png"
    options_button_path = "assets/button_surface.png"
    options_icon_path = "assets/options_icon.png"
    element_button_path = "assets/element_button.png"
    star_core_1_path = "assets/star_core_1.png"
    star_core_2_path = "assets/star_core_2.png"
    star_core_3_path = "assets/star_core_3.png"
    star_core_4_path = "assets/star_core_4.png"
    star_core_5_path = "assets/star_core_5.png"
    isotope_button_green_path = "assets/isotope_button_green.png"
    isotope_button_red_path = "assets/isotope_button_red.png"
    isotope_button_gray_path = "assets/isotope_button_gray.png"
    isotope_button_gold_path = "assets/isotope_button_gold.png"
    cursor_path = "assets/cursor.png"
    ID_chart_path = "ID_chart.xlsx"
    lock_icon_path = "assets/lock_icon.png"
    progression_record = "progression_record.dat"
    save_icon_path = "assets/save_icon.png"
    book_01_icon_path = "assets/book_01_icon.png"
    book_02_icon_path = "assets/book_02_icon.png"
    book_03_icon_path = "assets/book_03_icon.png"
    hand_icon_path = "assets/hand_icon.png"
    launcher_bg_path = "assets/launcher_bg.png"

else:

    text_font_path = "Game/assets/font/Aero.ttf"
    button_surface_path = "Game/assets/button_surface.png"
    arrow_button_path = "Game/assets/arrow_button.png"
    atoms_icon_path = "Game/assets/atom_icon.png"
    star_icon_path = "Game/assets/star_icon.png"
    star_icon_2_path = "Game/assets/star_icon2.png"
    options_button_path = "Game/assets/button_surface.png"
    options_icon_path = "Game/assets/options_icon.png"
    element_button_path = "Game/assets/element_button.png"
    star_core_1_path = "Game/assets/star_core_1.png"
    star_core_2_path = "Game/assets/star_core_2.png"
    star_core_3_path = "Game/assets/star_core_3.png"
    star_core_4_path = "Game/assets/star_core_4.png"
    star_core_5_path = "Game/assets/star_core_5.png"
    isotope_button_green_path = "Game/assets/isotope_button_green.png"
    isotope_button_red_path = "Game/assets/isotope_button_red.png"
    isotope_button_gray_path = "Game/assets/isotope_button_gray.png"
    isotope_button_gold_path = "Game/assets/isotope_button_gold.png"
    cursor_path = "Game/assets/cursor.png"
    ID_chart_path = "Game/ID_chart.xlsx"
    lock_icon_path = "Game/assets/lock_icon.png"
    progression_record = "Game/progression_record.dat"
    save_icon_path = "Game/assets/save_icon.png"
    book_01_icon_path = "Game/assets/book_01_icon.png"
    book_02_icon_path = "Game/assets/book_02_icon.png"
    book_03_icon_path = "Game/assets/book_03_icon.png"
    hand_icon_path = "Game/assets/hand_icon.png"
    launcher_bg_path = "Game/assets/launcher_bg.png"

#-------------------------------------------------------------------------------
--------------------------------------------------------------------------
# --- Glavne postavke igre --- #

# *** Postavke prostora igre *** #

pygame.init()
```

```
pygame.display.set_caption("Game")
space = pymunk.Space()
space.gravity = (0,0)
options = pymunk.SpaceDebugDrawOptions()
space.debug_draw(options)

# *** Postavke pozadine *** #

display_size = pygame.display.Info().current_w, pygame.display.Info().current_h -
60
screen_width = pygame.display.Info().current_w
screen_height = pygame.display.Info().current_h - 60
screen = pygame.display.set_mode((screen_width, screen_height))
screen_center_cord = (screen_width / 2, screen_height / 2)
screen_org_cord = (0, 0)

global_scale_x = round(screen_width / 1366, 2)
global_scale_y = round(screen_height / 768, 2)
global_scale = (global_scale_x, global_scale_y)

particle_space = screen_width - scale_for_x(400)


# *** Postavke sata i brojača *** #

clock = pygame.time.Clock()
FPS = 60

# *** Postavke zaslona za učitavanje *** #

loading_display_bg_load = pygame.image.load(star_core_1_path).convert_alpha()
loading_display_bg = pygame.transform.scale_by(loading_display_bg_load,
global_scale)
screen.fill(pygame.Color(80, 80, 100))
screen.blit(loading_display_bg, (0, 0))
loading_display_bg.set_alpha(125)

#-------------------------------------------------------------------------------
-------------------------------------------------------------------------
# --- Postavke kursora (pokazivača) --- #

cursor = Cursor()
cursor_group = pygame.sprite.Group()
cursor_group.add(cursor)
pygame.mouse.set_visible(False)

#-------------------------------------------------------------------------------
-------------------------------------------------------------------------
# --- Postavke fonta i teksta --- #

text_font_01_isotopes = pygame.font.Font(text_font_path, scale_for_y(25))
text_font_02_elements = pygame.font.Font(text_font_path, scale_for_y(28))
text_font_03_bin = pygame.font.Font(text_font_path, scale_for_y(30))
text_font_04_star = pygame.font.Font(text_font_path, scale_for_y(45))
text_font_05_particle = pygame.font.Font(text_font_path, scale_for_y(14))
text_font_08_loader = pygame.font.Font(text_font_path, scale_for_y(50))
text_font_09_decay_speed = pygame.font.Font(text_font_path, scale_for_y(40))

#-------------------------------------------------------------------------------
-------------------------------------------------------------------------
#-------------------------------------------------------------------------------
-------------------------------------------------------------------------
#-------------------------------------------------------------------------------
-------------------------------------------------------------------------
# --- Postavke launchera --- #

launcher_screen_width = scale_for_x(1280) - 10
launcher_screen_height = scale_for_y(720)
```

```
launcher_screen = pygame.display.set_mode((launcher_screen_width,
launcher_screen_height))

launcher_menu_surface = pygame.Rect(0, 0, launcher_screen_width,
launcher_screen_height)
launcher_menu_image = pygame.image.load(launcher_bg_path).convert_alpha()
launcher_menu_color = (20, 20, 50)

launcher_button_exploration = Button(scale_for_x(400), scale_for_y(300),
button_surface_path, "launcher_menu", 1.3)
launcher_button_creative = Button(scale_for_x(800), scale_for_y(300),
button_surface_path, "launcher_menu", 1.3)
launcher_button_instructions = Button(scale_for_x(600), scale_for_y(600),
button_surface_path, "launcher_menu", 1.3)
launcher_arrow_button = Button(6000 + scale_for_x(1210), scale_for_y(50),
arrow_button_path, "all_menu", 1.0, icon = True)

launcher_button_list = []
launcher_button_list.append(launcher_button_exploration)
launcher_button_list.append(launcher_button_creative)
launcher_button_list.append(launcher_button_instructions)
launcher_button_list.append(launcher_arrow_button)

text_font_06_launcher = pygame.font.Font(text_font_path, scale_for_y(45))
text_font_07_instructions = pygame.font.Font(text_font_path, scale_for_y(30))

launcher_text_surface_1 = text_font_06_launcher.render("Exploration", True, "Gray")
launcher_text_surface_2 = text_font_06_launcher.render("Creative", True, "Gray")
launcher_text_surface_3 = text_font_06_launcher.render("Instructions", True,
"Gray")
launcher_instruction_surface_00 = text_font_06_launcher.render("Instructions",
True, "Gray")
launcher_instruction_surface_01 = text_font_07_instructions.render("Welcome to the
Nuclear fusion simulator", True, "Gray")
launcher_instruction_surface_02 = text_font_07_instructions.render("Main goal of
the game is to create as many elements", True, "Gray")
launcher_instruction_surface_03 = text_font_07_instructions.render("      as you
can through Nuclear fusion", True, "Gray")
launcher_instruction_surface_04 = text_font_07_instructions.render("To start, first
select game mode you wish to play", True, "Gray")
launcher_instruction_surface_05 = text_font_07_instructions.render("In Exploration
mode you start with only few lighter", True, "Gray")
launcher_instruction_surface_06 = text_font_07_instructions.render("      elements
in your hand and in Creative there is ", True, "Gray")
launcher_instruction_surface_07 = text_font_07_instructions.render("      no
restrictions ... Do as you like", True, "Gray")
launcher_instruction_surface_08 = text_font_07_instructions.render("Once the game
loads simply open the top right menu", True, "Gray")
launcher_instruction_surface_09 = text_font_07_instructions.render("      where you
can select isotopes to collide and more", True, "Gray")
launcher_instruction_surface_10 = text_font_07_instructions.render("      will
unlock as you progress", True, "Gray")
launcher_instruction_surface_11 = text_font_07_instructions.render("Golden isotopes
will let you unlock new stars which", True, "Gray")
launcher_instruction_surface_12 = text_font_07_instructions.render("      you can
select in secund menu on the right", True, "Gray")
launcher_instruction_surface_13 = text_font_07_instructions.render("You'll need
stronger stars to produce heavier elements", True, "Gray")
launcher_instruction_surface_14 = text_font_07_instructions.render("      ... so
Good Luck! ", True, "Gray")

# --- Pokretanje launchera --- #

launcher = True
display_type = "launcher_menu"
creative = False

while launcher == True:
```

```
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
            mouse_pos = pygame.mouse.get_pos()

            if display_type != "launcher_menu" and
launcher_arrow_button.rect.collidepoint(mouse_pos) == True:
                display_type = "launcher_menu"
                for button_select in launcher_button_list:
                    if button_select.display_group == "launcher_menu" or
button_select.display_group == "all_menu":
                        button_select.arrange()

            if display_type == "launcher_menu" and
launcher_button_instructions.rect.collidepoint(mouse_pos) == True:
                display_type = "launcher_instructions"
                for button_select in launcher_button_list:
                    if button_select.display_group == "launcher_menu" or
button_select.display_group == "all_menu":
                        button_select.arrange()

            if display_type == "launcher_menu" and
launcher_button_exploration.rect.collidepoint(mouse_pos) == True:
                creative = False
                launcher = False
            if display_type == "launcher_menu" and
launcher_button_creative.rect.collidepoint(mouse_pos) == True:
                creative = True
                launcher = False

    display_UI(display_type, launcher_bg_path)

    cursor_group.draw(screen)
    cursor_group.update()

    pygame.display.update()
    clock.tick(FPS)
    space.step(1/FPS)

#-------------------------------------------------------------------------------
--------------------------------------------------------------------------
# --- Postavke sučelja i izbornika --- #

# *** Postavke izbornika *** #


main_menu_surface = pygame.Rect((particle_space), 0, screen_width-particle_space,
screen_height)
main_menu_color = pygame.Color(80, 80, 100)
elements_menu_surface = pygame.Rect(0, 0, screen_width, screen_height)
elements_menu_color = pygame.Color(20, 20, 50)
elements_menu_color_2 = pygame.Color(20, 20, 50)
options_menu_surface = pygame.Rect(0, 0, screen_width, screen_height)
options_menu_color = pygame.Color(50, 50, 80)
star_menu_surface = pygame.Rect(0, 0, screen_width, screen_height)
star_menu_color = pygame.Color(50, 50, 80)

screen.fill(pygame.Color(80, 80, 100))
loading_screen(0.0)

# *** Zastor za izotope da ne pređu preko strelice *** #

element_box_width = scale_for_x(48)
element_box_height = scale_for_y(76)
elements_menu_border = pygame.Rect(scale_for_x(screen_width - 100), 0,
```

```
scale_for_x(600), scale_for_y(90))

# *** Postavke padajučeg izbornika *** #

dropmenu_rect_pos_x = 1130
dropmenu_rect_pos_y = 120
dropmenu_scroll = scale_for_y(20)

# *** Postavke za dizanje i spuštanje izbornika s kolutičem miša *** #

scroll_up_limit = scale_for_y(75)
scroll_down_limit = screen_height - scale_for_y(25)
scroll_push_limit = scroll_up_limit + dropmenu_scroll * 2

#-------------------------------------------------------------------------------
-------------------------------------------------------------------
# --- Postavke ikona --- #

lock_icon_load = pygame.image.load(lock_icon_path).convert_alpha()
lock_icon = pygame.transform.scale_by(lock_icon_load, global_scale)
star_icon_2_load = pygame.image.load(star_icon_2_path).convert_alpha()
star_icon_2 = pygame.transform.scale_by(star_icon_2_load, global_scale)

#-------------------------------------------------------------------------------
-------------------------------------------------------------------
# --- Postavke zidova za čestice --- #

wall_group = pygame.sprite.Group()

for i in range(4):
    wall = Wall(space, i, 0)
    wall_group.add(wall)

#-------------------------------------------------------------------------------
-------------------------------------------------------------------
# --- Postavke čestica --- #

particle_group = pygame.sprite.Group()
particle_list = []
particle_archive = []
decay_chain_list = []

#-------------------------------------------------------------------------------
-------------------------------------------------------------------
# --- Postavke progresije --- #

progression = 1

progression_record_cache = []
progression_recorder = open(progression_record, "a+")
progression_reader = numpy.loadtxt(progression_record, dtype = float, unpack =
True, usecols = 0).astype(int)

if creative == True:
    progression = 5
else:
    if 57 in progression_record_cache or 58 in progression_record_cache:
            progression = 2
    if 106 in progression_record_cache:
            progression = 3
    if 506 in progression_record_cache or 508 in progression_record_cache or 509 in
progression_record_cache or 510 in progression_record_cache:
            progression = 4
    if 2814 in progression_record_cache or 2817 in progression_record_cache:
            progression = 5

#-------------------------------------------------------------------------------
-------------------------------------------------------------------
```

```
# --- Postavke elemenata i izotopa --- #

element_grid = [[  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  2],
                [  3,  4,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  5,  6,  7,  8,
9, 10],
                [ 11, 12,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 13, 14, 15, 16,
17, 18],
                [ 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
35, 36],
                [ 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52,
53, 54],
                [ 55, 56,  0, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
85, 86],
                [ 87, 88,
0,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118],
                [  0,  0, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70,
71,  0],
                [  0,  0, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98,
99,100,101,102,103,  0]]

element_grid_name = [[  "H",    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,    0,    0,    0,    0,    0,    0, "He"],
                     [ "Li", "Be",    0,    0,    0,    0,    0,    0,    0,    0,
0,    0,  "B",  "C",  "N",  "O",  "F", "Ne"],
                     [ "Na", "Mg",    0,    0,    0,    0,    0,    0,    0,    0,
0,    0, "AL", "Si",  "P",  "S", "Cl", "Ar"],
                     [  "K", "Ca", "Sc", "Ti",  "V", "Cr", "Mn", "Fe", "Co", "Ni",
"Cu", "Zn", "Ga", "Ge", "As", "Se", "Br", "Kr"],
                     [ "Rb", "Sr",  "Y", "Zr", "Nb", "Mo", "Tc", "Ru", "Rh", "Pd",
"Ag", "Cd", "In", "Sn", "Sb", "Te",  "I", "Xe"],
                     [ "Cs", "Ba",    0, "Hf", "Ta",  "W", "Re", "Os", "Ir", "Pt",
"Au", "Hg", "Tl", "Pb", "Bi", "Po", "At", "Rn"],
                     [ "Fr", "Ra",    0, "Rf", "Db", "Sg", "Bh", "Hs", "Mt", "Ds",
"Rg", "Cn", "Nh", "Fl", "Mc", "Lv", "Ts", "Og"],
                     [    0,    0, "La", "Ce", "Pr", "Nd", "Pm", "Sm", "Eu", "Gd",
"Tb", "Dy", "Ho", "Er", "Tm", "Yb", "Lu",    0],
                     [    0,    0, "Ac", "Th", "Pa",  "U", "Np", "Pu", "Am", "Cm",
"Bk", "Cf", "Es", "Fm", "Md", "No", "Lr",    0]]

#-------------------------------------------------------------------------------
-------------------------------------------------------------------------
# --- Uvoz iz baze podataka --- #

ID_chart_workbook = openpyxl.load_workbook(ID_chart_path)
ID_chart_sheet = ID_chart_workbook.active
ID_chart_isotopes = []
ID_chart_list = []
decay_list = []

for i in range(2, 3186):
    ID_chart_row = []
    for j in range(1, 19):
        ID_chart_cell = ID_chart_sheet.cell(i, j)
        ID_chart_row.append(ID_chart_cell)
    ID_chart_isotopes.append(ID_chart_row)
    if i % 100 == 0:
        loading_screen(i / 32)

for i in range(1, 119):
    ID_chart_elements = []
    for j in range(len(ID_chart_isotopes)):
        if ID_chart_isotopes[j][2].value == i:
            ID_chart_element = ID_chart_isotopes[j]
            ID_chart_elements.append(ID_chart_element)
        if ID_chart_isotopes[j][2].value > i:
            break
    ID_chart_list.append(ID_chart_elements)
```

```
for i in range(1, 17):
    ID_chart_cell = ID_chart_sheet.cell(1, i).value
    if i < 7:
        decay_list.append(None)
    else:
        decay_list.append(ID_chart_cell)


#-------------------------------------------------------------------------------
-------------------------------------------------------------------------
# --- Postavke gumba izbornika --- #

# *** Postavke tipki glavnog izbornika *** #

button_list = []

arrow_button = Button(6000 + scale_for_x(1300), scale_for_y(50), arrow_button_path,
"all_menu", icon = True)
button_list.append(arrow_button)

elements_button = Button(scale_for_x(1160), scale_for_y(200), button_surface_path,
"main_menu")
button_list.append(elements_button)
atoms_icon = Button(scale_for_x(1060), scale_for_y(200) - 10, atoms_icon_path,
"main_menu", icon = True)
button_list.append(atoms_icon)

star_button = Button(scale_for_x(1160), scale_for_y(400), button_surface_path,
"main_menu")
button_list.append(star_button)
star_icon = Button(scale_for_x(1260), scale_for_y(400) - 10, star_icon_path,
"main_menu", icon = True)
button_list.append(star_icon)

options_icon = Button(scale_for_x(70), scale_for_y(70), options_icon_path,
"main_menu", 0.5, icon = True)
button_list.append(options_icon)
save_button = Button(screen_width - scale_for_x(30), screen_height -
scale_for_y(30), save_icon_path, "main_menu", 0.6, icon = True)
button_list.append(save_button)

# *** Postavke tipki za elemente *** #

element_button_list = []
element_name_list = []
element_cord_list = []

for i in range (118):
    element_button_list.append(0)
    element_name_list.append(0)
    element_cord_list.append((0, 0))

for i in range(0, 9):
    for j in range(0, 18):
        if element_grid[i][j] != 0:
            element_cord_list[element_grid[i][j] - 1] = (scale_for_x(25 + 50 * j),
scale_for_y(55 + 76 * i + 36))
            element = Button(element_cord_list[element_grid[i][j] - 1]
[0],element_cord_list[element_grid[i][j] - 1][1], element_button_path,
"element_menu")
            element_button_list[element_grid[i][j] - 1] = element
            element_name_list[element_grid[i][j] - 1] = element_grid_name[i][j]

loading_screen(100.0)

# *** Postavke tipki za izotope *** #
```

```
element_selection_list = []

for i in range(len(element_button_list)):
    isotope_selection_list = []

    for j in range(len(ID_chart_list[i])):
        isotope_selection = [ID_chart_list[i][j][0].value, ID_chart_list[i][j]
[1].value, ID_chart_list[i][j][2].value]
        isotope_selection_list.append(isotope_selection)

    isotope_counter = len(ID_chart_list[i])
    isotope_selection_list.append(isotope_counter)
    element_selection_list.append(isotope_selection_list)

isotope_button_list = []


# *** Postavke gumba za zvjezde*** #

star_core_list = []

star_core_1_button = Button(6000 + scale_for_x(200), scale_for_y(600),
button_surface_path, "star_menu", 1.2)
button_list.append(star_core_1_button)
star_core_list.append([star_core_1_button, (scale_for_x(80), scale_for_y(560))])

star_core_2_button = Button(6000 + scale_for_x(300), scale_for_y(500),
button_surface_path, "star_menu", 1.2)
button_list.append(star_core_2_button)
star_core_list.append([star_core_2_button, (scale_for_x(180), scale_for_y(460))])

star_core_3_button = Button(6000 + scale_for_x(400), scale_for_y(400),
button_surface_path, "star_menu", 1.2)
button_list.append(star_core_3_button)
star_core_list.append([star_core_3_button, (scale_for_x(280), scale_for_y(360))])

star_core_4_button = Button(6000 + scale_for_x(500), scale_for_y(300),
button_surface_path, "star_menu", 1.2)
button_list.append(star_core_4_button)
star_core_list.append([star_core_4_button, (scale_for_x(380), scale_for_y(260))])

star_core_5_button = Button(6000 + scale_for_x(600), scale_for_y(200),
button_surface_path, "star_menu", 1.2)
button_list.append(star_core_5_button)
star_core_list.append([star_core_5_button, (scale_for_x(480), scale_for_y(160))])

# *** Postavke gumba za opcije*** #

decay_speed_text_surface_1 = text_font_09_decay_speed.render("Decay speed", True,
"Black")
decay_speed_text_surface_2 = text_font_09_decay_speed.render("Fast", True, "Gray")
decay_speed_text_surface_3 = text_font_09_decay_speed.render("Medium", True,
"Gray")
decay_speed_text_surface_4 = text_font_09_decay_speed.render("Slow", True, "Gray")

decay_speed_text_surface_pointer = text_font_09_decay_speed.render(" >", True,
"Gray")

decay_speed_rect_02 = pygame.Rect(scale_for_x(100), scale_for_y(220), 200, 50)
decay_speed_rect_03 = pygame.Rect(scale_for_x(100), scale_for_y(320), 200, 50)
decay_speed_rect_04 = pygame.Rect(scale_for_x(100), scale_for_y(420), 200, 50)


#-------------------------------------------------------------------------------
--------------------------------------------------------------------
# --- Inicijacija igre --- #

deley_counter = 0
```

```
delay_max = 200
display_type = "main_menu"
previus_display_type = ""
current_star = [star_core_1_path, 1]
collision_bin = []
screen = pygame.display.set_mode((screen_width, screen_height))

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
            mouse_pos = pygame.mouse.get_pos()

            if display_type != "main_menu" and
arrow_button.rect.collidepoint(mouse_pos) == True:
                display_type = "main_menu"
                for button_select in button_list:
                    if button_select.display_group == "main_menu" or
button_select.display_group == "all_menu" or button_select.display_group ==
previus_display_type:
                        button_select.arrange()
                previus_display_type = "main_menu"
                isotope_button_list = []

            if display_type == "main_menu" and
save_button.rect.collidepoint(mouse_pos) == True :
                for particle in progression_record_cache:
                    if particle in progression_reader:
                        continue
                    else:
                        progression_recorder.write("%4s\n" % particle)

            if elements_button.rect.collidepoint(mouse_pos) == True:

                for particle in particle_group:
                    kill_command(particle)
                collision_bin = []
                particle_list = []
                particle_group.empty()
                particle_archive = []
                deley_counter = 0

                display_type = "elements_menu"
                isotope_button_list = []
                for button_select in button_list:
                    if button_select.display_group == "elements_menu" or
button_select.display_group == "main_menu" or button_select.display_group ==
"all_menu":
                        button_select.arrange()
                previus_display_type = "elements_menu"

            if star_button.rect.collidepoint(mouse_pos) == True:

                if creative == True:
                    progression = 5
                else:
                    if 57 in progression_record_cache or 58 in
progression_record_cache:
                        progression = 2
                    if 106 in progression_record_cache:
                        progression = 3
                    if 506 in progression_record_cache or 508 in
progression_record_cache or 509 in progression_record_cache or 510 in
progression_record_cache:
                        progression = 4
```

```
                    if 2814 in progression_record_cache or 2817 in
progression_record_cache:
                        progression = 5

                for particle in particle_group:
                    kill_command(particle)
                collision_bin = []
                particle_list = []
                particle_group.empty()
                particle_archive = []
                deley_counter = 0

                display_type = "star_menu"
                for button_select in button_list:
                    if button_select.display_group == "star_menu" or
button_select.display_group == "main_menu" or button_select.display_group ==
"all_menu":
                        button_select.arrange()
                previus_display_type = "star_menu"

            if options_icon.rect.collidepoint(mouse_pos) == True:

                for particle in particle_group:
                    kill_command(particle)
                collision_bin = []
                particle_list = []
                particle_group.empty()
                particle_archive = []
                deley_counter = 0

                display_type = "options_menu"
                for button_select in button_list:
                    if button_select.display_group == "options_menu" or
button_select.display_group == "main_menu" or button_select.display_group ==
"all_menu":
                        button_select.arrange()
                previus_display_type = "options_menu"

            if display_type == "star_menu":
                if star_core_1_button.rect.collidepoint(mouse_pos) == True and
current_star[0] != star_core_1_path:
                    star_transition_event(current_star[0], star_core_1_path)
                    current_star = [star_core_1_path, 1]
                    display_type = "main_menu"
                    for button_select in button_list:
                        if button_select.display_group == "star_menu" or
button_select.display_group == "all_menu" or button_select.display_group ==
"main_menu":
                            button_select.arrange()
                    previus_display_type = "star_menu"

                if star_core_2_button.rect.collidepoint(mouse_pos) == True and
current_star[0] != star_core_2_path and progression >= 2:
                    star_transition_event(current_star[0], star_core_2_path)
                    current_star = [star_core_2_path, 2]
                    display_type = "main_menu"
                    for button_select in button_list:
                        if button_select.display_group == "star_menu" or
button_select.display_group == "all_menu" or button_select.display_group ==
"main_menu":
                            button_select.arrange()
                    previus_display_type = "star_menu"

                if star_core_3_button.rect.collidepoint(mouse_pos) == True and
current_star[0] != star_core_3_path and progression >= 3:
                    star_transition_event(current_star[0], star_core_3_path)
                    current_star = [star_core_3_path, 3]
                    display_type = "main_menu"
```

```
                    for button_select in button_list:
                        if button_select.display_group == "star_menu" or
button_select.display_group == "all_menu" or button_select.display_group ==
"main_menu":
                            button_select.arrange()
                    previus_display_type = "star_menu"

                if star_core_4_button.rect.collidepoint(mouse_pos) == True and
current_star[0] != star_core_4_path and progression >= 4:
                    star_transition_event(current_star[0], star_core_4_path)
                    current_star = [star_core_4_path, 4]
                    display_type = "main_menu"
                    for button_select in button_list:
                        if button_select.display_group == "star_menu" or
button_select.display_group == "all_menu" or button_select.display_group ==
"main_menu":
                            button_select.arrange()
                    previus_display_type = "star_menu"

                if star_core_5_button.rect.collidepoint(mouse_pos) == True and
current_star[0] != star_core_5_path and progression >= 5:
                    star_transition_event(current_star[0], star_core_5_path)
                    current_star = [star_core_5_path, 5]
                    display_type = "main_menu"
                    for button_select in button_list:
                        if button_select.display_group == "star_menu" or
button_select.display_group == "all_menu" or button_select.display_group ==
"main_menu":
                            button_select.arrange()
                    previus_display_type = "star_menu"

            if display_type == "elements_menu" or display_type ==
"elements_menu_isotopes":
                for element_button_select in element_button_list:
                    if element_button_select.rect.collidepoint(mouse_pos) == True:
                        display_type = "elements_menu_isotopes"
                        isotope_proton_number =
element_button_list.index(element_button_select) + 1
                        isotope_number =
element_selection_list[isotope_proton_number - 1][-1]
                        isotope_ID_list = []
                        isotope_button_list = []

                        for i in range(isotope_number):
                            isotope_ID =
element_selection_list[isotope_proton_number - 1][i][0]
                            isotope_ID_list.append(isotope_ID)

                            if creative == False:
                                if ID_chart_list[isotope_proton_number - 1][i]
[17].value != None:
                                    isotope_name_background =
isotope_button_gold_path
                                elif isotope_ID in progression_reader or isotope_ID
in progression_record_cache:
                                    if ID_chart_list[isotope_proton_number - 1][i]
[5].value == 1 and ID_chart_list[isotope_proton_number - 1][i][16].value <=
current_star[1]:
                                        isotope_name_background =
isotope_button_green_path
                                    else:
                                        isotope_name_background =
isotope_button_red_path
                                else:
                                    isotope_name_background =
isotope_button_gray_path
                            else:
                                if ID_chart_list[isotope_proton_number - 1][i]
```

```
[5].value == 1 and ID_chart_list[isotope_proton_number - 1][i][16].value <=
current_star[1]:
                                    isotope_name_background =
isotope_button_green_path
                                else:
                                    isotope_name_background =
isotope_button_red_path

                            isotope_button =
Button(scale_for_x(dropmenu_rect_pos_x),
                                                   scale_for_y(dropmenu_rect_pos_y
+ i * 55),
                                                   isotope_name_background,
"elements_menu")
                            isotope_button_list.append(isotope_button)

            if display_type == "elements_menu_isotopes":
                for isotope_button_select in isotope_button_list:
                    isotope_proton_number =
isotope_button_list.index(isotope_button_select)
                    isotope_ID =
isotope_ID_list[isotope_button_list.index(isotope_button_select)]
                    if isotope_button_select.rect.collidepoint(mouse_pos) == True:
                        if creative == False:
                            if isotope_ID in progression_reader or isotope_ID in
progression_record_cache:
                                if ID_chart_isotopes[isotope_ID][5].value == 1:
                                    if ID_chart_isotopes[isotope_ID][16].value <=
current_star[1]:
                                        collision_bin.append(isotope_ID)
                                        if len(collision_bin) == 2:
                                            collision_prep(collision_bin[0],
collision_bin[1])
                                            display_type = "main_menu"
                                            collision_bin = []
                                            for button_select in button_list:
                                                if button_select.display_group ==
"elements_menu" or button_select.display_group == "main_menu" or
button_select.display_group == "all_menu":
                                                    button_select.arrange()
                                            previus_display_type = "elements_menu"
                        if creative == True:
                            if ID_chart_isotopes[isotope_ID][5].value == 1:
                                if ID_chart_isotopes[isotope_ID][16].value <=
current_star[1]:
                                    collision_bin.append(isotope_ID)
                                    if len(collision_bin) == 2:
                                        collision_prep(collision_bin[0],
collision_bin[1])
                                        display_type = "main_menu"
                                        collision_bin = []
                                        for button_select in button_list:
                                            if button_select.display_group ==
"elements_menu" or button_select.display_group == "main_menu" or
button_select.display_group == "all_menu":
                                                button_select.arrange()
                                        previus_display_type = "elements_menu"

            if display_type == "options_menu":
                if decay_speed_rect_02.collidepoint(mouse_pos) == True:
                    delay_max = 100
                if decay_speed_rect_03.collidepoint(mouse_pos) == True:
                    delay_max = 200
                if decay_speed_rect_04.collidepoint(mouse_pos) == True:
                    delay_max = 300

        if display_type == "elements_menu_isotopes":
            if event.type == pygame.MOUSEWHEEL:
```

```
                if isotope_button_list[0].rect[1] <= scroll_up_limit:
                    if event.y == 1:
                        for isotope in isotope_button_list:
                            isotope.scroll_arrange(-event.y)

                if isotope_button_list[-1].rect[1] >= scroll_down_limit and
isotope_button_list[-1].rect[1] >= scroll_push_limit:
                    if event.y == -1:
                        for isotope in isotope_button_list:
                            isotope.scroll_arrange(-event.y)

    handler_1 = space.add_collision_handler(1, 1)

    for wall in wall_group:
        wall.draw()

    display_UI(display_type, current_star[0])

    if len(particle_group) != 0:
        deley_counter += 1

    if deley_counter == delay_max:
        for particle in particle_group:
            decay(particle)
        deley_counter = 0

    cursor_group.draw(screen)
    cursor_group.update()

    pygame.display.update()
    clock.tick(FPS)
    space.step(1/FPS)
```