# Python Final Project: Tag Game

**By: David**

**Final Project Link:** https://github.com/davidhalim22/Python-Final-Project-Tag-Game.git

## Project Specification

This project is a game of tag in space with the players being a spaceship and all. It allows the player to move the spaceship around and try not to be 'it' or else he/she will lose. The objective of making this game is to make the user experience the joy of playing the game of tag and to relief their stress or make it worse. This project was made by using a bunch of modules such as 'pygame', 'time', 'os', etc and there were classes for making the characters. The comments were perfectly fine and very detailed for explaining about the code. The python code for making the project were put in different python files to better organize the code and I put all the variables that's commonly use in one file so I can use the variable by importing the variable file to the other python files. The game mostly function as like how the user play tag in real life but, if I put more effort and implement cool effects and build maps for the game, it might be cool game. The most challenging part I face while making this project is probably making the enemy follow the player but, the annoying part I face is how I kept realizing ways to minimize the code I write by using classes and organize the code by writing them in different files when I was in the process of making the project which made me have to restart all over again in some part of the project.

## Solution Design

In this project, I used a bunch of different modules especially the 'pygame' module to make the project and the program usually works if when run it, it will open a screen which is the start menu of the game and the program will keep running until you click the exit button and also, the screen keeps updating 60 FPS since it is almost like making an animation except you can actually move the player. There were some problems that I ran to while making the project such as minimizing the code and make it a little organize and to solve that, I usually use classes to make characters and separate the code to different file to make it more tidier and another one of my problems that I ran to was to make the enemy the player or ran away from them, the problem I come up with was

to calculates the distance and direction to the player using the difference in their coordinates. If the enemy is "it," it moves toward the player; otherwise, it moves away from the player. The movement is proportional to the distance and constrained by a fixed speed. The enemy smoothly rotates to face its movement direction by adjusting its angle incrementally toward the target angle.

## Implementation and how it works

A player and an enemy compete in a tag-based challenge in the dynamic Space Tag game, which is implemented with Python and the Pygame library. Its core features include smooth movement of the player and enemy, collision detection using pixel-perfect masks, and real-time game mechanics like timing and scoring. The game architecture is modular, with classes like Player and Enemy encapsulating character behaviors like rotation, movement, and boundary handling; Menu interactions, like starting or ending the game, are managed by the Menu and Box_menu classes, which detect mouse gestures for easy navigation; the game loop coordinates events, updates visuals, and manages state transitions between gameplay, loading screens, and the game-over interface. A timer to enforce a set game duration, distance-based movement logic, and trigonometric computations for smooth rotations are some of the key algorithms. While features like input validation, modularity, and responsive user interface (UI) improve user experience, data structures like classes, masks, and constants guarantee maintainable and effective code. The implementation shows how algorithmic accuracy and visually appealing design can be combined to create a coherent and interactive solution.

# Evidence of Working Program

main.py | game_start.py | characters.py | terrrain_generator.py | variables.py

main.py > ...

```python
from pygame import *
import os
import math
from time import sleep
import characters
from variables import *
import game_start as GS
import terrrain_generator as TG
import time as tm
import random

init()
player_points = 0  # Player's initial score
enemy_points = 0  # Enemy's initial score
num = None  # Random number for determining tag

start_time = tm.time()  # Timer start time
timer_duration = 60  # Duration of the timer in seconds

display.set_caption("Space Tag")  # Set game window title

# Store initial positions of player and enemy
player_start_pos = characters.player_rect  # Initial player position
enemy_start_pos = characters.enemy_rect  # Initial enemy position

is_game_over = False  # Game over flag
running = True  # Main loop flag
play_game_over = True  # Game over sound flag

decider_text = 'You Won'  # Initial end-game text
decider_color = 'blue'  # Initial end-game text color

# Menu class for game UI
class Menu:
    def __init__(self, game_interface, coordinate_mask, size, colors, coordinate_text):
        self.game_interface = game_interface  # Menu text
        self.coordinate_mask = coordinate_mask  # Menu position
        self.size = size  # Menu size
        self.colors = colors  # Menu color
        self.coordinate_text = coordinate_text  # Text position
        self.menu_bar_rect = None  # Menu bar rectangle
        self.menu_bar_mask = None  # Menu bar mask

    def draw_menu(self):
        # Render menu text and draw menu bar
        interface_text = menu_pixel_font.render(self.game_interface, True, self.colors)
        interface_text_rect = interface_text.get_rect(center=(self.coordinate_text))
        menu_bar = Surface(self.size)  # Create menu bar surface
        menu_bar_rect = menu_bar.get_rect(center=(self.coordinate_mask))  # Set menu bar position
```

main.py | game_start.py | characters.py | terrrain_generator.py | variables.py

game_start.py > ...

```python
from pygame import *
from variables import *

# Initialize Pygame
init()

# Frames per second
FPS = 60

# Render the game title text
GAME_TITLE = title_pixel_font.render('Space Tag', True, 'blue')

# Render the "Start" button text in two states (normal and highlighted)
START_GAME = menu_pixel_font.render('Start', True, 'blue')
WHITE_START_GAME = menu_pixel_font.render('Start', True, 'white')

# Render the "Quit" button text in two states (normal and highlighted)
EXIT_GAME = menu_pixel_font.render('Quit', True, 'blue')
WHITE_EXIT_GAME = menu_pixel_font.render('Quit', True, 'white')

# Define the positions for the game title and buttons
game_title_rect = GAME_TITLE.get_rect(center=(WIDTH / 2, HEIGHT / 9))
start_game_rect = START_GAME.get_rect(center=(WIDTH / 2, HEIGHT - 450))
exit_game_rect = EXIT_GAME.get_rect(center=(WIDTH / 2, HEIGHT - 296))

# Create masks for the "Start" button
start_mask = mask.from_surface(START_GAME)
start_image = start_mask.to_surface()

# Create a cursor surface for collision detection
mouse_cursor = Surface((10, 10))
mouse_cursor_rect = mouse_cursor.get_rect()
mouse_cursor.fill('white')
mouse_cursor_mask = mask.from_surface(mouse_cursor)

# Initialize button state variables
start_menu = START_GAME
exit_menu = EXIT_GAME



# Class for creating interactive menu boxes
class Box_menu:
    def __init__(self, size, coordinate, colors):
        self.size = size
        self.coordinate = coordinate
        self.colors = colors
        self.box_mask = None
        self.box_rect = None
```

main.py | game_start.py | characters.py | terrrain_generator.py | variables.py

characters.py > ...

```python
from pygame import *
from variables import *
import terrrain_generator as TG
from math import *

init()

# Load images and scale them to the specified size
PLAYER = image.load(blue_ship).convert_alpha()
PLAYER = transform.scale(PLAYER, (character_width, character_height))
PMOVE = image.load(blue_ship_move).convert_alpha()
PMOVE = transform.scale(PMOVE, (character_width, character_height))

ENEMY = image.load(red_ship).convert_alpha()
ENEMY = transform.scale(ENEMY, (character_width, character_height))
EMOVE = image.load(red_ship_move).convert_alpha()
EMOVE = transform.scale(EMOVE, (character_width, character_height))

# Speed and initial positions
player_speed = 5  # Speed at which the player moves
enemy_push_speed = 10  # Speed at which the enemy pushes towards the player

player_rect = PLAYER.get_rect(center=(200, HEIGHT / 2))  # Set initial position of player
player_x, player_y = player_rect.x, player_rect.y  # Player's coordinates

player_angle = 0  # Player's initial rotation angle
target_angle = 0  # Target rotation angle for smooth rotation

enemy_angle = 0  # Enemy's initial rotation angle

enemy_rect = ENEMY.get_rect(center=(WIDTH - 200, HEIGHT / 2))  # Set initial position of enemy
enemy_x, enemy_y = enemy_rect.x, enemy_rect.y  # Enemy's coordinates

# Create masks for collision detection
player_mask = mask.from_surface(PLAYER)
player_image = player_mask.to_surface()

enemy_mask = mask.from_surface(ENEMY)
enemy_image = enemy_mask.to_surface()

# Set initial character and movement images
player_char = PLAYER
enemy_char = ENEMY
player_move = PMOVE
enemy_move = EMOVE

# Render tags for player and enemy
player_tag = font1.render('Player', True, 'white')
enemy_tag = font1.render('Enemy', True, 'red')
```

main.py | game_start.py | characters.py | terrrain_generator.py | variables.py

terrain_generator.py > ...

```python
from pygame import *
from variables import *

init()

# Class for creating and drawing borders
class Border:
    def __init__(self, size, coordinate, colors):
        self.size = size  # Size of the border (width, height)
        self.coordinate = coordinate  # Position of the border (center of the rectangle)
        self.colors = colors  # Color of the border
        self.border_mask = None  # Initialize mask for border collision detection (if needed)
        self.border_rect = None  # Rectangle representation of the border

    # Method to draw the border
    def draw_border(self, is_border_draw=True):
        border = Surface(self.size)  # Create a surface for the border with the given size
        border_rect = border.get_rect(center=(self.coordinate))  # Set the position of the border
        border.fill(self.colors)  # Fill the border surface with the specified color
        border_mask = mask.from_surface(border)  # Create a mask from the surface (used for collision detection)
        border_image = border_mask.to_surface()  # Convert the mask back into a surface
        self.border_mask = border_mask  # Store the mask for future use
        self.border_rect = border_rect  # Store the rectangle for future use

        if is_border_draw:  # If the flag is set to True, draw the border to the screen
            WIN.blit(border_image, border_rect)  # Blit (draw) the border image on the window


# Create instances of the Border class for each border (top, bottom, left, right)
border_top = Border((1085, 5), (WIDTH / 2, 50), 'red')  # Top border
border_bottom = Border((1085, 5), (WIDTH / 2, HEIGHT - 50), 'red')  # Bottom border
border_left = Border((5, 615), (100, HEIGHT / 2), 'red')  # Left border
border_right = Border((5, 615), (WIDTH - 100, HEIGHT / 2), 'red')  # Right border


# Function to draw the entire terrain (all borders)
def terrain():
    border_top.draw_border()  # Draw top border
    border_bottom.draw_border()  # Draw bottom border
    border_left.draw_border()  # Draw left border
    border_right.draw_border()  # Draw right border
    display.update()  # Update the display to show the drawn borders
```

main.py | game_start.py | characters.py | terrain_generator.py | variables.py

variables.py > ...

```python
from pygame import *  # Importing all pygame modules
import os  # Importing os module for file path handling

init()  # Initialize Pygame

# Get screen information (maximum screen size)
info = display.Info()
MAX_WIDTH, MAX_HEIGHT = info.current_w, info.current_h  # Maximum screen width and height
WIDTH, HEIGHT = 1280, 720  # Set custom screen width and height
WIN = display.set_mode((WIDTH, HEIGHT), RESIZABLE)  # Create a Pygame window with the specified dimensions

# Path to background music
BACKGROUND_MUSIC = os.path.join('D:\Documents\Python Final Project\Assets\Music', 'BG SOUND 2.wav')

# Directory where game assets (images, sounds) are stored
ASSETS_DIR = r'D:\Documents\Python Final Project\Assets'

# Load background images for the game
BACKGROUND_IMAGE = image.load(os.path.join(ASSETS_DIR, 'background', 'space.jpg'))  # Main background
BACKGROUND_IMAGE_RECT = BACKGROUND_IMAGE.get_rect(center=(WIDTH / 2, HEIGHT / 2))  # Get rect for background position
BACKGROUND_START = image.load(os.path.join(ASSETS_DIR, 'background', 'background start.jpg'))  # Start screen background
BACKGROUND_START_RECT = BACKGROUND_START.get_rect(center=(WIDTH / 2, HEIGHT / 2))  # Get rect for start background position

# Sound effect file paths
CLICK_SOUND = os.path.join(ASSETS_DIR, 'Music', 'click_effect.mp3')  # Sound when clicking
GAME_OVER_SFX = os.path.join(ASSETS_DIR, 'Music', 'game-over-arcade.mp3')  # Game over sound effect
KNOCKBACK_SOUND = os.path.join(ASSETS_DIR, 'Music', 'hitting.mp3')  # Sound for player knockback

# Rocket sound effect file path
ROCKET_FX = os.path.join(ASSETS_DIR, 'Music', 'rocket-launch.mp3')

# Set Frames Per Second (FPS) for the game
FPS = 60

# Initialize variables for handling input (keyboard and mouse)
keys_pressed = key.get_pressed()  # Get the current state of all keys
mouse_pos = mouse.get_pos()  # Get the current position of the mouse
mouse_pressed = mouse.get_pressed()  # Get the current state of mouse buttons

# File paths for character images
blue_ship = os.path.join('D:\Documents\Python Final Project\Assets\Characters', 'player1.png')  # Player ship image
blue_ship_move = os.path.join('D:\Documents\Python Final Project\Assets\Characters', 'player1_move.png')  # Player ship moving image

red_ship = os.path.join('D:\Documents\Python Final Project\Assets\Characters', 'player2.png')  # Enemy ship image
red_ship_move = os.path.join('D:\Documents\Python Final Project\Assets\Characters', 'player2_move.png')  # Enemy ship moving image

# Font setup for text rendering
font1 = font.Font(None, 24)  # Default font with size 24
title_pixel_font = font.Font('D:\Documents\Python Final Project\Assets\public-pixel-font\PublicPixel-rv0pA.ttf', 72)  # Title font (pixel style)
```