Odd Semester (2022)

BINUS UNIVERSITY

BINUS INTERNATIONAL

Algorithm and Programming Final Project

(Individual Work)

Student Information:

Surname: Munthe **Given Name:** Abigail **Student ID:** 2602109883

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's 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: Priscilla Abigail Munthe

A. Project Specification

I. Background

In recent years, the use of Discord as a communication platform has grown rapidly, particularly in the wake of the COVID-19 pandemic. As more and more people turn to online communities for connection and collaboration, the need for powerful, efficient bots to enhance the Discord experience has become increasingly apparent. In light of this trend, I have chosen to undertake a project to create a Discord bot that can provide a wide range of useful functionality to users. The purpose of this project is to explore the capabilities of Discord bot development and to create a tool that can improve the user experience of the platform. The bot will be designed to be easy to use. The development process will involve researching and utilizing various Discord API functions and libraries, as well as implementing features and functionality based on user feedback and testing. Overall, this project aims to contribute to the growing Discord community by providing a valuable tool that can enhance the way people interact and communicate on the platform.

II. Bot Description

The Discord bot is named "Angry Timmy" and it was developed as a fun and interactive tool for users to engage with on Discord. The bot was built using Nextcord library, Akinator library, Asyncio library, and Python's Random module. The use of these libraries and modules allowed for efficient and effective development, while also providing a wide range of functionality. The bot has three main features, which include a Battleship game, a Magic 8-ball, and an Akinator bot. The battleship game allows users to play game of Battleship with their friends on the server, providing a fun and interactive way for users to engage with each other. The Magic 8-ball feature allows users to ask the bot a question and receive a random fortune-telling response, which adds an element of surprise and excitement to conversations. The Akinator bot feature allows users to play a game where the bot guesses the name of either a fictional or non-fictional character by asking a series of questions. This feature adds an element of fun, entertainment and also a way to test the bot's knowledge. These features were developed to be easy to use and provide an enjoyable experience for users on the server.

III. Features

- **Battleship game**

  This feature allows users to play a game of battleship with each other on the server. The game requires two players and can be initiated by issuing a command to the bot. Once the game starts, players take turns guessing where the ships are located on the opponent's board. The bot keeps track of the guesses and displays the results of each turn, allowing players to strategize and plan their next move. The game ends when all of the ships are sunk and then the bot will announce the winner (the player with more points).

- **Magic 8ball**

  This feature allows users to ask the bot a question and receive a random fortune-telling response. The feature is activated by issuing a command to the bot followed by a question. The bot then randomly selects one of the classic responses of a magic 8 ball and sends it as a reply to the user. The responses include answers like "It is certain", "Without a doubt", "You may rely on it", "Outlook good", "Yes – definitely", "As I see it, yes", "Most likely" and "Signs point to yes" and others.

- **Akinator bot**

  This feature allows users to play a game where the bot attempts to guess the name of either a fictional or non-fictional character by asking a series of questions. The Akinator function utilizes the Akinator library to generate a series of questions that are designed to narrow down the possibilities and ultimately identify the character in question. Users can interact with the bot by answering the questions and providing feedback on the guesses. The bot will continue to ask questions and make guesses until it correctly identifies the character or the user decides to end the game. The Akinator feature provides an entertaining and engaging way for users to test their knowledge of fictional characters, and also a way for people to bond and have fun together trying to outsmart the bot. This feature can also be used as a party game, where players take turns to think of characters for the bot to guess.

IV. Libraries/Modules

- **Nextcord**

  The Nextcord library is a library that enables developers to create Discord bots. It provides a set of tools and functions that simplify the process of interacting with the Discord API and allows for the creation of powerful and efficient bots. In this bot project, I used the nextcord library to handle the various commands and events that were required for the bot to function properly. The library provided a simple and straightforward way to interact with the Discord API, allowing me to easily implement features such as the Battleship game, Magic 8-ball, and Akinator bot. Additionally, nextcord also provides features like caching, rate-limiting, and error handling which helped me in making the bot more stable and efficient.

- **Akinator**

  The Akinator library is a library that allows developers to create games and applications that can guess the name of fictional characters based on a series of questions. The library utilizes a set of algorithms and a large database of characters to generate questions and make guesses. In this bot project, I used the Akinator library to implement the Akinator feature of the bot which allows users to play a game where the bot attempts to guess the name of a fictional character based on a series of questions. The library provided an easy way to access the database of characters and generate questions, allowing me to quickly implement this feature without having to build the algorithm and the database from scratch. The library also provided features like error handling and feedback, which helped in making the game more engaging for the users.

- **Asyncio**

  This library allows developers to write concurrent code that can run without blocking the execution of other parts of the program. In this bot project, I used the Asyncio library to handle multiple tasks simultaneously and to improve the performance of the bot. The library provides an easy way to handle events and commands in an asynchronous manner, which allowed the bot to respond to multiple requests at the same time. For example, the Asyncio library was used to handle multiple games of Battleship at the same

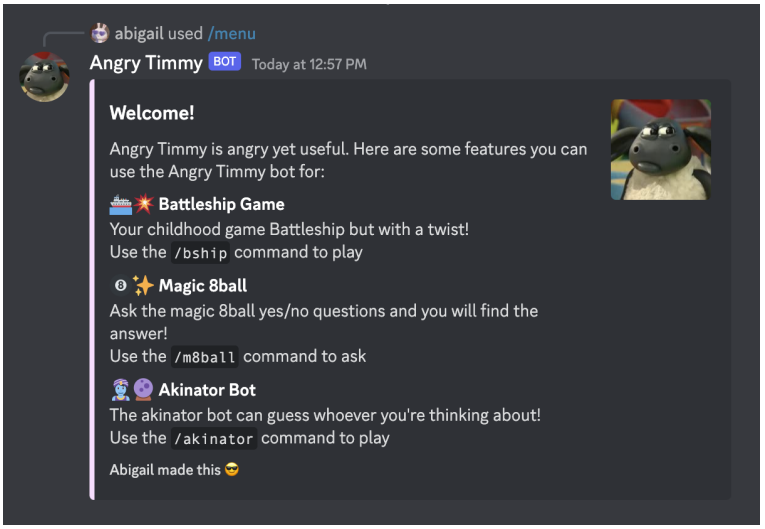
time, allowing multiple users to play the game concurrently. Additionally, it was also used to handle multiple Akinator games simultaneously, this allowed users to play with the bot at the same time without any delays. The Asyncio library was key to making the bot more efficient and responsive to user requests.

- **Random**

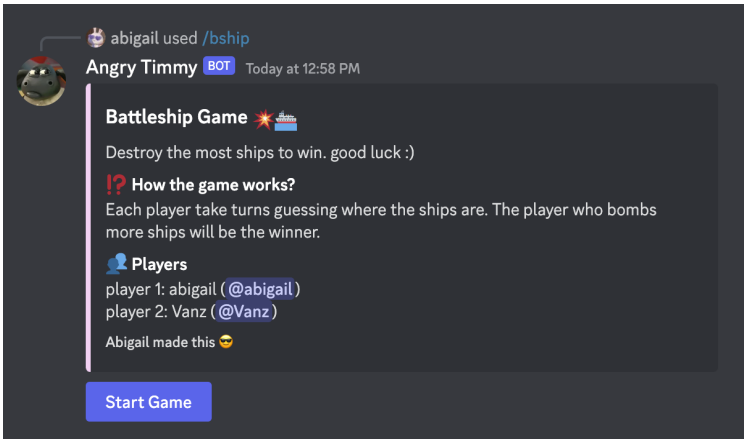
The random module is a built-in Python library that provides a suite of functions for generating random numbers and selecting random items from a collection. This module is used to incorporate randomness and unpredictability into the bot's features. In this bot project, the random module was utilized in the magic 8-ball feature, which allows users to ask the bot a question and receive a random fortune-telling response. The random module was used to randomly select one of the pre-defined responses from a list, providing a different answer each time the feature is used.

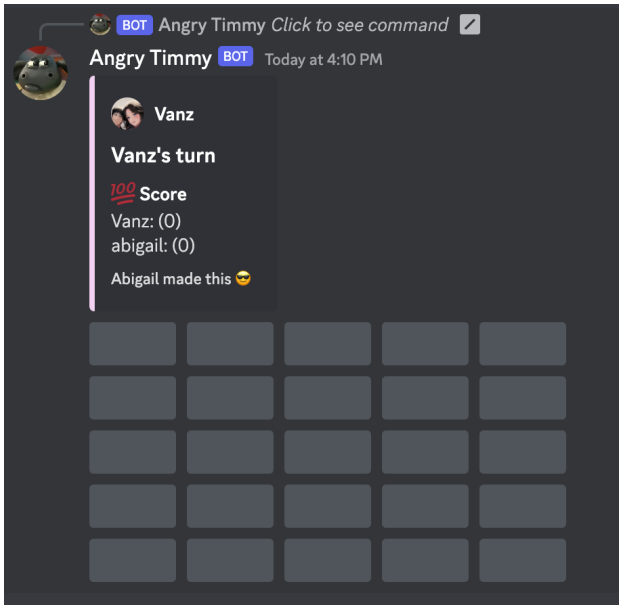
V. Screenshots of the Bot

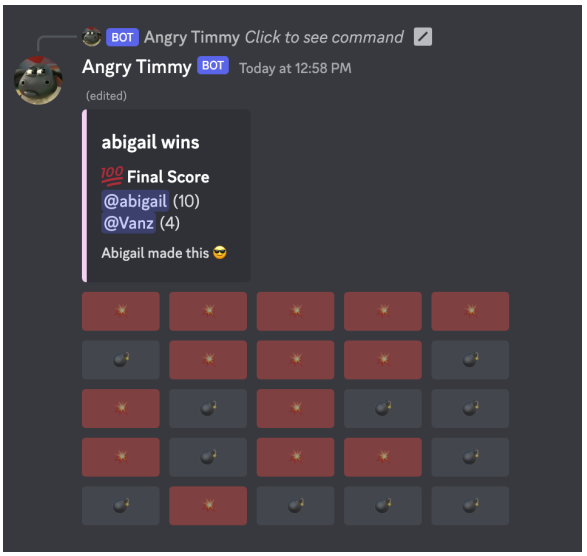
- **Main Menu**



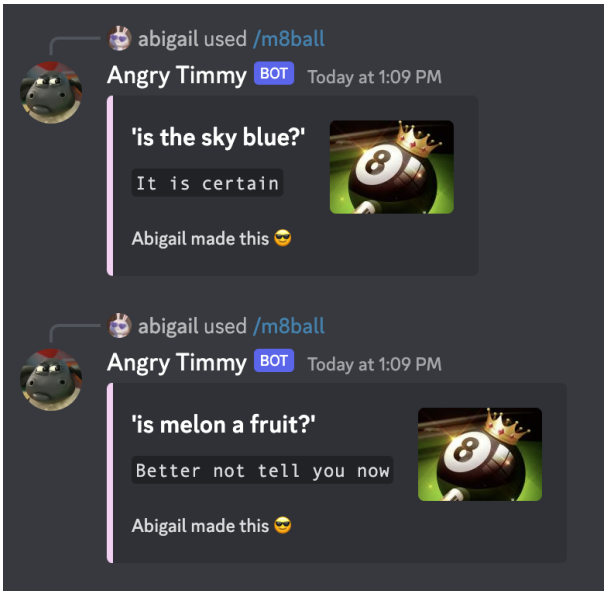
- **Battleship Game**

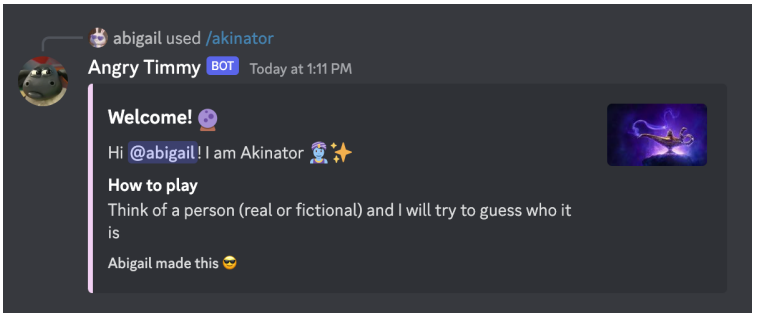


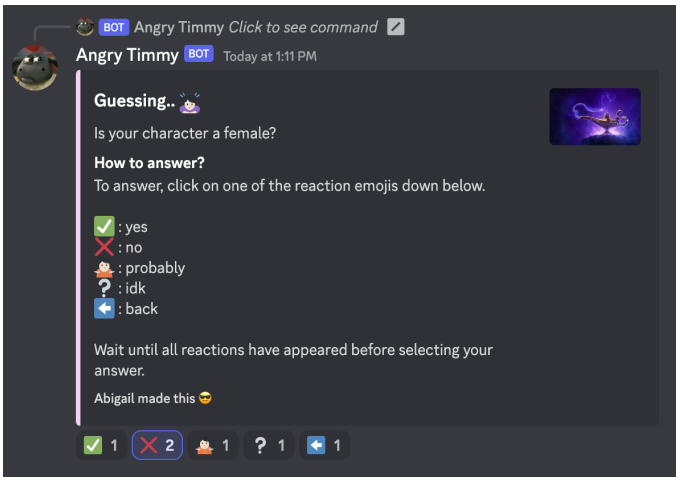




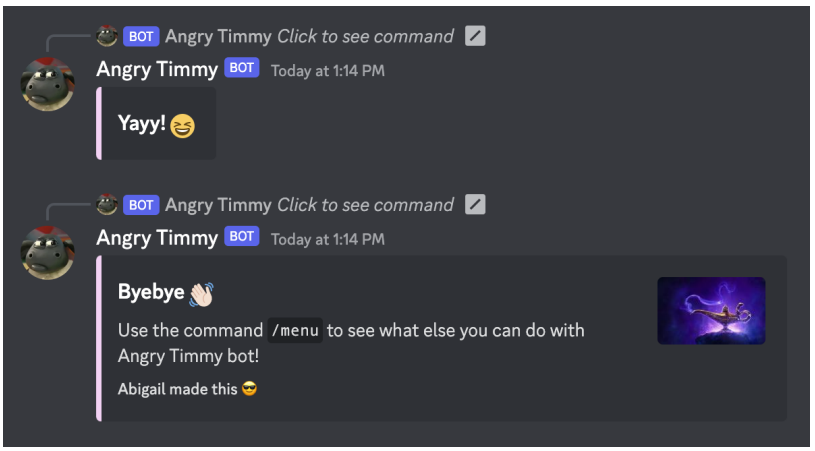
- **Magic 8ball**




abigail used /akinator
Angry Timmy BOT Today at 1:11 PM
Welcome!
Hi @abigail! I am Akinator
How to play
Think of a person (real or fictional) and I will try to guess who it is
Abigail made this

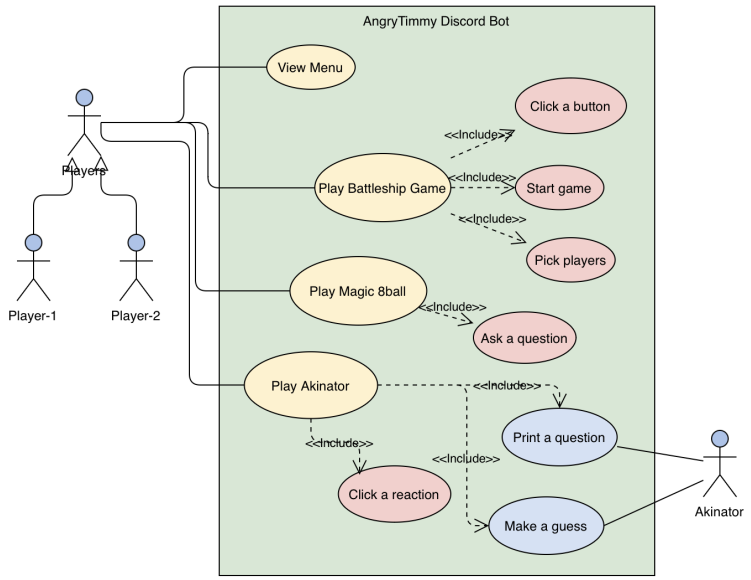

BOT Angry Timmy Click to see command
Angry Timmy BOT Today at 1:11 PM
Guessing..
Is your character a female?
How to answer?
To answer, click on one of the reaction emojis down below.
: yes
: no
: probably
: idk
: back
Wait until all reactions have appeared before selecting your answer.
Abigail made this
1 2 1 1 1


BOT Angry Timmy Click to see command
Angry Timmy BOT Today at 1:14 PM
My guess is..
Harry Potter!
Did I guess the right person?
2 1

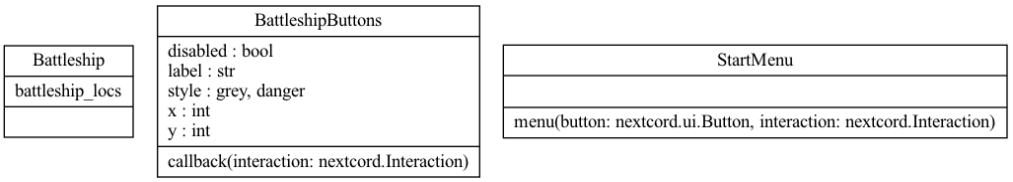

BOT Angry Timmy Click to see command
Angry Timmy BOT Today at 1:14 PM
Yayy!
BOT Angry Timmy Click to see command
Angry Timmy BOT Today at 1:14 PM
Byebye
Use the command /menu to see what else you can do with Angry Timmy bot!
Abigail made this

B. Diagrams

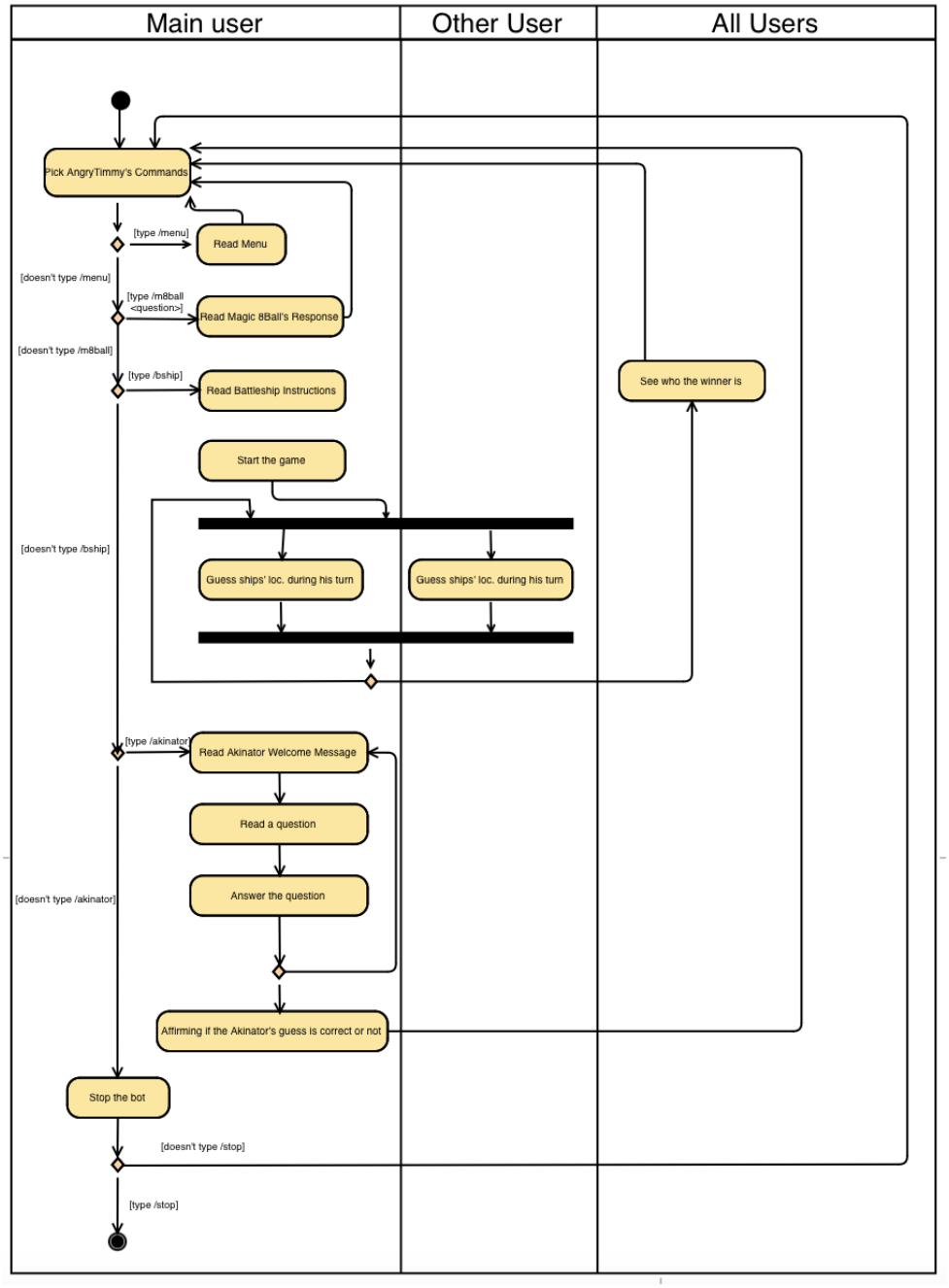
I. Use-case Diagram



II. Class Diagram



III. Activity Diagram



C. Program/Source Code

I start by importing several modules which are nextcord (a python library used to create discord bot), nextcord.ext.commands (a submodule within the nextcord library that allowed me to create commands that can be executed by users in a discord server), akinator (used to interact with the Akinator API), asyncio (a python library used for asynchronous programming which allows for multiple operations happening simultaneously without waiting for one to finish before moving on to the next), and lastly random which is a built-in library that I used to select a random element from a list.

```
import nextcord
from nextcord.ext import commands
import akinator as ak
import asyncio
import random
```

Here I define a variable called guilds to store all guild ids of the servers that the bot is in.

```
guilds = []
```

Next I assigned global variables to keep track of the players' info.

```
player_names = [None,None]
player_ids = [None,None]
player_avatar_urls = [None,None]
player_scores = [0,0]
who_is_playing = None
number_attempts = 0
```

```
# calls the bot and enables all intents
bot = commands.Bot(intents=nextcord.Intents.all())
```

Here I added the @bot_event decorator which tells the program that the function that follows is an event that should be triggered when the bot is ready. The decorator is then followed by the built-in event @on_ready which is triggered when the bot is ready to use. The function contains a single line of code which is an asynchronous function that prints "Bot is Online!" to the console to indicate that the bot is ready to be used.

```
# when bot is ready to be used -> prints bot is online
@bot.event
async def on_ready():
   print('Bot Is Online!')
```

This code defines the menu function which, when called, prints an embed containing a menu displaying various features that the bot provides, such as playing a Battleship game, using a Magic 8ball, and playing with an Akinator bot.

```
# if the /menu command is called, the menu function is used
@bot.slash_command(guild_ids = guilds, description="Main menu of Angry Timmy
bot")
async def menu(ctx):

    # create an embed containing the menu information
    menu = nextcord.Embed(
        title="Welcome!",
        description="Angry Timmy is angry yet useful. Here are some features
you can use the Angry Timmy bot for:",
        color=0xfbdcff
    )
    menu.add_field(name="🚢💥 Battleship Game", value="Your childhood game
Battleship but with a twist!\nUse the `/bship` command to play",
inline=False)
    menu.add_field(name="🎱✨ Magic 8ball", value="Ask the magic 8ball yes/no
questions and you will find the answer!\nUse the `/m8ball` command to ask",
inline=False)
    menu.add_field(name="🧞🔮 Akinator Bot", value="The akinator bot can guess
whoever you're thinking about!\nUse the `/akinator` command to play",
inline=False)

menu.set_thumbnail(url="https://i.pinimg.com/originals/00/59/bc/0059bc0ab9864
498107b173f6391d59e.jpg")
    menu.set_footer(text="Abigail made this 😎")

    # sends and prints the menu embed
    await ctx.send(embed=menu)
```

The first feature is the Battleship game feature and for this feature, I used three classes.

This first class manages the start menu of the game. This class includes a method called ‘menu’ which declares three global variables (player_names, player_scores, who_is_playing) when called. Then it creates an embed object called playing containing information about the game and the involved players. Lastly, the code will create an instance of the ‘Battleship’ class and sends the ‘playing’ embed and the ‘view’ instance to the interaction.

```
# class that manages the start menu
class StartMenu(nextcord.ui.View):
```

```
    def __init__(self):
        super().__init__()

    # displays a button used to start the battleship game -> starts the game
    @nextcord.ui.button(label="Start Game",
style=nextcord.ButtonStyle.blurple)
    async def menu(self, button: nextcord.ui.Button, interaction:
nextcord.Interaction) :

        # global variables
        global player_names # player names
        global player_scores # players' scores
        global who_is_playing # current player (player 1/2)

        # embed containing info about who is playing
        playing = nextcord.Embed(
            title=f"{player_names[who_is_playing-1]}'s turn",
            color=0xf9cff5
        )
        playing.set_author(
            name=player_names[who_is_playing-1],
            icon_url=player_avatar_urls[who_is_playing-1]
        )
        playing.add_field(
            name="💯 Score",
            value=f"{player_names[0]}:
({player_scores[0]})\n{player_names[1]}: ({player_scores[1]})"
        )
        playing.set_footer(text="Abigail made this 😎")

        # displays the battleship game (buttons)
        view = Battleship()

        # send and prints the who is playing embed
        await interaction.send(embed=playing, view=view)
```

This next class represents the buttons used in the battleship game. It has an __init__ method that takes two arguments, x and y and initializes the class by calling the __init__ method of the parent class using the super() function and setting the style and label of the button.

```
# class that represents the buttons
class BattleshipButtons(nextcord.ui.Button):
```

```
    def __init__(self, x: int, y: int):
        super().__init__(style=nextcord.ButtonStyle.secondary, label="\u200b",
row=x)
        self.x = x # x-coordinate of the buttons
        self.y = y # y-coordinate of the buttons
```

The class also has a method 'callback', which is executed when the associated button is pressed. The method asserts that the 'view' attribute is available and assigns it to a variable 'view'. It declares a number of global variables such as who_is_playing, player_ids, player_scores, player_names, player_avatar_urls, number_attempts. The callback function checks whose turn it is by comparing the user id of the interaction with the player_ids list, if it is the correct user's turn, it checks if the button pressed is a miss or a hit by checking the value in the view.battleship_locs matrix. it adds the number of attempts count by 1, and switch the players. The callback function creates an embed with the current player's turn, scores, and the footer.

```
    # function that is executed when its asociated button is pressed
    async def callback(self, interaction: nextcord.Interaction):
        assert self.view is not None # makes sure that self.view is available
to use
        view = self.view # stores self.view into the variable view -> needed
to display battleship buttons

        # global variables
        global who_is_playing # whos turn it is
        global player_ids # list of player ids
        global player_scores # list of player scores
        global player_names # list of player names
        global player_avatar_urls # list of player avatars
        global number_attempts # used to keep track of number of attempts

        # the loop runs/continues only if the user who clicked is the one
whose turn it is
        if player_ids[who_is_playing-1] == interaction.user.id:

            # adds the number of attempts count by 1
            number_attempts+=1

            # if the location of the button clicked is 0 in the matrix -> miss
            if(not view.battleship_locs[self.x][self.y]):
                self.style = nextcord.ButtonStyle.grey
                self.label = "💣"
                self.disabled = True
```

```
            else:
                # if the location of the button clicked is 1 in the matrix ->
hit
                self.style = nextcord.ButtonStyle.danger
                self.label = "💥"
                self.disabled = True

                # adds the player's score by 1
                player_scores[who_is_playing-1]+=1

             # switch players
            if who_is_playing == 1:
                who_is_playing = 2
            else:
                who_is_playing = 1

            # embed containing which player's turn it is and their scores
            embed = nextcord.Embed(
                title=f"{player_names[who_is_playing-1]}'s turn",
                color=0xf9cff5
            )
            embed.set_author(
                name=player_names[who_is_playing-1],
                icon_url=player_avatar_urls[who_is_playing-1]
            )
            embed.add_field(
                name="💯 Score",
                value=f"{player_names[0]}:
({player_scores[0]})\n{player_names[1]}: ({player_scores[1]})"
            )
            embed.set_footer(text="Abigail made this 😎")

            content = None
```

If it is not the correct user's turn, it sends an embed with the current player's turn, scores, the footer and a content message of "Oops it's not your turn 😅".

```
        else:
            # embed printed if its not the user's turn
            embed = nextcord.Embed(
                title=f"{player_names[who_is_playing-1]}'s turn",
                color=0xf9cff5
            )
```

```
            embed.set_author(
                name=player_names[who_is_playing-1],
                icon_url=player_avatar_urls[who_is_playing-1]
            )
            embed.add_field(
                name="💯 Score",
                value=f"{player_names[0]}:
({player_scores[0]})\n{player_names[1]}: ({player_scores[1]})"
            )
            embed.set_footer(text="Abigail made this 😎")

            content = "Oops it's not your turn 😅"
```

If the number of attempts is 25, it checks the scores of the players and generates an embed with the winner or draws.

```
        # if the number of attempts already reaches 25 (the number of buttons)
        if number_attempts == 25:

            # embed generated if player 1 wins
            if player_scores[0]>player_scores[1]:
                who_wins = 1
                embed = nextcord.Embed(
                    title=f"{player_names[who_wins-1]} wins",
                    color=0xf9cff5
                )
                embed.add_field(
                    name="💯 Final Score", # final scores
                    value=f"<@{player_ids[0]}>
({player_scores[0]})\n<@{player_ids[1]}> ({player_scores[1]})"
                )
                embed.set_footer(text="Abigail made this 😎")


            # embed generated if player 2 wins
            elif player_scores[1]>player_scores[0]:
                who_wins = 2
                embed = nextcord.Embed(
                    title=f"{player_names[who_wins-1]} wins",
                    color=0xf9cff5
                )
                embed.set_author(
                    name=player_names[who_is_playing-1],
                    icon_url=player_avatar_urls[who_is_playing-1]
```

```
                )
                embed.add_field(
                    name="💯 Final Score", # final scores
                    value=f"<@{player_ids[0]}>
({player_scores[0]})\n<@{player_ids[1]}> ({player_scores[1]})"
                )
                embed.set_footer(text="Abigail made this 😎")
            else:
                # embed generated if the score is a tie
                embed = nextcord.Embed(
                    title="it's a tie",
                    color=0xf9cff5
                )
                embed.add_field(
                    name="💯 Final Score", # final scores
                    value=f"<@{player_ids[0]}>
({player_scores[0]})\n<@{player_ids[1]}> ({player_scores[1]})"
                )
                embed.set_footer(text="Abigail made this 😎")

            # restarts the scores and number of attempts back to 0
            player_scores = [0,0]
            number_attempts = 0

            # disables all the buttons
            view.stop()

        # prints the wanted embed
        await interaction.response.edit_message(content=content,
embed=embed,view=view)
```

This third and last class manages the battleship game and is responsible for generating the grid of buttons that represent the game board. The class has an __init__ method which initializes the class by calling the __init__ method of the parent class using the super() function. The method also creates a while loop that runs until the number of 1s in the randomly generated binary matrix is between 10 and 15. It creates a 5x5 matrix of random binary numbers, which will be used as a reference for the buttons on the game board. The method also contains a print statement that prints the matrix to the terminal, which can be used as a cheat code for the game.

```
# class that manages the battleship game
class Battleship(nextcord.ui.View):
    def __init__(self):
        super().__init__()
```

The class also has a nested for loop that iterates through the rows and columns of the matrix and adds an instance of the BattleshipButtons class to the game board for each button on the grid. The nested for loop is used to create a 5x5 grid of buttons, each with their respective x and y coordinates.

```
        while True:
            # generates a random binary matrix to be used as a reference for
the buttons
            self.battleship_locs = [[random.randint(0, 1) for _ in range(5)]
for _ in range(5)]
            count = sum(sum(row) for row in self.battleship_locs)
            if 10 <= count <= 15:
                break

        # prints the matrix into the terminal -> cheat code
        print(self.battleship_locs)

        # making the grid of buttons
        for x in range(5):
            for y in range(5):
                self.add_item(BattleshipButtons(x, y))
```

This is the last snippet of code for the battleship game feature. When used, it takes two parameters which are player1 and player 2 which has the type nextcord.Member. These two parameters represents the two players in the game.

```
# slash command that calls the bship function
@bot.slash_command(guild_ids = guilds, description="Battleship game with a
twist")
async def bship(interaction: nextcord.Interaction, player1: nextcord.Member,
player2: nextcord.Member):
```

The command declares global variables such as player_names, player_ids, player_avatar_urls, who_is_playing. The function assigns the player1 and player2 name, id, and avatar to the global variables. It also randomly selects one of the players to go first using the random.randint(1,2) function.

```
    # global variables
    global player_names
    global player_ids
    global player_avatar_urls
    global who_is_playing

    if(player1.name):
```

```
        player_names[0] = player1.name
        player_ids[0] = player1.id
        player_avatar_urls[0] = player1.avatar

    if(player2.name):
        player_names[1] = player2.name
        player_ids[1] = player2.id
        player_avatar_urls[1] = player2.avatar

    # decide who's playing first
    who_is_playing = random.randint(1,2)
```

It also creates an embed object that displays the title, description, and color of the game. It also adds two fields to the embed object, one describing how the game works and another one displaying the players' information.

```
    # embed containing the info of the game
    embed = nextcord.Embed(
        title="Battleship Game 💥🚢",
        description="Destroy the most ships to win. good luck :)",
        color=0xf9cff5
    )

    embed.add_field(name="⁉ How the game works?", value="Each player take
turns guessing where the ships are. The player who bombs more ships will be
the winner.")
    embed.add_field(name="👥 Players", value=f"player 1: {player_names[0]}
(<@{player_ids[0]}>) \n player 2: {player_names[1]} (<@{player_ids[1]}>)",
inline=False)
    embed.set_footer(text="Abigail made this 😎")
```

Here the function creates an instance of the StartMenu class and sends the embed object and the view (which is the instance of the StartMenu class) to the interaction. The interaction sends the start menu embed and start game button to the user who used the command.

```
    # calls the StartMenu class
    view = StartMenu()

    # prints the start menu embed and start game button
    await interaction.send(embed=embed, view=view)
```

This next code snippet is for the second feature of the bot containing a slash command which starts a magic 8ball game. It takes one argument question which is a string representing the question the user

wants to ask. Here I made a list called list_of_ans which contains all the possible answers for the magic 8ball. The command then uses the random.choice() method to randomly select one of the answers from the list_of_ans list and assigns it to the variable selected_ans.

```
# list containing all possible answers
list_of_ans = ["It is certain", "Definitely not", "Reply hazy", "try again",
"As I see it, yes", "Dont count on it", "It is decidedly so", "Ask again
later",
   "My reply is no", "Without a doubt", "Of course", "Better not tell you
now", "My sources say no", "Yes definitely", "Cannot predict now",
   "Outlook not so good", "You may rely on it", "Concentrate and ask again",
"Very doubtful", "Most likely",
   "Outlook good", "Yes", "Signs point to yes"]

# a slash command using the m8ball function
@bot.slash_command(guild_ids = guilds, description="Magic 8ball that answers
your yes/no questions")
async def m8ball(interaction: nextcord.Interaction, question: str): #accepts
a parameter -> the question

   # randomly choose an answer out of the list_of_ans list
   selected_ans = random.choice(list_of_ans)
```

Then it creates an embed object that displays the title, description, and color of the embed. The title displays the question that was asked and the description displays the selected answer from the list_of_ans list. It also sets a thumbnail image and sets the footer.

```
   # makes the answer embed
   embed = nextcord.Embed(
       title=f"'{question}'",
       description=f"`{selected_ans}`",
       color=0xf9cff5
   )

embed.set_thumbnail(url="https://i.pinimg.com/originals/75/d3/df/75d3df783cfe
8380440051924f2ad200.jpg")
   embed.set_footer(text="Abigail made this 😎")
```

The function then sends the embed object to the interaction, which prints the answer to the question that was asked.

```
   # sends and prints the answer embed
   await interaction.send(embed=embed)
```

This code creates a slash command using the Akinator function, which is an AI game that guesses the person the user is thinking of.

```
# a slash command using the akinator function
@bot.slash_command(guild_ids = guilds, description="Akinator bot that guesses anyone you're thinking of")
async def akinator(Interaction: nextcord.Interaction):

    # constant
    AKINATOR_YES = "✅"
    AKINATOR_NO = "❌"
    AKINATOR_PROBABLY = "🤷"
    AKINATOR_IDK = "❓"
    AKINATOR_BACK = "⬅️"
```

The game starts with an introduction embed, which welcomes the user and explains the game's mechanics. The user is prompted to think of a person, real or fictional, and the AI will try to guess who it is.

```
    # intro embed
    intro = nextcord.Embed(title="Welcome! 🔮", description="Hi " + Interaction.user.mention + "! I am Akinator 🧞✨",
                           color=0xf9cff5)
    intro.add_field(name="How to play", value="Think of a person (real or fictional) and I will try to guess who it is", inline=False)

intro.set_thumbnail(url="https://www.designbolts.com/wp-content/uploads/2019/04/aladdin-Lamp_2019-Movie.jpg")
    intro.set_footer(text="Abigail made this 😎")

    # bye embed (outro)
    bye = nextcord.Embed(title="Byebye 👋", description="Use the command `/menu` to see what else you can do with Angry Timmy bot!", color=0xf9cff5)

bye.set_thumbnail(url="https://www.designbolts.com/wp-content/uploads/2019/04/aladdin-Lamp_2019-Movie.jpg")
    bye.set_footer(text="Abigail made this 😎")

    # send intro embed (print)
    await Interaction.send(embed=intro)

    # check if the person clicking on the reaction button is the same person who initiated the game
    def check(reaction, user):
```

```
        return user == Interaction.user

    # convert reaction into sring readable by akinator
    def reaction_to_str(str_reaction):
        if str_reaction == AKINATOR_YES:
            return 'y'
        elif str_reaction == AKINATOR_NO:
            return 'n'
        elif str_reaction == AKINATOR_PROBABLY:
            return 'p'
        elif str_reaction == AKINATOR_IDK:
            return 'idk'
        else:
            return 'b'

    try:
        # calls akinator class -> imports the class from akinator library here
        aki = ak.Akinator()

        # initiate the game
        q = aki.start_game()
```

The game then enters a loop that continues until the Akinator's confidence level reaches 80. The game then sends an embed with a question and four reaction options for the user to choose from (yes, no, probably, idk) as well as a back button. The user then reacts to the embed with one of the options, and the game uses the reaction to generate the next question.

```
        # loops while the akinator's confidence level is still below 80 (after
80 it will guess)
        while aki.progression <= 80:

            # question embed
            question = nextcord.Embed(title="Guessing.. 🧞‍♀️", description=q,
color=0xf9cff5)
            question.add_field(
                name="How to answer?",
                value="To answer, click on one of the reaction emojis down
below.\n\n✅ : yes\n❌ : no\n🤷‍♀️ : probably\n❓ : idk\n⬅️ : back\n\nWait until
all reactions have appeared before selecting your answer.",
                inline=False)

question.set_thumbnail(url="https://www.designbolts.com/wp-content/uploads/20
19/04/aladdin-Lamp_2019-Movie.jpg")
```

```
            question.set_footer(text="Abigail made this 😎")

            # sends the question embed containing the question
            question_sent = await Interaction.send(embed=question)

            # adding reactions into(below) the embed -> so the user only has
to click on the existing reaction options
            await question_sent.add_reaction(AKINATOR_YES)
            await question_sent.add_reaction(AKINATOR_NO)
            await question_sent.add_reaction(AKINATOR_PROBABLY)
            await question_sent.add_reaction(AKINATOR_IDK)
            await question_sent.add_reaction(AKINATOR_BACK)

            try:

                # waits for a reaction by the user
                # function wait_for outputs 2 variables -> reaction, user
                # "reaction, _" makes sure only the reaction is used
                reaction, _ = await bot.wait_for('reaction_add', timeout=30,
check=check)
            except asyncio.TimeoutError:
                await Interaction.send("You took too long to respond:(")
                await Interaction.send(embed=bye)
                return # game is stopped if no input for 30 seconds

            # if the back reaction is clicked
            if str(reaction.emoji) == AKINATOR_BACK:
                try:
                    q = aki.back() # calls back function
                except ak.CantGoBackAnyFurther:
                    await Interaction.send(e) # sends error message (cant go
back any further)
                    continue
            else:
                try:
                    # calls reaction_to_str funtion and assigns the converted
reaction emoji
                        # into the variable akinator_str
                    akinator_str = reaction_to_str(str(reaction.emoji))
                    # calls answer function from the akinator library and uses
akinator_str as a parameter
                    q = aki.answer(akinator_str)
                except ak.InvalidAnswer as e:
```

```
                    await Interaction.send(e) # sends error message that the
answer is invalid
                    continue
```

If the Akinator's confidence level reaches 80, the game exits the loop, and the AI will make a guess. After the game is finished, the user is sent a byebye embed, which contains a message encouraging the user to use the command /menu to see what else they can do with the Angry Timmy bot.

```
        # after confidence level reaches 80
        # signals akinator to make a guess
        aki.win()

        # creates an embed using the keys from the guesses -> the outputs of
the aki.win function
        answer = nextcord.Embed(title="My guess is..",
description=aki.first_guess['name']+"! ✨",
                                color=0xf9cff5)
                                # aki.first_guess['description']
        answer.set_image(url=aki.first_guess['absolute_picture_path'])
        answer.set_footer(text="Did I guess the right person?")
        question_sent = await Interaction.send(embed=answer)
        await question_sent.add_reaction(AKINATOR_YES)
        await question_sent.add_reaction(AKINATOR_NO)

        try:
            # waits for a reaction by the user
            # function wait_for outputs 2 variables -> reaction, user
            # "reaction, _" makes sure only the reaction is used
            reaction, _ = await bot.wait_for('reaction_add', timeout=30,
check=check)
        except asyncio.TimeoutError:
            await Interaction.send("You took too long to respond :(")
            await Interaction.send(embed=bye) # sends bye embed after no
response for 30 seconds
            return

        # if the guess is correct
        if str(reaction.emoji) == AKINATOR_YES:
            yes = nextcord.Embed(title="Yayy! 😆", color=0xf9cff5)
            await Interaction.send(embed=yes)

        # if the guess is incorrect
        else:
```

```
            no = nextcord.Embed(title="OOPSIEE 😅", color=0xf9cff5)
            await Interaction.send(embed=no)
        await Interaction.send(embed=bye)

    # if there is an error
    except Exception as e:
        await Interaction.send(e)
```

And lastly, I added this line of code to start the bot. This method is provided by the Nextcord library. The empty string inside the parentheses is where the bot's token should be placed. The token is a secret key that is used to authenticate the bot and allow it to connect to the server.

```
# calls the run function and uses the discord bot token as a parameter
bot.run("")
```

D. Lessons learned/Reflection

I have learned quite a lot during the process of making this project. One of the main difficulties was learning how to incorporate other libraries and modules into the project. The use of various libraries such as Nextcord, Akinator library, Asyncio library, and random module, allows for efficient and effective development. Even so, they also presented challenges in terms of understanding how to properly utilize them. This required a significant amount of research and experimentation to fully understand the capabilities and limitations of each library.

Additionally, while creating the bot, I encountered several bugs that needed to be overcome. This was a challenging process as it required a lot of debugging and testing to identify and fix the issues. Overall, I learned that incorporating other libraries and modules into a project can be a complex process and requires a lot of patience and perseverance. However, through this experience, I also gained valuable knowledge and experience in debugging and problem-solving which will be useful in future projects.

E. Resources

https://docs.nextcord.dev/en/stable/

https://github.com/nextcord/nextcord

http://stackoverflow.com

http://geeksforgeeks.org

https://www.w3schools.com/python/module_random.asp

https://docs.python.org/3/library/asyncio.html

http://pinterest.com

https://www.youtube.com/watch?v=zid-MVo7M-E