Міністерство освіти і науки України

Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Програмування інтелектуальних інформаційних систем

**ЗВІТ**

до лабораторних робіт

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Виконав**  **студент** |  | Бакун Нікіта Валерійович |  |  |
|  |  | (№ групи, прізвище, ім’я, по батькові ) |  |  |
|  |  |  |  |  |
| **Прийняв** |  | ас. Очеретяний О. К. |  |  |
|  |  | (посада, прізвище, ім’я, по батькові ) |  |  |

Київ 2021

# **Завдання лабораторної роботи**

Завданням даної лаболаторної роботи було створення гри Space Inviders з Ви маєте додати життя вашим ворогам, тому:

При генерації ви вказуєте кількість згенерованих ворогів одного з двох типів. Перший тип шукає дорогу до гравця (оптимальність цього пошуку на вашій совісті). Другий тип рухається випадково

Ви маєте реалізувати алгоритми перемоги за вашою грою - minimax з alpha-beta pruning та expectimax.

Ви маєте зібрати статистику по іграм. Запишіть в csv файл наступні данні по грі: перемога чи поразка, час, кількість очок, алгоритм

# **Опис використаних технологій**

Алгоритм Minimax основан на систематическом поиске, или, точнее говоря, на грубой силе и простой функции оценки. Предположим, что каждый раз, когда мы решаем следующий ход, мы просматриваем все дерево, вплоть до листьев. По сути, мы изучали все возможные результаты и каждый раз могли определить лучший из возможных ходов.

Однако для нетривиальных игр такая практика неприменима. Даже поиск на определенной глубине иногда занимает неприемлемое количество времени. Таким образом, Minimax применяет поиск к дереву довольно низкой глубины с соответствующей эвристикой и хорошо продуманной, но простой оценочной функцией.

При таком подходе мы теряем уверенность в нахождении наилучшего возможного хода, но в большинстве случаев решение, которое принимает минимакс, намного лучше, чем решение любого человека.

Теперь давайте более подробно рассмотрим функцию оценки, о которой мы упоминали ранее. Чтобы определить хороший (не обязательно лучший) ход для определенного игрока, мы должны каким-то образом оценить узлы (позиции), чтобы иметь возможность сравнивать один с другим по качеству.

Функция оценки - это статическое число, которое в соответствии с характеристиками самой игры присваивается каждому узлу (позиции).

Важно отметить, что функция оценки не должна полагаться ни на поиск предыдущих узлов, ни на следующее. Он должен просто анализировать состояние игры и обстоятельства, в которых находятся оба игрока.

Необходимо, чтобы функция оценки содержала как можно больше релевантной информации, но, с другой стороны, поскольку она вычисляется много раз, она должна быть простой.

Обычно он отображает набор всех возможных позиций в симметричный сегмент:

F:п→[-M,M]

Значение M присваивается только листьям, где победителем является первый игрок, а значение -M - листьям, где победителем является второй игрок.

В играх с нулевой суммой значение оценочной функции имеет противоположное значение - что лучше для первого игрока, хуже для второго, и наоборот. Следовательно, значение для симметричных позиций (если игроки меняются ролями) должно отличаться только знаком.

Распространенной практикой является изменение оценок листьев путем вычитания глубины этого точного листа, чтобы из всех ходов, ведущих к победе, алгоритм мог выбрать тот, который сделает это за наименьшее количество шагов (или выберет ход, который откладывает убыток, если он неизбежен).

# **Опис програмного коду**

В головному файлі проекту main.py визиваються по натиску клавіші алгоритми та відбувається зміна алгоритмів

import GlobalVariables as gv  
  
  
def main():  
 gv.CAN\_EDIT = True  
 gv.PG\_LIB.display.set\_caption(**"Space Invaders"**) *# Назва застосунку* run = True  
 clock = gv.PG\_LIB.time.Clock()  
 while run:  
 clock.tick(gv.FPS) *#Виставлено кількість оновлень у секунду* gv.FrameCreator\_LIB.updateFrame()  
 if gv.LIVES <= 0 or gv.GOOD\_SHIP.health <= 0: *# умови програшу* gv.LOST = True  
 gv.LOST\_COUNT += 1  
  
 if gv.LOST: *# час показання екрану програшу* if gv.LOST\_COUNT > gv.FPS \* 3:  
 run = False  
  
 else:  
 continue  
 if len(gv.ENEMIES) == 0: *# збільшення рівня складності* gv.LEVEL += 1  
 gv.WAVE\_LENGTH += 5  
 for i in range(gv.WAVE\_LENGTH): *# створення масиву з ворогів* enemy = gv.ShipCreator.Enemy(gv.RANDOM\_LIB.randrange(50, gv.WIDTH - 100),  
 gv.RANDOM\_LIB.randrange(-1500, -100),  
 gv.RANDOM\_LIB.choice([**"red"**, **"blue"**, **"purple"**])) *# створення місця та колір ворога* gv.ENEMIES.append(enemy)  
 for event in gv.PG\_LIB.event.get():  
 if event.type == gv.PG\_LIB.QUIT:  
 run = False  
  
 keys = gv.PG\_LIB.key.get\_pressed() *# рух корабля по натиску на клавішу та стрільба* if keys[gv.PG\_LIB.K\_a] and gv.GOOD\_SHIP.x - gv.PLAYER\_VEL > 0: *# left* gv.GOOD\_SHIP.x -= gv.PLAYER\_VEL  
 if keys[gv.PG\_LIB.K\_d] and gv.GOOD\_SHIP.x + gv.PLAYER\_VEL + gv.GOOD\_SHIP\_SIZEX < gv.WIDTH: *# right* gv.GOOD\_SHIP.x += gv.PLAYER\_VEL  
 if keys[gv.PG\_LIB.K\_w] and gv.GOOD\_SHIP.y - gv.PLAYER\_VEL > 0: *# up* gv.GOOD\_SHIP.y -= gv.PLAYER\_VEL  
 if keys[gv.PG\_LIB.K\_s] and gv.GOOD\_SHIP.y + gv.PLAYER\_VEL + gv.GOOD\_SHIP\_SIZEY < gv.HEIGHT: *# down* gv.GOOD\_SHIP.y += gv.PLAYER\_VEL  
 if keys[gv.PG\_LIB.K\_SPACE]:  
 gv.GOOD\_SHIP.shoot()  
 for enemy in gv.ENEMIES[:]:  
 enemy.move(gv.ENEMY\_VEL)  
 enemy.move\_lasers(gv.LASER\_VEL, gv.GOOD\_SHIP)  
  
 if gv.RANDOM\_LIB.randrange(0, 20) == 1: *# рандомізація пострілу ворога* enemy.shoot()  
  
 if gv.LaserCreator.collide(enemy, gv.GOOD\_SHIP): *# дотик до ворога* gv.GOOD\_SHIP.health -= 10  
 gv.ENEMIES.remove(enemy)  
 gv.SCORE -= 20  
  
 if enemy.y + gv.BAD\_SHIP\_SIZEY + 10 > gv.HEIGHT: *# проходження ворогу до низу екрану* gv.LIVES -= 1  
 gv.ENEMIES.remove(enemy)  
 gv.SCORE -= 100  
  
 gv.GOOD\_SHIP.move\_lasers(-gv.LASER\_VEL, gv.ENEMIES)  
  
 gv.FrameCreator\_LIB.updateFrame()  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 title\_font = gv.PG\_LIB.font.SysFont(**"comicsans"**, 70)  
 run = True  
 main\_menu = True  
 while run:  
 gv.WINDOW.blit(gv.BACKGROUND\_PNG, (0, 0)) *#створення графіки для гри* title\_label = title\_font.render(**"Press any button to begin..."**, 1, (255, 255, 255))  
  
 gv.WINDOW.blit(title\_label, (gv.WIDTH / 2 - title\_label.get\_width() / 2,350))  
 if main\_menu:  
 main\_menu = False  
  
 gv.PG\_LIB.display.update()  
 for event in gv.PG\_LIB.event.get():  
 if event.type == gv.PG\_LIB.QUIT:  
 run = False  
 if event.type == gv.PG\_LIB.KEYDOWN or event.type == gv.PG\_LIB.MOUSEBUTTONDOWN:  
 main()  
 gv.PG\_LIB.quit()

У файлі LaserCreator.py реалізовано клас лазер та показано методи та поля даного класу. Змім відносно попередної лр не відбулося

import GlobalVariables  
  
  
class Laser:  
 def \_\_init\_\_(self, x, y, img):  
 self.x = x  
 self.y = y  
 self.img = img  
 self.mask = GlobalVariables.PG\_LIB.mask.from\_surface(self.img)  
  
 def draw(self, window): *# створення лазеру* window.blit(self.img, (self.x, self.y))  
  
 def move(self, vel): *# рух лазеру* self.y += vel  
  
 def off\_screen(self, height): *# вихід лазеру за рамки* return not (self.y <= height and self.y >= 0)  
  
 def collision(self, obj): *# дотикання до лазеру* return collide(self, obj)  
  
  
def collide(obj1, obj2): *# чи дотикнулись об'єкти* offset\_x = obj2.x - obj1.x  
 offset\_y = obj2.y - obj1.y  
 return obj1.mask.overlap(obj2.mask, (offset\_x, offset\_y)) is not None

У файлі GlobalVariables.py розміщені усі глобальні змінні. Вони винесені в окремий файл для зручного використання та доступу. В подальшому таке розміщення повинно спростити роботу з проектом. У даній лабораторній роботі додано деякі змінні

import pygame  
import os  
import time  
import random  
import FrameCreator  
import ShipCreator as sc  
import LaserCreator  
*# created global variables of libraries*OS\_LIB = os  
PG\_LIB = pygame  
TIME\_LIB = time  
RANDOM\_LIB = random  
FrameCreator\_LIB = FrameCreator  
ShipCreator = sc  
LaserCreator = LaserCreator  
*# Created global variables of Integer values*LOST = False  
FPS = 60  
LEVEL = 1  
LIVES = 5  
SCORE = 0  
WIDTH, HEIGHT = 750, 750  
PLAYER\_VEL = 5  
LASER\_VEL = 4  
ENEMIES = []  
WAVE\_LENGTH = 5  
ENEMY\_VEL = 1  
LOST\_COUNT = 0  
GOOD\_SHIP\_SIZEX, GOOD\_SHIP\_SIZEY = 50, 50  
BAD\_SHIP\_SIZEX, BAD\_SHIP\_SIZEY = 40, 40  
LASER\_SIZEX = 10  
LASER\_SIZEY = 20  
*# Created global variables of Font*PG\_LIB.font.init()  
MAIN\_FONT = PG\_LIB.font.SysFont(**"comicsans"**, 50)  
LOST\_FONT = PG\_LIB.font.SysFont(**"comicsans"**, 60)  
  
WINDOW = PG\_LIB.display.set\_mode((WIDTH, HEIGHT))  
  
*# Created global variables of boats images*GOOD\_SHIP\_PNG = PG\_LIB.transform.scale(PG\_LIB.image.load(os.path.join(**"assets"**, **"good\_ship.png"**)), (GOOD\_SHIP\_SIZEX, GOOD\_SHIP\_SIZEY))  
  
BAD\_SHIP\_RED\_PNG = PG\_LIB.transform.scale(PG\_LIB.image.load(os.path.join(**"assets"**, **"bad\_ship\_red.png"**)), (BAD\_SHIP\_SIZEX, BAD\_SHIP\_SIZEX))  
BAD\_SHIP\_BLUE\_PNG = PG\_LIB.transform.scale(PG\_LIB.image.load(os.path.join(**"assets"**, **"bad\_ship\_blue.png"**)), (BAD\_SHIP\_SIZEX, BAD\_SHIP\_SIZEX))  
BAD\_SHIP\_PURPLE\_PNG = PG\_LIB.transform.scale(PG\_LIB.image.load(os.path.join(**"assets"**, **"bad\_ship\_purple.png"**)), (BAD\_SHIP\_SIZEX, BAD\_SHIP\_SIZEX))  
  
BAD\_BULLET\_PNG = PG\_LIB.transform.scale(PG\_LIB.image.load(os.path.join(**"assets"**, **"bad\_bullet.png"**)), (LASER\_SIZEX, LASER\_SIZEY))  
GOOD\_BULLET\_PNG = PG\_LIB.transform.scale(PG\_LIB.image.load(os.path.join(**"assets"**, **"good\_bullet.png"**)), (LASER\_SIZEX, LASER\_SIZEY))  
  
*# Created global variables of background images*BACKGROUND\_PNG = PG\_LIB.transform.scale(PG\_LIB.image.load(os.path.join(**"assets"**, **"background.png"**)), (WIDTH, HEIGHT))  
  
*# Created ships*GOOD\_SHIP = sc.Player(300, 650)  
  
COLOR\_MAP = {  
 **"red"**: BAD\_SHIP\_RED\_PNG,  
 **"blue"**: BAD\_SHIP\_BLUE\_PNG,  
 **"purple"**: BAD\_SHIP\_PURPLE\_PNG  
}

У файлі ShipCrearor.py реалізовано клас астероїдів.

import GlobalVariables as gv  
  
  
class Ship:  
 COOLDOWN = 30 *# час перезарядки човна* def \_\_init\_\_(self, x, y, health=100):  
 self.x = x  
 self.y = y  
 self.health = health  
 self.ship\_img = None  
 self.laser\_img = None  
 self.lasers = []  
 self.cool\_down\_counter = 0  
  
 def draw(self, window): *# створення човна* gv.WINDOW.blit(self.ship\_img, (self.x, self.y))  
 for laser in self.lasers:  
 laser.draw(gv.WINDOW)  
  
 def move\_lasers(self, vel, obj): *# рух пострілів* self.coldown()  
 for laser in self.lasers:  
 laser.move(vel)  
 if laser.off\_screen(gv.HEIGHT):  
 self.lasers.remove(laser)  
 elif laser.collision(obj):  
 obj.health -= 10  
 self.lasers.remove(laser)  
  
 def shoot(self): *# постріл* if self.cool\_down\_counter == 0:  
 laser = gv.LaserCreator.Laser(self.x + 20, self.y, self.laser\_img)  
 self.lasers.append(laser)  
 self.cool\_down\_counter = 1  
  
 def coldown(self): *# механіка перезарядки* if self.cool\_down\_counter >= self.COOLDOWN:  
 self.cool\_down\_counter = 0  
 elif self.cool\_down\_counter >= 0:  
 self.cool\_down\_counter += 1  
  
  
class Player(Ship):  
 def \_\_init\_\_(self, x, y, health=100):  
 super().\_\_init\_\_(x, y, health)  
 self.ship\_img = gv.GOOD\_SHIP\_PNG  
 self.laser\_img = gv.GOOD\_BULLET\_PNG  
 self.mask = gv.PG\_LIB.mask.from\_surface(self.ship\_img)  
 self.max\_health = health  
  
 def move\_lasers(self, vel, objs): *#рух лазерів* self.coldown()  
 for laser in self.lasers:  
 laser.move(vel)  
 if laser.off\_screen(gv.HEIGHT):  
 self.lasers.remove(laser)  
 else:  
 for obj in objs:  
 if laser.collision(obj):  
 objs.remove(obj)  
 gv.SCORE += 50  
 if laser in self.lasers:  
 self.lasers.remove(laser)  
  
 def healthbar(self, window): *# створення здоров'я* gv.PG\_LIB.draw.rect(window, (255, 0, 0), (self.x, self.y + gv.GOOD\_SHIP\_SIZEY + 10, gv.GOOD\_SHIP\_SIZEX, 10))  
 gv.PG\_LIB.draw.rect(window, (0, 255, 0), (self.x, self.y + gv.GOOD\_SHIP\_SIZEY + 10, gv.GOOD\_SHIP\_SIZEX \* (  
 1 - ((self.max\_health - self.health) / self.max\_health)), 10))  
  
 def draw(self, window):  
 super().draw(window)  
 self.healthbar(window)  
  
  
class Enemy(Ship):  
 def \_\_init\_\_(self, x, y, color, health=100):  
 super().\_\_init\_\_(x, y, health)  
 self.ship\_img = gv.COLOR\_MAP[color]  
 self.laser\_img = gv.BAD\_BULLET\_PNG  
 self.mask = gv.PG\_LIB.mask.from\_surface(self.ship\_img)  
  
 def move(self, vel):  
 self.y += vel  
  
 def shoot(self):  
 if self.cool\_down\_counter == 0:  
 laser = gv.LaserCreator.Laser(self.x + 10, self.y, self.laser\_img)  
 self.lasers.append(laser)  
 self.cool\_down\_counter = 1

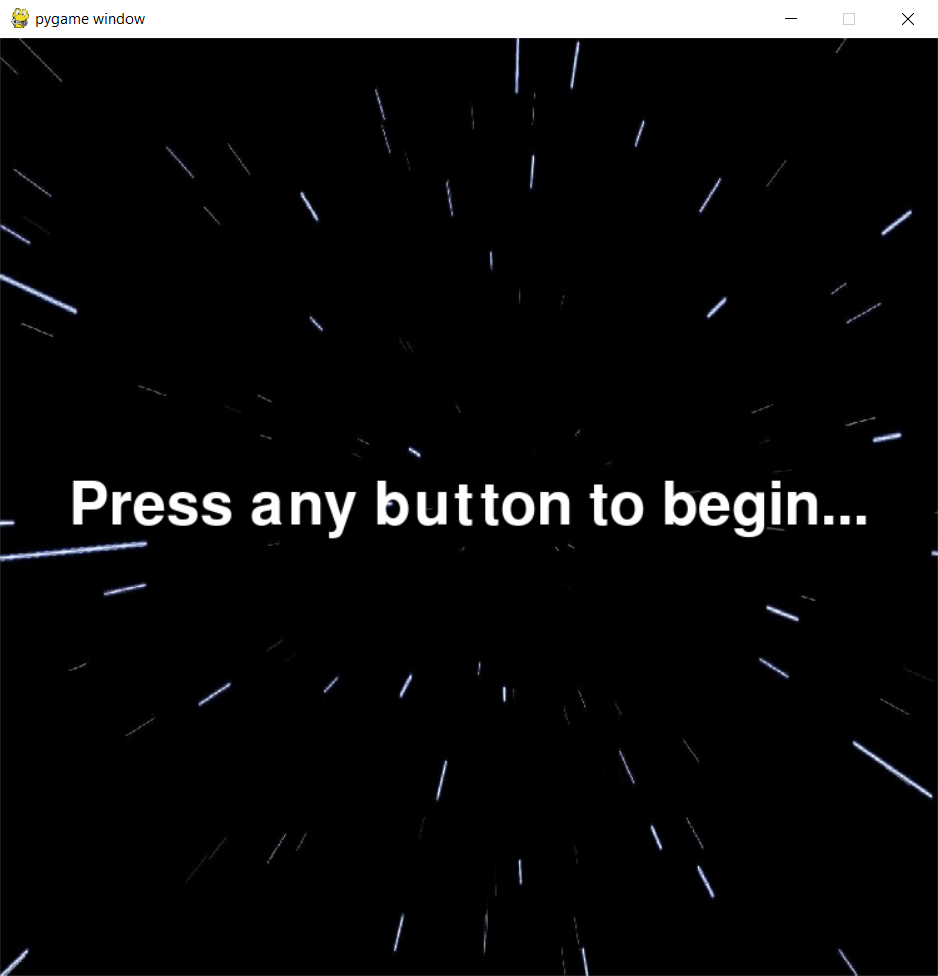
в FrameCreator.py описано процес оновлення фрейму

import GlobalVariables as gv  
import Algorytms  
  
def updateFrame(): *# оновлення фрейму* gv.WINDOW.blit(gv.BACKGROUND\_PNG, (0, 0))  
  
 lives\_label = gv.MAIN\_FONT.render(**f"Lives:** {gv.LIVES}**"**, 1, (255, 255, 255))  
 level\_label = gv.MAIN\_FONT.render(**f"Level:** {gv.LEVEL}**"**, 1, (255, 255, 255))  
 score\_label = gv.MAIN\_FONT.render(**f"Score:** {gv.SCORE}**"**, 1, (255, 255, 255))  
 gv.WINDOW.blit(lives\_label, (10, 10))  
 gv.WINDOW.blit(level\_label, (gv.WIDTH - level\_label.get\_width() - 10, 10))  
 gv.WINDOW.blit(score\_label,(10,40))  
 for enemy in gv.ENEMIES:  
 enemy.draw(gv.WINDOW)  
 for asteroid in gv.ASTEROIDS:  
 asteroid.draw()  
 *# if gv.RANDOM\_LIB.randrange(0, 2000) == 1:  
 # gv.ShipCreator.create\_asteroids()* for pixel in gv.pixelPath:  
 pixel.draw()  
 gv.GOOD\_SHIP.draw(gv.WINDOW)  
  
 if gv.LOST: *# програш* lost\_label = gv.LOST\_FONT.render(**f"YOU LOST!"**, 1, (255, 255, 255))  
 score\_label = gv.LOST\_FONT.render(**f"YOUR SCORE IS:** {gv.SCORE}**"**, 1, (255, 255, 255))  
 gv.WINDOW.blit(lost\_label, (gv.WIDTH / 2 - lost\_label.get\_width() / 2, 250))  
 gv.WINDOW.blit(score\_label, (gv.WIDTH / 2 - score\_label.get\_width() / 2, 350))  
  
 Algorytms.enemyArray = []  
 Algorytms.arrayOfPath = []  
 gv.pixelPath = []  
 Algorytms.createVisitMatrix(Algorytms.matrix)  
 Algorytms.fillMatrix(Algorytms.matrix)  
 *# for i in Algorytms.matrix:  
 # print(\*i)* while Algorytms.enemyArray:  
 Algorytms.emptyMatrix(Algorytms.lenMatrix, Algorytms.startPoint)  
 Algorytms.markThree(Algorytms.matrix)  
 for i in Algorytms.matrix:  
 print(\*i)  
 Algorytms.path = []  
 Algorytms.astar(Algorytms.curr)  
  
 Algorytms.enemyArray.remove(Algorytms.enemyArray[0])  
 Algorytms.arrayOfPath.append(Algorytms.path)  
 for j in Algorytms.arrayOfPath:  
 for i in j:  
 pixel = gv.ShipCreator.Pixel(int(i[1] \* 50), int(i[0] \* 50))  
 gv.pixelPath.append(pixel)  
 print(Algorytms.enemyArray)  
 *# for i in Algorytms.lenMatrix:  
 # print(\*i)* Algorytms.enemyArray = []  
 Algorytms.matrix = Algorytms.numpy.full((int(750 / 50), int(750 / 50)), 0)  
 if gv.GOOD\_SHIP and Algorytms.arrayOfPath:  
 if len(Algorytms.arrayOfPath[0]) > 1:  
 *# if not gv.GOOD\_SHIP.y == Algorytms.arrayOfPath[0][1][0]:  
 # gv.GOOD\_SHIP.y = Algorytms.arrayOfPath[0][1][0] \* 50  
 # Algorytms.curr = [Algorytms.arrayOfPath[0][1][0], int(gv.GOOD\_SHIP.x / 50)]* if not gv.GOOD\_SHIP.x == Algorytms.arrayOfPath[0][1][1]:  
 gv.GOOD\_SHIP.x = Algorytms.arrayOfPath[0][1][1] \* 50  
 Algorytms.curr = [int(gv.GOOD\_SHIP.y / 50), Algorytms.arrayOfPath[0][1][1]]  
 Algorytms.startPoint = [int(gv.GOOD\_SHIP.y / 50), Algorytms.arrayOfPath[0][1][1]]  
 Algorytms.arrayOfPath = []  
 for i in gv.ENEMIES:  
 if Algorytms.lenToFromPointtoPoint([int(gv.GOOD\_SHIP.y / 50), int(gv.GOOD\_SHIP.x/ 50)], [int(i.y / 50), int(i.x/ 50)]) < 3:  
 gv.ENEMIES.remove(i)  
 if [int(i.y / 50), int(i.x/ 50)] in Algorytms.enemyArray:  
 Algorytms.enemyArray.remove([int(i.y / 50), int(i.x/ 50)])  
 if [int(i.x / 50), int(i.y/ 50)] in Algorytms.enemyArray:  
 Algorytms.enemyArray.remove([int(i.x / 50), int(i.y/ 50)])  
 *# gv.GOOD\_SHIP.x = Algorytms.pointToResp[1]  
 # gv.GOOD\_SHIP.y = Algorytms.pointToResp[0]* Algorytms.enemyArray = []  
 Algorytms.arrayOfPath = []  
 Algorytms.createVisitMatrix(Algorytms.matrix)  
 Algorytms.fillMatrix(Algorytms.matrix)  
 if gv.RANDOM\_LIB.randrange(1, 200) == 1:  
 gv.currPoint = [int(Algorytms.pointToResp[1] / 50), int(Algorytms.pointToResp[0] / 50)]  
 Algorytms.startPoint = [int(Algorytms.pointToResp[1] / 50), int(Algorytms.pointToResp[0] / 50)]  
 *# print(len(Algorytms.arrayOfPath))* Algorytms.moveEnemy()  
  
 gv.PG\_LIB.display.update()  
 *# for k in gv.dfsArrayOfPath:  
 # for i in range(len(k)):  
 # for j in range(len(k)):  
 # if k[i][j] == 1 or k[i][j] == 4:  
 # gv.WINDOW.set\_at((i,j), (255,0,0))  
  
  
 # print(gv.enemyCount)  
  
  
  
 # print(len(Algorytms.Matrix))  
 # print(gv.currPoint)*

У Algoryhtmms.py реалізовано усі алгоритми

import GlobalVariables as gv  
import numpy  
  
*# empty = 0  
# current = 1  
# enemy = 2  
# asteroids = 3*matrix = numpy.full((int(750 / 50), int(750 / 50)), 0)  
lenMatrix = numpy.full((int(750 / 50), int(750 / 50)), 0)  
pointToResp = [600, 350]  
path = []  
numofEnemy = 9  
startPoint = [13, 7]  
curr = [13, 7]  
enemyArray = []  
arrayOfPath = []  
  
def createVisitMatrix(matrix):  
 for i in range(len(matrix)):  
 for j in range(len(matrix[i])):  
 if matrix[i][j] == 2:  
 enemyArray.append([i, j])  
  
def emptyMatrix(matr, cur):  
 global curr  
 for i in range(0, len(matr) - 1):  
 for j in range(0, len(matr[i]) - 1):  
 if not matr[i][j] == 0:  
 matr[i][j] = 0  
 matr[cur[0]][cur[1]] = 1  
 *# for i in matr:  
 # print(\*i)  
#TODO: add baricades(3)*def lenToFromPointtoPoint(cur, startPoiint):  
 distance = 0  
 if cur[0] > startPoiint[0]:  
 distance += cur[0] - startPoiint[0]  
 else:  
 distance += startPoiint[0] - cur[0]  
 if cur[1] > startPoiint[1]:  
 distance += cur[1] - startPoiint[1]  
 else:  
 distance += startPoiint[1] - cur[1]  
 return distance  
  
  
def lenFinal(curr):  
 return lenToFromPointtoPoint(curr, startPoint) + lenToFromPointtoPoint(curr, enemyArray[0]) \* 10  
  
  
*# print(lenToFromPointtoPoint([12, 5], [1, 7]))  
# print(enemyArray)*def isEnemyClose(cur):  
 if 0 <= cur[0] + 1 < 15 and 0 <= cur[1] + 1 < 15 and lenMatrix[cur[0] + 1][cur[1] + 1] == -1:  
 return True  
 elif 0 <= cur[0] - 1 < 15 and 0 <= cur[1] - 1 < 15 and lenMatrix[cur[0] - 1][cur[1] - 1] == -1:  
 return True  
 elif 0 <= cur[0] + 1 < 15 and 0 <= cur[1] - 1 < 15 and lenMatrix[cur[0] + 1][cur[1] - 1] == -1:  
 return True  
 elif 0 <= cur[0] - 1 < 15 and 0 <= cur[1] + 1 < 15 and lenMatrix[cur[0] - 1][cur[1] + 1] == -1:  
 return True  
 elif 0 <= cur[0] < 15 and 0 <= cur[1] + 1 < 15 and lenMatrix[cur[0]][cur[1] + 1] == -1:  
 return True  
 elif 0 <= cur[0] + 1 < 15 and 0 <= cur[1] < 15 and lenMatrix[cur[0] + 1][cur[1]] == -1:  
 return True  
 elif 0 <= cur[0] < 15 and 0 <= cur[1] - 1 < 15 and lenMatrix[cur[0]][cur[1] - 1] == -1:  
 return True  
 elif 0 <= cur[0] - 1 < 15 and 0 <= cur[1] < 15 and lenMatrix[cur[0] - 1][cur[1]] == -1:  
 return True  
 else:  
 return False  
  
  
def markPoints(cur):  
 if 0 < cur[0] + 1 < len(lenMatrix) and 0 < cur[1] + 1 < len(lenMatrix) and not lenMatrix[cur[0] + 1][cur[1] + 1] == 999:  
 lenMatrix[cur[0] + 1][cur[1] + 1] = lenFinal([cur[0] + 1, cur[1] + 1])  
 if 0 < cur[0] - 1 < len(lenMatrix) and 0 < cur[1] - 1 < len(lenMatrix) and not lenMatrix[cur[0] - 1][cur[1] - 1] == 999:  
 lenMatrix[cur[0] - 1][cur[1] - 1] = lenFinal([cur[0] - 1, cur[1] - 1])  
 if 0 < cur[0] + 1 < len(lenMatrix) and 0 < cur[1] - 1 < len(lenMatrix) and not lenMatrix[cur[0] + 1][cur[1] - 1] == 999:  
 lenMatrix[cur[0] + 1][cur[1] - 1] = lenFinal([cur[0] + 1, cur[1] - 1])  
 if 0 < cur[0] - 1 < len(lenMatrix) and 0 < cur[1] + 1 < len(lenMatrix) and not lenMatrix[cur[0] - 1][cur[1] + 1] == 999:  
 lenMatrix[cur[0] - 1][cur[1] + 1] = lenFinal([cur[0] - 1, cur[1] + 1])  
 if 0 < cur[0] < len(lenMatrix) and 0 < cur[1] + 1 < len(lenMatrix) and not lenMatrix[cur[0]][cur[1] + 1] == 999:  
 lenMatrix[cur[0]][cur[1] + 1] = lenFinal([cur[0], cur[1] + 1])  
 if 0 < cur[0] + 1 < len(lenMatrix) and 0 < cur[1] < len(lenMatrix) and not lenMatrix[cur[0] + 1][cur[1]] == 999:  
 lenMatrix[cur[0] + 1][cur[1]] = lenFinal([cur[0] + 1, cur[1]])  
 if 0 < cur[0] < len(lenMatrix) and 0 < cur[1] - 1 < len(lenMatrix) and not lenMatrix[cur[0]][cur[1] - 1] == 999:  
 lenMatrix[cur[0]][cur[1] - 1] = lenFinal([cur[0], cur[1] - 1])  
 if 0 < cur[0] - 1 < len(lenMatrix) and 0 < cur[1] < len(lenMatrix) and not lenMatrix[cur[0] - 1][cur[1]] == 999:  
 lenMatrix[cur[0] - 1][cur[1]] = lenFinal([cur[0] - 1, cur[1]])  
 if 0 < cur[0] < len(lenMatrix) and 0 < cur[1] < len(lenMatrix) and not lenMatrix[cur[0]][cur[1]] == 999:  
 lenMatrix[cur[0]][cur[1]] = lenFinal([cur[0], cur[1]])  
  
def moveEnemy():  
 global arrayOfPath  
 for i in gv.ENEMIES:  
 if i.x == gv.GOOD\_SHIP.x and 600 > i.x > 50:  
 if [int(i.y / 50), int(i.x / 50)] in enemyArray:  
 enemyArray.remove([int(i.y / 50), int(i.x / 50)])  
 if [int(i.x / 50), int(i.y / 50)] in enemyArray:  
 enemyArray.remove([int(i.x / 50), int(i.y / 50)])  
 buf = gv.RANDOM\_LIB.choice([1, 0])  
 if buf == 0:  
 i.x += 50  
 else:  
 i.x -= 50  
  
 arrayOfPath = []  
 createVisitMatrix(matrix)  
 fillMatrix(matrix)  
  
def getCoordsOfSmallest(matrix):  
 value = 9999  
 currentVay = []  
 for i in range(0, len(matrix)):  
 for j in range(0, len(matrix[i])):  
 if not matrix[i][j] == -1 and matrix[i][j] > 0:  
 if matrix[i][j] < value:  
 value = matrix[i][j]  
 currentVay = [i, j]  
 return currentVay  
  
  
def astar(curr):  
 lenMatrix[enemyArray[0][0]][enemyArray[0][1]] = -1  
 if not isEnemyClose(curr):  
 markPoints(curr)  
 path.append(curr)  
 curr = getCoordsOfSmallest(lenMatrix)  
 *#print(curr)* astar(curr)  
  
def markThree(matr):  
 for i in range(0, len(matr)):  
 for j in range(0, len(matr[i])):  
 if matr[i][j] == 3:  
 lenMatrix[i][j] = 999  
  
  
def fillMatrix(matrix):  
 for i in gv.ENEMIES:  
 if 0 < int(i.y / 50) < 15 and 0 < int(i.x / 50) < 15:  
 matrix[int(i.y / 50)][int(i.x / 50)] = 2  
 for i in gv.ASTEROIDS:  
 if 0 < int(i.y / 50) < 15 and 0 < int(i.x / 50) < 15:  
 matrix[int(i.y / 50)][int(i.x / 50)] = 3

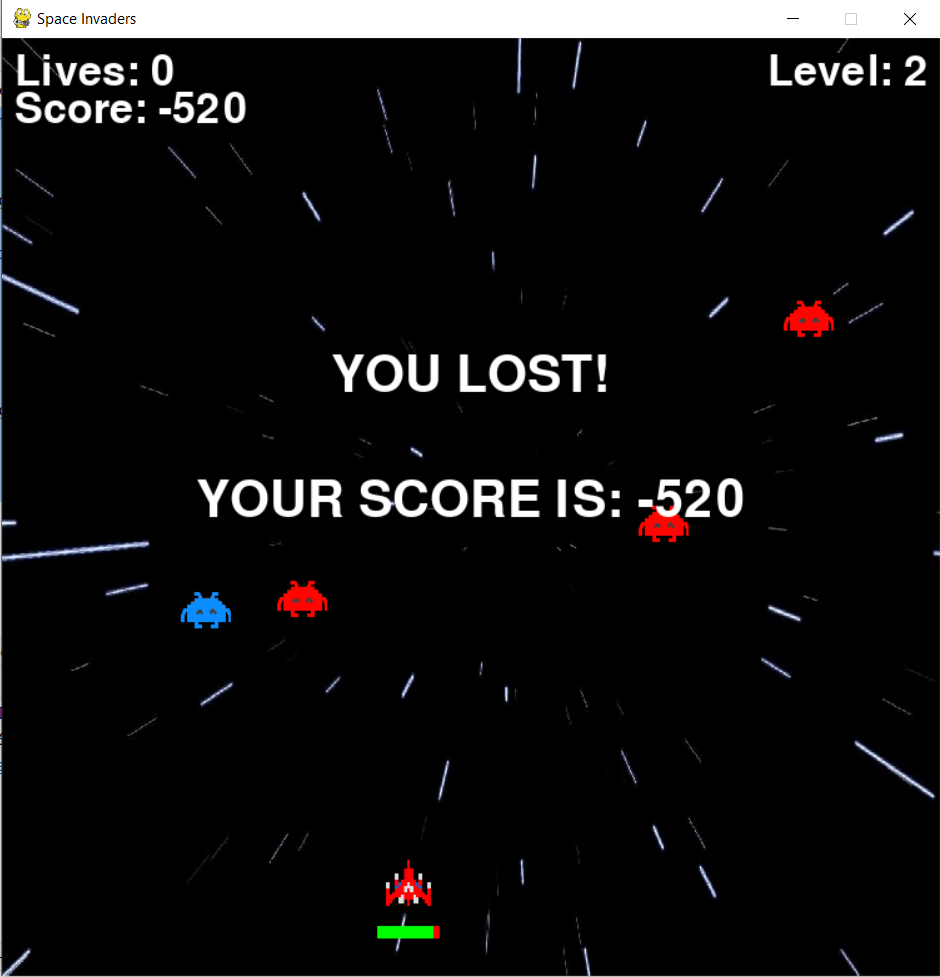
# **Скріншоти роботи програмного застосунку**



1 – Початкове меню гри



2 – момент з гри



3 – фрейм програшу