

BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



BÁO CÁO BÀI TẬP LỚN
Môn: Lập trình Python
Giảng Viên: Nguyễn Trọng Khánh

Nhóm 9

B22DCCN482 Trịnh Quang Lâm

B22DCCN381 Lê Đức Huy

B22DCCN194 Nguyễn Đức Đạt

B22DCCN470 Nguyễn Đức Khởi

B22DCCN384 Vũ Nhân Kiên

Hà Nội, 02/12/2024

Mục lục

Lời nói đầu	2
I.Giới thiệu về dự án	3
<i>Mục đích dự án (Giới thiệu về Tank Game):</i>	<i>3</i>
<i>Mục tiêu:</i>	<i>3</i>
II.Phân công công việc các thành viên trong nhóm	3
III.Phân tích yêu cầu.....	4
<i>1.Yêu cầu chức năng:.....</i>	<i>4</i>
<i>2.Yêu cầu phi chức năng:.....</i>	<i>4</i>
IV. Thiết kế hệ thống – Kiến trúc hệ thống của game.....	5
<i>1. File Main game: Chứa các file Asset, Map, AI (tìm đường) và các file code liên quan đến việc chạy logic chương trình như sau:.....</i>	<i>5</i>
<i>2. File Setting game: chứa các file hỗ trợ cài đặt chương trình</i>	<i>5</i>
V. Triển khai cho dự án Tank Game	8
<i>1. Tiền xử lý game.....</i>	<i>8</i>
<i>2. Chạy game (cách thức gameloop hoạt động và các thành phần chính ở gameloop)</i>	<i>8</i>
Map	8
Xe tank	9
Hiệu ứng hình ảnh và âm thanh	19
Một số logic quan trọng của game	20
Item	23
VI. Demo trò chơi.....	25
VII. Một số tính năng đặc biệt	29
VIII. Khó khăn khi thực hiện dự án và giải pháp	29
IX. Tài liệu tham khảo.....	30

Lời nói đầu

Python là một trong những ngôn ngữ lập trình phổ biến và mạnh mẽ, được ứng dụng rộng rãi trong nhiều lĩnh vực như phát triển phần mềm, trí tuệ nhân tạo, xử lý dữ liệu và phát triển game. Trong bối cảnh đó, dự án này được thực hiện nhằm vận dụng những kiến thức đã học để xây dựng một trò chơi 2D với lối chơi đơn giản nhưng đầy hấp dẫn, qua đó rèn luyện kỹ năng lập trình và tư duy logic.

Bài báo cáo này sẽ trình bày toàn bộ quá trình phát triển dự án **Tank Game**, từ thiết kế hệ thống đến triển khai và kiểm thử, giúp người đọc hiểu rõ hơn về cách xây dựng một trò chơi hoàn chỉnh bằng ngôn ngữ Python và thư viện Pygame.

I. Giới thiệu về dự án

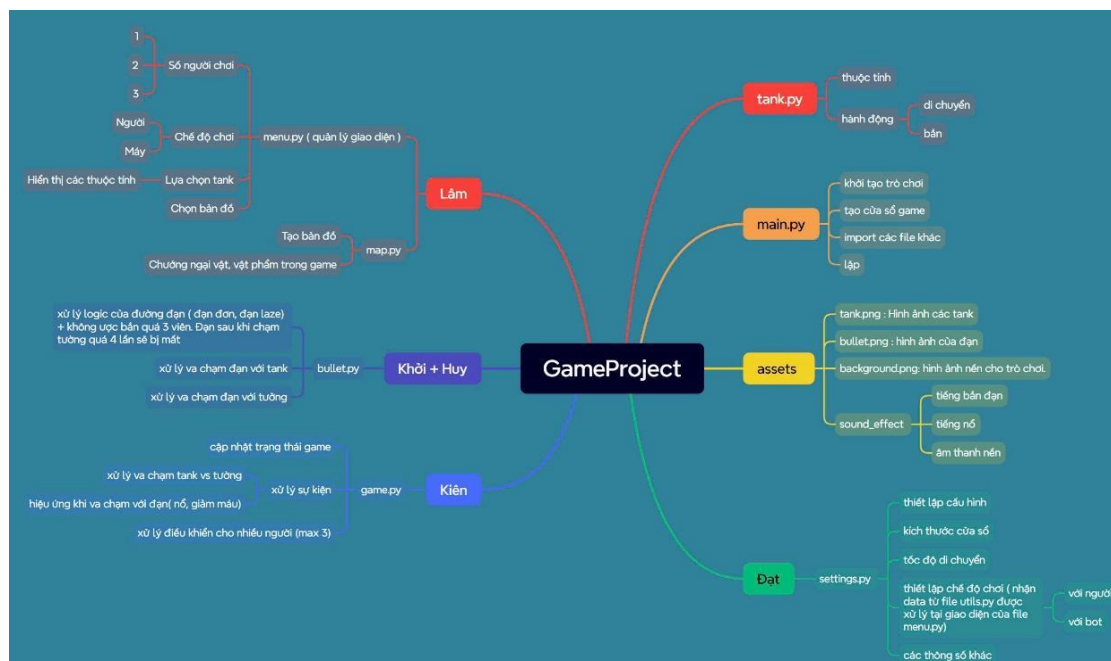
Mục đích dự án (Giới thiệu về Tank Game):

Là 1 trò chơi chiến đấu chiến thuật giữa các xe tăng với đa dạng các bản đồ

Mục tiêu:

Xây dựng 1 trò chơi giải trí tương tác đầy đủ tính năng của 1 game đối kháng chiến thuật, cung cấp trải nghiệm điều khiển và chiến đấu xe tăng trong môi trường 2D

II. Phân công công việc các thành viên trong nhóm



III. Phân tích yêu cầu

1. Yêu cầu chức năng:

- Người chơi có thể chọn số lượng xe tăng tùy thuộc vào số lượng người chơi
- Người chơi cũng có thể tùy chọn bản đồ trước khi bắt đầu trò chơi
- Điều khiển xe tăng bằng bàn phím và chuột, và tất cả mọi thao tác trong khi trò chơi diễn ra phải đủ mượt mà, làm tăng trải nghiệm game
- Tương tác vật phẩm và những vật phẩm đó mang lại những tính năng hỗ trợ nhất định cho xe tăng (ví dụ như bắn các loại đạn khác nhau, hồi máu, giáp, tăng tốc độ)
- Hiện thị thông báo khi một người chơi chiến thắng
- Có tùy chọn chơi lại hoặc thoát khi kết thúc trò chơi

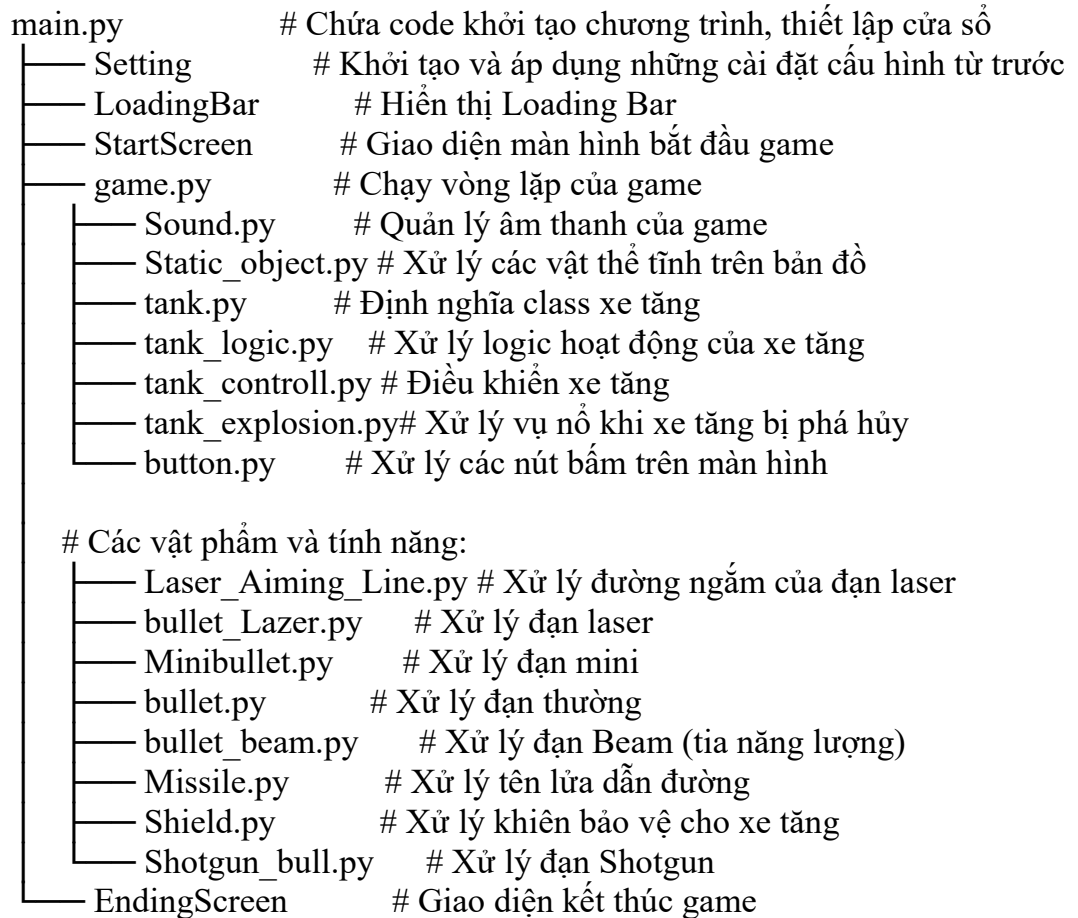
2. Yêu cầu phi chức năng:

- Trò chơi phải chạy mượt mà, yêu cầu phân xử lý logic phải đủ nhanh
- Giao diện đồ họa 2D phải đơn giản, tránh màu mè khiến người chơi bị rối mắt, nhưng đồng thời phải rõ ràng, dễ nhìn, thân thiện với người chơi.
- Độ trễ của phản hồi thấp khi điều khiển xe tăng

IV. Thiết kế hệ thống – Kiến trúc hệ thống của game

Dự án Tank Game được chia thành các phần chính như sau:

1. File Main game: Chứa các file Asset, Map, AI (tìm đường) và các file code liên quan đến việc chạy logic chương trình như sau:



Hình 1: Cấu trúc thư mục dự án

2. File Setting game: chứa các file hỗ trợ cài đặt chương trình
 - Mô tả ý tưởng chương trình: Trước khi run chương trình ta sẽ chạy một preprocess để đọc cấu hình màn hình thông số của máy để lấy recommended resolution mà máy đang dùng ! Ở phần pre-process này nhóm em dùng ngôn ngữ C++ để có thể đọc cấu hình của máy, vì ngôn ngữ C++ hỗ trợ mạnh với việc các thao tác liên quan đến phần cứng hay hệ điều hành và chạy nhanh hơn so với Python. Ở đây chúng em có tham khảo thư viện C++ GLFW hỗ trợ cho việc đọc cấu hình này!

```

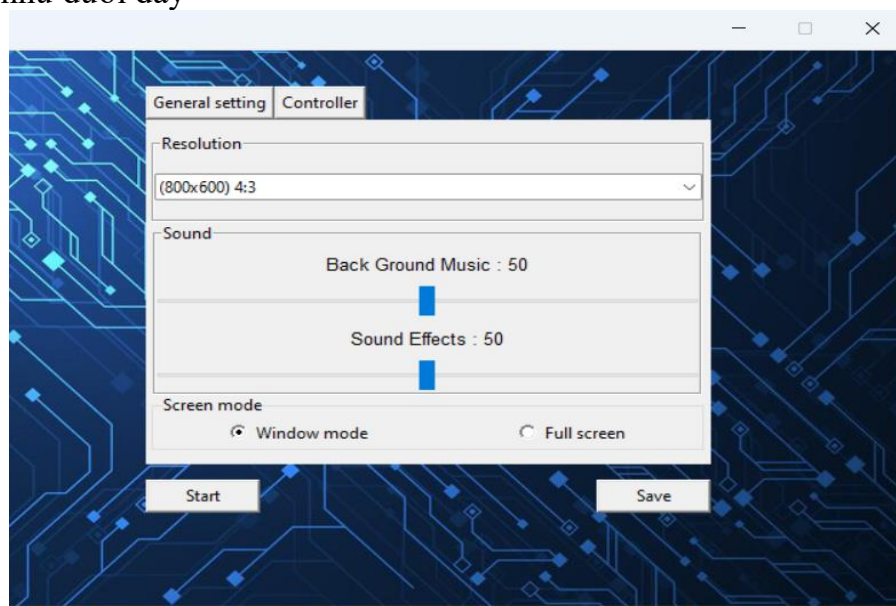
import re
import tkinter as tk
import subprocess
from tkinter import PhotoImage, Button, Canvas, messagebox
from tkinter import ttk
import pygame
from pygame import mixer

subprocess.run(["test reso.exe"])
resolution = ""
try:
    with open('screen_resolution.txt', 'r') as f:
        resolution = f.readline().strip()
except FileNotFoundError:
    resolution = "Error"

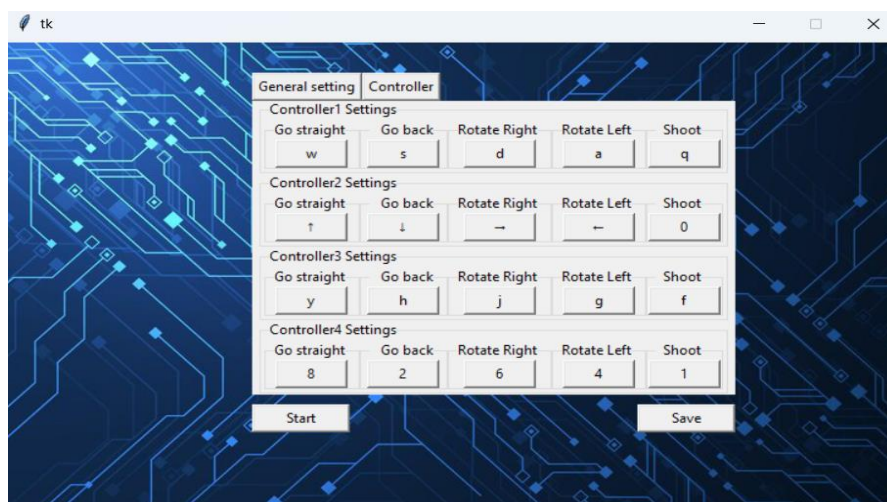
```

Hình 2: Đọc cấu hình

- Sau đó sẽ hiện thị một bảng setting như ảnh dưới đây và ta có thể tùy chỉnh thông số của game để vào như dưới đây



Hình 3.1: Bảng Setting



Hình 3.2: Bảng Setting

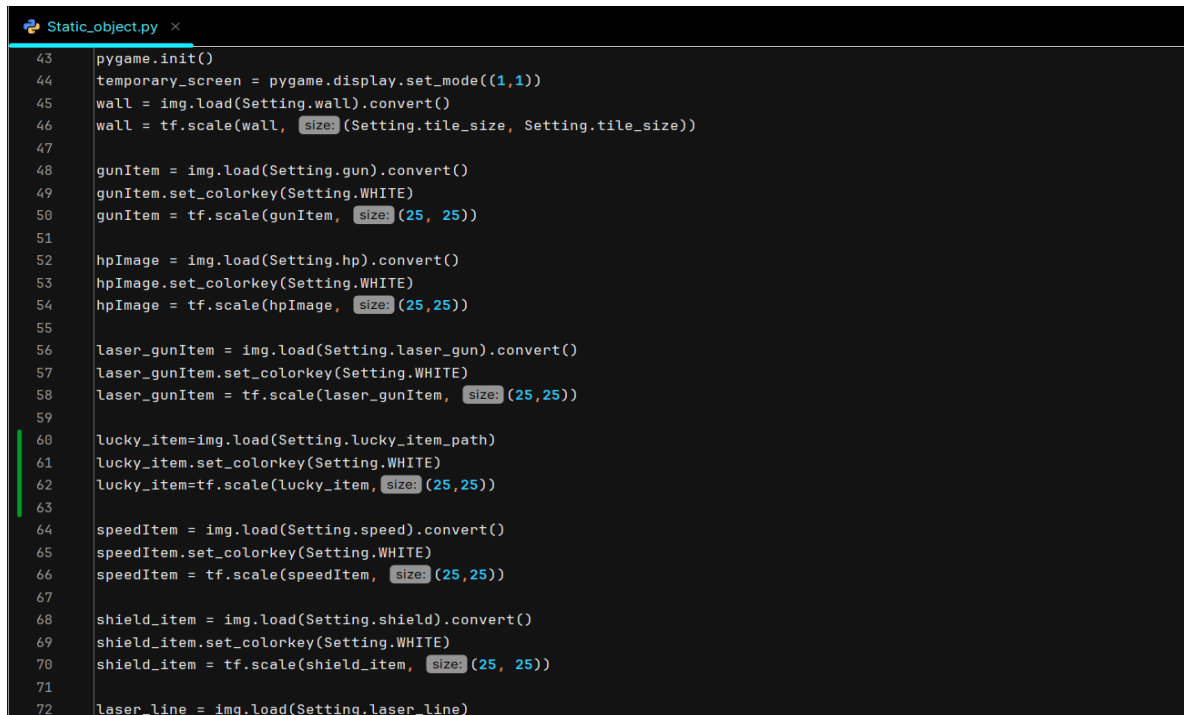
- Khi ấn nút “Save” thì mọi thông số đang hiện ở màn hình Setting sẽ được lưu vào file có tên “Saved.txt”

- Còn khi ta bấm start thì chương trình sẽ chạy một sub-process có tên là mid.exe (chương trình này được viết bằng ngôn ngữ C++, có tác dụng tắt một process khác và bật một process mới lên ở đây là setting.exe sẽ bị tắt và game.exe được bật. Tại sao lại không chạy luôn game.exe khi bấm nút “Start” vì điều này sẽ dẫn đến việc crash chương trình, ta nên đóng setting.exe xong rồi mới khởi động game.exe)

V. Triển khai cho dự án Tank Game

1. Tiền xử lý game

- Trong pygame, nhà phát triển cung cấp cho ta một hàm vẽ hình ảnh `pygame.image.load(path_image)` dựa trên những bức ảnh có đuôi như `.jpg` hay `.png`, để ta có thể vẽ vào cửa sổ giao diện người dùng. Hàm này có độ phức tạp $O(W*H)$ (với W và H là chiều ngang và chiều cao của bức ảnh). Nếu như trong chương trình main, ta mới gọi tới những hàm này để xử lý việc vẽ các hoạt ảnh thì sẽ làm chương trình trở nên nặng hơn về mặt thời gian xử lý, do đó nhóm em có một giải pháp đó là trước khi vào game chúng em sẽ xử lý hết các tài nguyên hoạt ảnh trước và lưu vào các biến trong file được gọi là `Static_object` sau đó thì trong gameloop chỉ đơn giản gọi lại các hình ảnh đã được xử lý rồi (đây là một kỹ thuật tối ưu khá phổ biến trong ngành phát triển game)



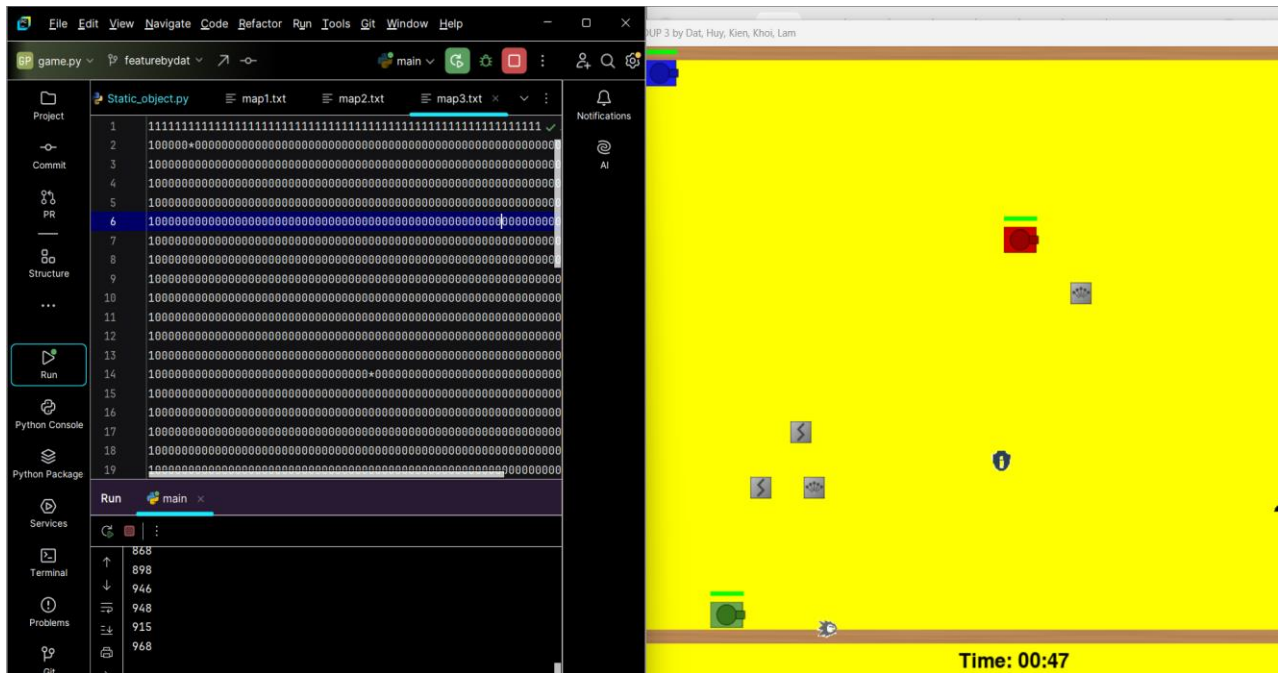
```
43 pygame.init()
44 temporary_screen = pygame.display.set_mode((1,1))
45 wall = img.load(Setting.wall).convert()
46 wall = tf.scale(wall, size=(Setting.tile_size, Setting.tile_size))
47
48 gunItem = img.load(Setting.gun).convert()
49 gunItem.set_colorkey(Setting.WHITE)
50 gunItem = tf.scale(gunItem, size=(25, 25))
51
52 hpImage = img.load(Setting.hp).convert()
53 hpImage.set_colorkey(Setting.WHITE)
54 hpImage = tf.scale(hpImage, size=(25, 25))
55
56 laser_gunItem = img.load(Setting.laser_gun).convert()
57 laser_gunItem.set_colorkey(Setting.WHITE)
58 laser_gunItem = tf.scale(laser_gunItem, size=(25,25))
59
60 lucky_item=img.load(Setting.lucky_item_path)
61 lucky_item.set_colorkey(Setting.WHITE)
62 lucky_item=tf.scale(lucky_item, size=(25,25))
63
64 speedItem = img.load(Setting.speed).convert()
65 speedItem.set_colorkey(Setting.WHITE)
66 speedItem = tf.scale(speedItem, size=(25,25))
67
68 shield_item = img.load(Setting.shield).convert()
69 shield_item.set_colorkey(Setting.WHITE)
70 shield_item = tf.scale(shield_item, size=(25, 25))
71
72 laser_line = img.load(Setting.laser_line)
```

Hình 4: Lưu giá trị hình ảnh trước khi chạy game

- Tương tự với xử lý âm thanh, chúng em cũng thực hiện tiền xử lý với những âm thanh đuôi như `.mp3` `.wav`,..
- ### 2. Chạy game (cách thức gameloop hoạt động và các thành phần chính ở gameloop)

Map

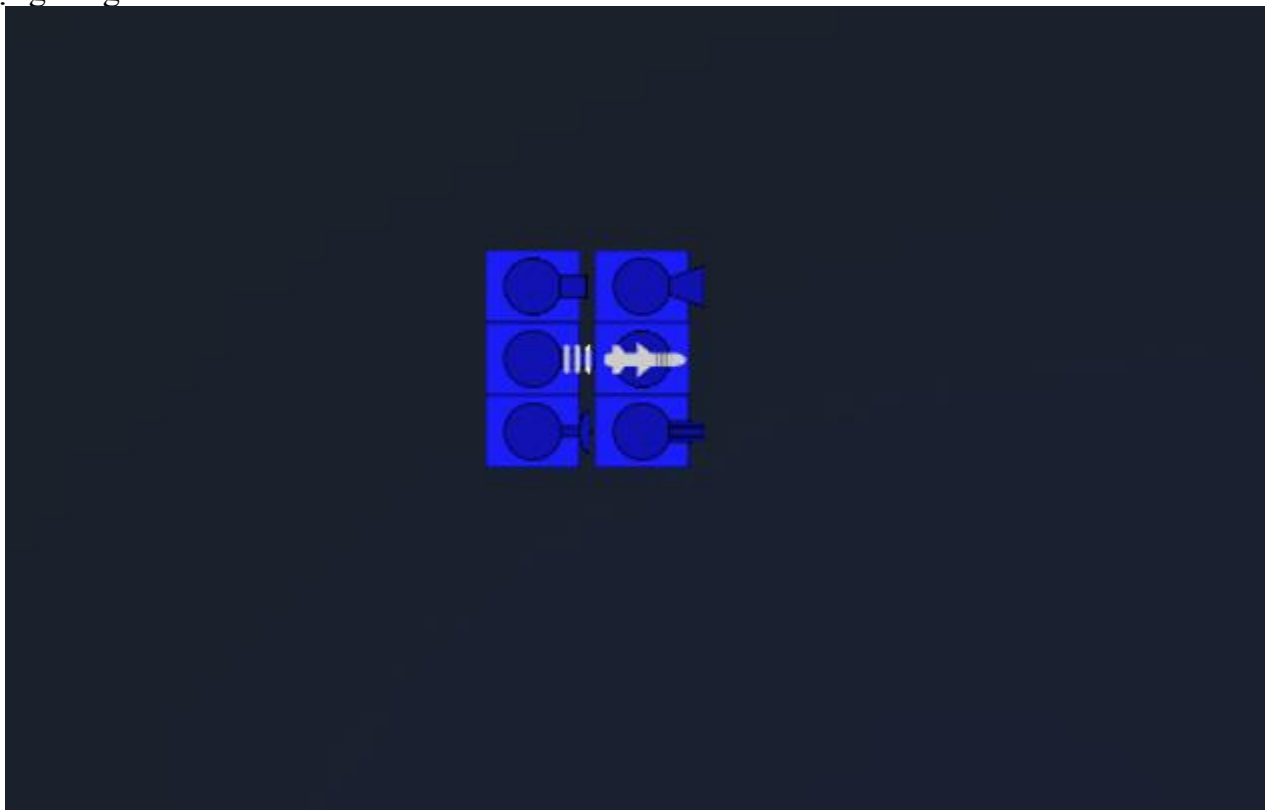
Trước khi vào màn chơi chính, ta có thể chọn số lượng người chơi và chọn map thi đấu. Map ở đây được nhóm chúng em vẽ dựa theo một ma trận có kích cỡ 43x64 bao gồm kí tự 0,1,* có tỉ lệ tùy thuộc vào cấu hình ví dụ như tỉ lệ 1:16 thì một phần tử ở ma trận tương ứng với 16 pixel hình ảnh trong map. Với '0' là những nơi ta có thể di chuyển, '1' là nơi ta không thể di chuyển và '*' là nơi người các người chơi được đặt vào khi mới bắt đầu trò chơi.



Hình 5: Hình ảnh tương quan giữa ma trận mô phỏng map và map thực

Xe tank

- Đây là class quan trọng của game với các thuộc tính như hình ảnh, id, dạng chiến đấu, điều khiển,...
- Hình ảnh của xe tank được lấy ở phần tiền xử lý (nơi ta đã xử lý các hình ảnh được thiết kế có đuôi.png hay .jpg). Có tổng cộng 6 hình ảnh khác nhau với mỗi xe tăng tương ứng với 6 dạng súng của mỗi xe.

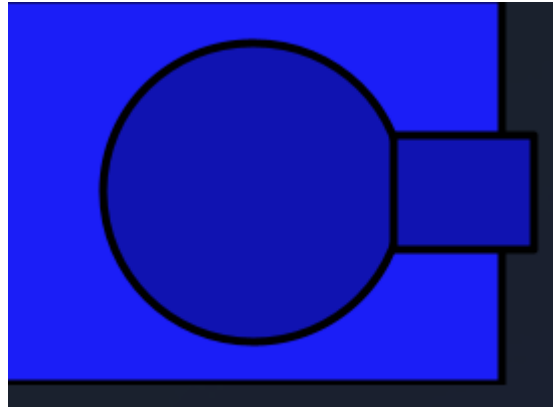


Hình 6: Các dạng xe tank tương ứng với các dạng súng

- Xe tank của mỗi người chơi đều có 6 dạng súng khác nhau. Với mỗi dạng súng thì người chơi sẽ sở hữu những đặc trưng khác nhau của súng, để có thể chuyển đổi dạng chiến đấu thì các người chơi sẽ phải điều khiển xe tank của mình tới vị trí item để chuyển đổi. Đây

cũng là một phần quan trọng góp trò chơi không bị tẻ nhạt, sau đây nhóm em sẽ trình bày về những đặc điểm tiêu biểu của từng dạng chiến đấu đó :

- Dạng thường



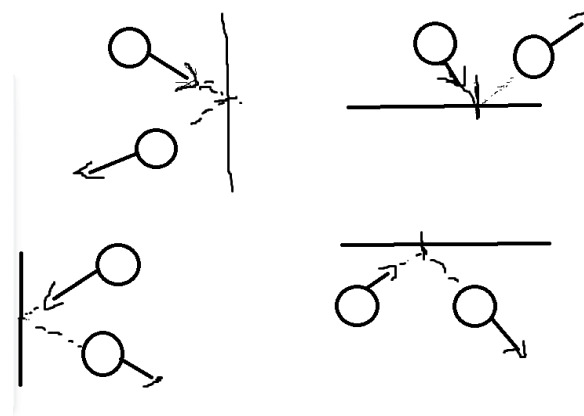
Hình 7: Hình ảnh xe tank dạng thường

- Ở dạng chiến đấu này, xe tăng của người chơi sẽ bắn ra viên đạn màu đen có hình tròn.

```
class Bullet: 6 usages  ▲ datbuudien +1*
def __init__(self, tank, x, y, angle, speed=1):  ▲ datbuudien +1*
    self.tank=tank
    self.radius = 5 # Bán kính của hình tròn
    self.color = tank.bullet_color
    self.name="Normal"
    self.rect = pygame.Rect(x - self.radius, y - self.radius, self.radius * 2, self.radius * 2)
    self.angle = angle #góc ban đầu để tính ra vector vận tốc
    self.speed = speed
    self.bullet_x = self.rect.x
    self.bullet_y = self.rect.y
    self.creation_time = pygame.time.get_ticks()
    self.bounce_count = 0 #số lần chạm tường
    #self.dame=10
    # tính vector vận tốc
    self.direction_x = math.cos(math.radians(self.angle)) * self.speed
    self.direction_y = -math.sin(math.radians(self.angle)) * self.speed
```

Hình 8: Lớp Bullet mô phỏng đạn

- Khi đạn gặp tường thì sẽ nảy ngược lại tùy thuộc vào vector vận tốc của viên đạn



Hình 9: Các trường hợp tiêu biểu của đạn khi gặp tường

```

# Canh ben trai
if map_data[int(b/16)][int((a+self.radius)/16)]=='1':
    self.direction_x *=-1
    self.bounce_count +=1
#Canh tren
elif map_data[int((b-self.radius)/16)][int(a/16)]=='1':
    self.direction_y *=-1
    self.bounce_count +=1
#canh duoi
elif map_data[int((b+self.radius)/16)][int(a/16)]=='1':
    self.direction_y *=-1
    self.bounce_count+=1
#canh ben phai
elif map_data[int(b/16)][int((a-self.radius)/16)]=='1':
    self.direction_x *=-1
    self.bounce_count+=1
elif map_data[int((b+self.radius)/16)][int((a+self.radius)/16)]=='1':
    x= int((a+self.radius)/16)*16
    y= int((b+self.radius)/16)*16
    if TankLogic.check_in_circle(point: (x,y),a,b,self.radius):
        if self.direction_x>0:
            self.direction_x *=-1
        else: self.direction_y *=-1
elif map_data[int((b - self.radius) / 16)][int((a + self.radius) / 16)] == '1':
    x= int((a+self.radius)/16)*16
    y=int((b - self.radius) / 16)+16
    if TankLogic.check_in_circle(point: (x,y),a,b,self.radius):

```

Hình 10: Xử lý logic ứng với từng loại đạn nảy

- Người chơi chỉ có thể bắn tối đa 4 viên đạn cùng một lúc, mỗi viên khai hỏa phải cách nhau 1 giây. Nếu muốn bắn thêm thì phải đợi cho các viên đạn ở lần bắn trước biến mất (điều kiện là đạn va chạm tường 10 lần)

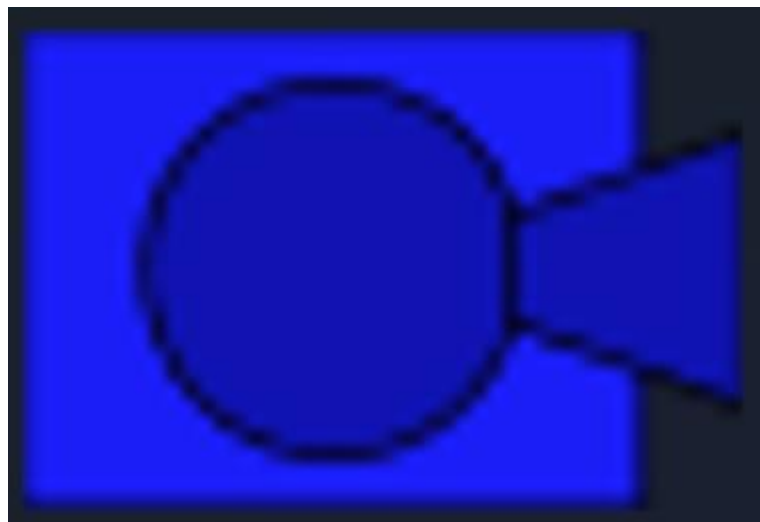
```

def is_expired_bullet(self): 2 usages (2 dynamic)  datbuudien +1
return self.bounce_count >= 10

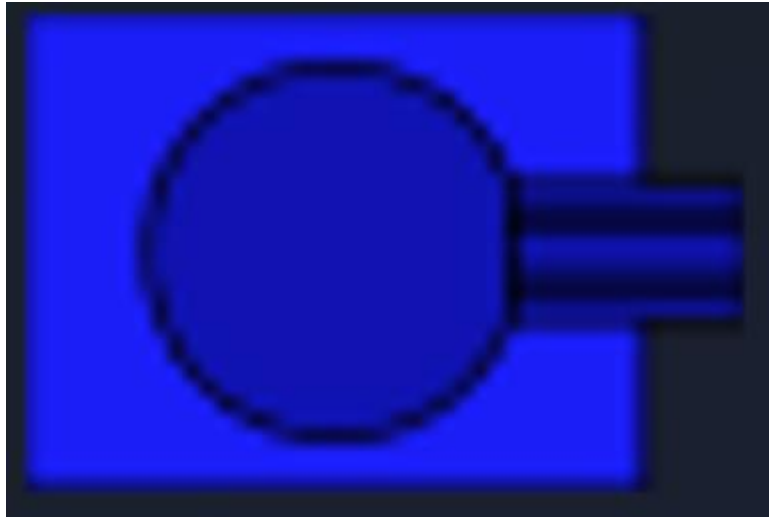
```

Hình 11: Logic kiểm tra đạn cần được xóa bỏ

- Dạng shotgun và dạng minigun



Hình 12: Dạng shotgun



Hình 13: Dạng minigame

- Ở 2 dạng này người chơi vẫn khai hỏa ra loại đạn tròn đen tuy nhiên có kích thước bé hơn so với đạn thường đều được kế thừa từ lớp đạn thường ở phía trên (nên mọi logic của 2 loại đạn này giống với đạn thường)
- Điểm khác biệt ở chỗ: Với đạn minigun người chơi có thể bắn liên tục 3 viên đạn cùng lúc, cùng vector vận tốc(chỉ khác điểm được vẽ) với thời gian khai hỏa chỉ cách nhau 0.2s. Còn với shotgun người chơi có thể bắn ra một chùm đạn với sát thương lớn tuy nhiên vector vận tốc của các viên đạn lại lệch nhau. Khi bắn hết băng đạn thì cả 2 dạng súng này sẽ trở về lại dạng mặc định

```
elif self.tank.gun_mode == 3: #minigun
    if (current_time - self.last_shoot) > 200:
        minibull_1=Minigun(self.tank,
            self.tank.tank_rect.centerx + 29 * math.cos(math.radians(self.tank.tank_angle+15)),
            self.tank.tank_rect.centery - 29 * math.sin(math.radians(self.tank.tank_angle+15)),
            self.tank.tank_angle)
        minibull_2=Minigun(self.tank,
            self.tank.tank_rect.centerx + 29 * math.cos(math.radians(self.tank.tank_angle)),
            self.tank.tank_rect.centery - 29 * math.sin(math.radians(self.tank.tank_angle)),
            self.tank.tank_angle)
        minibull_3=Minigun(self.tank,
            self.tank.tank_rect.centerx + 29 * math.cos(math.radians(self.tank.tank_angle-15)),
            self.tank.tank_rect.centery - 29 * math.sin(math.radians(self.tank.tank_angle-15)),
            self.tank.tank_angle)
        self.bullets.append(minibull_1)
```

Hình 14: Chùm đạn của minigun

```

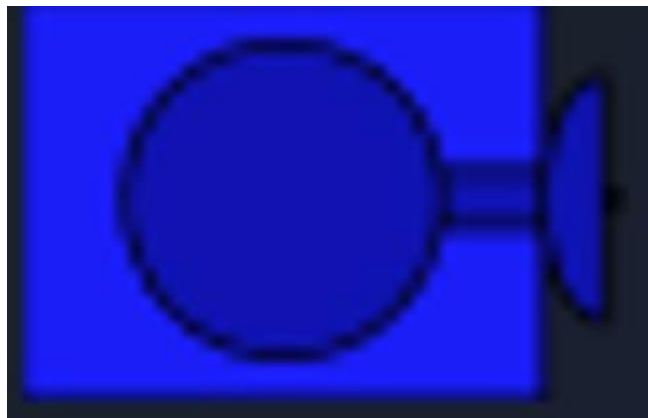
elif self.tank.gun_mode ==6: #shotgun
    if (current_time - self.last_shoot) > 400:
        shotgun_1 = Shotgun(self.tank,
                               self.tank.tank_rect.centerx + 29 * math.cos(
                                   math.radians(self.tank.tank_angle)),
                               self.tank.tank_rect.centery - 29 * math.sin(
                                   math.radians(self.tank.tank_angle)),
                               self.tank.tank_angle+6)
        shotgun_2= Shotgun(self.tank,
                               self.tank.tank_rect.centerx + 29 * math.cos(
                                   math.radians(self.tank.tank_angle)),
                               self.tank.tank_rect.centery - 29 * math.sin(
                                   math.radians(self.tank.tank_angle)),
                               self.tank.tank_angle+4)

        shotgun_3 = Shotgun(self.tank,
                               self.tank.tank_rect.centerx + 29 * math.cos(
                                   math.radians(self.tank.tank_angle)),
                               self.tank.tank_rect.centery - 29 * math.sin(
                                   math.radians(self.tank.tank_angle)),
                               self.tank.tank_angle+2)
        shotgun_4= Shotgun(self.tank,
                               self.tank.tank_rect.centerx + 29 * math.cos(
                                   math.radians(self.tank.tank_angle)),
                               self.tank.tank_rect.centery - 29 * math.sin(
                                   math.radians(self.tank.tank_angle)),

```

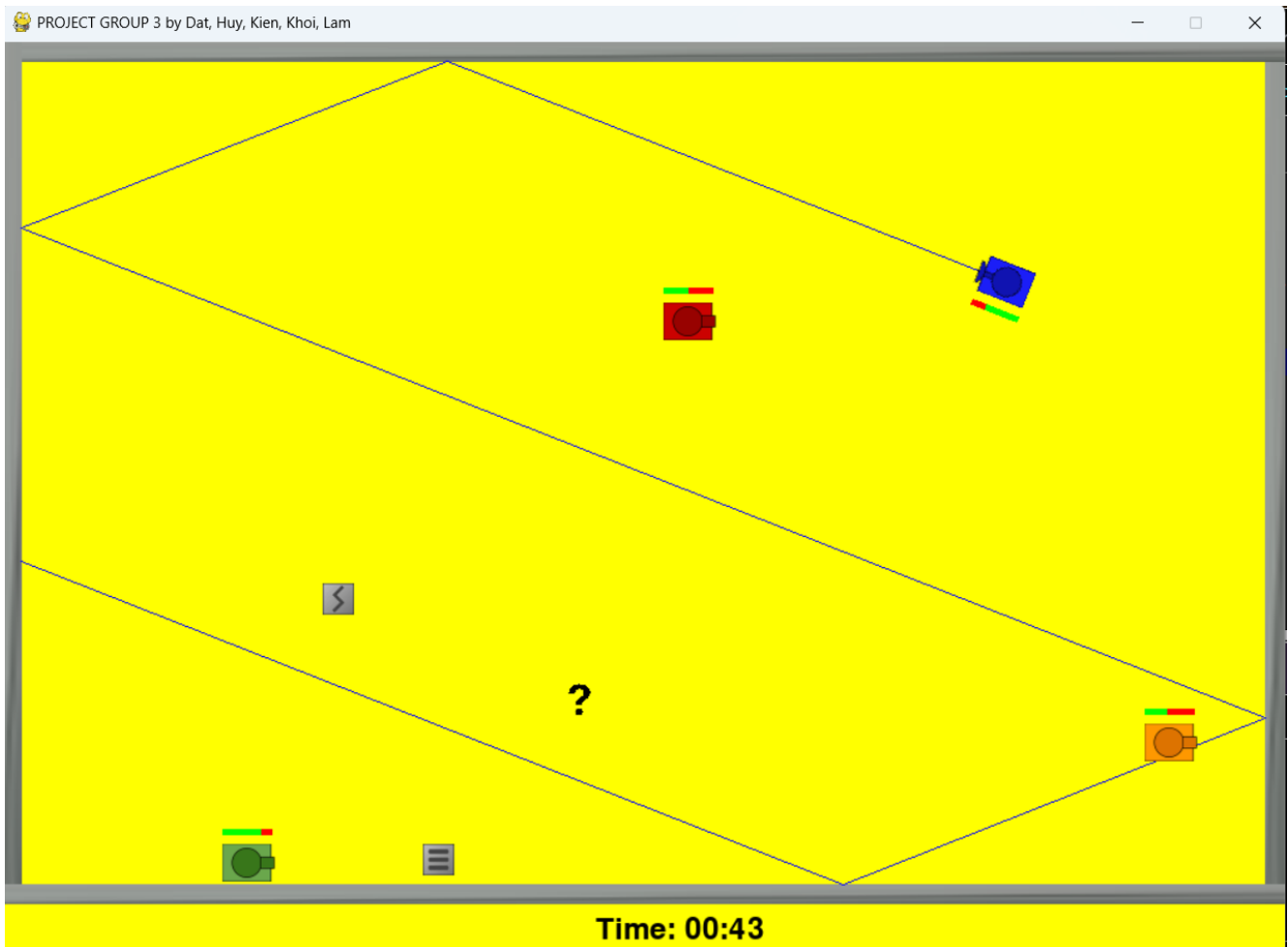
Hình 15: Chùm đạn của shotgun

- Dạng laser



Hình 16: Xe tank ở dạng Lazer

Ở dạng này người chơi sẽ được nhận một đường ngắm. Đường ngắm này là một đường thẳng và nó sẽ phản xạ khi gặp tường.



Hình 17: Hình ảnh đường ngầm

Đường ngầm sẽ hoạt động theo cách sau, đầu tiên ta vẽ một đường thẳng có độ dài lớn

```
def draw_2_line(self, window, color): 1 usage  datbuudien
    pygame.draw.line(window, color, start_pos=[self.x, self.y], end_pos=[self.end_x, self.end_y], width=1)
```

Hình 18: Hàm vẽ đường thẳng được hỗ trợ trong pygame

```
self.end_x = self.x + 10000 * math.cos(math.radians(self.angle))
self.end_y = self.y + 10000 * math.sin(math.radians(self.angle))
```

Hình 19: Tính tọa độ điểm cuối của tia Lazer

Kế tiếp ta sử dụng hàm clipline trong pygame để kiểm tra xem một line đã cắt một hình chữ nhật (rectengal) hay chưa (tuy nhiên hình chữ nhật này không bị xoay điều này rất phù hợp bởi vì ở map thì các bức tường là vật thể tĩnh)

```
for i in collision_map:
    line = i.clipline(self.x, self.y, self.end_x, self.end_y)
```

Hình 20: Kiểm tra va chạm của tia Lazer với tường

Nếu kiểm tra va chạm đúng thì ta sẽ vẽ thêm một tia có điểm bắt đầu tại chỗ va chạm và lấy đó làm điểm kết thúc mới cho tia ban đầu, ta cứ lặp lại các bước này sẽ có được một đường ngầm như hình.

```

for i in collision_map:
    line = i.clipline(self.x, self.y, self.end_x, self.end_y)
    if line:
        length = (line[0][0] - self.x) ** 2 + (line[0][1] - self.y) ** 2 # line[0]: collision point
        if 0 < length < line_length:
            line_length = length
            temp_min = line[0]
            if line[0][0] == i.x:
                self.normal = [-1, 0]
            elif abs(line[0][0] - i.x - i.width) <= 1.5:
                self.normal = [1, 0]
            elif line[0][1] == i.y:
                self.normal = [0, -1]
            elif abs(line[0][1] - i.y - i.height) <= 1.5:
                self.normal = [0, 1]
if temp_min:
    self.end_x = temp_min[0]
    self.end_y = temp_min[1]
# self.remaining_length -= math.sqrt((self.end_x - self.x) ** 2 + (self.end_y - self.y) ** 2)
self.remaining_length -= 1

```

Hình 21: Logic hàm vẽ đường ngắm

Đạn ở dạng này sẽ khác đạn ở ba dạng trên đó sẽ là một đường vẽ thẳng dài có màu sắc tùy theo màu chiếc xe tank. Để đạn này có thể hiển thị trên màn hình thì ta sẽ dựa vào các điểm mà đường ngắm đã đi qua mà vẽ lại.

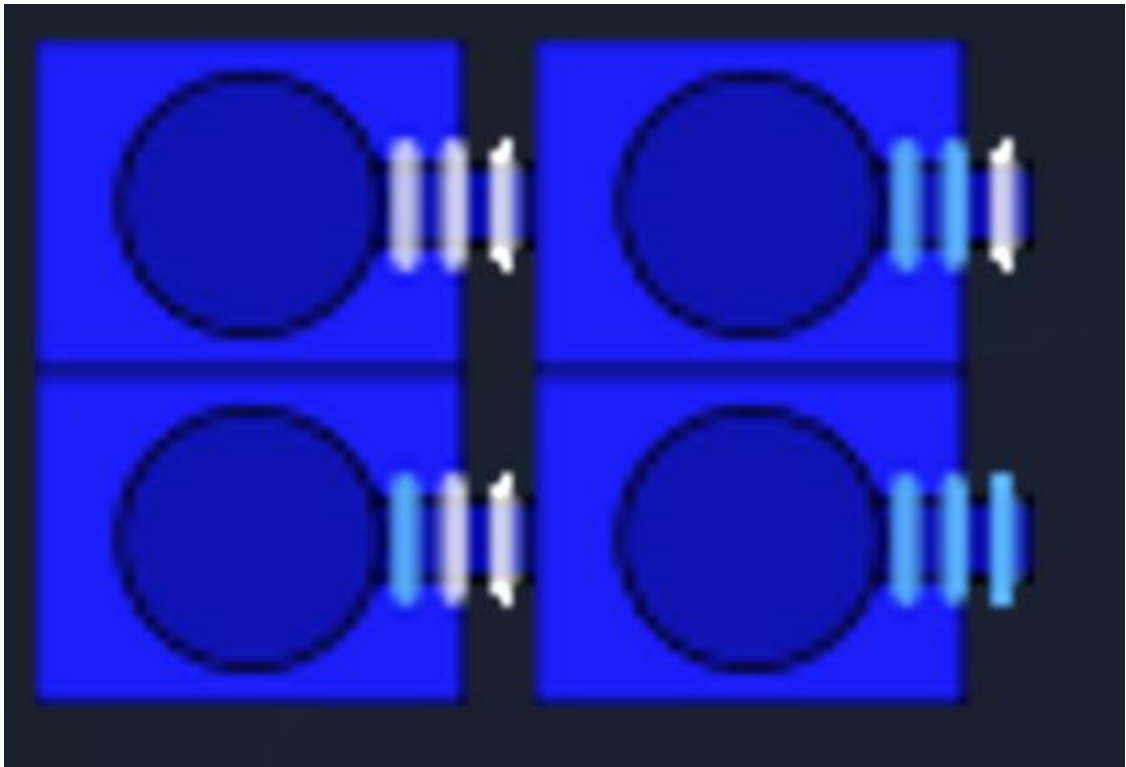
```

elif self.tank.gun_mode == 2:
    laser=Laser(self.tank,self.tank.tank_rect.centerx + 29 * math.cos(math.radians(self.tank.tank_angle)),
                self.tank.tank_rect.centery - 29 * math.sin(math.radians(self.tank.tank_angle)),
                self.tank.tank_angle)
    self.lasers.append(laser)
    self.tank.tank_laser.active = True #Khi bấm shot tức là tia laser đã được active
    self.tank.gun_mode = 1 #lặp tục chuyển gun_mode về mặc định
    #image_path = Setting.asset + Setting.tanks_img[self.tank.id]
    self.tank.update_tank_image(normal_tanks[self.tank.id])
    Sound.laser_sound.play()
    self.last_shoot = current_time

```

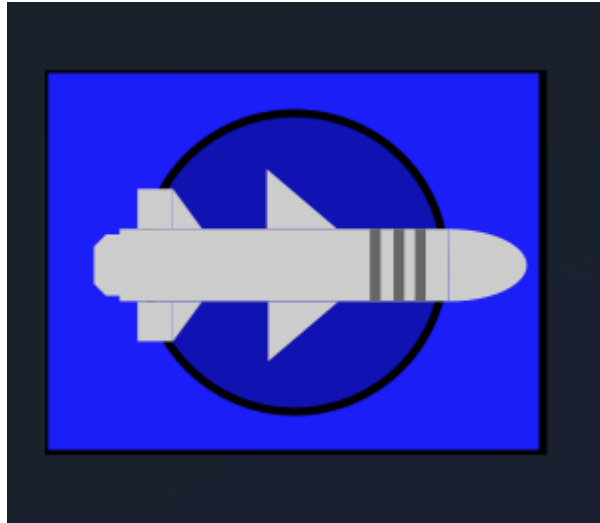
Hình 22: Khởi tạo đường Lazer khi khai hỏa

- Dạng beam



Hình 23: Hình ảnh Tank ở dạng Beam

- Ở dạng này, khi người chơi bấm khai hỏa thì sẽ phải đợi một lúc sau thì mới có thể bắn, trong thời gian đó hình ảnh của xe tank cũng sẽ thay đổi như hình trên.
- Khi đạn đã được khai hỏa người chơi sẽ chỉ có thể di chuyển bằng cách quay xe tank
- Dạng tên lửa



Hình 24: Hình ảnh Tank ở dạng tên lửa

- Ở dạng này người chơi sẽ bắn ra một quả tên lửa, tên lửa này sẽ theo đuổi người chơi gần với nó nhất bất kể đó có là người khai hỏa hay không.
- Để thực tên lửa có thể theo đuổi mục tiêu thì nhóm chúng em đã sử dụng thuật toán A^* để tìm đường đi ngắn nhất tới một xe tank. Nhưng trước đó sẽ phải tính toán vị trí xe tank nào gần nhất với tên lửa.

```

def missile_path(self, map_data): 1 usage (1 dynamic)  datbuudien
    if self.change:
        self.path=pathfinder.a_star(self.start_point,self.end_point,map_data)
        self.change=False
    if self.path:
        if pygame.time.get_ticks()-self.last_sound_time >1000:
            mts.play()
            self.last_sound_time=pygame.time.get_ticks()
        #mts.play()
        self.start_point = pathfinder.Node(self.path[-1][0], self.path[-1][1])
        target_x,target_y=self.path[0][1] * Setting.tile_size, self.path[0][0] * Setting.tile_size
        dx,dy=target_x-self.x,target_y-self.y
        d=math.sqrt(dx*dx+dy*dy)
        if d:
            self.x += self.speed*dx/d
            self.y += self.speed*dy/d
            a=dx/d
            b=dy/d
            angle_radian =math.atan2(a,b)
            self.angle=math.degrees(angle_radian) - 90
            #print(self.angle)
        if abs(dx) < self.speed and abs(dy) < self.speed:
            self.x, self.y = target_x, target_y # Snap to the exact target position
            self.path.pop(0)
        #self.start_point = pathfinder.Node(int(self.y / 16), int(self.x / 16))

```

Hình 25: Logic của thuật toán A* tìm đường đi

Việc áp dụng thuật toán A* để hiệu quả nhất thì nhóm chúng em sẽ sử dụng nó trên ma trận mô phỏng lại map với kích cỡ bé hơn qua việc tính toán và làm tròn vị trí của tên lửa và xe tank trên chính ma trận đó rồi ta sẽ sử dụng thuật toán A* để tìm đường đi. Để cho chuyển động của tên lửa không bị khựng lại thì cứ sau mỗi lần cập nhật, kiểm tra lại path thì điểm khởi đầu sẽ được gán là điểm gần nhất mà tên lửa đã đi. Sau đó tính khoảng cách tới điểm kế tiếp. Sau đó ta tính đc dx, dy là hai giá trị mà tên lửa cần dịch chuyển để có thể đến được vị trí trong điểm đi gần nhất (hay có thể nói là vx và vy của tên lửa). Sau đó ta cần tính v tổng hợp để rồi có thể tìm ra được góc cho tên lửa.

Điều khiển xe tank trong hệ tọa độ của pygame, người chơi có 4 nút để có thể điều khiển xe tank đó là đi thẳng, đi lùi, quay trái, quay phải và khai hỏa. Việc chọn nút điều khiển này sẽ tùy thuộc vào từng người chơi ở phần cài đặt settinggame.

```
def move_tank(tank, speed, window_width, window_height):
    rad_angle = math.radians(tank.tank_angle) # chuyển đổi góc quay sang radian vì hàm trong python làm việc với radian
    tank.dx = speed * math.cos(rad_angle) # đoạn này để vẽ hình nhé :V
    tank.dy = -speed * math.sin(rad_angle) # dấu trừ là bởi khi đi lên trên thì Y giảm

    #sao phải chia ra tank_x tank_y mà không lấy luôn tank_rect.x vì đơn giản một cái nó luôn là kiểu int còn cái
    #điều này giúp chuyển động trở nên mượt mà hơn
    tank.tank_x = max(0, min(tank.tank_x + tank.dx, window_width - tank.tank_width)) # đoạn lấy min là để không bị
    tank.tank_y = max(0, min(tank.tank_y + tank.dy, window_height - tank.tank_height))
    tank.d_angle=0
    tank.check=True

@staticmethod
def rotate_tank(tank, angle): #quay xe tank
    tank.d_angle = angle
    tank.tank_angle = (tank.tank_angle + angle) % 360
    # tank.dx, tank.dy=0,0
```

Hình 26: Hướng di chuyển của Xe Tank

Kiểm tra va chạm của xe tank với những bức tường thì nhóm chúng em đã sử dụng một thư viện có sẵn trong pygame đó là `pygame.mask`, thư viện này sẽ giúp kiểm tra xem các pixel của xe tank đã va chạm với các pixel của map hay chưa (phương pháp này gọi là perfect pixel collision) tuy nhiên hàm kiểm tra này có độ phức tạp lớn nên nhóm em hạn chế dùng phương pháp này đối với sự va chạm của 2 vật thể động

Va chạm giữa xe tank và xe tank cũng có cùng một logic với xe tank với map

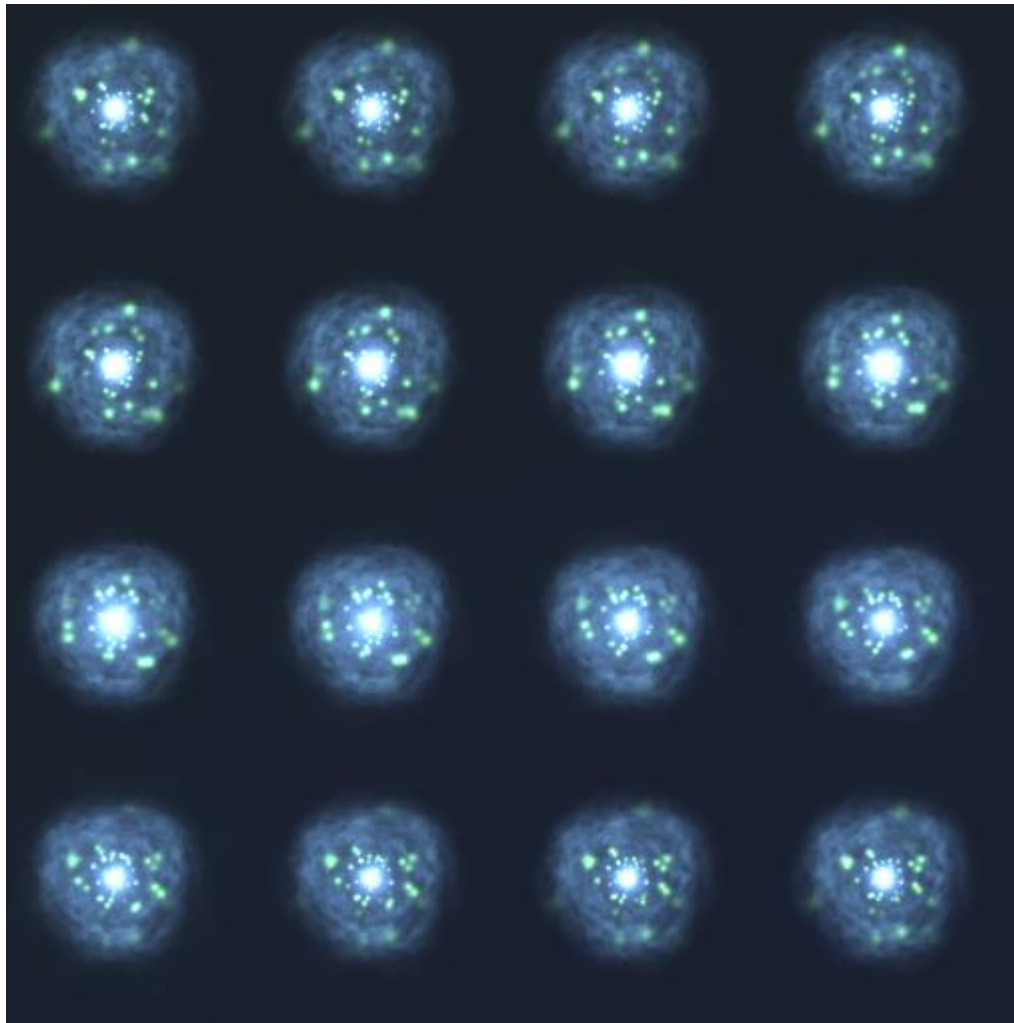
Khi xe tank hết máu hay khai hỏa đều sẽ có những hiệu ứng hình ảnh và âm thanh đi kèm theo



Hình 27: Hiệu ứng xe nổ

Hiệu ứng hình ảnh và âm thanh

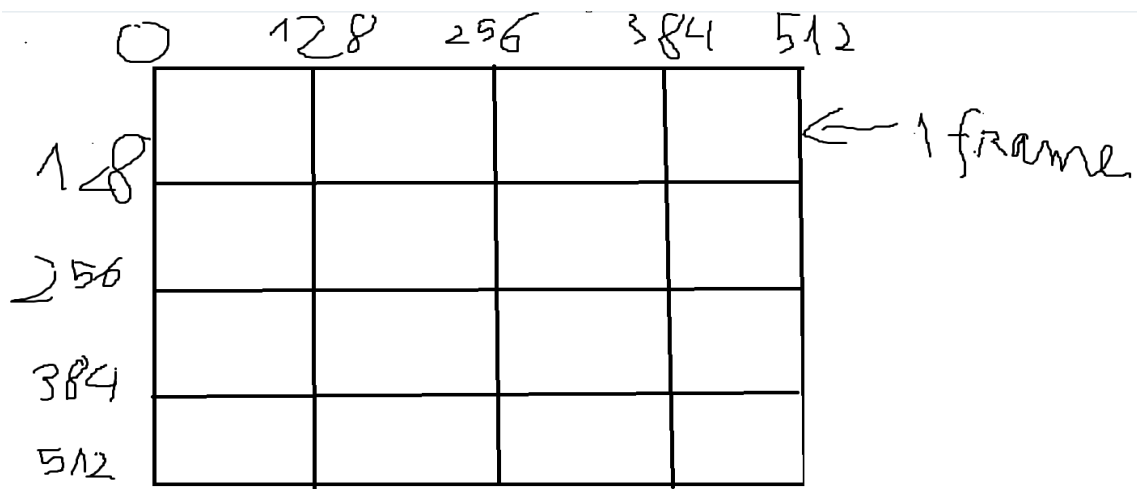
Để có những hình ảnh hiệu ứng chuyển động mượt mà thay vì những hình tĩnh cứng ngắt thì nhóm đã lên trên các trang web tìm kiếm những *spritesheet* ảnh. Sau đó tạo ra các hàm để có thể chia và lưu trữ những bức ảnh trong sheet đấy



Hình 28: *spritesheet*

Ví dụ sheet này có kích cỡ 512x512 và ta thấy có 16 bức ảnh do đó mỗi bức sẽ có kích cỡ 128x128, nhóm em sử dụng 2 vòng for với các chỉ số i,j tương ứng với vị trí góc trái bên cùng để lấy hình ảnh.

```
def calculate_frame(sheet_path, frame_width, frame_height): 6 usages  ▲ datbuudien
    image= pygame.image.load(sheet_path).convert_alpha()
    frames=[]
    total_frame_width =image.get_width() // frame_width
    total_frame_height=image.get_height() // frame_height
    for i in range(total_frame_height):
        for j in range(total_frame_width):
            frame=image.subsurface(j * frame_width, i*frame_height, frame_width,frame_height)
            frames.append(frame)
    return frames
```



Hình 29: Chia Frame hoạt động

Sau khi có được các khung hình thì việc còn lại cần làm chỉ là vẽ các khung hình đó trong một khoảng thời gian nào đó, ví dụ ta có 10 khung hình và hiển thị chúng trong 10 giây thì giây thứ 0 cho đến giây thứ 1 sẽ hiển thị khung hình một, và cứ tiếp tục hoạt động. Và đây cũng chính là cách thức hàm vẽ khung hình của nhóm em hoạt động.

```
def update(self): 1 usage (1 dynamic)  datbuudien
    elapsed_time=pygame.time.get_ticks()-self.start_time #tong thoi gian ma da chay khi dc goi den ham update
    if elapsed_time<self.animation: #cho animation chay khong qua animation giay
        self.current_frame=(elapsed_time //(self.animation//len(self.frames)) ) #chia ra xem la thoi diem nao ung
        if self.current_frame >= len(self.frames):
            self.current_frame= len(self.frames)-1
        self.image=self.frames[self.current_frame] #cap nhap hinh anh
    else:
        self.image=None
```

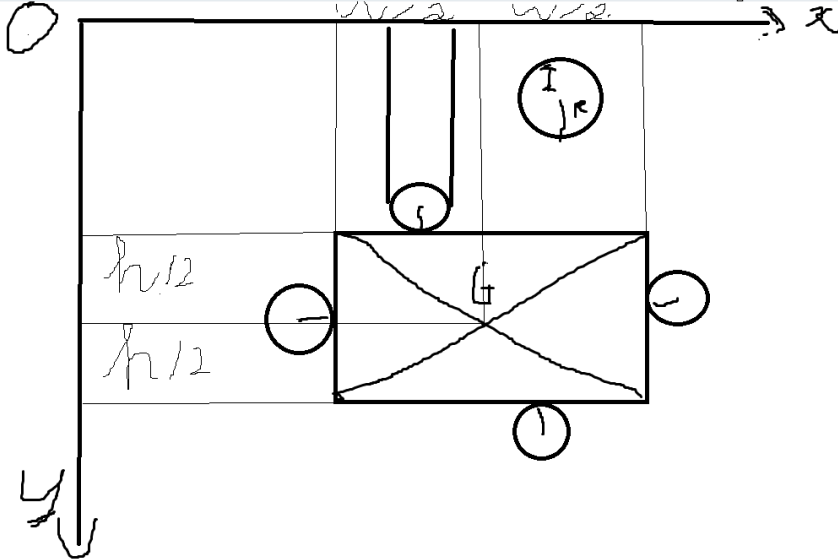
Hình 30: Logic hàm vẽ khung hình

Hiệu ứng âm thanh của nhóm chúng em đều dùng các hàm có sẵn trong pygame đó là mixer, setvolume và play.

Một số logic quan trọng của game

- Như có đề cập ở trên nhóm em hoàn toàn có thể dùng pygame.mask để kiểm tra tất cả sự va chạm giữa các vật thể trong game tuy nhiên hàm pygame.mask có độ phức tạp khá lớn vì hàm đó sẽ kiểm tra từng pixel của những surface, nếu các surface đủ lớn và đủ nhiều dẫn đến chương trình chạy bị chậm, nhóm chúng em cũng muốn tối ưu hóa tựa game này nhiều nhất có thể.
- Trong khi đó pygame cũng không hỗ trợ các hàm kiểm tra va chạm phức tạp nào khác do đó phần này nhóm chúng em đã thêm vào những hàm kiểm tra va chạm giữa các hình, sau đây là một số hàm tiêu biểu:
 - Kiểm tra va chạm giữa đạn tròn và xe tăng:
 - Đầu tiên ta sẽ coi xe tăng như là một hình chữ nhật (điều này có tỉ lệ chính xác gần tuyệt đối), việc kiểm tra va chạm bây giờ sẽ chỉ còn là logic giữa hình chữ nhật và hình tròn.
 - Vì xe tăng có thể quay các hướng khác nhau, do đó ta phải bắt đầu từ trường hợp đơn giản nhất đó là góc quay của xe là 0 độ, gọi tâm đường tròn là I(x,y) tâm hình chữ nhật G(a,b) có chiều ngang =w chiều cao =h (G là giao của 2 đường chéo)

- Ta nhận thấy đường tròn sẽ giao với hình chữ nhật khi và chỉ khi $\max(a-w/2, x-r) \leq \min(a+w/2, x+r)$ and $\max(b-h/2, y-r) \leq \min(b+h/2, y+r)$



Hình 31: Mô phỏng cho thuật toán

- Để tiếp tục cho các trường hợp khi hình chữ nhật xoay ta sẽ giúp thêm phép toán rời trục tọa độ, trục tọa độ của ta bây giờ cũng sẽ bị xoay theo hướng của hình chữ nhật, do đó tọa độ của các điểm cũng cần được tính lại, còn 4 trường hợp đặc biệt đó là khi hình tròn chạm 4 đỉnh của hình chữ nhật thì ta phải dùng đến công thức $(x-a)^2 + (y-b)^2 \leq r^2$ với (a,b) là tọa độ của tâm đường tròn.

```
def check_collision(tank, bullet):
    r=bullet.radius
    x,y=tank.tank_rect.center
    alpha=float(tank.tank_angle)
    a,b=bullet.rect.center
    w,h=tank.tank_width,tank.tank_height
    check_x,check_y= a-x,b-y
    new_x=float(check_x*math.cos(math.radians(alpha))-check_y*math.sin(math.radians(alpha)))
    new_y=float(check_y*math.cos(math.radians(alpha))+check_x*math.sin(math.radians(alpha)))
    tmp = TankLogic.calculate_rotated_corners(tank)
    if max(-w / 2, new_x - r) <= min(w / 2, new_x + r) and max(-h / 2, new_y - r) <= min(h / 2, new_y + r):
        if -w / 2 <= new_x <= w / 2 or -h / 2 <= new_y <= h / 2:
            return True
        else:
            for corner in tmp:
                if TankLogic.check_in_circle(corner, a, b, r):
                    return True
            return False
    else:
        return False
```

Hình 32: Xử lý Logic va chạm giữa hình tròn và hình chữ nhật

- Kiểm tra va chạm giữa đường thẳng và hình chữ nhật bị xoay
 - Ở phần trên có đề cập tới rằng pygame có hỗ trợ hàm clipline để kiểm tra xem một đường thẳng có cắt một hình chữ nhật hay không. Tuy nhiên hàm clipline trong pygame chỉ hỗ trợ kiểm tra một đường thẳng với một hình chữ nhật không bị xoay, mà nếu dùng pygame.mask thì dễ bị dẫn tới việc chương trình bị giảm hiệu năng vì ví dụ nếu như đường thẳng đó có độ dài 9999 tương đương 9999 điểm ảnh thì điều này dẫn

- tới việc kiểm tra rất lâu
- Do đó nhóm em cũng đã tạo ra một hàm kiểm tra va chạm giữa một đường thẳng và một hình chữ nhật
 - Ý tưởng khá đơn giản, đó là ta sẽ tìm ra tọa độ 4 đỉnh của hình chữ nhật sau khi đã bị xoay dùng công thức:
 - 1 $-(x) * \mathbf{cos}(\mathbf{angle}) - (y) * \mathbf{sin}(\mathbf{angle}) + \mathbf{cx}$,
 2. $-(y) * \mathbf{cos}(\mathbf{angle}) + (x) * \mathbf{sin}(\mathbf{angle_rad}) + \mathbf{cy}$
 với \mathbf{cx}, \mathbf{cy} là tâm của hình chữ nhật còn \mathbf{x}, \mathbf{y} là tọa độ của 1 trong 4 đỉnh khi hình chữ nhật ở góc 0 độ
 - Sau đó từ tọa độ 4 đỉnh ta sẽ viết phương trình đường thẳng của 4 cạnh hình chữ nhật và sử dụng thư viện numpy để tính toán xem có nghiệm với đường thẳng ta đang xét hay không, nếu có nghiệm thì phải kiểm tra lại xem điểm đó có thuộc vào cạnh nào của hình chữ nhật.

```
def check_collision_with_laser(tank, laser):
    #phương trình đường thẳng của laser Ax+By+C=0
    #ví dụ end_x=4 end_y =5 x=2 y=3
    A= laser.end_y-laser.y #2
    B= laser.x-laser.end_x # -2
    C=laser.end_x*laser.y-laser.end_y*laser.x #4*3 -5*2 =2 => 2x-2y+2=0

    #4 góc hình chữ nhật
    rects=TankLogic.calculate_rotated_corners(tank)
    for rect in rects:
        if TankLogic.check_point_between(laser.x, laser.y, rect[0], rect[1], laser.end_x, laser.end_y) :
            return rect[0],rect[1]

    res= [TankLogic.line_from_point(rects[0], rects[1]), TankLogic.line_from_point(rects[1], rects[3]),
          TankLogic.line_from_point(rects[2], rects[3]), TankLogic.line_from_point(rects[0], rects[2])]
    #lần lượt là cạnh bên trên, cạnh bên phải, cạnh o dưới, cạnh bên trái
    TankLogic.line_from_point(rects[2], rects[3]), TankLogic.line_from_point(rects[0], rects[2])
    #lần lượt là cạnh bên trên, cạnh bên phải, cạnh o dưới, cạnh bên trái
    try:
        tmp=res[0] #cạnh bên trên
        a = np.array([[A, B], [tmp[0], tmp[1]]])
        b = np.array([-C, -tmp[2]])
        solution = np.linalg.solve(a, b)
        x, y = solution

        if TankLogic.check_point_between(laser.x, laser.y, x, y, laser.end_x, laser.end_y) and TankLogic.check_po
            return x,y
    except np.linalg.LinAlgError:
        pass

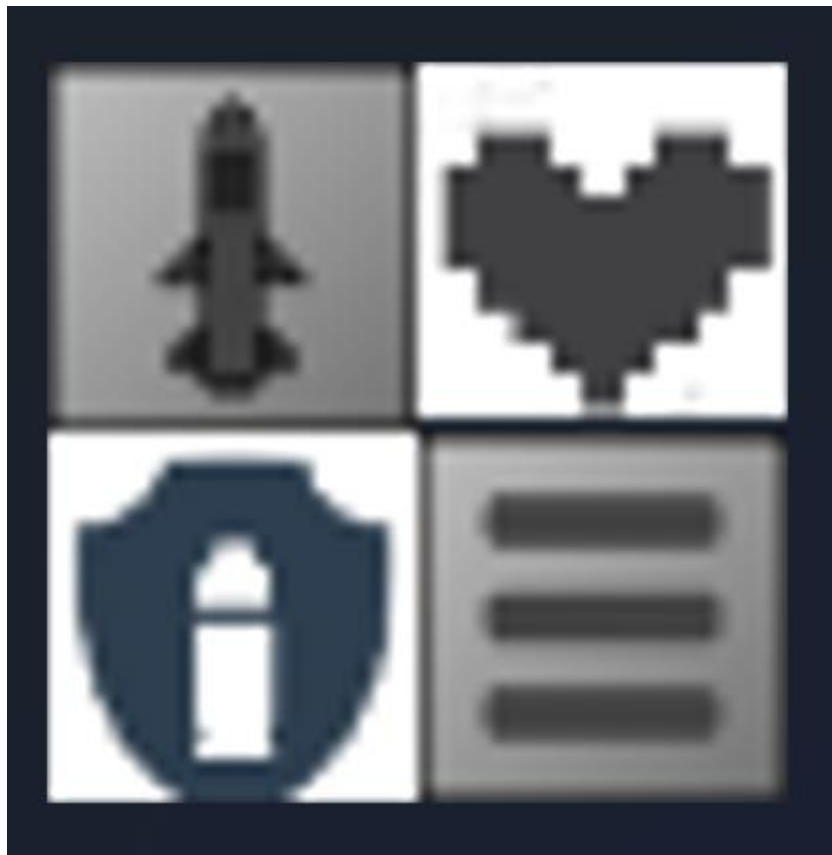
    try:
        tmp=res[1] #cạnh bên phải
```

Hình 33: Hiện thực hóa ý tưởng

Ngoài ra còn có những hàm logic giúp sinh ra thêm những khung hình, bởi vì nhóm em không có một designer thực thụ, nên để thiết kế ra những khung hình là một điều rất khó. Do đó những hiệu ứng như khói thì nhóm em đã chỉ dùng duy nhất một hình ảnh, sau đó dùng hàm làm mờ trong pygame để tạo ra các hiệu ứng của khói

```
def faded_eff(image): 5 usages  ⓘ datbuudien
    alpha = 255
    frames = [image]
    while alpha > 0:
        faded_image = image.copy()
        alpha = max(0, alpha - 51)
        faded_image.set_alpha(alpha)
        frames.append(faded_image)
    return frames
```

Item



Hình 34: Hình ảnh một số Item trong trò chơi

Item sẽ được sinh ra ngẫu nhiên nhờ hàm random trong python, khi random phải kiểm tra xem vị trí đó có phải là vật cản hay không thì mới có thể sinh ra.


```

def random_item(map_data, tile_size):
    x = random.randint(0, 63)
    y = random.randint(0, 42)
    while map_data[int(y)][int(x)] == '1':
        x = random.randint(0, 62)
        y = random.randint(0, 41)
    item_num = random.randint(0, 9)
    items.append((item_num, x * tile_size, y * tile_size))

```

Hình 35: Sinh Item ngẫu nhiên

Mỗi item trong game sẽ đều được gán id, chúng có thể đem cho người chơi những sức mạnh khác nhau, và để kiểm tra va chạm giữa item và xe tăng thì nhóm em đã dùng pygame.mask bởi vì số lượng item ít và kích thước của chúng rất bé.

VI. Demo trò chơi



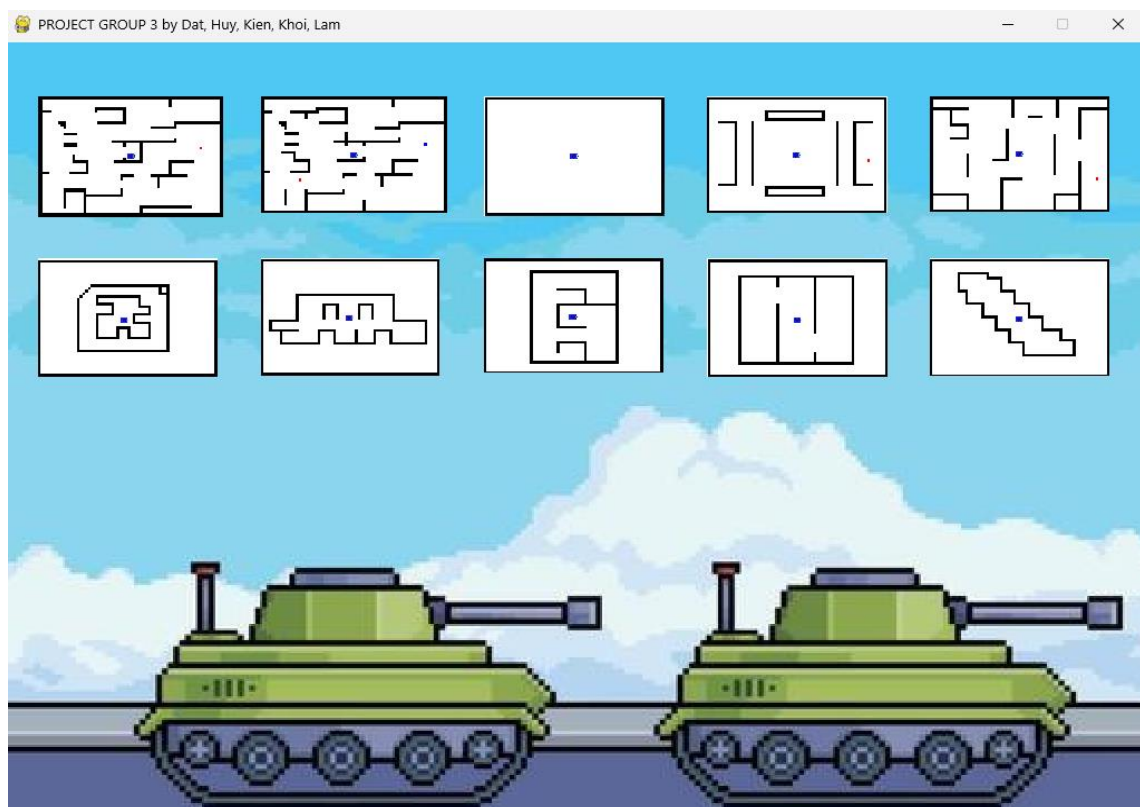
Hình 1: Loading Bar



Hình 2: Màn hình StartGame



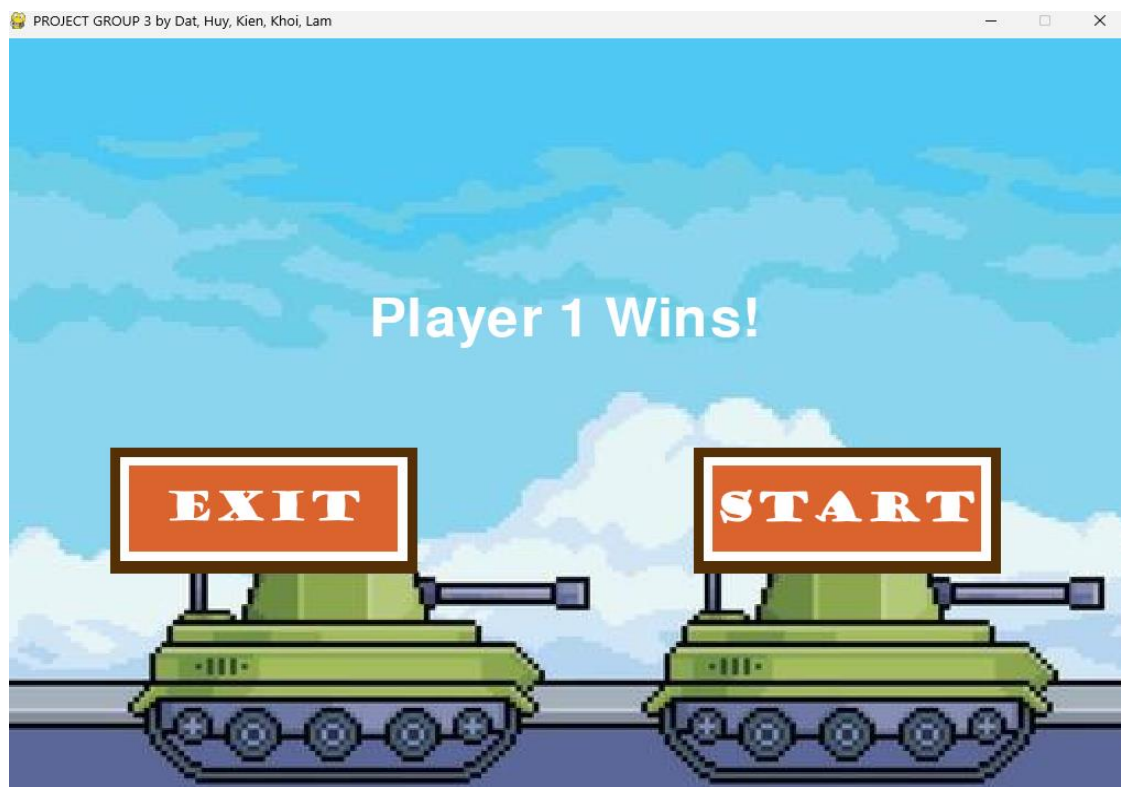
Hình 3: Chọn số lượng người chơi



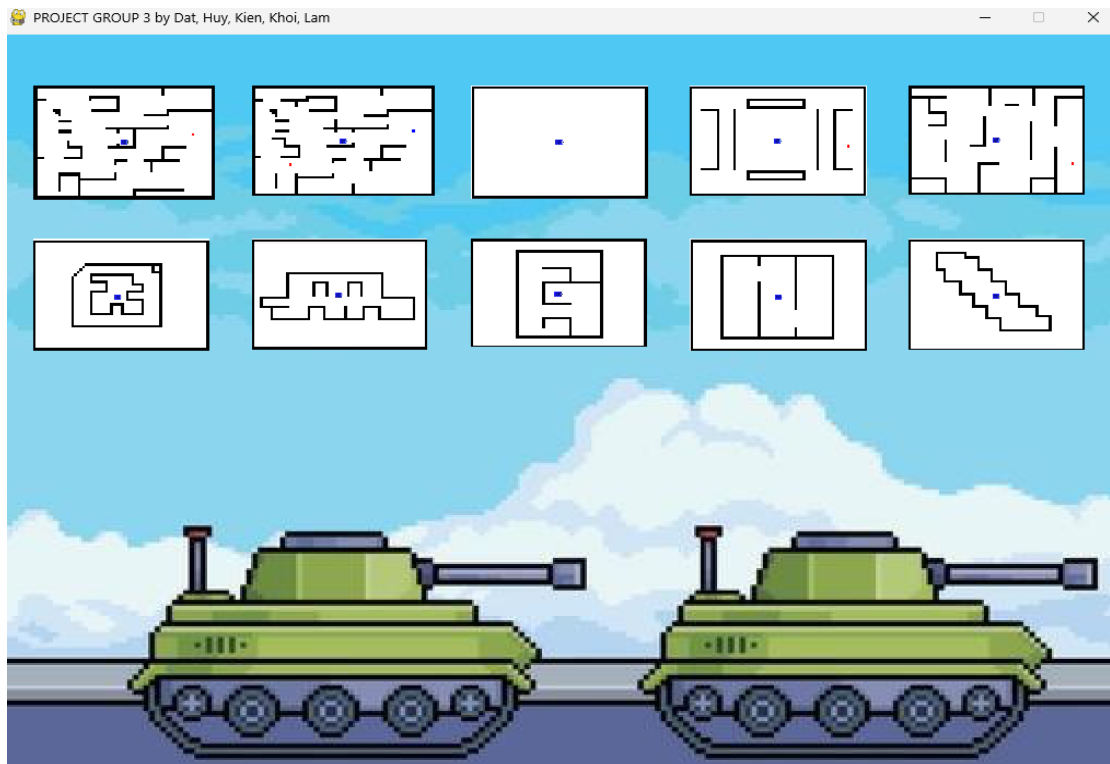
Hình 4: Chọn map chơi



Hình 5: Quá trình thao tác với trò chơi



Hình 6: Kết thúc trò chơi và thông báo người thắng cuộc - Đưa ra lựa chọn Chơi tiếp/Thoát



Hình 7: Quay trở lại quá trình chọn lại map (nếu người chơi chọn chơi tiếp)

VII. Một số tính năng đặc biệt

- Vẽ lại tank sau khi ăn vật phẩm
- Tên lửa bay theo thuật toán tìm đường
- Xử lý logic tất cả các loại đạn và các loại vật phẩm.

VIII. Khó khăn khi thực hiện dự án và giải pháp

- Lần đầu làm dự án còn nhiều bỡ ngỡ
- Cố gắng phát huy khả năng làm việc nhóm. Trong quá trình thực hiện dự án, mọi người cùng hỗ trợ nhau
- Quản lý nhiều trạng thái trong trò chơi: LoadingBar, Starting Game, Game, Ending Game
- Cần phải phân tích logic game 1 cách cẩn thận, bài bản
- Một số logic trong trò chơi còn khá khó, đòi hỏi suy nghĩ tốn nhiều thời gian và công sức ví dụ như:
 - thuật toán tìm đường rồi cài đặt vào tên lửa
 - phản xạ Laser sao cho đúng logic
- Do là 1 game đối kháng chiến thuật theo thời gian thực, yêu cầu đòi hỏi trò chơi phải chạy mượt mà mặc dù trong logic trò chơi bắt buộc cần phải dùng nhiều vòng lặp
- Cải thiện bằng logic và thuật toán
- Viết thêm những file, những chương trình trong Setting giúp chương trình ổn định hơn, có thể cấu hình được giao diện
 - Game engine cổ nên mọi thứ hỗ trợ không được hoàn toàn! Nhiều lúc phải code tay một vài phần

IX. Tài liệu tham khảo

- [1]. <https://www.pygame.org/wiki/tutorials>
- [2]. Pygame tutorial Documentation (Release 2019) - Raphael Holzer
- [3]. Tracey, S. M. (2021). Game development with Python and Pygame. Packt Publishing.
- [4]. GeeksforGeeks. (n.d.). Breadth first search (BFS). Retrieved from <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>
- [5]. GeeksforGeeks. (n.d.). A search algorithm*. Retrieved from <https://www.geeksforgeeks.org/a-search-algorithm/>