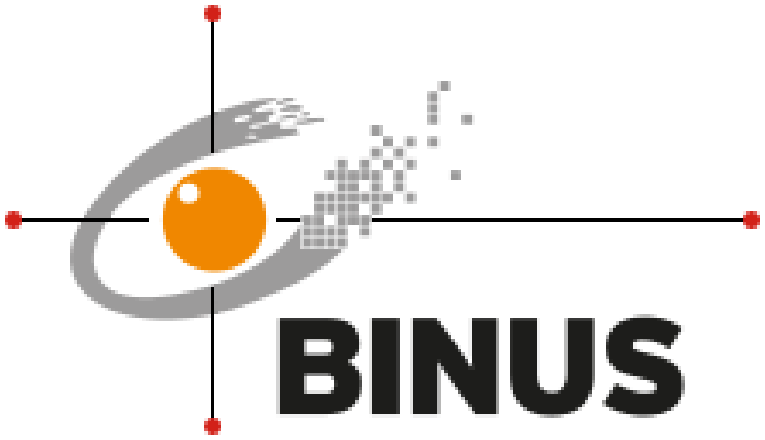
Final Project Report



BINUS UNIVERSITY INTERNATIONAL

Algorithm and Programming

COMP6047001

Lecturer

Jude Joseph Lamug Martinez, MCS

Report by

Joelliane Anggra (2802466322)

L1BC

Type of Assignment : Final Project Report

Submission Pattern:

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 offences 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: Joelliane Anggra

Table of Contents

I. Project Specification

a. Background

When I was informed of this final project and its specifications, my first idea was to create a game. This is because I have often heard about the pygame library and I thought this project would be the perfect opportunity for me to explore its capabilities. Then, comes the question about what kind of game I should make, as considering the flexibility of the pygame library, the possibilities are endless. This is when I was hit with the idea of a claw machine game.

Ever since I was young, I have always been captivated by claw machines in local arcades, and consequently, its digital game adaptations, an example of which is the mobile game, Clawbert. I wondered how exactly one could program something like it, the physics of the prizes moving around was something I could not wrap my head around, additionally, how the claw was to interact with the prize objects was something unknown to me too. I also enjoyed collecting games, which drew me to combine these two concepts in my final project. Hence, I decided that making a claw game would be a good title for my project.

b. Project Description

The claw machine game I created is titled Gacha Grab, as inspired by its collaboration with an Entrepreneurship Hatchery Group, Cup o' Collects, from which I received the game assets for the prizes that can be drawn by the machine. I kept the game fairly simple. At first load, the gacha balls (the prizes to be grabbed) will be spawned into the machine. The user can then either click on the arrow buttons on the window or press the left/right arrow keys to move the claw and press the space key to descend the claw. If the claw detects a gacha ball within its grasp, it is carried along with the claw upwards and disappears. The prize book is updated with the prize the user won. The user can view all the prizes he/she has collected by clicking on the show prizes button.

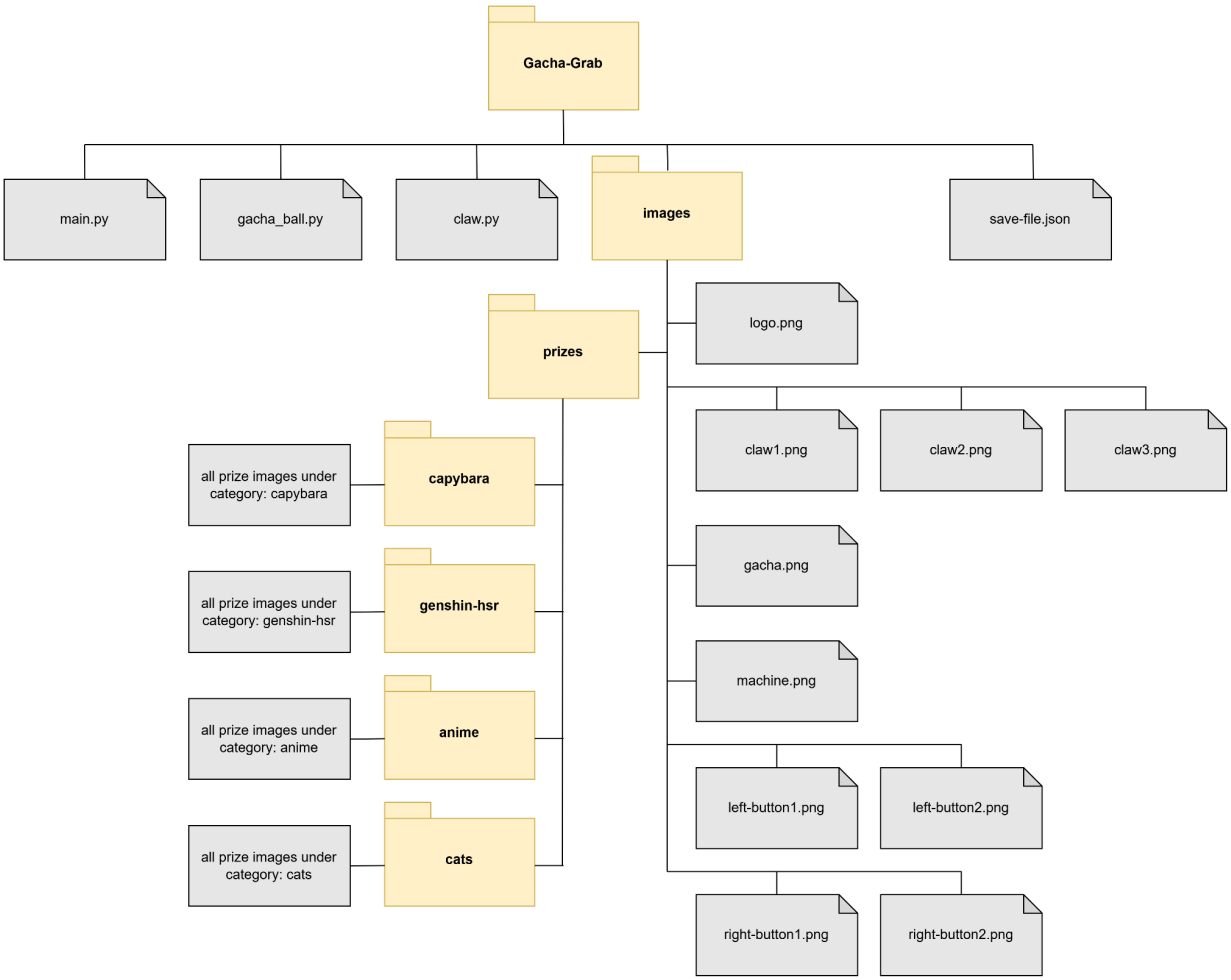
I made sure game progress is saved in a JSON save file. This makes it so that the next time the user starts the game again, the user can pick off from where he/she left off the last time she played. I also added a shuffle function to shuffle the gacha balls around the machine, in case of when they fall into an area unreachable by the claw. In addition to that, to mimic real-life claw machine mechanics as well as promote the longevity of the game's playtime, I added a coin

feature which limits the number of tries the user has before having to wait for his/her coins to replenish as well as limiting the number of gacha balls inside the machine, forcing the user to come back another day before being able to complete the game.

c. Libraries Used

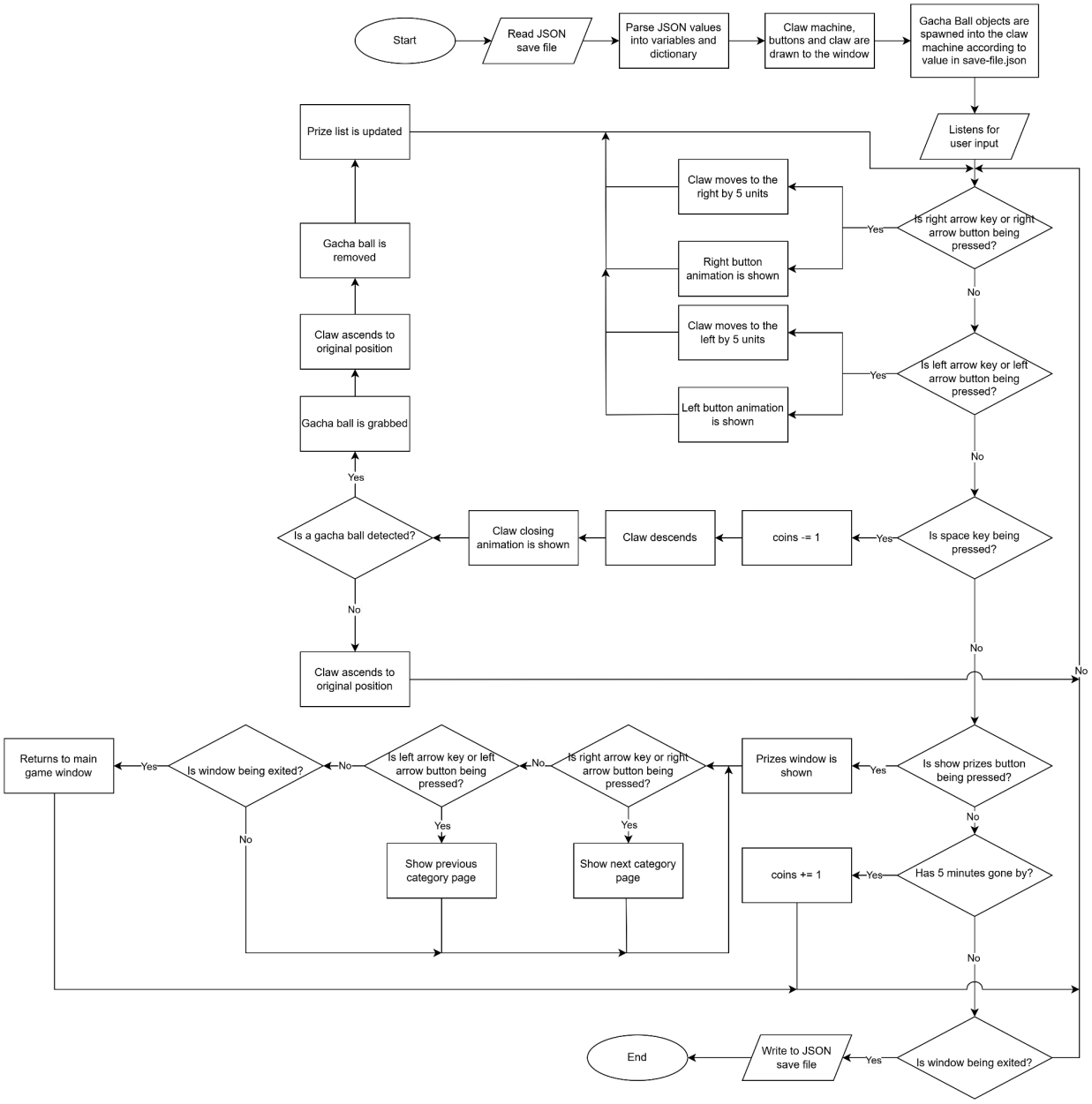
1. Python Standard Library:
    - os: For interacting with the operating system (file manipulation, environment variables, etc.).
    - json: For working with JSON data (encoding and decoding).
    - math: For mathematical functions (sqrt, sin, cos, etc.).
    - datetime: For working with dates and times.
    - tempfile: For creating temporary files and directories.
2. Pygame: Core Pygame library for game development (graphics, sound, input).
3. Pymunk: 2D physics engine library for drawing Pymunk objects with Pygame.
4. NumPy: for numerical computing (arrays, matrices, mathematical operations).
5. Pillow (PIL Fork): For image processing and manipulation.
6. SciPy: For computing the convex hull of a set of points.

d. File Structure



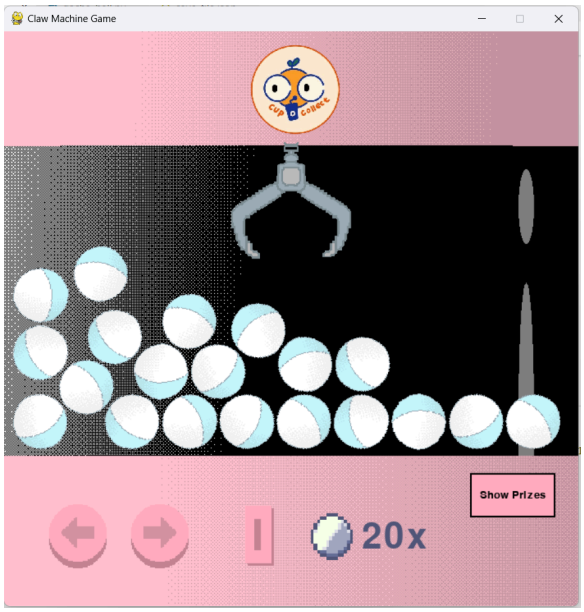
II. Solution Design

a. Flowchart

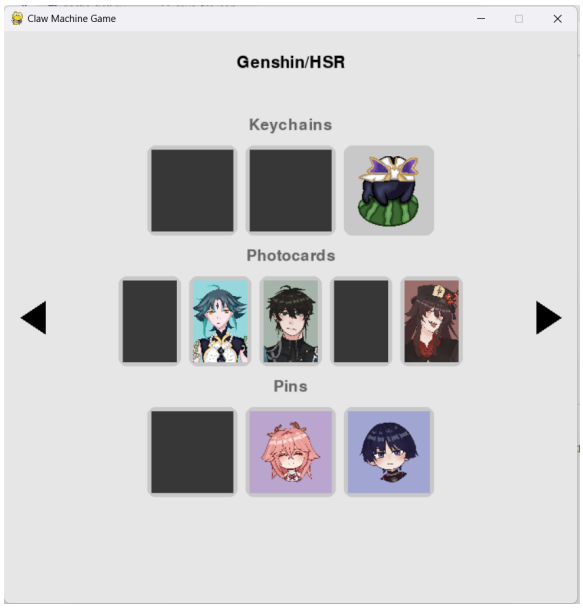


b. Program Layout

Main Game Screen



Prize Book



III. Code Implementation

a. Important Algorithms

GachaBall class

```
class GachaBall:
    # Constants declaration
    SPAWN_X_RANGE = (29, 30)
    SPAWN_Y = 150
    RADIUS = 35
    MASS = 1
    ELASTICITY = 0.8
    FRICTION = 0.5
    COLLISION_TYPE = 1

    def __init__(self, space):
        self.space = space
```

```
        # Use constants for initialization
        spawn_x = random.randint(*self.SPAWN_X_RANGE)
        spawn_y = self.SPAWN_Y

        # Create the body and shape
        self.body = pymunk.Body(self.MASS,
pymunk.moment_for_circle(self.MASS, 0, self.RADIUS))
        self.body.position = spawn_x, spawn_y
        self.shape = pymunk.Circle(self.body, self.RADIUS)
        self.shape.elasticity = self.ELASTICITY
        self.shape.friction = self.FRICTION
        self.shape.collision_type = self.COLLISION_TYPE

        # Add the body and shape to the space
        self.space.add(self.body, self.shape)
```

I implemented modular programming by separating my code into classes, one of which is the GachaBall class. This allows me to create separate gacha ball (pymunk) objects with the same (physics) properties, of which constants were derived from trial and error to mimic real-life physics, and have them added to the game screen. The method below is one within this class.

shuffle method

```
    def shuffle(self, intensity=500): # Apply random forces to shuffle the
gacha ball
        # Generate random force components
        force_x = random.uniform(-intensity, intensity)
        force_y = random.uniform(-intensity, intensity)

        # Apply the impulse to the ball's body
        self.body.apply_impulse_at_local_point((force_x, force_y))
```

This method utilizes the pre-defined method **`apply_impulse_at_local_point((force_x, force_y))`** from the pymunk library. How this method works is that it applies an impulse to the body at a specific point in its local coordinate system (relative to the body itself). This applied

impulse alters the body's linear and angular velocity, which causes the body (gacha ball) to move in random directions when triggered. So the method takes an impulse vector, which I had set to be a tuple of 2 random values in the range of -500 to 500 and then based upon this value, the ball gets deflected accordingly.

Claw class

```
class Claw:
    ORIGINAL_Y = 210
    TARGET_Y = 450
    SPEED = 5
    GRAB_RADIUS = 40
    GRAB_CHANCE = 1

    def __init__(self, screen_width, space):
        # Initializes the claw properties
        self.x = screen_width // 2  # Initial claw X position
        self.body = pymunk.Body(body_type=pymunk.Body.KINEMATIC)  #
Initialize pymunk body
        self.body.position = (self.x, self.ORIGINAL_Y)  # Initial claw
position
        self.state = "idle"
        self.space = space  # Pymunk space
        self.current_frame = 0  # Track current frame in animation
        self.frame_delay = 5  # Delay before switching to the next frame
        self.frame_counter = 0  # Counter to track frame delays
        self.grabbed_ball = None
        self.shape = None
```

I also created a class for the claw, which is largely the same as the GachaBall class with the addition of an animation. Whenever the claw descends and ascends back up, a very simple animation is shown. These two methods are shown below.

descend method

```
def descend(self, gacha_prizes, claw_points_close_list):
```

```
        self.body.position = (self.x, self.body.position.y + self.SPEED)
        # Check if claw reached target
        if self.body.position.y >= self.TARGET_Y:
            self.grabbed_ball = self.check_grab(gacha_prizes)
            self.state = "ascending"
            self.frame_counter = 0
            self.frame_counter += 1
            if self.frame_counter >= self.frame_delay: # Switch to next
frame after delay
                self.frame_counter = 0
                self.current_frame = min(self.current_frame + 1,
len(claw_points_close_list) - 1)
        self.shape = self.update_claw_shape(claw_points_close_list) #
Update claw shape based on current frame
```

The descend method updates the claw's position, incrementing its vertical coordinate by 5 then calling for the update_claw_shape function, which I will show below. It also sets the claw's state to ascending when it is done descending.

ascend function

```
def ascend(self, gacha_prizes, claw_points_open_list):
        flag = False
        self.body.position = (self.x, self.body.position.y - self.SPEED)
        if self.grabbed_ball:
            ball_body, _ = self.grabbed_ball.get_body_and_shape()
            ball_body.position = self.body.position + pymunk.Vec2d(0, 40)
# Offset below claw

        # Reset claw to idle state after ascending
        if self.body.position.y <= self.ORIGINAL_Y:
            self.state = "idle"
            self.current_frame = 0  # Reset animation
            self.frame_counter = 1
            if self.frame_counter >= self.frame_delay:
                self.frame_counter = 0
```

```
                self.current_frame = min(self.current_frame + 1,
len(claw_points_open_list) - 1)
            if self.grabbed_ball:
                    flag = True
                    gacha_prizes.remove(self.grabbed_ball)
                    self.space.remove(self.grabbed_ball.body,
self.grabbed_ball.shape)
                    self.grabbed_ball = None
        self.shape = self.update_claw_shape(claw_points_open_list)
        return flag
```

The ascend function mainly performs a similar job, but has the additional check for a grabbed ball and its removal if it is true.

update_claw_shape function

```
    # Update the claw shape based on the current animation frame
    def update_claw_shape(self, claw_points_list):
        # Remove the old shapes from the body and the space
        for shape in self.body.shapes:
            self.space.remove(shape)

        # Add the body to the space (if not already added)
        if self.body not in self.space.bodies:
            self.space.add(self.body)

        # Create a new claw shape
        new_claw_points = claw_points_list[self.current_frame]
        self.shape = pymunk.Poly(self.body, new_claw_points)
        self.shape.elasticity = 0.4
        self.shape.friction = 0.5

        # Add the new shape to the space
        self.space.add(self.shape)

        return self.shape
```

This function creates the illusion of an animation occurring. It works by removing the old shape with the old frame situated at a particular position and replacing it with the new updated shape with potentially a new image and/or position.

get_claw_points_from_surface function

```
def get_claw_points_from_surface(surface, scale=1.0):

    # Save the surface to a file
    temp_file = os.path.join(tempfile.gettempdir(), "claw_temp.png")
    pygame.image.save(surface, temp_file)

    # Load it with PIL
    img = Image.open(temp_file).convert("RGBA")

    # Extract alpha channel and compute convex hull
    alpha = np.array(img)[:, :, 3]
    non_transparent_coords = np.column_stack(np.where(alpha > 0))
    hull = ConvexHull(non_transparent_coords)
    hull_points = non_transparent_coords[hull.vertices]

    # Center and scale points
    center_x = (hull_points[:, 1].max() + hull_points[:, 1].min()) // 2
    center_y = (hull_points[:, 0].max() + hull_points[:, 0].min()) // 2
    scaled_points = [
        ((x - center_x) * scale, (y - center_y) * scale)
        for y, x in hull_points
    ]

    # Remove temporary file
    os.remove(temp_file)

    return scaled_points
```

This function returns the points around the shape of the claw animation and ensures the game recognizes the different claw shapes. How it works is that it makes use of the numpy library to detect the transparency of each pixel of the image using the alpha channel (the 4th channel in RGBA). It collects the non-transparent pixels and then computes their convex hull (the smallest convex polygon that encloses all the points) using the SciPy library. The points (gathered from computing the center of these convex hulls) are then returned in the appropriate format). This function is essential for more complicated shapes such as the claw.

b. Data Structures Used

- Lists – used to store the gacha ball objects as well as frames for the claw animations
- Dictionaries – used in storing the data to be saved/load into/from the JSON save file
- Tuples – mainly used as arguments for functions requiring data types with tuples format such as vectors and image/screen size
- Classes – used to implement modularity (GachaBall, Claw and Main class)

c. Further Development

Acknowledging that the game is incredibly basic, here are a few improvements that can be added in future updates to enhance the playability of the game:

1. More Prizes – The current very limited number of prizes that can be won reduces the game's playtime significantly.
2. Bonus Minigames – The introduction of bonus minigames to win extra coins or refill the claw machine may improve the variability of gameplay which in turn reduces the user's tendency to get bored over repetitive mechanics (improve user retention).
3. Customizability – Future installments may include allowing the user to change the style of their claw machines, starting from the machine itself, to the claw and the gacha balls.
4. Levelling System – A level system may be added, with experience being accumulated every time the user makes a turn to promote user engagement.

IV. Resources

- Game Assets from Cup o' Collects Team (link classified)
- External Libraries:
    - Pygame = http://www.pygame.org
    - Pymunk = http://www.pymunk.org
    - NumPy = https://numpy.org/
    - Pillow = https://pypi.org/
    - SciPy = https://scipy.org/
- ChatGPT = https://chatgpt.com/