Odd Semester (2022)

BINUS UNIVERSITY

BINUS INTERNATIONAL

Algorithm and Programming

Final Project Report

Student Information:

Surname: Kusnadi **Given Name:** Clarissa Audrey Fabiola Kusnadi **Student ID:** 2602118490

Course Code : COMP6047001 **Course Name :** Algorithm and Programming

Class : L1AC **Lecturer :** Jude Joseph Lamug Martinez, MCS

Type of Assignment: Final Project Report

Submission Pattern

Due Date **:** 15 January 2023 **Submission Date** **:** 15 January 2023

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating, and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student:

Clarissa Audrey Fabiola Kusnadi

Table of Contents

A. Introduction

1. Background

Students are expected and encouraged to create and design a program or game that is beyond the topic of the Algorithm and Programming course, by using all the lessons that we have learned about the Python programming language.

After some ideas consultation and research, I finally decided to make a computer vision game named ‘Catch the Penguin’ by using libraries of programming functions that have never been covered during this semester of Algorithm and Programming, which are OpenCV (Open Source Computer Vision) and MediaPipe.

2. Problem Identification

Due to the rapid changes in technology and the growing use of computers, there is a desire and demand for amiable and simple interfaces to communicate with machines. Therefore, the phrase “human-machine interaction” refers to communication and interaction between a human and a machine via a user interface. A more effective method of communicating with a computer through more natural and intuitive behavior is to use hand gesture recognition as a real-time input. Gaming with computer attachments in the past, such as a keyboard, joystick, mouse, etc., has always been a repetitive activity for players. It becomes repetitive work when used for extended periods, making players weary. Thus, a fresh approach to playing games is required to make the gaming world more dynamic, alluring, and interactive. A revolution is required to create new advances and give gamers new opportunities to play games in innovative ways.

In my game, I am concentrating on two main things. A) To make the relationship and interaction between people and robots seem natural, mimicking human interactions. B) To remove the necessity for any input space, i.e., such as the input devices, thus making its input data by just making bare hand gestures.

B. Project Specification

1. Game Name

- Catch the Penguin

2. Game Concept

- 'Catch the Penguin' is a real-time game where the player uses their hand to capture as many penguins as you can while avoiding bombs on the screen, which is done by tracking the player's hand gesture using a webcam.
- The game consists of objects such as penguins and bombs running around in the center of the window's screen. The player earns a point for each penguin they catch, but they lose a point if they catch a bomb. The game has a time limit, and the player needs to gain a score as high as possible before the timer runs out.
- To increase the game difficulty, I incorporated some probability logic to the amount of objects spawn over time along with randomizing the running velocity, object sizes, and spawn positions to give unpredictability to the game.

3. Game Flow Summary

- The player has to capture as many penguins as possible before the timer runs out by using a hand to control the game by moving it in front of a webcam, opening and then closing the hand to capture the penguin. If the player captures a bomb, they lose 1 point, while capturing a penguin gains 1 point. The game becomes progressively harder as time goes on, as the probability of a bomb spawning increases. The player has to catch as many penguins as possible to get points while avoiding bombs before the timer runs out.

4. Game Objective

- To score as many points as possible by catching penguins that are running around on the screen while avoiding bombs that also appear on the screen using the player's hand within a limited time of the game.

5. **Game Display**
    - 2D game that utilizes bits of pixel art as its graphic style. The background of the game is a snow landscape, with the foreground containing various elements that are animated with running animation such as penguins and bombs that the player must interact with. The player's hand is also rendered on the screen, allowing them to physically interact with the game using hand tracking technology. When opened, the game displays an opened hand object image, and when closed, it displays a closed hand object image. The game also features a simple user interface that displays the player's score and remaining time, as well as buttons for starting the game, going back to the game's menu, and quitting the game.
6. **Game Mechanics**
    - The player is able to control a hand on the game screen using their own hand in front of a webcam, opening and closing it to capture the penguin while avoiding bombs.
7. **Game Physics**
    - For the objects, the penguins and bombs will spawn at random positions and move using randomized velocity to either four directions: right, left, up, or down of the screen. The player controls a hand that can be moved by tracking their own hand moving using a webcam. The hand object in the game will move along with the player's real hand movement. Moreover, the game physics include collision detection between the player's hand and the objects that is done by comparing the rectangles of the player's hand and the objects. The game detects the collision and performs a specific action based on the type of object. For example, when the player's hand closes and overlaps the penguin, the penguin is captured and the player earns a point. On the other hand, if the player's hand closes and overlaps with a bomb, the player loses a point.
8. **Game Input**
    - A player's hand - to catch penguins and avoid bombs.
    - Webcam - to capture the player's hand.

- Mouse left button/trackpad - to click either the start, quit, or back button and can also close the game window by clicking the X button on the top left corner of the game's window.
- Keyboard escape key - close the game's window.

9. Game Output

- Player hand image (open and close) - can move and open and close based on a player's real-time hand movement captured by the webcam (Hand folder containing hand1.png and hand2.png)
- Penguin image - will randomly spawn and move using randomized velocity to either four directions: right, left, up, or down of the screen (Penguin folder containing 1.png, 2.png, 3.png, 4.png, 5.png, and 6.png)
- Bomb image - will randomly spawn and move using randomized velocity to either four directions: right, left, up, or down of the screen (Bomb folder containing 1.png, 2.png, 3.png, 4.png, 5.png, and 6.png)
- Background image (background.jpg)
- Player's score
- Remaining time
- Background music (Komiku_Glouglou.mp3)
- Button click sound effect (click.mp3)
- Bomb explosion sound effect (explosion.wav)
- Penguin capture sound effect (grab.mp3)

10. Game Libraries/Modules

- Pygame - a set of Python modules designed for writing video games. Used in the game to create the window, handling user input, drawing images on the screen, and playing sounds.
- OpenCv - a library of programming functions mainly aimed at real-time computer vision. Used in the game for capturing and processing webcam frames to track the player's hand in real-time.
- MediaPipe - an open-source framework developed by Google for building cross-platform multimodal applied ML (Machine Learning) pipelines. Used in the game for hand detection and tracking, for drawing collision

boxes around the detected hand, and for determining the position of the hand in the webcam frame.

- time - a built-in module that provides various functions to work with time-related operations. Used in the game to keep track of the time left in the game, and to control the spawning of objects on the screen.
- random - a built-in library that allows the generating of random numbers, select random elements from a list, and perform other random operations. Used in the game to randomly select what objects are to be spawned, and to randomly adjust the size and speed of the objects.
- sys - to properly terminate the game when the user wants to exit.

11. Game Files

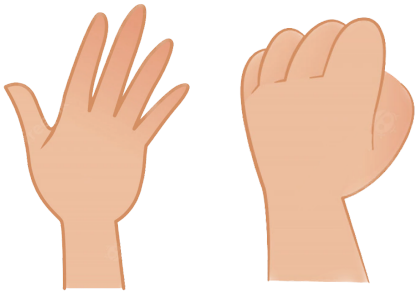
- 'Documents' folder - contains the report pdf file, screenshots of the game, and diagram images for this game.
- 'Assets' folder - contains all fonts, images, and sounds used in the game.
- background.py - contains the class Background in charge of displaying the background of the game on the screen.
- bomb.py - contains the class Bomb in charge of the bomb object.
- constant.py - contains a set of constant variables that are used throughout the game
- game.py - contains the class Game in charge of creating the logic behind the game.
- hand_tracking.py - contains the class HandTracking in charge of tracking and detecting the position of a hand using the MediaPipe library.
- hand.py - contains the class Hand in charge of the hand object that represents the player's hand in the game.
- main.py - contains the main file that runs the game.
- menu.py - contains the class Menu in charge of creating the menu of the game.
- penguin.py - contains the class Penguin in charge of the penguin object.
- ui.py - contains the function for the user interface.

12. Game Images

➢ All of the images were not made by me.

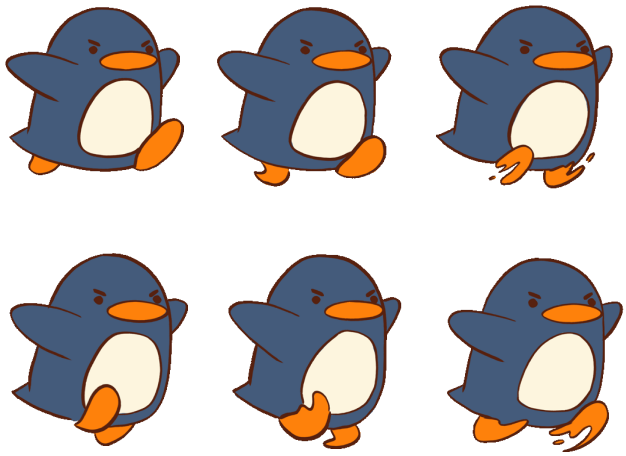
- Background image (https://www.freepik.com/free-photo/abstract-background-light-steel-blue-wallpaper-image_15554723.htm)

- Player hand image (https://www.istockphoto.com/id/vektor/gerakan-tangan-set-ilustrasi-vektor-datar-gm1196938818-341573454)



- Bomb image
  (https://media.giphy.com/media/Tcya3RuTKfPil2rTml/giphy.gif)

- Penguin image
  (https://tenor.com/bc9mr.gif)



13. Game Sounds

- Background sound effect
  (https://freemusicarchive.org/music/Komiku/Captain_Glouglous_Incredible_Week_Soundtrack)
- Button click sound effect
  (https://www.youtube.com/watch?v=OClCzFJUVqM)
- Bomb explosion sound effect
  (https://pixabay.com/sound-effects/search/explosion/)
- Penguin capture sound effect

(https://www.youtube.com/watch?v=up5J30atI_Q)

14. Game Fonts

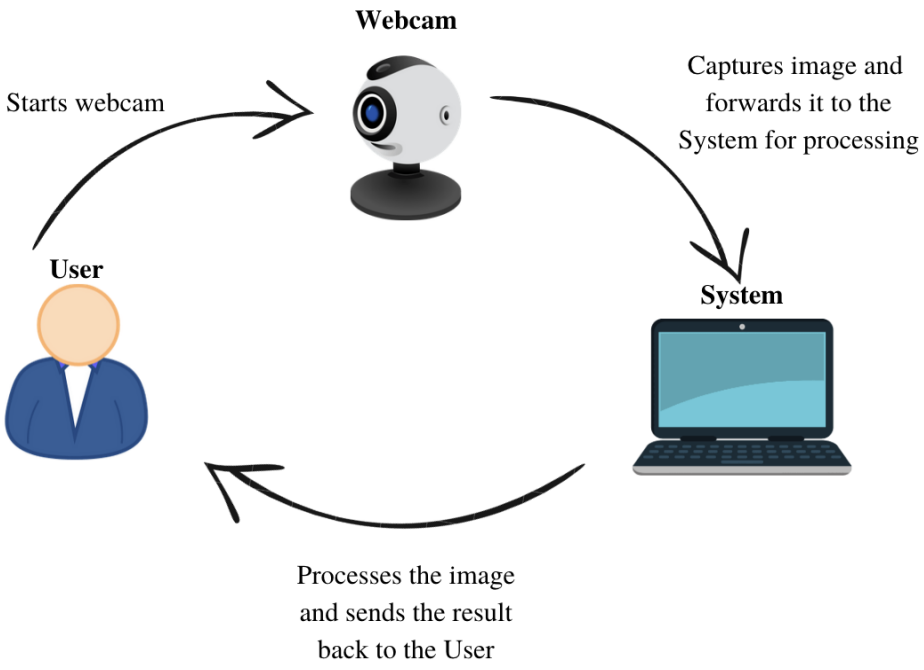
- 04B_30 font is used for the whole text in the game

(https://www.dafont.com/04b-30.font)

C. Solution Design

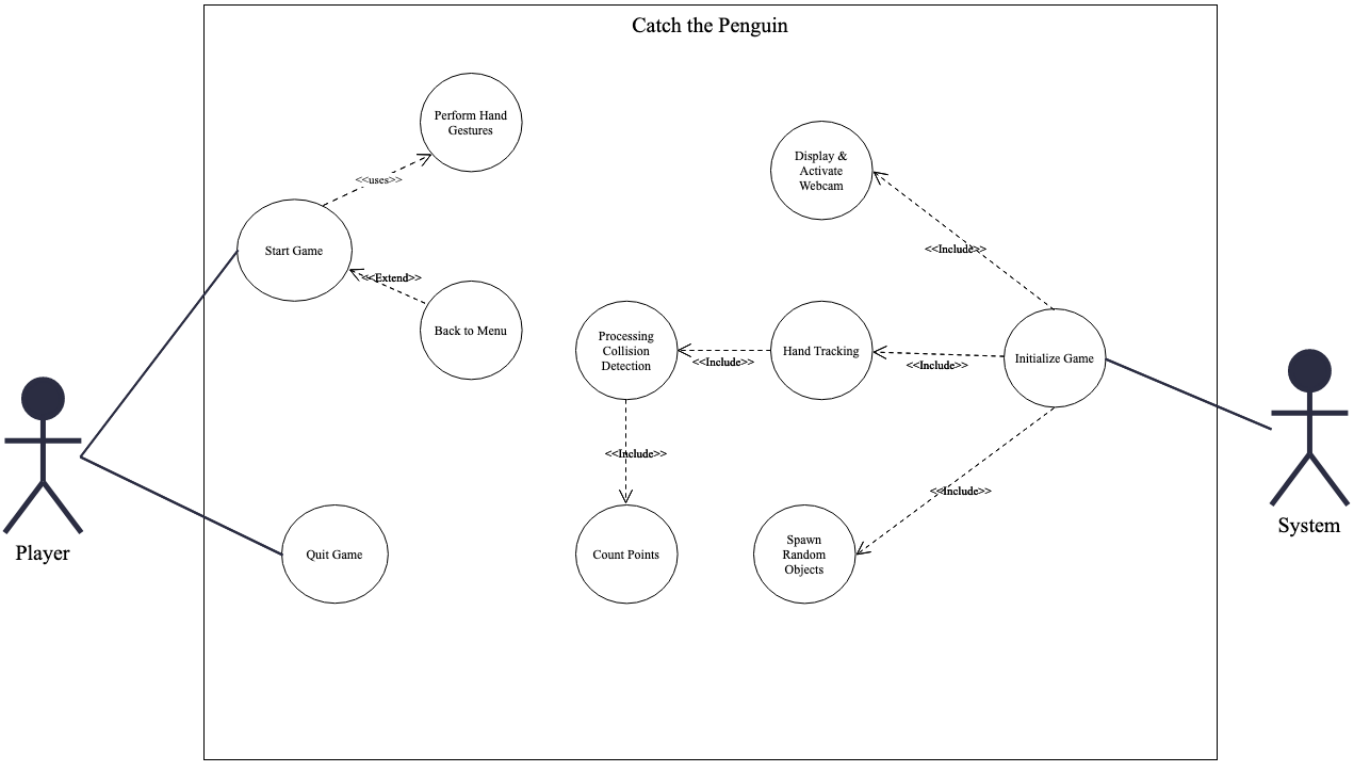
1. System Architecture

The following is the basic idea of the system's structure:



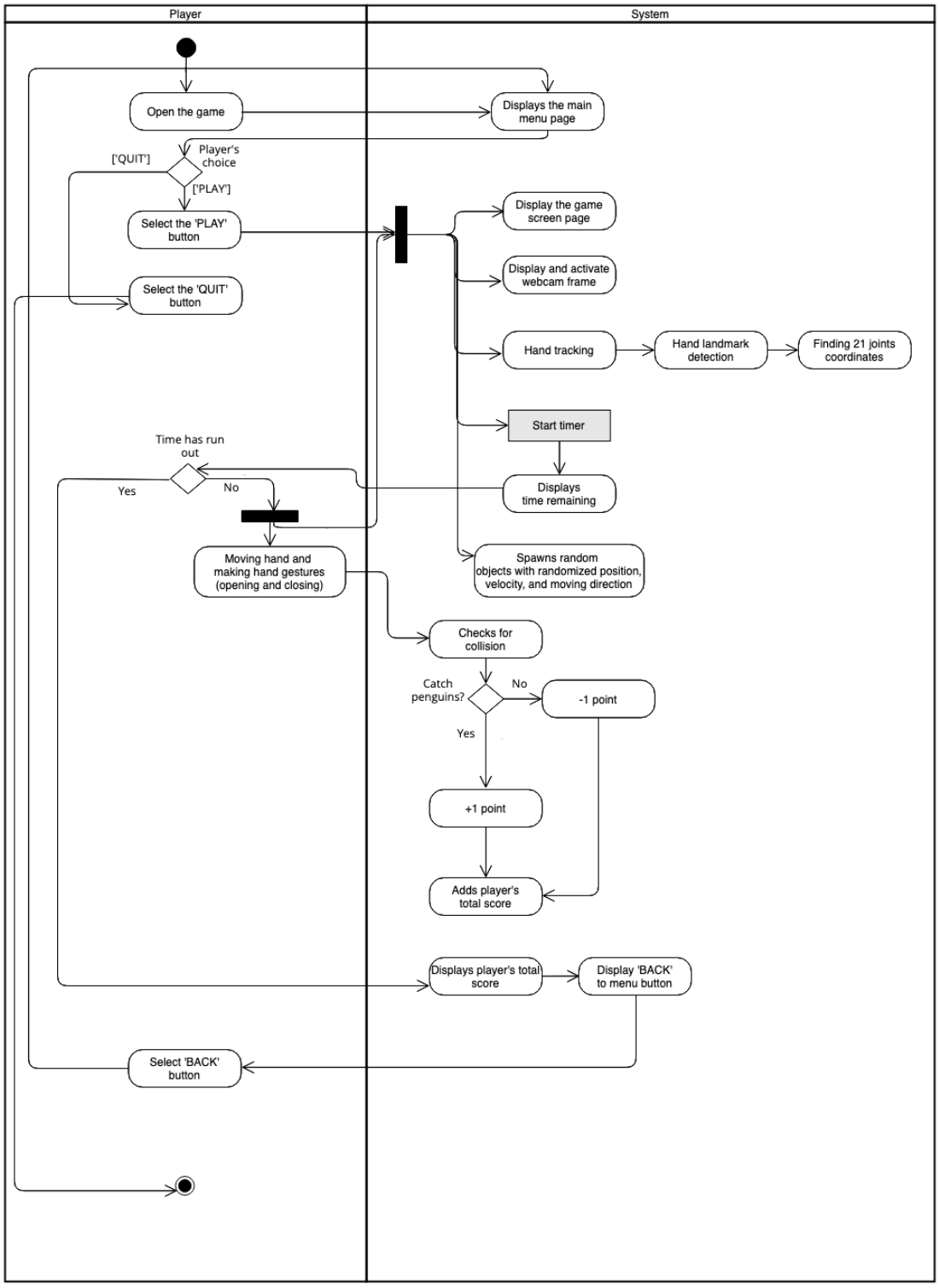
2. Use Case Diagram

The following is an use case diagram that summarizes the action of a system and a player within the game:



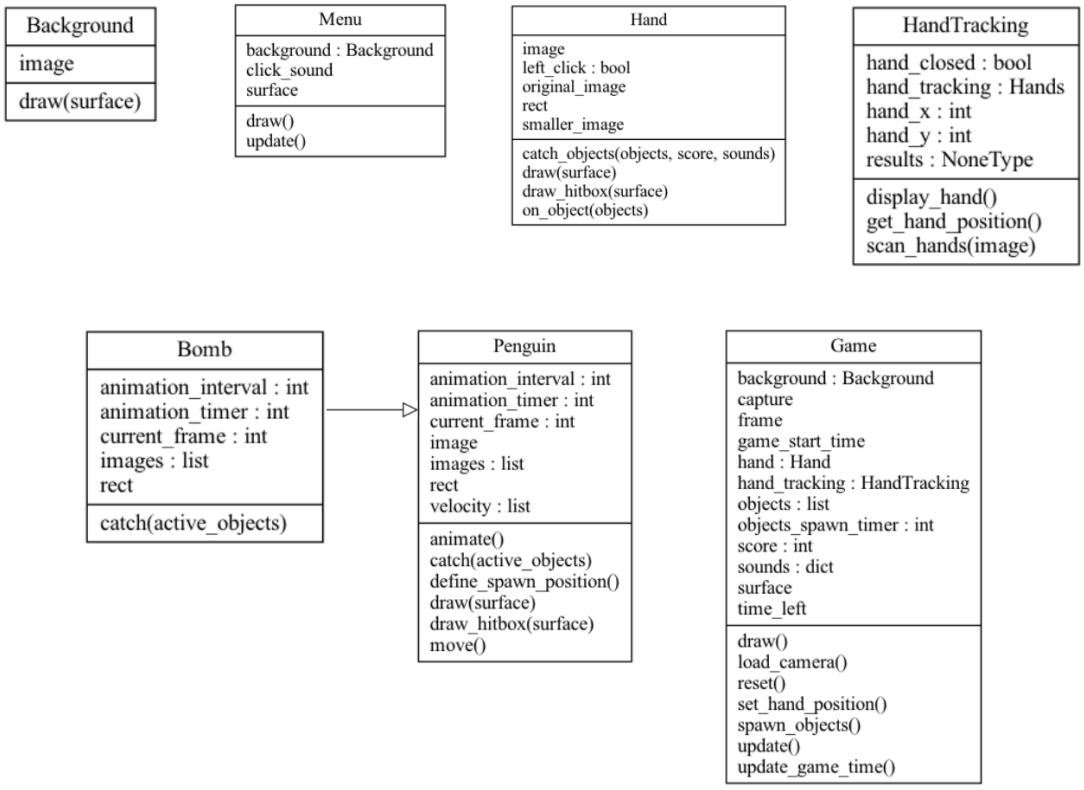
3. Activity Diagram

The following is an activity diagram that denotes an action on an activity:



4. Class Diagram

The following is a class diagram that describes the structure and relationships between the classes existing in the game:



D. Essential Algorithms

1. constant.py

- This file contains a set of constant variables that are used throughout the game.

```
import pygame
```

- Imports pygame library.

```
# Window
WINDOW_NAME = "Catch the Penguin"
GAME_TITLE = WINDOW_NAME
SCREEN_WIDTH, SCREEN_HEIGHT = 1100, 600
```

- The ‘Window’ section defines the name of the game window, the title of the game, and the dimensions of the game screen.

```
# FPS
FPS = 90
DRAW_FPS = True
```

- The ‘FPS’ section defines the frame rate of the game and whether the frame rate should be displayed on the screen.

```
# Animation
ANIMATION_SPEED = 0.08 # The frame of the objects will change every 0.08
seconds
```

- The ‘Animation’ section defines the speed at which the frames of the object in the game will change.

```
# Penguin
PENGUIN_SIZE = (60, 60)
PENGUIN_SIZE_RANDOMIZE = (1.2, 2) # For each new penguin, it will
multiply the size with an random value between 1.2 and 2
PENGUIN_MOVE_SPEED = {"min": 2, "max": 5}
PENGUIN_SPAWN_TIME = 1
```

- The ‘Penguin’ section defines the size of the penguins, the range of random size multipliers that can be applied to newly spawned penguins, the movement speed of the penguins, and the time between each penguin spawn.

```
# Collision Box
DRAW_HITBOX = False
```

- The ‘Collision Box’ section defines whether the collision boxes of the objects in the game should be displayed on the screen. It is set to ‘False’, which means that it is not displayed on the screen.

```
# Bomb
BOMB_SIZE = (100, 80)
BOMB_SIZE_RANDOMIZE = (1, 1.5)
BOMB_PENALITY = 1 # Will remove X of the score of the player (if capture
a bomb)
```

- The ‘Bomb’ section defines the size of the bombs, the range of random size multipliers that can be applied to newly spawned bombs, and the penalty for capturing a bomb.

```
# Hand
HAND_SIZE = 200
HAND_HITBOX_SIZE = (60, 80)
```

- The ‘Hand’ section defines the size of the hand object image and the size of the collision box for the hand.

```
# Game
GAME_DURATION = 60 #The game will last 60s
```

- The ‘Game’ section defines the duration of the game.

```
# UI
```

```
pygame.font.init() #Initializes the pygame font module so that we can use
it to render text
FONTS = {}
FONTS["medium"] = pygame.font.Font("Assets/04B_30__.ttf", 35)
FONTS["big"] = pygame.font.Font("Assets/04B_30__.ttf", 68)
```

- The 'UI' section initializes the pygame font module and defines a dictionary of font styles that can be used to render text on the screen.

```
# Button Sizes
BUTTONS_SIZES = (240, 90)
```

- The 'Button Sizes' section defines the size of buttons in the game.

```
# Colors
COLORS = {"title": (38, 61, 39),
          "score": (38, 61, 39),
          "timer": (38, 61, 39),
          "buttons": {"default": (6, 67, 200), "hover":  (87, 99, 255),
"text": (255, 255, 255), "shadow": (46, 54, 163)}}
          #Hover is the color when the mouse if hovering over the button
```

- The 'Colors' section defines the colors used for various elements of the game, such as title, score, timer, and buttons.

```
# Sounds
MUSIC_VOLUME = 0.16
SOUNDS_VOLUME = 1
```

- The 'Sounds' section defines the volume levels for the background music and sound effects in the game.

2. background.py

- The class 'Background' is used to create and display the background of the game on the screen.

```
import pygame
from constant import *
```

- Imports pygame library and constant module.

```
class Background:
     def __init__(self):
        self.image = pygame.image.load("Assets/background.jpg").convert()
#Load the image file and convert() is used to improve the game's speed
        self.image = pygame.transform.smoothscale(self.image,
(SCREEN_WIDTH, SCREEN_HEIGHT)) #Scaling the image to fit the window's
size
```

- The '__init()__' function is called when a new instance of the class is created. It loads the image file "Assets/background.jpg" using the pygame.image.load() function, and converts it for faster rendering using the convert() function. It also scales the image to fit the window's size

using the pygame.transform.smoothscale() function, using the SCREEN_WIDTH and SCREEN_HEIGHT constants as the screen size imported from the constant.py file.

```
   # Drawing the background image onto the window's screen, game evolves
here inside
   def draw(self, surface):
       surface.blit(self.image, (0, 0)) #blit function is used to draw
one image onto another
```

- The 'draw()' function takes in a surface parameter and is called to draw the background image onto the game window's surface. It uses the blit() function to draw the background image at the coordinates (0,0), which is the top-left corner of the surface.

3. **penguin.py**
   - The class 'Penguin' is used to create and manage the behavior and appearance of the penguin objects that appear in the game.

```
import pygame
import random
from constant import *
```

- Imports pygame library, random library, and constant module.

```
class Penguin:
   def __init__(self):
       # Generate random size for the penguin
       random_size_value = random.uniform(PENGUIN_SIZE_RANDOMIZE[0],
PENGUIN_SIZE_RANDOMIZE[1])
       size = (int(PENGUIN_SIZE[0] * random_size_value),
int(PENGUIN_SIZE[1] * random_size_value))
       # Movement of the penguin
       moving_direction, start_position = self.define_spawn_position()
       # Sprite
       self.rect = pygame.Rect(start_position[0], start_position[1],
size[0]//1.4, size[1]//1.4) #Takes in x, y, width, height arguments
       self.images = [] # initialize the images list
       for nb in range(1, 7):
           image =
pygame.image.load(f"Assets/Penguin/{nb}.png").convert_alpha()
           # Scale the image
           image = pygame.transform.smoothscale(image, size)
           # Flip the image if moving_direction is "left", because
penguin image is facing to the right
           if moving_direction == "left":
               image = pygame.transform.flip(image, True, False)
           self.images.append(image) # Append the image to the list
```

```
        self.current_frame = 0 #Setting the starting frame of the
animation to the first image in the list
        self.animation_timer = 0 #Used to synchronize the animation with
other events in the game
        self.animation_interval = 50 #Interval
```

- The '__init__()' function is used for initializing the attributes and properties of the penguin object, such as its size, position, images, animation, and movement. It first generates a random size for the penguin by generating a random value between the range of the constants PENGUIN_SIZE_RANDOMIZE[0] and PENGUIN_SIZE_RANDOMIZE[1], multiplying it by PENGUIN_SIZE. Then, it returns a random position and movement direction for the penguin to start at by calling the function define_spawn_position(). The 'self.rect' attribute is initialized with the starting position and the size of the penguin, which is used to define the collision box of the penguin. The 'self.images' attribute is initialized as an empty list, and then it is filled with the images of the penguin in the for loop. The 'self.current_frame' attribute is set to 0, which will be used to keep track of the current animation frame of the penguin. The 'self.animation_timer' attribute is set to 0, which will be used to synchronize the animation with other events in the game. The 'self.animation_internal' attribute is set to 50 and it is used to define the animation speed of the penguin.

```
    def define_spawn_position(self):
        velocity = random.uniform(PENGUIN_MOVE_SPEED["min"],
PENGUIN_MOVE_SPEED["max"])
        # Choose a random direction for the penguin to move in
        moving_direction = random.choice(("right", "left", "up", "down"))

        # Generate a random starting position
        offset = 50 # Spawns at least 50 pixels away from the edges of the
screen
        start_x = random.uniform(offset, SCREEN_WIDTH - offset)
        start_y = random.uniform(offset, SCREEN_HEIGHT - offset)
        start_position = (start_x, start_y)

        # Set the velocity of the penguin based on the moving direction
        if moving_direction == "right":
            self.velocity = [velocity, 0]

        if moving_direction == "left":
            self.velocity = [-velocity, 0]
```

```
        if moving_direction == "up":
            self.velocity = [0, -velocity]

        if moving_direction == "down":
            self.velocity = [0, velocity]

        return moving_direction, start_position
```

- The ‘define_spawn_position()’ function is used to define the initial spawn position and movement direction of the penguin object. The function starts by generating a random velocity for the penguin within a specified range. Then, it generates a random starting position for the penguin and randomly chooses a direction for the penguin to move (either right, left, up, or down). Finally, the moving_direction and start_position are returned from the function.

```
    # Move the penguin based on its current velocity
    def move(self):
        # Update the rect based on the velocity
        self.rect.x += self.velocity[0]
        self.rect.y += self.velocity[1]
```

- The ‘move()’ function updates the position of the penguin object on the game screen. The position of the penguin is represented by the ‘self.rect’ attribute. The x and y coordinates of the rect are updated by adding the values in the ‘self.velocity’ attribute is set in the ‘define_spawn_position()’ function, where it is determined randomly. This ‘move’ function is called every frame of the game, causing the penguin to continuously move in the direction specified by its velocity.

```
    # Updates the frame of the penguin animation based on the elapsed time
since the last frame
    def animate(self):
        self.animation_timer += 1
        if self.animation_timer >= ANIMATION_SPEED:
            self.animation_timer = 0 #Reset the animation timer
            self.current_frame += 1 #Increment the current frame

            if self.current_frame >= len(self.images):
                self.current_frame = 0
```

- The ‘animate()’ function is used for animating the penguin sprite by updating its current frame based on the elapsed time since the last frame. The function increments the animation timer on each call, and when it is bigger than the constant ANIMATION_SPEED, the animation frame is resetted to 0 if it has reached the end of the list of images, thus creating an

illusion of animation by displaying different images of the penguin in quick succession.

```
    # Draw a visual representation of the penguin's collision box (hitbox)
    def draw_hitbox(self, surface):
        pygame.draw.rect(surface, (200, 60, 0), self.rect) #Draw rect on
the surface using orange color with self.rect as its size and position
```

- The 'draw_hitbox()' function is for visualizing the collision box of the penguin on the game screen. It uses the function 'pygame.draw.rect()' which takes in three arguments, the surface to draw on, the color of the rectangle in RGB format, and the size and position of the rectangle which is represented by the 'self.rect' object.

```
    # Draw elements on the game screen
    def draw(self, surface):
        self.animate()
        self.image = self.images[self.current_frame]
        # Draw the image using the self.rect as its position
        surface.blit(self.image, self.rect)
        if DRAW_HITBOX: #The draw_hitbox constant is set to 'False',
making it invisible to the player
            self.draw_hitbox(surface)
```

- The 'draw()' function is used for drawing the elements on the game screen. It takes in one argument 'surface', which represents the game screen. The function first calls the animate method to update the current frame of the animation. It then sets the 'self.image' variable to the current frame of animation. Then, the function uses the 'blit()' method of the surface object to draw the current frame of the animation on the screen, using the 'self.rect' as the position. It then checks if the DRAW_HITBOX constant is true or not. Since the constant is set to False, it will always be invisible to the player.

```
    # Remove the penguin object from the list of active penguins when it
is caught by the player
    def catch(self, active_objects): #using active_objects as parameter,
representing a list of penguin objects that represents all of the active
penguins in the game.
        active_objects.remove(self)
        return 1 #+1 point from catching the penguin
```

- The 'catch()' function is used for removing the current penguin object from the list of active penguins when it is caught by the player. This is done by using the 'remove' method of the list, which takes the penguin object as a parameter and removes it from the list. The function then

returns a value of 1, which represents the point earned by the player for catching the penguin.

4. **bomb.py**
    - The class ‘Bomb’ is a subclass of the ‘Penguin’ class. It creates an object of the class Bomb, which is an instance of the class Penguin. The Bomb class inherits all the attributes and methods of the Penguin class.

```
import pygame
import random
from constant import *
from penguin import Penguin
```

    - Imports pygame library, random library, constant module, and penguin module.

```
class Bomb(Penguin):
    def __init__(self):
        # Generate random size for the bomb
        random_size_value = random.uniform(BOMB_SIZE_RANDOMIZE[0],
BOMB_SIZE_RANDOMIZE[1])
        size = (int(BOMB_SIZE[0] * random_size_value), int(BOMB_SIZE[1] *
random_size_value))
        # Movement of the bomb
        moving_direction, start_position = self.define_spawn_position()
        # Sprite
        self.rect = pygame.Rect(start_position[0], start_position[1],
size[0]//1.4, size[1]//1.4)
        self.images = [] # initialize the images list
        for nb in range(1, 7):
            image =
pygame.image.load(f"Assets/Bomb/{nb}.png").convert_alpha()
            # Scale the image
            image = pygame.transform.smoothscale(image, size)
            # Flip the image if moving_direction is "left", because
penguin image is facing to the right
            if moving_direction == "left":
                image = pygame.transform.flip(image, True, False)
            self.images.append(image) # Append the image to the list
        self.current_frame = 0
        self.animation_timer = 0
        self.animation_interval = 50

    def catch(self, active_objects): # Remove the active bombs from the
list
        active_objects.remove(self)
```

```
        return -1 #-1 point from catching the bomb
```

- The Bomb class overrides the 'catch()' method of the parent class, Penguin. Instead of returning 1 point as in the parent class, the 'catch()' method in the Bomb class returns -1 point for catching the bomb. The Bomb class also uses different image assets and size specifications for the bomb object. The rest of the attributes and methods of the Bomb class are inherited from the Penguin class and work in the same way as the parent class.

5. hand.py

- The class 'Hand' is used to create an object that represents the player's hand in the game.

```
import pygame
from constant import *
```

- Imports pygame library and constant module.

```
class Hand:
    def __init__(self):
        self.original_image =
pygame.image.load("Assets/Hand/hand1.png").convert_alpha() #Opened hand.
Using 'self.original_image', allowing original image to be used as a base
for creating new scaled images
        self.original_image =
pygame.transform.smoothscale(self.original_image, (HAND_SIZE, HAND_SIZE))
#Scales the image to size specified by the HAND_SIZE
        self.image = self.original_image.copy() #Creates a copy of the
image
        self.smaller_image =
pygame.image.load("Assets/Hand/hand2.png").convert_alpha() #Closed hand
        self.smaller_image =
pygame.transform.smoothscale(self.smaller_image, (HAND_SIZE - 60,
HAND_SIZE - 60))
        self.rect = pygame.Rect(0, 0, HAND_HITBOX_SIZE[0],
HAND_HITBOX_SIZE[1]) #To determine whether hand is colliding with any
other objects in the game
```

- The '__init__()' function initializes the object by loading an opened hand image and then for the smaller image, loading in a closed hand image and also scaling it to a smaller specified size. It also creates a copy of the original image, and creates a rectangle object that is used as the hitbox for the hand.

```
    # Draw a visual representation of the hand's collision box (hitbox)
    def draw_hitbox(self, surface):
        pygame.draw.rect(surface, (200, 60, 0), self.rect) #Draw rect on
the surface using orange color with self.rect as its size and position
```

- The ‘draw_hitbox()’ function is used to draw a visual representation of the hand’s collision box on the game’s surface.

```
    def draw(self, surface):
        # Draw the image using the self.rect as its position
        surface.blit(self.image, self.rect)
        if DRAW_HITBOX:
            self.draw_hitbox(surface)
```

- The ‘draw’ function is used to draw the hand’s image on the game’s surface, using ‘self.rect’ as its position. Like the penguin and bomb class, it also calls the draw_hitbox function if the DRAW_HITBOX constant is set to True. In this case, it is always set to False, thus being invisible to the player’s screen.

```
    def on_object(self, objects):
        # Return a list with all objects that are colliding with the
hand's hitbox
        return [object for object in objects if
self.rect.colliderect(object.rect)]
```

- The ‘on_object()’ function is used to check if the hand’s hitbox is currently colliding with any other objects in the game, and returns a list of objects that are colliding.

```
    def catch_objects(self, objects, score, sounds):
        if self.left_click:
            # Get a list of objects that are colliding with the hand
            caught_objects = self.on_object(objects)
            # If colliding
            if caught_objects:
                sounds["grab"].play()
                for object in caught_objects:
                    object_score = object.catch(objects)
                    score += object_score
                    # If the object's score is negative
                    if object_score < 0:
                        sounds["explosion"].play()
        else:
            self.left_click = False
        # Return the updated score
        return score
```

- The ‘catch_objects’ function is used to check if the player has clicked the left mouse button, and if so, it uses the ‘on_object()’ function to check if the hand is colliding with any objects by getting a list of it. If colliding, it calls the catch function on those objects, which removes them from the

game and adds to the player's score. It also plays sound effects depending on what objects are caught. Finally, it returns the updated score to the player.

6. **hand_tracking.py**
   - Source code: https://google.github.io/mediapipe/solutions/hands.html
   - The class 'HandTracking' is used for tracking and detecting the position of a hand in an image and its gesture using the MediaPipe library.

```
import cv2
import mediapipe as mp
from constant import *
```

   - Imports the cv2 (OpenCv library) and MediaPipe library to detect and track hand movements using a camera, along with a constant module.

```
# Source code: https://google.github.io/mediapipe/solutions/hands.html

# Import modules for drawing hand annotations and for hand tracking
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_hands = mp.solutions.hands
```

   - Importing modules from the MediaPipe library. 'mp_drawing' is a module that provides utility functions for drawing various types of annotations on images, such as landmarks and connections. 'mp_drawing_styles' is a module that provides predefined styles for the annotations. 'mp_hands' is a module that provides functionality for detecting and tracking hands in an image or video stream.

```
class HandTracking:
    def __init__(self):
        self.hand_tracking = mp_hands.Hands(min_detection_confidence=0.5,
min_tracking_confidence=0.5, max_num_hands=1) #To detect and track hand,
with the maximum number of hand on camera is set to 1
        self.hand_x = 0
        self.hand_y = 0
        self.results = None
        self.hand_closed = False
```

   - The '__init__()' function is used to initialize objects of the class. It sets the minimum detection and tracking confidence for the hand detection, and sets the maximum number of hands to be tracked to 1. It also sets the initial position of the hand as (0,0) and sets the results and hand_closed attributes to None and False respectively.

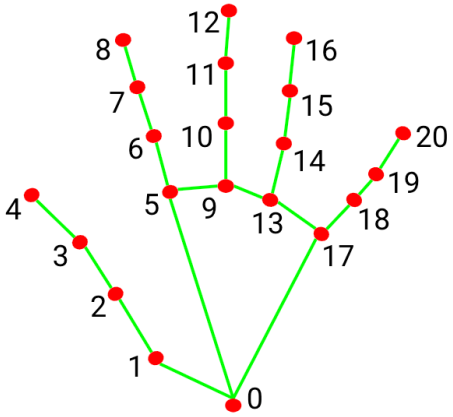
```
    # Scans the frame for any hands and returns the frame with the hand
position marked
    def scan_hands(self, image):
```

```
        # Flip the image horizontally for a later selfie-view display, and
convert the BGR image to RGB
        image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)

        # Resize the image
        image = cv2.resize(image, (350, 200))

        # To improve performance, optionally mark the image as not
writeable to pass by reference
        image.flags.writeable = False
        # Process the image and track hands within the image
        self.results = self.hand_tracking.process(image)

        # Draw the hand annotations on the image
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) #Image is converted
back to BGR so that it can be saved using OpenCv

        self.hand_closed = False

        if self.results.multi_hand_landmarks: #multi_hand_landmarks =
collection of detected/tracked hands, each hand represented as a list of
21 hand landmarks
            for hand_landmarks in self.results.multi_hand_landmarks:
                x, y = hand_landmarks.landmark[9].x,
hand_landmarks.landmark[9].y #9th landmark = MIDDLE_FINGER_MCP

                # Convert coordinates to screen coordinates
                self.hand_x = int(x * SCREEN_WIDTH)
                self.hand_y = int(y * SCREEN_HEIGHT)

                x1, y1 = hand_landmarks.landmark[12].x,
hand_landmarks.landmark[12].y #12th landmark = MIDDLE_FINGER_TIP

                if y1 > y:  #If the 12th landmark is below than the 9th
landmark
                    self.hand_closed = True  #the program will detect that
the hand is closed
                else:
                    self.hand_closed = False
```

```
                # Draw the hand landmarks and connections on the 'image'
object
                mp_drawing.draw_landmarks(
                    image,
                    hand_landmarks,
                    mp_hands.HAND_CONNECTIONS,
                    mp_drawing_styles.get_default_hand_landmarks_style(),

mp_drawing_styles.get_default_hand_connections_style())
        return image
```

- The ‘scan_hands()’ function is used to scan the frame for any hands present and returns the frame with the hand position marked. It first takes an image as an input, flips the image horizontally, converts it from BGR to RGB and resizes the webcam frame to a specific size.
- Then, the image is marked as not writable to improve performance when passing by reference. Image writeable specifies whether the image’s data can be modified or not. Then, the method uses the ‘self.hand_tracking.process(image)’, which uses MediaPipe’s hand tracking solution to process the image and detect and track any hands within the image. This returns a ‘results’ object, which is stored as an attribute of the ‘HandTracking’ class.
- After the image is processed, the image is marked as writeable again, and then the image is converted back to BGR color space, which is the default color space used by OpenCv. This is done so that the image can be saved using OpenCv later on.
- The method then checks if there are any ‘multi_hand_landmarks’ detected, and if there are, it iterates over them. The code first extracts the position of the 9th landmark which is the MIDDLE_FINGER_MCP using the ‘landmark[9].x’ and ‘landmark[9].y’ attributes and assigns the x,y coordinates to the variables x and y respectively.
- It then converts the coordinates from the image to screen coordinates by multiplying with the ‘SCREEN_WIDTH’ and ‘SCREEN_HEIGHT’, and assigns the result to ‘self.hand_x’ and ‘self.hand_y’ respectively.
- Then, it extracts the position of the 12th landmark which is the MIDDLE_FINGER_TIP using the ‘landmark[12].x’ and ‘landmark[12].y]’ attributes and assigns the x1,y1 coordinates to the variables x1 and y1 respectively.
- And then it checks if the 12th landmark is below the 9th landmark by comparing the y1 and y. If it is, it means that the hand is closed, so it sets the ‘self.hand_closed’ to ‘True’. Else, it sets it to ‘False’.

- Finally, the method draws the hand landmarks and connections on the image using MediaPipe's 'draw_landmarks' method and returns the image. The 'HAND_CONNECTIONS' constant is a list of tuples that define the connections between different hand landmarks, used to draw lines between landmarks in order to better visualize the hand's shape and structure.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Hand landmarks. Image taken from:
https://google.github.io/mediapipe/solutions/hands.html

```
    # Return the current position of the hand
    def get_hand_position(self):
        return (self.hand_x, self.hand_y)
```

- The 'get_hand_position()' function returns the current position of the hand as a tuple of (x,y) coordinates.

```
    # Display the image with the hand annotations on the screen
    def display_hand(self):
        cv2.imshow("image", self.image)
        cv2.waitKey(1) #Waits for 1ms before the next frame to create
illusion of a live video stream
```

- The 'display_hand()' function displays the image with the hand annotations on the screen using the 'cv2.imshow()' function from OpenCv library. It also uses the 'cv2.waitKey()' function to wait for 1ms before the next frame to create the illusion of a live video stream.

7. ui.py

- The 'UI.py' file contains two functions that are used to render texts and display buttons in the game.

```
import pygame
from constant import *
```

- Imports pygame library and constant module.

```
def draw_text(surface, text, position, color, font=FONTS["medium"],
position_mode="top_left",
              shadow=False, shadow_color=(0,0,0), shadow_offset=5):
   label = font.render(text, 0, color) #The 0 is ignored, as the size is
used from the constant
   label_rect = label.get_rect() #To position the text
   # Set the position of the text on the surface
   if position_mode == "top_left":
       label_rect.x, label_rect.y = position
   elif position_mode == "center":
       label_rect.center = position
   # Make the shadow
   if shadow:  #Checks if shadow parameter is set to True
       label_shadow = font.render(text, 0, shadow_color)
       shadow_rect = label_rect.copy()
       shadow_rect.x -= shadow_offset
       surface.blit(label_shadow, shadow_rect)

   surface.blit(label, label_rect) # draw the text
```

- The 'draw_text()' function is used to render text on the screen taking parameters such as the surface to draw the text on, the text to be displayed, position, color, and font. It also allows for the option to add a shadow to the text.

```
def button(surface, position_y, text=None, click_sound=None):
   button_rect = pygame.Rect((SCREEN_WIDTH//2 - BUTTONS_SIZES[0]//2,
position_y), BUTTONS_SIZES)

   # Check if the mouse cursor is currently hovering over the button or
not
   on_button = False
   if button_rect.collidepoint(pygame.mouse.get_pos()): #If it is on the
button, get the current mouse position
       color = COLORS["buttons"]["hover"] #Change the button color
       on_button = True
   else:
       color = COLORS["buttons"]["default"]

   # Draw the shadow rectangle
   pygame.draw.rect(surface, COLORS["buttons"]["shadow"], (button_rect.x
- 8, button_rect.y - 8, button_rect.w, button_rect.h))

   # Draw the rectangle
```

```
    pygame.draw.rect(surface, color, button_rect)

    # Draw the text
    if text is not None:
        draw_text(surface, text, button_rect.center,
COLORS["buttons"]["text"], position_mode="center",
                  shadow=True, shadow_color=COLORS["buttons"]["shadow"])

    if on_button and pygame.mouse.get_pressed()[0]: # if the user press on
the button using the left mouse button
        if click_sound is not None: # play the sound if needed
            click_sound.play()
        return True
```

- The ‘button()’ function is used to create and display a button on the screen. It takes in parameters such as the surface to draw the button on, the y position of the button, the text to be displayed on the button, and the sound effect when it's clicked.

8. menu.py

- The class ‘Menu’ is a class that creates the menu of the game.

```
import pygame
from constant import *
from background import Background
import ui
import sys
```

- Imports the pygame library, constant module, background module, ui library, and sys library.

```
class Menu:
    def __init__(self, surface):
        self.surface = surface
        # Create a background object as an attribute
        self.background = Background()
        # Create a Pygame sound object as an attribute
        self.click_sound = pygame.mixer.Sound(f"Assets/Sounds/click.mp3")
```

- The ‘__init__()’ function sets up the surface, creates a background object, and creates a pygame sound object as an attribute.

```
    def draw(self):
        self.background.draw(self.surface)
        # Draw the game title
        ui.draw_text(self.surface, GAME_TITLE, (SCREEN_WIDTH//2, 180),
COLORS["title"], font=FONTS["big"],
                  shadow=True, shadow_color=(255,255,255),
position_mode="center")
```

- The ‘draw()’ function is used to draw the background and the game title text on the surface.

```
    def update(self):
        # Draw the menu to the screen
        self.draw()
        # If the 'Start' button was clicked
        if ui.button(self.surface, 320, "START",
click_sound=self.click_sound):
            return "game"

        # If the 'Quit' button was clicked
        if ui.button(self.surface, 455, "QUIT",
click_sound=self.click_sound):
            # Close the Pygame window and exit the program
            pygame.quit()
            sys.exit()
```

- The ‘update()’ function is used to update the menu and to draw the buttons. It calls the draw function and checks if the “START” or “QUIT” button was clicked. If the “START” button was clicked, it returns “game” which is used to transition to the game state. If the “QUIT” button was clicked, it closes the pygame window and exits the program.

9. game.py

- The class ‘Game’ is used to create the logic behind the game.

```
import pygame
import time
import random
from constant import *
from background import Background
from hand import Hand
from hand_tracking import HandTracking
from penguin import Penguin
from bomb import Bomb
import cv2
import ui
```

- Imports pygame library, time library, random library, constant module, background module, hand module, hand_tracking module, penguin module, bomb module, cv2 library, and ui library.

```
class Game:
    def __init__(self, surface):
        self.surface = surface
        self.background = Background()
```

```
        self.capture = cv2.VideoCapture(0) # Load camera using the default
camera (0)
        self.sounds = {}
        self.sounds["grab"] =
pygame.mixer.Sound(f"Assets/Sounds/grab.mp3")
        self.sounds["grab"].set_volume(SOUNDS_VOLUME)
        self.sounds["explosion"] =
pygame.mixer.Sound(f"Assets/Sounds/explosion.wav")
        self.sounds["explosion"].set_volume(SOUNDS_VOLUME-0.6)
        self.sounds["click"] =
pygame.mixer.Sound(f"Assets/Sounds/click.mp3")
        self.sounds["click"].set_volume(SOUNDS_VOLUME)
```

- The '__init__()' function is used for initializing the game by creating a background object, loading the camera using OpenCv, and loading the sound effects.

```
    # Reset all the needed variables when a new game starts
    def reset(self):
        self.hand_tracking = HandTracking()
        self.hand = Hand()
        self.objects = []
        self.objects_spawn_timer = 0
        self.score = 0
        self.game_start_time = time.time()
```

- The 'reset()' function is used to reset all the needed variables when a new game starts.

```
    def spawn_objects(self):
        t = time.time() #Gets the current time in seconds
        if t > self.objects_spawn_timer:
            self.objects_spawn_timer = t + PENGUIN_SPAWN_TIME #A new
penguin will be spawned every 1 second

            # Increase the probability that the object will be a bomb over
time
            # nb = number of bombs
            nb = (GAME_DURATION-self.time_left)/GAME_DURATION * 100 / 2
#Increase from 0 to 50 during all the game (linear)

            # Add randomness and unpredictability to the game
            if random.randint(0, 100) < nb:
                self.objects.append(Bomb())
            else:
                self.objects.append(Penguin())
```

```
            # Spawn additional bomb and penguin after the half of the game
to increase difficulty
            if self.time_left < GAME_DURATION/2:
                self.objects.append(Bomb())

            if self.time_left < GAME_DURATION/2:
                self.objects.append(Penguin())
```

- The 'spawn_objects()' function is used for periodically spawning new objects in the game. It uses the current time and a timer that is set to the current time plus a predefined spawn time ('PENGUIN_SPAWN_TIME') to determine when to spawn a new object. The function also increases the probability of spawning a bomb object as the time left in the game decreases, to increase difficulty. Additionally, the function also spawns additional penguin and bomb objects after half of the game to increase difficulty. It also adds more randomness and unpredictability to the game by randomly choosing between spawning a penguin or a bomb object.

```
    # Keep track of the remaining time in the game
    def update_game_time(self):
        self.time_left = max(round(GAME_DURATION - (time.time() -
self.game_start_time), 1), 0) #The result is 1 decimal place, with the
maximum value between the result and 0
```

- The 'update_game_time()' function is used to keep track of the remaining time in the game. It subtracts the current time obtained from the 'time.time()' function from the time when the game started 'self.game_start_time', gaining the elapsed time since the game started. Then, it subtracts this elapsed time from the total duration of the game to get the remaining time in the game. The 'round()' function is used to round the remaining time to 1 decimal place, and the 'max()' function is then used to ensure that the time left cannot be negative. Finally, the result is assigned to the 'self.time_left' attribute.

```
    # Capture single frame from the camera
    def load_camera(self):
        _, self.frame = self.capture.read() #method to read the next frame
from the camera
        # '_' used to unpack tuple returned by the method, containing
boolean whether frame was read successfully
        # frame assigned to the self.frame
```

- The 'load_camera()' function is used to capture a single frame from the camera.

```
    # Track player's hand in real-time
```

```
    def set_hand_position(self):
        self.frame = self.hand_tracking.scan_hands(self.frame) #Scanning
the hands in the frame
        (x, y) = self.hand_tracking.get_hand_position()
        self.hand.rect.center = (x, y) #Sets the hand object position on
the screen to match the position of the hand in the webcam frame
                                        # thus hand object positioned on
the screen at the same location as hand in the webcam frame
```

- The ‘set_hand_position()’ is used to keep track of the player’s hand in real-time by scanning the hands in the frame and setting the position of the hand object on the screen to match the position of the hand in the webcam frame.

```
    # Drawing elements of the game on the screen
    def draw(self):
        self.background.draw(self.surface) #Draw the background
        for object in self.objects: #Draw the objects
            object.draw(self.surface)
        self.hand.draw(self.surface) #Draw the hand
        ui.draw_text(self.surface, f"Score : {self.score}", (5, 5),
COLORS["score"], font=FONTS["medium"],
                     shadow=True, shadow_color=(255,255,255)) #Draw the
score
        timer_text_color = (160, 40, 0) if self.time_left < 5 else
COLORS["timer"] #Change the text color remaining time less than 5s
        ui.draw_text(self.surface, f"Time left : {self.time_left}",
(SCREEN_WIDTH//2, 5),  timer_text_color, font=FONTS["medium"],
                     shadow=True, shadow_color=(255,255,255)) #Draw the
remaining time
```

- The ‘draw()’ function is used to draw elements of the game on the screen.

```
    # Update the game's state
    def update(self):
        self.load_camera()
        self.set_hand_position()
        self.update_game_time()
        self.draw()

        if self.time_left > 0:
            self.spawn_objects()
            (x, y) = self.hand_tracking.get_hand_position()
            self.hand.rect.center = (x, y)
            self.hand.left_click = self.hand_tracking.hand_closed #Sets
the mouse left click to hand closing
```

```
            # print("Hand closed", self.hand.left_click)
            if self.hand.left_click: #If hand is closed
                self.hand.image = self.hand.smaller_image.copy() #Use the
hand closed image that has been scaled smaller
            else:
                self.hand.image = self.hand.original_image.copy() #Use the
hand opened image
            self.score = self.hand.catch_objects(self.objects, self.score,
self.sounds)
            for object in self.objects:
                object.move() #Updates its position on the screen

        else: # When the game is over
            if ui.button(self.surface, 400, "BACK",
click_sound=self.sounds["click"]):
                return "menu"

        cv2.imshow("Frame", self.frame) #Shows webcam frame on the screen
        cv2.waitKey(1) #Waits for 1ms before the next frame to create
illusion of a live video stream
```

- The ‘update()’ function is responsible for updating the state of the game. It loads the camera, sets hand position, updates the game time, drawing the game elements on the screen, spawning objects, and handling user input.
- If the game is over, the function calls the ‘ui.button()’ method to draw a “BACK” button on the screen, which allows the user to return to the “menu” when clicked.

10. main.py

- The ‘main.py’ file is the main file that runs the game.

```
import pygame
from constant import *
from game import Game
from menu import Menu
from penguin import *
import sys
```

- Imports pygame library, constant module, game module, menu module, penguin module, and sys library.

```
# Initialize Pygame
pygame.init()
```

- Initializes pygame.

```
# Create Window/Display
pygame.display.set_caption(WINDOW_NAME)
SCREEN = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
```

- Sets the window caption and screen size

```
# Control frame rate of the game
clock = pygame.time.Clock()
```

- Sets the frame rate of the game using a clock object.

```
# Music
pygame.mixer.music.load("Assets/Sounds/Komiku_Glouglou.mp3") #Loads
background music
pygame.mixer.music.set_volume(MUSIC_VOLUME)
pygame.mixer.music.play(-1) #Loop the music indefinitely
```

- Loads the background music, sets its volume, and loop it indefinitely.

```
# Set initial state of the game to menu screen
state = "menu"
```

- Setting the initial state of the game to the menu screen.

```
# State of the game, if it's on game or menu
game = Game(SCREEN)
menu = Menu(SCREEN)
```

- Create a representation of the states of the game.

```
# Handle user input events (pressing escape key on keyboard or closing
the game window)
def user_events():
    for event in pygame.event.get(): #Get events
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_ESCAPE: #If the pressed key is the
escape key
                pygame.quit()
                sys.exit()
```

- The ‘user_events()’ function is used to handle user input events such as closing the game window or pressing the escape key or pressing the buttons.

```
# Update the game state and display
def update(): # The variable can be accessed and modified from anywhere
within the program.
    global state
    if state == "menu":
        if menu.update() == "game":
            game.reset() # Reset all necessary variables to start a new
game
            state = "game"
```

```
    elif state == "game":
        if game.update() == "menu":
            state = "menu"
    pygame.display.update()
    # Set FPS
    clock.tick(FPS)
```

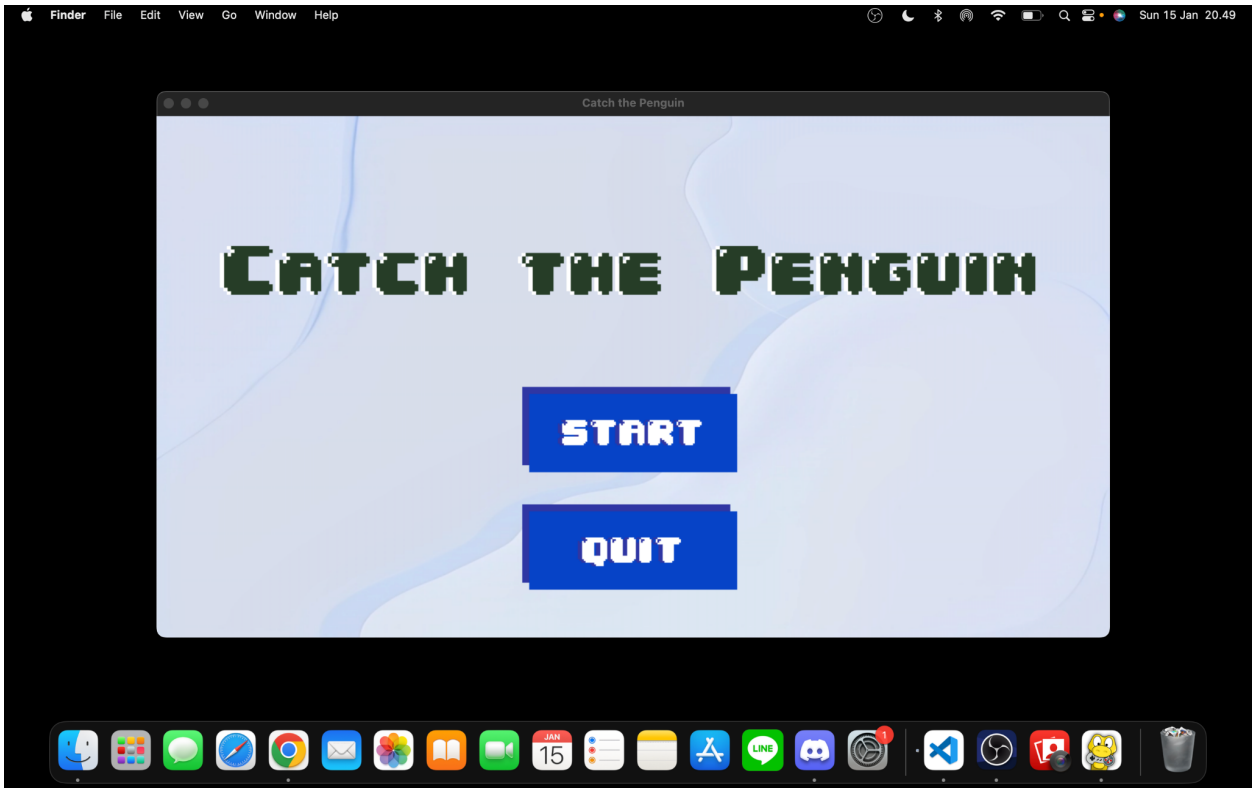
- The ‘update()’ function is used to update the current state of the game, whether it is in the menu or in the game itself. It also updates the display by calling the ‘pygame.display.update()’ function and sets the frame rate of the game using the ‘clock.tick(FPS’ function.

```
# Main Loop
while True:
    user_events()
    update()
```

- The main game loop is an infinite loop that calls the ‘user_events()’ function and ‘update’ function. The loop will end when the user closes the window or presses the escape key or clicks on the exit button.

E. Screenshots of the Game

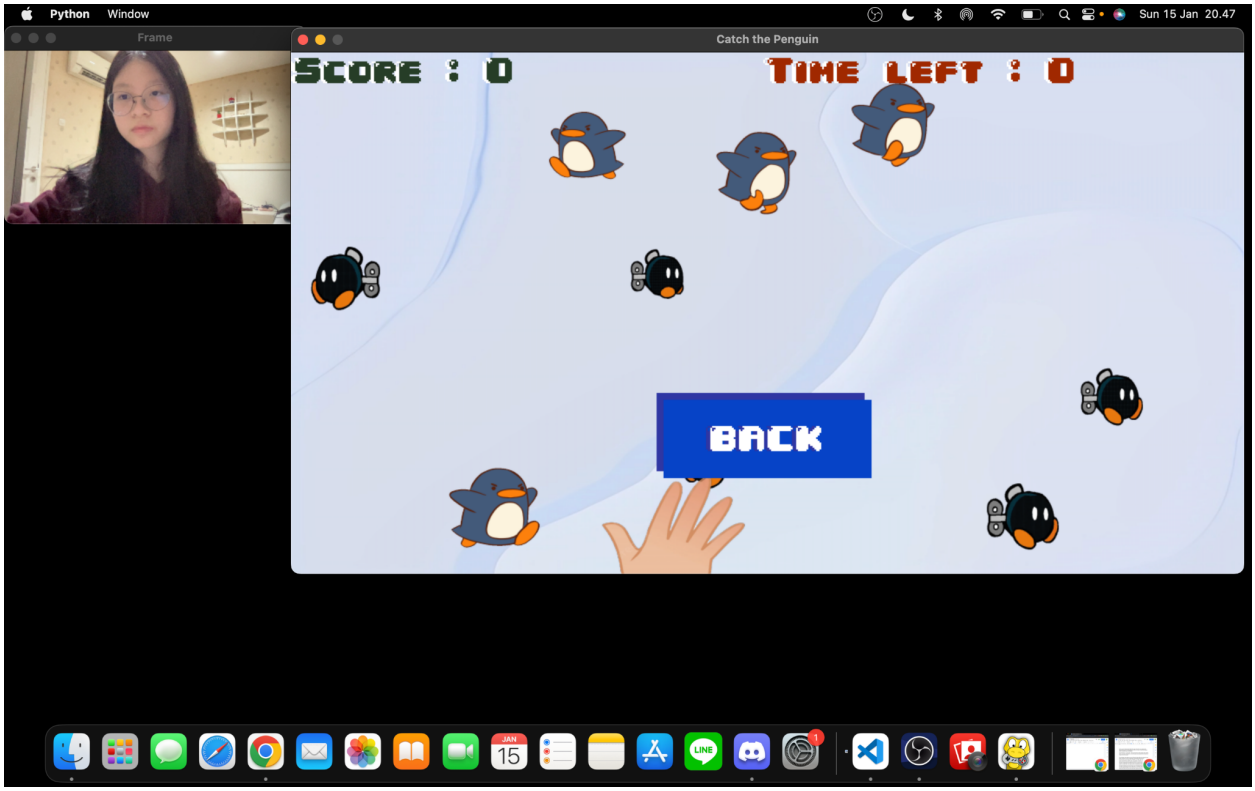
1. Menu Screen



2. **Game Screen (Hand Opened)**

3. **Game Screen (Hand Closed)**

4. Game Over Screen



F. Lesson Learned/Reflection

As this was my first ever programming project using Python, I was, to say the least, pressured about it. I had a hard time finding ideas to fit this project, as my skills and knowledge in Python were overall still very basic and at a beginner level. However, after some time contemplating and consulting ideas with Mr Jude, I finally decided to step out of my boundaries and decided to create a game inspired using my favorite animal, a penguin, by using and learning two completely new libraries that have never been covered during the semester 1 of Algorithm and Programming course, which are OpenCv and MediaPipe libraries, while implementing a library that I have been taught before in the course, which is Pygame.

By creating the game, I was able to learn game development by using the Pygame library and was also able to learn for the first time about the implementation of machine learning using OpenCv and MediaPipe by using a laptop's webcam and performing hand-tracking. All of the libraries, logic, and functions used in the game were very challenging and

complicated, as I ran into several issues when importing needed libraries and also when applying different logic and functions for the games. However, through the help of sources from Youtube and Stack Overflow, I was able to debug and solve the issues and get the game to function properly with how I imagined it to be in my head. I was also able to come across a website named Pygame Front Game that has helped me immensely in understanding functions available in the Pygame library. Overall, this was a really fun yet stressful journey. I can say that I am proud of myself for being able to create this project using outside knowledge of the course.

G. Resources

1. **Troubleshooting**
   - https://stackoverflow.com
2. **Design**
   - https://online.visual-paradigm.com/
3. **Music and Sound Effects**
   - https://www.y2mate.com/id/youtube-mp3/up5J30atI_Q
4. **Pygame Learning**
   - https://www.pygame.org/docs/index.html
5. **OpenCv and MediaPipe Learning**
   - https://www.youtube.com/watch?v=lDfplevUWRw