Report 1

Jayden Jung

My team, me, Jason, and Jueun, met up for the first time to work on our project: CritterCraft, a game where you can pick a critter to take care of by feeding it, giving it love, cleaning its room, and playing games with it, ensuring that your pet doesn't die from neglect. We scoped out the overall game with plans to implement text to speech, a dynamic soundtrack, and some random events.

I personally worked on the branding/logo for the game, and also one of the core mini-games: Memorization (official name in the works). I used TKinter and Python to build a game where you must memorize the order of a 5-step sequence between up, left, right, and down, that you must then imitate, with an interactive GUI. Going forward, I will clean up Memorization, and am focusing on writing music for the soundtrack and designing the critters with pixel art.

Jueun Kang

Today, I met up with my team members to discuss the details of our CritterCraft game. I outlined what the mini games would look like on a document shared to my group, and we decided how we were going to visualize the pet's health, hunger, and love levels. I then, started to work on TicTacBone, which is one of the mini games that will be implemented in our game. I have been adding comments to each line of code that I write so that my team members can easily understand my work. Going forward, I will continue to work on TicTacBone, and then will also work on designing one of the characters.

Jason Ngo

Today we met up and discussed the details of the mini games we plan on implementing. Afterwards, I implemented the title screen of our game using Tkinter using a logo drawn by Jayden. The title screen shows off the logo as well as a call to action to Create Your Critter! Moving forward, I will work on the next couple screens after hitting the Create Your Critter button collaboratively with my team members.

Report 2

Jayden Jung

This week, I designed 2 of the 3 animal characters for the game, the sheep and the panda, and also all the additional colour variation for each character for an added 6 more recoloured designs. I also designed the background art for the game: the main “hub” which is a nice green field with blue skies that the player mainly stays at while taking care of their pet, and the game over “funeral” background that has a gravestone for if the player fails to take care of their pet and it dies. I finished writing all the music for the game: one happy hub theme, an urgent hub theme if your pet is low in all stats, a mini-game theme, and the gameover theme. I also continued working on the code for the Memorization game, fixing previous bugs and adding more to the GUI to make it user friendly. I will be wrapping up this game and moving on to help Jason with coding the main hub and also the last mini-game.

My code:

```python
import tkinter as tk
import tkinter.font as font
from tkinter import messagebox
import random

class MemorizationGame:
    def __init__(self, master):
        self.master = master
        self.master.title("Memorization Game")

        # Initialize sequences
        self.sequence = []
        self.player_sequence = []

        critterfont = font.Font(family = "Helvetica")
        critterFont = font.Font(size = 20)

        # Create labels
        label_frame = tk.Frame(self.master)
        label_frame.pack(pady=10)

        self.lbWelcome = tk.Label(label_frame, text="Welcome to Memorize 
Me!").grid(row=0, column=1)
        self.lbInstructions = tk.Label(label_frame, text="Press start to see the 
pattern, memorize and copy it, and click submit.").grid(row=1, column=1)
        self.lbStatus = tk.Label(label_frame, text="").grid(row=3, column=1)

        # Create buttons
        button_frame = tk.Frame(self.master)
        button_frame.pack(pady=10)

        # Up and Down buttons
        self.btUp = tk.Button(button_frame, text="Up", command=lambda: 
self.add_to_player_sequence("Up"))
        self.btUp.grid(row=0, column=1, pady=5)
        self.btUp['font'] = critterfont

        self.btDown = tk.Button(button_frame, text="Down", command=lambda: 
self.add_to_player_sequence("Down"))
        self.btDown.grid(row=2, column=1, pady=5)
        self.btDown['font'] = critterfont

        # Left and Right buttons
        self.btLeft = tk.Button(button_frame, text="Left", command=lambda: 
self.add_to_player_sequence("Left"))
        self.btLeft.grid(row=1, column=0, padx=5)
        self.btLeft['font'] = critterfont

        self.btRight = tk.Button(button_frame, text="Right", command=lambda: 
self.add_to_player_sequence("Right"))
        self.btRight.grid(row=1, column=2, padx=5)
        self.btRight['font'] = critterfont

        # Start and Submit buttons
        start_button = tk.Button(button_frame, text="Start", 
command=self.start_game)
        start_button.grid(row=1, column=1, pady=10)
        start_button['font'] = critterfont
```

```python
        submit_button = tk.Button(button_frame, text="Submit Sequence",
command=self.check_sequence)
        submit_button.grid(row=3, column=1)
        submit_button['font'] = critterfont

        # Disable buttons initially
        self.disable_buttons()

    def disable_buttons(self):
        for direction in ["Left", "Right", "Up", "Down"]:
            self.get_button(direction).config(state=tk.DISABLED)

    def enable_buttons(self):
        for direction in ["Left", "Right", "Up", "Down"]:
            self.get_button(direction).config(state=tk.NORMAL)

    def add_to_player_sequence(self, direction):
        self.player_sequence.append(direction)

    def start_game(self):
        self.sequence = []
        self.player_sequence = []
        self.disable_buttons()
        self.generate_sequence()
        self.show_sequence()
        self.enable_buttons()

    def generate_sequence(self):
        directions = ["Left", "Right", "Up", "Down"]
        for i in range(5):  # Adjust the number of moves in the sequence as
needed
            self.sequence.append(random.choice(directions))

    def show_sequence(self):
        for i, direction in enumerate(self.sequence):
            self.master.after(i * 1500, lambda d=direction:
self.highlight_button(d))
            self.master.after((i + 1) * 1800, self.clear_highlight)
    def highlight_button(self, direction):
        self.get_button(direction).config(fg="red")

    def clear_highlight(self):
        for direction in ["Left", "Right", "Up", "Down"]:
            self.get_button(direction).config(fg="black")

    def check_sequence(self):
        if self.player_sequence == self.sequence:
            messagebox.showinfo("Success", "Correct sequence! You won!")
        else:
            messagebox.showerror("Error", "Incorrect sequence. Try again.")
        self.disable_buttons()
        self.sequence = []
        self.player_sequence = []

    def get_button(self, direction):
        if direction == "Up":
            return self.btUp
        elif direction == "Down":
            return self.btDown
        elif direction == "Left":
            return self.btLeft
        elif direction == "Right":
            return self.btRight

if __name__ == "__main__":
    root = tk.Tk()
    game = MemorizationGame(root)
    root.mainloop()
```

Jueun Kang

During this past week, I have been individually working on the TicTacBone game which is almost at completion – I will be reviewing the code with my group in our meeting tomorrow. I also completed designing the duck which is one of the characters in our game. After our meeting tomorrow, I will confirm where else I can help out the team with, though I plan to contribute to making random events for our characters.

Jason Ngo

This week I coded most of the front-end of the game. All images are appropriately resized before loaded. The user can enter the game, select their critter with 3 buttons that each take the user to different functions, select their color through a radio button and intvar, and name their critter through stringvar. Right now there is no appropriate sizing and scaling of the selections due to the prototype nature of the game. In future weeks, I will focus on the main hub of the game, coding the interactions between triggering mini games and caring for your critter, and the interface of the main hub.

Report 3

Jayden Jung

This week, I implemented the Care functionalities into the game by working with Jason who added the critter's stats (Health, Hunger, and Love), that increase the base values of these based on the player pressing a corresponding button. I also brought together the 3 mini games that Jueun and I made over the last couple weeks into the main game file and integrated them so they are playable from the critter hub. Jason and I are in the middle of debugging an issue where the outcome of these games aren't being properly reflected (when you win, the max value of a certain stat should increase).

My code:

```
        self.btn_care = Button(self.window, text = "Care", font = 
self.critterFont, bg = "#8cc45c", command = self.openCareWindow)
        self.btn_play = Button(self.window, text = "Play", font = 
self.critterFont, bg = "#8cc45c", command = self.openPlayWindow)
```

(…)

```
    def openCareWindow(self):
        self.activeTimer = 1
        print("Care button pressed")
        careWindow = Toplevel(self.window)
        careWindow.title("Care")
        careWindow.geometry("200x100")
        framecare = Frame(careWindow) # Create and add a frame to window
        framecare.pack()
        btHealth = Button(framecare, text = "Clean", command=lambda: self.care(1,
careWindow))
        btHealth.grid(row = 1, column = 1)
        btHunger = Button(framecare, text = "Feed", command=lambda: self.care(2,
careWindow))
        btHunger.grid(row = 2, column = 1)
        btLove = Button(framecare, text = "Cuddle", command=lambda: self.care(3,
careWindow))
        btLove.grid(row = 3, column = 1)

    def care(self, num, window):
        dying = 0
        if self.critterHealth < 2 or self.critterHun < 2 or self.critterLove < 2:
            dying = 1

        if num == 1:
            print("hp care")
            if self.critterHealth < self.critterHealthMax:
                self.critterHealth += 1
                self.canvas.delete("hp")
                self.canvas.create_text(200, 100, text = f"Health:
{self.critterHealth} / {self.critterHealthMax}", font = "Helvetica 20", tag =
"hp")
            else:
                None
        elif num == 2:
            print("hgr care")
            if self.critterHun < self.critterHunMax:
                self.critterHun += 1
                self.canvas.delete("hgr")
                self.canvas.create_text(400, 100, text = f"Hunger:
{self.critterHun} / {self.critterHunMax}", font = "Helvetica 20", tag = "hgr")
            else:
                None
        elif num == 3:
            print("lov care")
            if self.critterLove < self.critterLoveMax:
                self.critterLove += 1
                self.canvas.delete("lov")
                self.canvas.create_text(600, 100, text = f"Love:
{self.critterLove} / {self.critterLoveMax}", font = "Helvetica 20", tag = "lov")
            else:
                None

        window.destroy()
        self.activeTimer = 0

        if dying == 1 and self.critterHealth > 1 and self.critterHun > 1 and
self.critterLove > 1:
            pygame.mixer.music.stop()
            pygame.mixer.music.load(f"{p.parent}/hub-music.mp3")
            pygame.mixer.music.play(loops = -1)

    def openPlayWindow(self):
        self.activeTimer = 1
        gameswindow = Toplevel(self.window)
        gameswindow.title("Play")
        gameswindow.geometry("200x100")
        framegames = Frame(gameswindow) # Create and add a frame to window
        framegames.pack()
        btMem = Button(framegames, text = "Brain Booster!",
                       command = lambda: self.game(1, gameswindow))
        btMem.grid(row = 1, column = 1)
        btTic = Button(framegames, text = "Tic Tac Bone!",
                       command = lambda: self.game(2, gameswindow))
        btTic.grid(row = 2, column = 1)
        btGuess = Button(framegames, text = "Guess the love!",
                         command = lambda: self.game(3, gameswindow))
        btGuess.grid(row = 3, column = 1)

    def game(self, num, window):
        window.destroy()
        pygame.mixer.music.stop()
        pygame.mixer.music.load(f"{p.parent}/minigame-music.mp3")
        pygame.mixer.music.play(loops = -1)
        if num == 1:
            print("Memorization game")
            self.memGame()
        elif num == 2:
            print("Bone game")
            self.boneGame()
        elif num == 3:
            print("Love game")
            self.loveGame()

    def memGame(self):
        print(self.gameOutcome)
        class MemorizationGame:
            def __init__(self):
                self.master = Tk()
                self.master.title("Brain Booster")

                self.outcome = 0

                # Initialize sequences
                self.sequence = []
                self.player_sequence = []

                self.critterfont = font.Font(family = "Helvetica")
                self.critterFont = font.Font(size = 20)

                # Create labels
                label_frame = Frame(self.master)
                label_frame.pack(pady=10)

                self.lbWelcome = Label(label_frame, text="Welcome to Memorize
Me!").grid(row=0, column=1)
                self.lbInstructions = Label(label_frame, text="Press start to see
the pattern, memorize and copy it, and click submit.").grid(row=1, column=1)
                self.statusVar = StringVar()
                self.statusVar.set("Press start!")
                self.lbStatus = Label(label_frame, textvariable = self.statusVar,
fg = 'blue').grid(row=3, column=1)
```

```
                # Create buttons
                button_frame = Frame(self.master)
                button_frame.pack(pady=10)

                # Up and Down buttons
                self.btUp = Button(button_frame, text="Up", command=lambda:
self.add_to_player_sequence("Up"))
                self.btUp.grid(row=0, column=1, pady=5)
                self.btUp['font'] = self.critterfont

                self.btDown = Button(button_frame, text="Down", command=lambda:
self.add_to_player_sequence("Down"))
                self.btDown.grid(row=2, column=1, pady=5)
                self.btDown['font'] = self.critterfont

                # Left and Right buttons
                self.btLeft = Button(button_frame, text="Left", command=lambda:
self.add_to_player_sequence("Left"))
                self.btLeft.grid(row=1, column=0, padx=5)
                self.btLeft['font'] = self.critterfont

                self.btRight = Button(button_frame, text="Right", command=lambda:
self.add_to_player_sequence("Right"))
                self.btRight.grid(row=1, column=2, padx=5)
                self.btRight['font'] = self.critterfont

                # Start and Submit buttons
                start_button = Button(button_frame, text="Start",
command=self.start_game)
                start_button.grid(row=1, column=1, pady=10)
                start_button['font'] = self.critterfont

                submit_button = Button(button_frame, text="Submit Sequence",
command=self.check_sequence)
                submit_button.grid(row=3, column=1)
                submit_button['font'] = self.critterfont

                # Disable buttons initially
                self.disable_buttons()
                self.master.mainloop()

            def disable_buttons(self):
                for direction in ["Left", "Right", "Up", "Down"]:
                    self.get_button(direction).config(state=DISABLED)

            def enable_buttons(self):
                for direction in ["Left", "Right", "Up", "Down"]:
                    self.get_button(direction).config(state=NORMAL)

            def add_to_player_sequence(self, direction):
                self.player_sequence.append(direction)

            def start_game(self):
                self.statusVar.set("Wait and watch... When the pattern finishes,
repeat it, and press submit!")
                self.sequence = []
                self.player_sequence = []
                self.disable_buttons()
                self.generate_sequence()
                self.show_sequence()
                self.enable_buttons()

            def generate_sequence(self):
                directions = ["Left", "Right", "Up", "Down"]
                for i in range(5):  # Adjust the number of moves in the sequence
as needed
                    self.sequence.append(random.choice(directions))

            def show_sequence(self):
                for i, direction in enumerate(self.sequence):
                    self.master.after(i * 1500, lambda d=direction:
self.highlight_button(d))
                    self.master.after((i + 1) * 1800, self.clear_highlight)

            def highlight_button(self, direction):
                self.get_button(direction).config(fg="red")
            def clear_highlight(self):
                for direction in ["Left", "Right", "Up", "Down"]:
                    self.get_button(direction).config(fg="black")

            def check_sequence(self):
                if self.player_sequence == self.sequence:
                    messagebox.showinfo("Success", "Correct sequence! You won!
Health max will increase by 1.")
                    print(3)
                    crittercraft.critterHealthMax += 1
                    crittercraft.canvas.delete("hp")
                    crittercraft.canvas.create_text(200, 100, text = f"Health:
{self.critterHealth} / {self.critterHealthMax}", font = "Helvetica 20", tag =
"hp")
                    self.master.destroy()
                else:
                    messagebox.showerror("Error", "Incorrect sequence. Try again
next time.")
                    self.master.destroy()
                self.statusVar.set("Press start!")
                self.disable_buttons()
                self.sequence = []
                self.player_sequence = []

            def get_button(self, direction):
                if direction == "Up":
                    return self.btUp
                elif direction == "Down":
                    return self.btDown
                elif direction == "Left":
                    return self.btLeft
                elif direction == "Right":
                    return self.btRight

        MemorizationGame()
        self.activeTimer = 0
        pygame.mixer.music.stop()
        pygame.mixer.music.load(f"{p.parent}/hub-music.mp3")
        pygame.mixer.music.play(loops = -1)
```

```python
    def boneGame(self):
        class TicTacBoneGUI:
            def __init__(self):  # Set up intial state of tictacbone
                self.window = Tk()  # Create a window
                self.window.title("Tic-Tac-Bone")  # Set window title

                # Initialize the tictacbone board as a 3 by 3 grid
                self.board = [["", "", ""], ["", "", ""], ["", "", ""]]
                # Initialize list to store buttons
                self.buttons = [[None, None, None], [
                    None, None, None], [None, None, None]]
                # The current player is 'X'
                self.current_player = "X"
                # Store game outcome to 0 for on-going
                self.outcome = 0

                # Create buttons for the 3 by 3 tictacbone grid
                for i in range(3):
                    for j in range(3):
                        self.buttons[i][j] = Button(self.window, text="",
font=("Arial", 24),
                                                    width=5, height=2,
command=lambda row=i, column=j: self.on_click(row, column))
                        # Put buttons in the grid
                        self.buttons[i][j].grid(row=i, column=j)

                self.computer = "O"  # Initalize symbol for computer
                self.user_turn = True  # Track whether it is the player's turn

                self.window.mainloop()

            def on_click(self, row, column):
                # Player's turn when player can click, button is empty, and there
is no winner yet
                if self.user_turn and self.board[row][column] == "" and not
self.winner_check():
                    # Player prevented from playing when computer's turn
                    self.user_turn = False
                    # Update board with the current player's symbol
                    self.board[row][column] = self.current_player
                    # Update onto the corresponding button on the board
                    self.button_update(row, column)

                    if self.winner_check():
                        # Sends message to show win if player wins
                        messagebox.showinfo(title = "TicTacBone", message =
"Congratulations, you won! Hunger max will increase by 1.")
                        crittercraft.critterHunMax += 1
                        crittercraft.canvas.delete("hgr")
                        crittercraft.canvas.create_text(400, 100, text =
f"Hunger: {self.critterHun} / {self.critterHunMax}", font = "Helvetica 20", tag =
"hgr")
                        self.window.destroy()
                    elif self.tie_check():
                        # Sends message to show tie if result is a tie
                        messagebox.showinfo(title = "TicTacBone", message = "It's
a tie! Try again next time.")
                        self.window.destroy()
                    else:
                        self.current_player = self.computer  # If it is currently
the computer's turn
                        self.computer_turn()  # And allow computer to make
computer's move

            def button_update(self, row, column):
                # Change symbols to bone and paw
                symbol = "🦴" if self.current_player == "X" else "🐾"
                # Update button and disable it
                self.buttons[row][column].config(text=symbol, state=DISABLED)

            def winner_check(self):
                for i in range(3):
                    # Horizontal: check if all symbols in the current row are the
same and not empty
                    if self.board[i][0] == self.board[i][1] == self.board[i][2]
!= "":
```

```
                        return True  # TRUE if row has the same elements and is
not empty

                    # Vertical: check if all symbols in the current column are
the same and not empty
                    if self.board[0][i] == self.board[1][i] == self.board[2][i]
!= "":
                        return True  # TRUE if column has the same elements and
is not empty

                # Diagonal: check if all symbols top-left to bottom-right are the
same and not empty
                if self.board[0][0] == self.board[1][1] == self.board[2][2] !=
"":
                    return True  # TRUE if diagonal from top-left to bottom-right
has the same elements and is not empty

                # Diagonal: check if all symbols top-right to bottom-left are the
same and not empty
                if self.board[0][2] == self.board[1][1] == self.board[2][0] !=
"":
                    return True  # TRUE if diagonal from top-right to bottom-left
has the same elements and is not empty

                # If none of the conditions above are met, meaning no one has
won, return False
                return False  # False if there is not a winning condition

            def tie_check(self):
                # Iterate over rows and columns
                for i in range(3):
                    for j in range(3):
                        if self.board[i][j] == "":  # Check for empty slot
                            return False  # Game is not a tie if there is an
empty slot
                return True  # Game is a tie if no empty slot are found

            def computer_turn(self):
                # Look for empty slots in the 3 by 3 grid
                empty_slot = [(i, j) for i in range(3)
                            for j in range(3) if self.board[i][j] == ""]

                # Choose empty slot randomly
                if empty_slot:
                    row, column = random.choice(empty_slot)
                    # Place symbol for computer in randomly chosen slot
                    self.board[row][column] = "O"
                    # Update onto the corresponding button on the board
                    self.button_update(row, column)

                    # Check for potential win by computer
                    if self.winner_check():
                        messagebox.showinfo(title = "TicTacBone", message = "You
lost! Try again next time.")
                        self.window.destroy()
                    # Check for a tie
                    elif self.tie_check():
                        # Sends message to show tie if result is a tie
                        messagebox.showinfo(title = "TicTacBone", message = "It's
a tie! Try again next time.")
                        self.window.destroy()
                    # Else, allow user to conduct their next turn
                    else:
                        self.current_player = "X"
                        self.user_turn = True
        TicTacBoneGUI()
        self.activeTimer = 0
        pygame.mixer.music.stop()
        pygame.mixer.music.load(f"{p.parent}/hub-music.mp3")
        pygame.mixer.music.play(loops = -1)

    def loveGame(self):
        class HeartGuessing:
            def __init__(self):
                # Initalize game
                self.master = Tk()
                self.master.title("Guess the Number of Hearts Game")
                # Generate a random number of hearts from 1-10
                self.no_of_hearts = random.randint(1, 10)
                self.attempts = 0
                self.no_of_attempts = 3 # Number of allowed attempts for player

                # Add GUI components
                self.label = Label(
                    self.master, text="Enter your guess for the number of hearts
❤️ (enter a number from 1-10):")
                self.label.pack()

                self.entry = Entry(self.master)
                self.entry.pack()

                self.guess_button = Button(
                    self.master, text="Guess", command=self.guess_hearts)
                self.guess_button.pack()

                self.window.mainloop()

            # Handle the user's guess
            def guess_hearts(self):
                try:
                    guess = int(self.entry.get())
                    self.check_guess(guess)
                # Display an error message if the user's input is not a valid
number
                except ValueError:
                    messagebox.showerror("Error", "Please enter a valid number.")

            # Check the user's guess
            def check_guess(self, guess):
                self.attempts += 1

                if guess == self.no_of_hearts:
                    # Display message to inform win if user guesses correctly
                    messagebox.showinfo(message=f"Congratulations You guessed
{self.no_of_hearts} number of hearts in {self.attempts} attempts. You win! Love
max will increase by 1.")
```

```
                    crittercraft.critterLoveMax += 1
                    crittercraft.canvas.delete("lov")
                    crittercraft.canvas.create_text(600, 100, text = f"Love:
{self.critterLove} / {self.critterLoveMax}", font = "Helvetica 20", tag = "lov")
                    self.master.destroy()
                    exit()
                elif guess < self.no_of_hearts:
                    # Prompt the user to go higher if the guess is too low
                    messagebox.showinfo("Incorrect", "Try again. Go higher.")
                else:
                    # Prompt the user to go lower if the guess is too high
                    messagebox.showinfo("Incorrect", "Try again. Go lower.")

                if self.attempts == self.no_of_attempts:
                    # Display message to inform user if they are out of attempts
                    messagebox.showinfo(message=f"Sorry, you've out of attempts.
The correct number of hearts was {self.no_of_hearts} ❤️.")
                    self.master.destroy()

        HeartGuessing()
        self.activeTimer = 0
        pygame.mixer.music.stop()
        pygame.mixer.music.load(f"{p.parent}/hub-music.mp3")
        pygame.mixer.music.play(loops = -1)
```

Jueun Kang

In continuing to work on CritterCraft, I have completed writing the code for two out of three of our mini games: Guess Hearts and TicTacBone. I then worked in collaboration with Jason to import the music file into our code and get it to play in the game. I have also been writing up the README file and updating it as we make iterations to our code – for the output example, I have been going through the game to take screenshots to include in the README, to guide the reader.

Code:

```python
import random
import tkinter as tk
from tkinter import messagebox

# Create class for the Heart Guessing Game
class HeartGuessing:
   def __init__(self, master):
       # Initalize game
       self.master = master
       self.master.title("Guess the Number of Hearts Game")

       # Generate a random number of hearts from 1-10
       self.no_of_hearts = random.randint(1, 10)
       self.attempts = 0
       self.no_of_attempts = 3 # Number of allowed attempts for player

       # Add GUI components
       self.label = tk.Label(
           master, text="Enter your guess for the number of hearts ❤️ (enter a number
from 1-10):")
       self.label.pack()

       self.entry = tk.Entry(master)
       self.entry.pack()

       self.guess_button = tk.Button(
           master, text="Guess", command=self.guess_hearts)
       self.guess_button.pack()

   # Handle the user's guess
   def guess_hearts(self):
       try:
           guess = int(self.entry.get())
           self.check_guess(guess)
       # Display an error message if the user's input is not a valid number
       except ValueError:
           messagebox.showerror("Error", "Please enter a valid number.")

   # Check the user's guess
   def check_guess(self, guess):
       self.attempts += 1
```

```
        if guess == self.no_of_hearts:
            # Display message to inform win if user guesses correctly
            messagebox.showinfo(message=f"Congratulations You guessed
{self.no_of_hearts} number of hearts in {self.attempts} attempts. You earned 1 ❤️")
            self.master.destroy()
            exit()
        elif guess < self.no_of_hearts:
            # Prompt the user to go higher if the guess is too low
            messagebox.showinfo("Incorrect", "Try again. Go higher.")
        else:
            # Prompt the user to go lower if the guess is too high
            messagebox.showinfo("Incorrect", "Try again. Go lower.")

        if self.attempts == self.no_of_attempts:
            # Display message to inform user if they are out of attempts
            messagebox.showinfo(message=f"Sorry, you've out of attempts. The correct
number of hearts was {self.no_of_hearts} ❤️.")
            self.master.destroy()

# Function to run game
def main():
    root = tk.Tk()
    game = HeartGuessing(root)
    root.mainloop()

# Entry point of program
if __name__ == "__main__":
    main()
```

```
import tkinter as tk
from tkinter import messagebox  # Import tkinter.messagebox
import random
import time


class TicTacBoneGUI:
    def __init__(self):  # Set up intial state of tictacbone
        self.window = tk.Tk()  # Create a window
        self.window.title("Tic-Tac-Bone")  # Set window title

        # Initialize the tictacbone board as a 3 by 3 grid
        self.board = [["", "", ""], ["", "", ""], ["", "", ""]]
        # Initialize list to store buttons
        self.buttons = [[None, None, None], [
            None, None, None], [None, None, None]]
        # The current player is 'X'
        self.current_player = "X"
        # Store game outcome to 0 for on-going
        self.outcome = 0

        # Create buttons for the 3 by 3 tictacbone grid
        for i in range(3):
            for j in range(3):
                self.buttons[i][j] = tk.Button(self.window, text="", font=("Arial",
24),
                                              width=5, height=2, command=lambda row=i,
column=j: self.on_click(row, column))
                # Put buttons in the grid
                self.buttons[i][j].grid(row=i, column=j)

        self.computer = "O"  # Initalize symbol for computer
        self.user_turn = True  # Track whether it is the player's turn

        self.window.mainloop()

    def on_click(self, row, column):
        # Player's turn when player can click, button is empty, and there is no winner
yet
        if self.user_turn and self.board[row][column] == "" and not
self.winner_check():
            # Player prevented from playing when computer's turn
            self.user_turn = False
            # Update board with the current player's symbol
            self.board[row][column] = self.current_player
            # Update onto the corresponding button on the board
            self.button_update(row, column)

            if self.winner_check():
                # Sends message to show win if player wins
                messagebox.showinfo(title = "TicTacBone", message = "You earn 1 🦴!")
                self.game_reset()  # Resets the game
            elif self.tie_check():
                # Sends message to show tie if result is a tie
                messagebox.showinfo(title = "TicTacBone", message = "It's a tie! Play
again to earn 🦴")
                self.game_reset()  # Reset the game
            else:
                self.current_player = self.computer  # If it is currently the
computer's turn
                self.computer_turn()  # And allow computer to make computer's move

    def button_update(self, row, column):
        # Change symbols to bone and paw
        symbol = "🦴" if self.current_player == "X" else "🐾"
        # Update button and disable it
        self.buttons[row][column].config(text=symbol, state=tk.DISABLED)

    def winner_check(self):
        for i in range(3):
            # Horizontal: check if all symbols in the current row are the same and not
empty
            if self.board[i][0] == self.board[i][1] == self.board[i][2] != "":
                return True  # TRUE if row has the same elements and is not empty

            # Vertical: check if all symbols in the current column are the same and not
empty
            if self.board[0][i] == self.board[1][i] == self.board[2][i] != "":
                return True  # TRUE if column has the same elements and is not empty

        # Diagonal: check if all symbols top-left to bottom-right are the same and not
empty
        if self.board[0][0] == self.board[1][1] == self.board[2][2] != "":
            return True  # TRUE if diagonal from top-left to bottom-right has the same
elements and is not empty

        # Diagonal: check if all symbols top-right to bottom-left are the same and not
empty
        if self.board[0][2] == self.board[1][1] == self.board[2][0] != "":
            return True  # TRUE if diagonal from top-right to bottom-left has the same
elements and is not empty

        # If none of the conditions above are met, meaning no one has won, return False
        return False  # False if there is not a winning condition

    def tie_check(self):
```

```python
        # Iterate over rows and columns
        for i in range(3):
            for j in range(3):
                if self.board[i][j] == "":  # Check for empty slot
                    return False  # Game is not a tie if there is an empty slot
        return True  # Game is a tie if no empty slot are found

    def game_reset(self):
        self.window.destroy()  # Close the window
        start_new_game = TicTacBoneGUI()  # Start a new game

    def computer_turn(self):
        # Look for empty slots in the 3 by 3 grid
        empty_slot = [(i, j) for i in range(3)
                      for j in range(3) if self.board[i][j] == ""]

        # Choose empty slot randomly
        if empty_slot:
            row, column = random.choice(empty_slot)
            # Place symbol for computer in randomly chosen slot
            self.board[row][column] = "O"
            # Update onto the corresponding button on the board
            self.button_update(row, column)

            # Check for potential win by computer
            if self.winner_check():
                messagebox.showinfo(title = "TicTacBone", message = "You lost! Play
again to earn 🦴")
                self.window.destroy()
                print(0)
            # Check for a tie
            elif self.tie_check():
                # Sends message to show tie if result is a tie
                messagebox.showinfo(title = "TicTacBone", message = "It's a tie! Play
again to earn 🦴")
                self.window.destroy()
                print(0)
            # Else, allow user to conduct their next turn
            else:
                self.current_player = "X"
                self.user_turn = True
                print(1)
```

```
if __name__ == "__main__":
   TicTacBoneGUI()  # Create GUI
```

Jason Ngo

This week I coded music to change upon certain health levels. Additionally I added a flag to the internal timer so that it stops when the player is caring or playing. I also was able to successfully link the mini-game outcomes to the stats of a critter (wins will increase the respective stat by 1), and have the game successfully operate on a hub-play-hub or a hub-care-hub loop. The game is mainly finished now, so all we need to do is finish the readme and maybe work on aesthetics.