Colegiul Național „Roman-Vodă”

Proiect pentru obținerea atestatului de competențe digitale

Elev,
Filip Tudor-Mihail

Profesor,
Gavril Petru Florin

Sesiunea – Iunie 2020

Temă:

Programare joc în python

Nume proiect:

Chicken Invaders

Argument

,,Chicken Invaders" este numele unei serii de jocuri video de acțiune, petrecute în spațiu, devenită foarte virală la începutul acestui mileniu. Jocurile erau inițial valabile doar pentru platformele de tip PC, cu Windows, dar versiunile mai noi pot fi jucate atât pe diferite sisteme de operare, cât și pe diverse platforme mobile. În joc ai rolul unui comandant de navă spațială, a cărui misiune este de a omorî gâini pentru a le vinde și a putea trece mai departe la noi nivele. Întreaga acțiune are loc într-un cadru relativ amuzant, jocul căpătând astfel o reputație mare, atrăgând, de-a lungul anilor, milioane de jucători. Fiind chiar primul joc video pe care l-am jucat vreodată, am hotărât să încerc să creez chiar eu o versiune a sa, folosindu-mă de cunoștințele pe care le-am căpătat în liceu, cât și pe cele pe care le-am căpătat în particular, mai ales cele legate de Python, limbajul în care am scris aplicația.

Cuprins

1. Python – Limbaj de programare

Python este un limbaj de programare dinamic multi-paradigmă (suportă mai multe forme de programare), creat în 1989 de programatorul olandez Guido van Rossum. Van Rossum este și în ziua de astăzi un lider al comunității de dezvoltatori de software care lucrează la perfecționarea limbajul Python și implementarea de bază a acestuia, CPython, scrisă în C. Python este un limbaj multifuncțional folosit de exemplu de către companii ca *Google* sau *Yahoo!* pentru programarea aplicațiilor web, însă există și o serie de aplicații științifice sau de divertisment programate parțial sau în întregime în Python.

Python pune accentul pe curățenia și simplitatea codului, iar sintaxa sa le permite dezvoltatorilor să exprime unele idei programatice într-o manieră mai clară și mai concisă decât în alte limbaje de programare ca C. În ceea ce privește metoda de programare, Python poate servi ca limbaj pentru software de tipul *object-oriented*, dar permite și programarea imperativă, funcțională sau procedurală.

1.1. Bibliotecile în Python

Includerea tuturor structurilor, precum și a funcțiilor ce permit manipularea și prelucrarea lor, precum și multe alte biblioteci de funcții sunt prezente datorită conceptului *"Batteries Included"*, concept ce îi conferă limbajului de programare utilitate practică, având un set de biblioteci importante pentru majoritatea dezvoltatorilor.

Din acest motiv Python include biblioteci pentru a ajuta la lucrul cu fișiere, arhive, fișiere XML și un set de biblioteci pentru a lucra cu rețeaua și principalele protocoale de comunicare pe internet (HTTP, Telnet, FTP). Totodată, un număr mare de platforme Web sunt construite cu Python, iar tot cu acesta se poate cu ușurință crea realitate augumentată.

1.2. Pygame

Una dintre cele mai renumite și complexe biblioteci este Pygame. A fost creată în anul 2000, cu scopul de a ușura programarea jocurilor video în Python, iar de atunci se tot extinde. Cuprinde, la rândul ei, o întreagă serie de librării și module pentru gestionarea tuturor elementelor grafice și audio.

2. Chicken Invaders

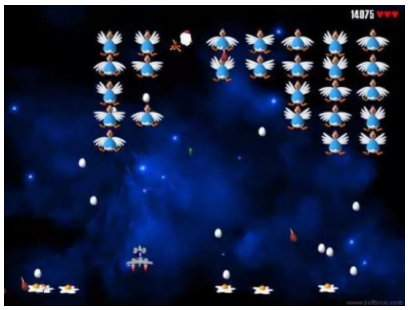
2.1. Gameplay

Scopul jocului este de a împușca găinile extraterestre care atacă Pământul. De asemenea, există și nivele cu asteroizi. Jucătorul controlează o navă spațială, având posibilitatea să se miște pe toată suprafața ecranului, trăgând în găini. Când o găină este împușcată, un copan va apărea, iar jucătorul trebuie să colecteze cât mai multe pentru a obține rachete, care sunt mai puternice decât arma obișnuită și sunt limitate. Totodată, jucătorul va trebui să colecteze cadouri pentru a-și îmbunătăți arma. La finalul rundei trebuie să te confrunți cu un *boss,* care este puțin mai greu de învins decât un inamic obișnuit.

Inițial gameplay-ul nu a fost prea complex, dar odată cu trecerea anilor au apărut noi variante, care presupuneau noi caracteristici și tipuri noi de nivele. La momentul de față sunt 5 versiuni diferite ale jocului, lansate pentru PC, unele având ediții separate, cea de a 6-a versiune aflată încă în stadiul de dezvoltare.

2.2. Versiuni

Prima variantă a jocului, care reprezintă și versiunea originală a acestuia, a apărut în 1999, sub denumirea generică de *Chicken Invaders*, creată de *InterAction Studios*. La baza ei se află o librărie grafică scrisă în Pascal. Jocul era ceva unic la acea perioadă. Te puteai juca alături de un prieten, dificultatea creștea proporțional cu cât avansai în misiuni, armele puteau fi dezvolate și exista un tabel cu scoruri, dar valabil doar pentru tine.



Chicken Invaders: The Next Wave este cea de a doua versiune a jocului, din 2002, și nu a adus multe îmbunătățiri față de ediția precedentă. Singurele schimbări majore au fost legate de controlul navei, acum putând să te miști și în sus sau în jos, nu doar stânga și dreapta, iar, pe lângă asta, culorile armelor puteau fi schimbate.

Cel de al treilea joc din serie este *Chicken Invaders: Revenge of the Yolk* care a fost lansat în 3 ediții separate (2006, 2007 și 2010). A prezentat o dezvoltare considerabilă față de predecesorul său, venind în plus cu 120 de nivele, arme cu super-puteri variate și care se supraîncalzesc dacă tragi încontinuu, o grafică și gameplay mult mai îmbunătățite, fiind adaugată o coloană sonoră pe fundal. Totodată, este primul joc din serie care se poate conecta la internet și la care ai ocazia să te întreci cu jucători din toată lumea.

A patra versiune a jocului este *Chicken Invaders 4: Ultimate Omelette*, lansată oficial în 2010. Acest nou joc nu aduce aproape nimic nou, în schimb este o versiune mult mai bine optimizată, eliminand majoritatea erorilor din versiunile trecute. La rândul ei, are 3 ediții diferite, una de Paște, alta de Crăciun și una de Ziua Recunoștinței.

Cea de a cincea și ultima versiune apărută este denumită *Chicken Invaders 5: Cluck of the Dark Side*. A fost făcută publică în 2014. Este cel mai dezvoltat joc din serie, cu

caracteristici noi față de predecesorii lui, nava putând fi acum personificată pentru prima oară, iar gamplay-ul devenind foarte complex.

2.3. Critici

Seria de jocuri a fost bine primită de toți pasionații de jocuri video din toată lumea. Deși primele jocuri au fost criticate că nu aveau o complexitate mărită și că se bazau pe aceleași idei repetitive, a ajuns rapid cunoscut deoarece este ușor de jucat și distractiv. Versiunea din 2010 a fost inclusă într-un top 100 a celor mai jucate jocuri de acest tip existente.

3. Aplicație

Aplicația pe care am decis să o creez pentru proiectul de atestat este creată în programul *PyCharm*. Am ales să folosesc *PyCharm* deoarece este ușor de folosit pentru utilizatorii începători în programarea cu python și pentru că oferă un suport sigur pentru jocul pe care l-am creat.

Jocul propriu-zis constă în manevrarea unei nave spațiale, cu ajutorul săgeților stânga și dreapta, într-o consolă de 800x600 pixeli, cu scopul este de a elimina cât mai mulți inamici posibili (găini), folosind o rază laser (proiectil). Proiectilul este lansat prin apăsarea tastei *Space*, iar un inamic eliminat va adăuga un punct la scor. La fiecare 10 inamici eliminați, numărul inamicilor crește cu unu. De asemenea, jocul are o muzică de fundal care poate fi oprită apăsând tasta *M*, și repornită apăsând tasta *P*. Alte sunete prezente sunt cele făcute în momentul în care tragi proiectilul și în momentul în care un inamic a fost lovit. Jocul se termină când unul dintre inamici ajunge la aceeași înălțime cu nava spațială, pe ecran apărând textul *Game Over* și punctajul final al jucătorului, însoțite de un sunet specific.

3.1. Inițializare

Codul aplicației este reprezentat de o singură pagină principală, denumită *main.py*, căreia îi sunt incluse mai multe librării, module și fișiere externe.

Pentru început, trebuie să introducem principala librărie cu care vom lucra, anume *Pygame*. Rolul acestei librării este explicat mai sus, la pagina 4, capitolul 1.2., dar, pe scurt, aceasta ne permite să folosim toate funcțiile și modulele care ne vor trebui să creem jocul. Următorul pas constă în introducerea modulelor. Vom folosi *math* și *random*, module implicitate alte limbajului python, și *mixer*, modul specific librăriei *Pygame*.

- *math* – ne permite să folosim formule matematice specifice limbajului C;
- *random* – permite generarea de valori aleatorii, de diferite tipuri;
- *mixer* – ajută la controlul sunetelor folosite în aplicație.

```
import pygame
import math
import random
from pygame import mixer

# Initializarea librariei
pygame.init()
```

3.2. Funcție principală

Principala funcție a programului este reprezentată de o instrucțiune repetitivă de tip *while*, ce are un singur parametru, variabila *ruleaza*. Variabila este de tip *bool*, având la declarare valoarea *True*. Dacă sunt îndeplinite anumite condiții în timpul rulării secvențelor de cod din cadrul funcției *while*, variabila *ruleaza* își schimbă valoarea în *False*, ducând astfel la oprirea aplicației.

În cadrul funcției principale sunt introduse mai multe subfuncții, cu roluri diferite, cât și secvențe separate de cod, care sunt descrise mai jos.

```
# Baza
ruleaza = True
while ruleaza:
```

3.3. Creare pagină de joc

Pagina de joc este o consolă de dimensiuni fixe, 800x600 pixeli, care va fi afișată ori de câte ori rulăm programul. Cu rolul de a o face mai atrăgătoare, aceasta are un fundal care se mișcă, muzică de acompaniament și un titlu și logo pentru bara de stare.

3.3.1. Fundal

După inițializarea paginii, primul lucru pe care îl facem este să adăugam un fundal jocului nostru. Creăm variabila globală *fundal*, care va prelua poza pe care o dorim afișată, cât și variabiala globală *y*, ceva va avea inițial valoarea 0, adică punctul pe axa de coordonate Y din care va începe afișarea pozei (colțul din stânga sus).

În funcția principală *while* sunt scrise o serie de instrucțiuni care vor face ca imaginea din spate să pară că se mișcă, de sus in jos, la finalul cărora variabilei *y* îi va fi adunată o valoare numerică, reprezentând viteza cu care se va mișca fundalul.

```
# Fundal
fundal = pygame.image.load("Fundal_Spatiu.png").convert()
y = 0
```

```
    # Ecranul principal
    screen.fill((0, 0, 50))
    # Fundal in pagina - miscare poza
    y_rel = y % fundal.get_rect().height
    screen.blit(fundal, (0, y_rel - fundal.get_rect().height))
    if y_rel < 600:
        screen.blit(fundal, (0, y_rel))
    y += 3  # Viteza cu care se misca imaginea din fundal
```

3.3.2. Titlu si logo

Deoarece am dorit să personalizez cât mai mult aplicația, am adăugat un logo și titlu, specifice temei jocului. Aceste instrucțiuni sunt scrise înaintea funcției *while*.

```
# Titlul si logoul paginii
pygame.display.set_caption("Chicken Invaders")
logo = pygame.image.load("logo_31px.png")
pygame.display.set_icon(logo)
```

3.3.3. Muzică fundal

Prin adăugarea la început a modulului *mixer*, am introdus jocului unul dintre elementele sale cheie, muzica. Sunt necesare două funcții predefinite pentru realizarea acestui lucru *mixer.music.load("Muzica_Fundal.wav")* și *mixer.music.play(-1)*. Prima dintre ele are rolul de a selecta melodia, a doua de a "spune" programului de câte ori vrem ca melodia să cânte. În cazul de față *-1* sugerează faptul că melodia aleasă va merge, în buclă, până la oprirea jocului, sau până când este oprită de o altă funcție.

```
# Muzica de fundal
mixer.music.load("Muzica_Fundal.wav")
mixer.music.play(-1)
```

3.4. Elemente tip text

Pentru a face o experiență de joc cât mai plăcută am adăugat pe ecranul de joc al aplicației câteva elemente text. Acestea sunt cele care arată scorul curent al jucătorului, cât și cele care prezintă butoanele care trebuie apăsate pentru a opri, respectiv pentru e reporni, muzica de fundal.

3.4.1. Scor

Fiind un joc în care scopul principal este de a obține un scor cât mai mare, este strict necesar ca afișarea acestuia să se întâmple în mod continuu, pe ecran. Inițial este nevoie de patru variabile globale: *val_scor* (care reține valoarea curentă a scorului; este inițializată cu 0), *font* (precizează tipul de font pe care vrem să îl folosim, cât și

caracteristicile acestuia), iar ultimele două, *textX* și *textY*, vor memora coordonatele pe axele X și Y ale colțului stânga sus a textului.

Urmează crearea funcției *afis_scor()*, cu două variabile, x și y, ce are ca scop afișarea pe ecran a textului. Aceasta conține exact textul pe care dorim să îl afișăm și folosind x și y va preciza programului unde va fi, mai exact, afișarea lui. Funcția este mai apoi introdusă în cadrul funcției principale *while*, având ca parametri *textX* și *textY*.

```
# Scor
val_scor = 0
font = pygame.font.Font('segoeui.ttf', 25)
textX = 10
textY = 6


def afis_scor(x, y):
    scor = font.render("Scor: " + str(val_scor), True, (255, 255, 255))
    screen.blit(scor, (x, y))
```

```
    afis_scor(textX, textY)
```

3.4.2. Butoane sonor

Pe lângă scorul care va fi afișat încontinuu, pe ecran se mai află și un text ce prezintă cele două butoane care au rolul de a porni și opri muzica de fundal, *P* și *M*. Sunt necesare patru variabile globale: *font_sunet* (denumește tipul de font folosit și caracteristicile lui), *text_volum_x* (reprezintă distanța pe axa de coordonate X la care se află plasat textul) și două variabile *text_volum_p_Y* și *text_volum_m_Y* ce memorează coordonatele pe care le au cele două linii de text pe axa Y. Funcția *afis_sunet()*, ulterior creată, este folosită pentru a preciza textul care se afișează, cât și poziția lui, folosind trei parametri: x, y și z. Aceasta este introdusă în funcția principală *while*, iar parametrii ei voi prelua valorile variabilelor *text_volum_x*, *text_volum_p_Y* și *text_volum_m_Y*, în această ordine.

```
# Text taste volum
font_sunet = pygame.font.Font('segoeui.ttf', 12)
text_volum_X = 590
text_volum_p_Y = 25
text_volum_m_Y = 10
```

```
def afis_sunet(x, y, z):
    tasta_p = font_sunet.render("P - activeaza muzica fundal", True, (255, 255, 255))
    screen.blit(tasta_p, (x, y))
    tasta_m = font_sunet.render("M - dezactiveaza muzica fundal", True, (255, 255, 255))
    screen.blit(tasta_m, (x, z))
```

```
            if event.key == pygame.K_m:
                mixer.music.pause()
            if event.key == pygame.K_p:
                mixer.music.unpause()
```

```
    afis_sunet(text_volum_X, text_volum_p_Y, text_volum_m_Y)
```

3.5. Erou (naveta spațială)

La pornirea aplicației, jucătorul pornește având plasată naveta spațială în partea de jos a ecranului, pe mijloc. De acolo, el are posibilitatea să o mute în stânga și dreapta cu ajutorul săgeților de pe tastatură. Sunt folosite patru variabile globale și o funcție denumită *jucator()*, cu doi parametri, x și y, ce fixează în timp real poziția navei pe ecran. Variabilele folosite sunt: *AvatarJucator* (reprezintă imaginea eroului, în cazul de față este o navetă spațială), *jucatorX* și *jucatorY* (memorează valorile la care se află plasată naveta pe axele de coordonate X și Y) și *jucatorX_miscare* (este valoarea care precizează viteza de mișcare a navetei; inițial are valoarea 0, naveta stând nemișcată).

În cadrul funcției *while* precizăm ca la apăsarea săgeții stânga, respectiv dreapta, variabila *jucatorX_miscare* capătă o anumită valoare. Valoarea respectivă este adaugată valorii variabilei *jucatorX*, creându-se astfel efectul de mișcare al navetei. Pentru a ne asigura că nava rămâne pe ecran, vom impune o limită de poziție în stânga și în dreapta, pe axa de coordonate X. Funcția *jucator()* este adaugată la final și folosește ca paremetri valoarea lui *jucatorX* și *jucatorY*. Este folosită pentru a se asigura că poziția navei rămâne mereu actualizată.

```
# Initializarea Eroului
AvatarJucator = pygame.image.load("Naveta_64px.png")
jucatorX = 370
jucatorY = 480
jucatorX_miscare = 0


def jucator(x, y):
    screen.blit(AvatarJucator, (x, y))
```

```
            if event.key == pygame.K_LEFT:
                # Viteza de miscare a navetei la stanga
                jucatorX_miscare = -2
            if event.key == pygame.K_RIGHT:
                # Viteza de miscare a navetei la drepta
                jucatorX_miscare = 2
```

```
    # Asigurare ca naveta nu iese din pagina
    if jucatorX <= 0:
        jucatorX = 0
    elif jucatorX >= 736:
        jucatorX = 736

    jucatorX += jucatorX_miscare
```

```
    jucator(jucatorX, jucatorY)
```

3.6. Inamici (găinile)

Inițial, în joc sunt prezenți un număr de cinci inamici care se mișcă aleatoriu pe ecran, înaintând spre navă. Inițializarea lor se face folosind liste. Pentru început vom declara cinci liste goale, cărora le vor fi atribute câte cinci valori asociate câte unui inamic. Astfel, vom avea: *AvatarInamic* – listă ce va memora imaginea fiecărui inamic; *inamicX* – listă ce memorează valorea pe axa de coordonare X a fiecăruia dintre inamici; *inamicY* – listă ce memorează valorea pe axa de coordonare Y a fiecăruia dintre inamici; *inamicX_miscare* – viteza cu care se mișcă fiecare inamic pe axa de coordonate X; *inamicY_miscare* – valoarea cu care sare fiecare inamic pe axa de coordonate Y.

Pe lângă acestea mai este declarată o variabilă globală, *numar_inamici*, care reprezintă numarul curent de inamici, implicit numărul de elemente din fiecare listă.

Vom mai avea nevoie de o funcție numită *inamic()*, ce are ca parametri trei variabile: x, y și i, unde i reprezintă al câtelea element din listă este un inamic. Funcția are rolul de a afișa pe ecran inamicul respectiv, într-o zonă anume, desemnată de x și y.

În cadrul funcției *while* vom folosi o instrucțiune de tip *for* ce va modifica poziția pe axa X a fiecăruia dintre inamici (*inamicX[i]*), adăugând valoarea specifică din lista *inamicX_miscare*, creându-se astfel efectul de mișcare al inamicilor. În același timp, trebuie să avem în vedere și pozițiile fiecăruia. Dacă unul dintre ei atinge marginea din stânga, acesta își schimbă direcția, mișcându-se spre dreapta, iar dacă atinge marginea din dreapta își schimbă direcția, mutâdu-se spre stânga. Vitezele lor rămân constante în tot acest timp.

Totodată, atunci când o margine este atinsă, inamicul respectiv coboară cu o anumită valoare pe axa de coordonate Y, așa cum este specificat în lista *inamicY_miscare*. La finalul instrucțiunii *for* este apelată funcția *inamic()*, actualizându-se astfel pozițiile pe ecran a fiecărui inamic în parte.

```
# Initializarea Inamicului
AvatarInamic = []
inamicX = []
inamicY = []
inamicX_miscare = []
inamicY_miscare = []
numar_inamici = 5

for i in range(numar_inamici):
    AvatarInamic.append(pygame.image.load('Gaina_62X51px.png'))
    inamicX.append(random.randint(0, 736))
    inamicY.append(random.randint(50, 150))
    inamicX_miscare.append(1.2)
    inamicY_miscare.append(30)


def inamic(x, y, i):
    screen.blit(AvatarInamic[i], (x, y))
```

```
    # Miscarea inamicului
    for i in range(numar_inamici):
        # Ecuatia prin care pozitia pe axa x a inamicului creste sau scade
        inamicX[i] += inamicX_miscare[i]
        # Daca coordonata X a inamicului i este egala sau mai mica decat 0,
        # acesta va incepe sa se miste spre dreapta si in acelasi
        # timp se va muta in jos pe axa Y
        if inamicX[i] <= 0:
            inamicX_miscare[i] = 1.2
            inamicY[i] += inamicY_miscare[i]
        # Daca coordonata X a inamicului i este egala sau mai mare decat 736,
        # acesta va incepe sa se miste spre stanga si in acelasi
        # timp se va muta in jos pe axa Y
        elif inamicX[i] >= 736:
            inamicX_miscare[i] = -1.2
            inamicY[i] += inamicY_miscare[i]
```

```
        inamic(inamicX[i], inamicY[i], i)
```

3.7. Proiectil (raza laser)

Pentru a elimina un inamic este nevoie ca acesta să fie lovit de un singur proiectil. Fiecare rază laser este trasă prin apăsarea tastei *Space* și este însoțită de un sunet specific în momentul în care tragem. Atunci când ajunge în partea de sus a ecranului sau atunci când lovește una dintre găini, raza revine la locul ei inițial. Glonțul este inițializat cu ajutorul a cinci variabile declarate global: *AvatarGlont* (reprezintă imaginea proiectilului), *glontX* și *glontY* (sunt valorile coordonatelor glonțului pe axa X, respectiv Y), *glontY_miscare* (valoarea acestei variabile determină viteza cu care se mișcă raza laser) și *glont_stare* (variabilă ce poate memora doar două valori, *„pregatit”* și *„tras”*, de care depinde starea glonțului).

Pentru a menționa dacă glonțul este tras sau nu, vom crea funcția *trage_glont()*. Aceasta are doi parametri: x și y. De fiecare dată când este apelată în cadrul instrucțiunii principale *while*, setează variabila *glont_stare* ca având valoarea *„tras”* și va afișa pe ecran imaginea razei laser la coordonatele specificate de către cei doi parametri, care au valorile lui *glontX* și *glontY*. La prima apelare a funcției, *glontX* ia valorea variabilei *jucatorX*, prezentată mai sus, deoarece dorim ca raza laser să pornească din dreptul navei spațiale. La următoarele apeluri, atâta vreme cât *glont_stare* are valoarea *“tras”*, funcția va avea același parametru *glontX*, dar *glontY* se va micșora de data aceasta cu valoarea variabilei *glontY_miscare*, fapt ce determină mișcarea pe o traiectorie fixă a razei laser. Tot în instrucțiunea *while* vom introduce și secvența care verifică dacă valoarea pe axa Y a glonțului este mai mică sau egală cu 0 (*glontY <= 0*). În caz afirmativ, proiectilul revine în starea lui inițială, iar *glont_stare* preia valorea *“pregatit”*, așteptând apăsarea tastei *Space*.

```
# Initializare Glont
AvatarGlont = pygame.image.load('glont_31px.png')
glontX = 0
glontY = 480
glontY_miscare = 10  # Viteza de miscare a glontului
glont_stare = "pregatit"


def trage_glont(x, y):
    global glont_stare
    glont_stare = "tras"
    screen.blit(AvatarGlont, (x + 17, y + 10))
```

```
            if event.key == pygame.K_SPACE:
                if glont_stare == "pregatit":
                    # Adauga sunet cand tragi
                    sunet_glont = mixer.Sound("Sunet_Laser.wav")
                    sunet_glont.play()
                    # Glontul netras preia aceeasi coordonata X ca si nava
                    glontX = jucatorX
                    trage_glont(glontX, glontY)
```

```
    # Miscarea Glontului
    # Daca valoarea pe axa Y a glontului devine mai mica sau egala cu 0,
    # glontul revine la starea sa initiala
    if glontY <= 0:
        glontY = 480
        glont_stare = "pregatit"
    # Daca glontul a fost tras, apasand pe tasta SPACE, acesta va fi
    # activat si in acelasi timp valoarea sa pe axa Y va scade
    if glont_stare == "tras":
        trage_glont(glontX, glontY)
        glontY -= glontY_miscare
```

3.7.1. Lovire inamic

Un inamic este declarat lovit dacă raza laser se apropie la o anumită distanță de el. Calculul unei astfel de distanțe se face folosind o funcție pe care am denumit-o *inamic_lovit()*. Funcția calculează distanța dintre punctul determinat de coordonatele proiectilului și punctul determinat de coordonatale unuia dintre inimici. Dacă distanța este mai mică sau egală decât o valoare dată, în cazul de față 27, funcția returnează valoarea *True*, iar în caz contrar valoarea *False*.

Această funcție este ulterior folosită în interiorul instrucțiunii *while*, unde o variabilă numită *lovire* preia valoarea returnată (*True* – dacă proiectilul a atins ținta; *False* – dacă proiectilul nu a atins ținta). Dacă rezultatul este afirmativ, glonțul revine la starea sa inițială, *glont_stare* schimbându-și valoarea în *„pregatit”* și *glontY* va lua din nou valoarea 480. În același timp, *val_scor* (variabilă ce memorează valoarea scorului, prezentată mai sus, la capitolul 3.4.1.) crește cu 1, în fundal este redat un sunet specific de găină lovită, iar inamicul respectiv va primi noi coordonate de afișare. Dacă *val_scor* este un multiplu de 10, atunci numărul inamicilor va crește cu 1 (*numar_inamici*), iar celor cinci liste care memorează datele despre ceilalți inamici le vor fi adăugate câte un nou element. Toate aceste instrucțiuni se vor afla sub instrucțiunea *for* ce este specifică

mișcării inamicilor, instrucțiune prezentată mai sus, la capitolul 3.6.. Acest lucru este datorat faptului că verificăm dacă a fost atins de proiectil fiecare inamic în parte.

```python
# Calculeaza daca glontul ajunge la o anumita distanta fata de inamic
def inamic_lovit(inamicX, inamicY, glontX, glontY):
    distanta = math.sqrt(math.pow(glontY - inamicY, 2) + math.pow(glontX - inamicX, 2))
    if distanta <= 27:
        return True
    else:
        return False
```

```python
        # Inamic Lovit
        lovire = inamic_lovit(inamicX[i], inamicY[i], glontX, glontY)
        if lovire:
            sunet_inamic_mort = mixer.Sound('Sunet_Gaina_Moarta.wav')
            sunet_inamic_mort.play()
            glontY = 480
            glont_stare = "pregatit"
            val_scor += 1
            # Adaugare inamic in plus la fiecare 10 inamici morti
            if val_scor % 10 == 0:
                numar_inamici += 1
                AvatarInamic.append(pygame.image.load('Gaina_62X51px.png'))
                inamicX.append(random.randint(0, 736))
                inamicY.append(random.randint(50, 150))
                inamicX_miscare.append(1.2)
                inamicY_miscare.append(30)
            inamicX[i] = random.randint(0, 736)
            inamicY[i] = random.randint(50, 150)
```

3.8. Sfârșit joc (Game Over)

Ca la aproape orice alt joc, în momentul în care pierzi, pe ecran apare textul „Game Over”, jocul luând sfârșit. Textul este însoțit de afișarea scorului final. Din acest punct nu se mai poate face nimic, singura opțiune fiind să ieși din aplicație. Pentru ca toate aceste lucruri să aibă loc, înainte de instrucțiunea *while* am declarat trei variabile globale și doua funcții, denumite *game_over_text()* și *sunet_final()*. Variabilele sunt: *font_game_over*, *font_scor_final* (acestea două conțin caracteristicile tipului de font folosit pentru afișarea mesajelor „Game Over” și a scorului final), cât și *sunet_rep*, ce este inițializată având valoarea 1 (variabila va avea ca scop verificarea dacă sunetul de la final este emis o singură dată). La rândul lor, funcțiile au ca scop, ca în momentul în care sunt apelate, să afișeze pe ecran cele două texte, în anumite poziții, acompaniate de un sunet specific de final.

În instrucțiunea *while*, secvența de cod, necesară introducerii evenimentelor ce se petrec la finalul jocului, este introdusă sub instrucțiunea *for* ce verifică informațiile pentru fiecare inamic în parte (instrucțiune prezentată mai sus, la capitolul 3.6.). Este necesar acest lucru deoarece este nevoie ca un singur inamic, din toti cei prezenți pe ecran, să corespundă cerințelor menționate, astfel, jocul terminându-se.

În cadrul secvenței este verificată poziția curentă pe axa Y a fiecăruia dintre inamici (*inamicY[i]*). Dacă aceasta depășește o valoare dată, coordonatele tuturor inamicilor sunt schimbate în așa fel încât să nu mai apară pe ecran, la fel procedându-se și cu coordonatele textelor ce apăreau până în momentul actual (adică scorul curent și cele două butoane pentru modificarea sunetului, explicate la capitolul 3.4.). În același timp sunt apelate cele două funcții mai sus amintite, legate de afișarea mesajului „Game Over” și a redării sunetului de final. Totodată, este oprită muzica de fundal, iar variabilei globale *y*, ce memora viteza cu care se mișca imaginea de fundal, îi este adăugată o nouă valoare, care va da impresia că fundalul se mișcă și mai repede. Instrucțiunea *break* se asigură că funcția *for*, în care se află încadrată secvența, nu mai este rulată, evitându-se astfel afișarea inamicilor din nou pe ecran.

```
# Game Over
font_game_over = pygame.font.Font('segoeuib.ttf', 70)
font_scor_final = pygame.font.Font('seguisb.ttf', 22)


def game_over_text():
    game_over = font_game_over.render("GAME OVER", True, (255, 255, 255))
    scor_final = font_scor_final.render("SCOR FINAL: " + str(val_scor), True, (255, 255, 255))
    screen.blit(game_over, (200, 200))
    screen.blit(scor_final, (325, 285))


sunet_rep = 1  # Variabila ce verifica daca sunetul de final a fost pornit o singura data


def sunet_final():
    sunet_game_over = mixer.Sound("Sunet_Game_Over.wav")
    sunet_game_over.play()
```

```
        # Afisarea pe ecran a textului "Game Over" (Joc terminat)
        if inamicY[i] > 420:  # Distanta pe axa Y, la care, daca ajunge inamicul, jocul se termina
            for j in range(numar_inamici):
                inamicY[j] = 2000
            game_over_text()
            mixer.music.pause()
            if sunet_rep == 1:
                sunet_final()
            sunet_rep += 1
            text_volum_X = 2000
            textY = 2000
            textX = 2000
            y +=5
            break
```

3.9. Evenimente

Deoarece folosim librăria pygame, aceasta ne permite sa atribuim fiecărei atingeri de taste sau click câte o anumită funcție. Astfel, dacă este apăsat butonul de *Ieșire* din fereastra aplicației, variabiala *ruleaza*, declarată global cu valoarea *True*, va prelua valoarea *False*, însemnând oprirea instrucțiunii principale *while*, adică închiderea ferestrei jocului.

La apăsarea tastei *Săgeată stânga*, naveta va merge în stânga, iar la apăsarea tastei *Săgeată dreapta*, va merge în dreapta. De asemenea, tasta *Space* activează raza laser, tasta *M* oprește muzica de fundal, iar tasta *P* repornește muzica de fundal.

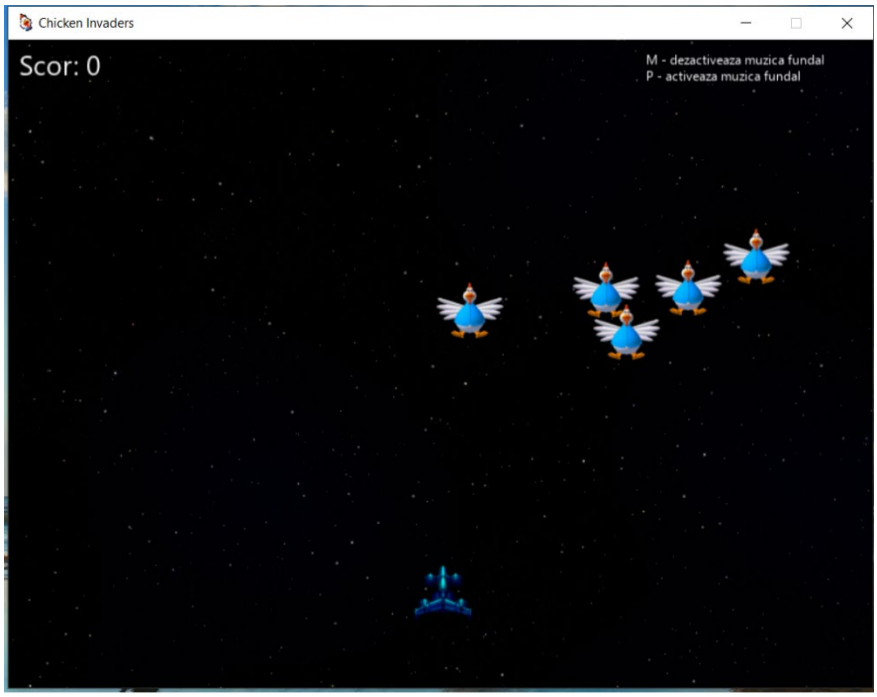
Pentru a ne asigura că jucatorul rămâne cu naveta în poziție fixă atunci când nu are niciuna dintre săgeți apăsate, folosim o instrucțiune care specifică ca atunci când una dintre cele două taste nu este apăsată, viteza navei scade la valoarea 0 (*jucatorX_miscare = 0*).

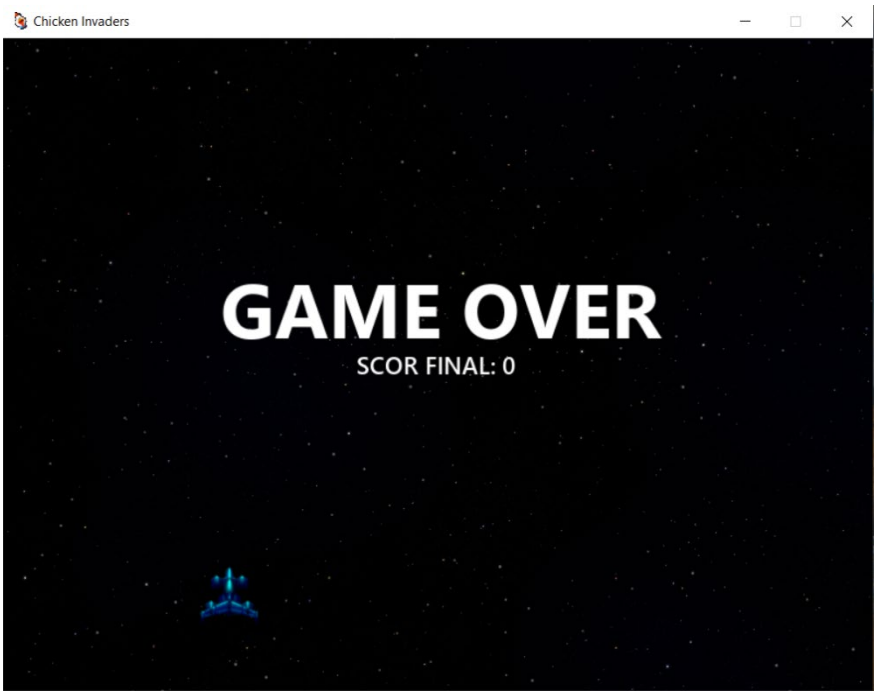
Secvențele de instrucțiuni sunt plasate sub instrucțiunea *while*.

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

        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                # Viteza de miscare a navetei la stanga
                jucatorX_miscare = -2
            if event.key == pygame.K_RIGHT:
                # Viteza de miscare a navetei la drepta
                jucatorX_miscare = 2
            if event.key == pygame.K_m:
                mixer.music.pause()
            if event.key == pygame.K_p:
                mixer.music.unpause()
            if event.key == pygame.K_SPACE:
                if glont_stare == "pregatit":
                    # Adauga sunet cand tragi
                    sunet_glont = mixer.Sound("Sunet_Laser.wav")
                    sunet_glont.play()
                    # Glontul netras preia aceeasi coordonata X ca si nava
                    glontX = jucatorX
                    trage_glont(glontX, glontY)

        if event.type == pygame.KEYUP:
            if event.key == pygame.K_LEFT or event.key == pygame.K_RIGHT:
                jucatorX_miscare = 0
```

3.10. Aplicație finală

După finalizarea scrierii și a optimizării codului, aplicația finală va putea fi afișată pe ecran în momentul rulării. Programul poate fi pornit fie prin rularea efectivă a întregului cod în PyCharm (sau în orice alt compilator care suportă Python), fie prin deschiderea aplicației *main.exe* ce se află în folderul *Executabil Aplicatie.*





3.11. Note

Este prima aplicație de acest gen pe care am făcut-o. Din această cauză, am remarcat două tipuri de erori ce apar în momentul în care este rulată. Primul tip este legat de mișcarea navei, stânga-dreapta. Dacă sunt ținute ambele taste apăsate (*Săgeată stânga*, *Săgeată dreapta*) și dăm drumul doar la una dintre ele, nava va rămâne fixă, în loc să se miște spre direcția săgeții încă apăsate. Celălalt tip de eroare este legat de viteza de afișare a jocului. Nu am realizat că nefolosirea unor variabile relative poate crea decalaje de redare. Jocul a fost scris și rulat pe un ecran de 1366x768 pixeli, la o rată de refresh de 60 Hz. Dacă aplicația este deschisă pe un sistem cu alte specificații este posibil ca elementele mobile să se miște mult prea repede decât au fost intenționate.

La următoarele aplicații pe care le voi mai crea voi ține cont de aceste mici detalii. Totuși, acesta a fost și scopul acestui proiect, să învăț să fac lucruri noi, folosind informațiile pe care le aveam deja. Prin urmare, întregul proces de creare a reprezentat o provocare nouă pentru mine și tocmai ducerea lui până la capăt reprezintă un succes personal.

4. Bibliografie

1. https://www.interactionstudios.com/chickeninvaders.php - capitolul 2
2. https://ro.wikipedia.org/wiki/Chicken_Invaders - capitolul 2
3. https://en.wikipedia.org/wiki/Chicken_Invaders - capitolul 2
4. https://en.wikipedia.org/wiki/Python_(programming_language) - capitolul 1
5. https://en.wikipedia.org/wiki/Pygame - capitolul 1