AUTOMATIC MUSIC SYNTHESIZER USING AI

UIT2511 – SOFTWARE DEVELOPMENT PROJECT – II

A PROJECT REPORT

Submitted by

HARSH BANSAL	3122215002036
HARSHA NANDHINI K M	3122215002037
HARSHINI D	3122215002038

SSN COLLEGE OF ENGINEERING,
KALAVAKKAM

NOVEMBER 2023

Sri Sivasubramaniya Nadar College of Engineering

(An Autonomous Institution, Affiliated to Anna University)

BONAFIDE CERTIFICATE

Certified that this project titled “Automatic music synthesizer using AI” is the bonafide work of “Harsh Bansal – 3122215002036, Harsha Nandhini K M – 3122215002037, Harshini D - 3122215002038”, and is submitted for project viva-voce examination held on 27th November 2023.

Signature of examiner

Submitted on ------------------------------

Internal Examiner **External Examiner**

1. ABSTRACT

The "Automatic Music Synthesizer Using AI" stands at the forefront of innovation in the field of automated music composition, introducing a pioneering method driven by genetic algorithms (GAs). Genetic algorithms, inspired by natural selection processes, are employed to iteratively refine, and evolve musical melodies. The emphasis is on adhering to objective criteria such as beauty, harmonics, and rhythmic structure. This approach allows the system to generate compositions that not only meet mathematical criteria but also resonate with the subjective sensibilities of human listeners.

A distinguishing feature of the synthesized compositions is their ability to exhibit intervals that are inherently pleasing to the human ear. Additionally, the algorithm ensures that the resulting musical pieces possess coherent rhythms, and it introduces subtle modifications to bring delightful variations. This demonstrates a nuanced understanding of not just the theoretical aspects of music but also the emotional and perceptual dimensions that contribute to a captivating musical experience.

The success of this study lies in the effective regulation of composition quality and form, achieved through the implementation of sophisticated coding techniques. These techniques provide the algorithm with precise control over musical elements such as tones, intervals, and rhythm. This level of control is essential in crafting compositions that achieve a harmonious blend, contributing to the overall aesthetic appeal of the generated music.

One of the standout aspects of the "Automatic Music Synthesizer Using AI" is its adaptability. Users are empowered to define specific criteria and reference individuals, influencing the selection and creation of compositions. This customization adds a personal touch to the automated music generation process, allowing for a diverse range of musical outputs that cater to individual creative preferences.

In essence, this project represents a creative fusion of music theory and computational intelligence. By navigating the expansive space of musical possibilities, the algorithm not only captures the fundamental aspects of musical aesthetics but also adapts to the unique preferences of users. It emerges as a promising tool for musicians and composers seeking automated yet inspiring music generation, offering a bridge between the structured world of algorithms and the nuanced realm of artistic expression. The "Automatic Music Synthesizer Using AI" exemplifies the harmonious convergence of art and technology, presenting a novel and accessible avenue for exploration in the ever-evolving landscape of automated music composition.

2. INTRODUCTION

The "Automatic Music Synthesizer Using AI" represents a groundbreaking foray into the fusion of music theory and computational intelligence, offering an innovative solution for the automated generation of musical compositions. Harnessing the power of genetic algorithms (GAs), this system pioneers a sophisticated approach to refining and evolving musical melodies. With a keen focus on objective criteria such as beauty, harmonics, and rhythmic structure, the synthesizer produces compositions that not only meet mathematical standards but also resonate with the subjective preferences of human listeners. In this technological endeavor, the study aims to bridge the gap between the structured world of algorithms and the nuanced realm of artistic expression, providing a novel avenue for musicians and composers seeking automated yet inspiring music generation.

At its core, the success of the "Automatic Music Synthesizer Using AI" lies in its ability to effectively regulate composition quality and form. This is achieved through the application of advanced coding techniques, which afford the algorithm precise control over essential musical elements, including tones, intervals, and rhythm. The system's adaptability emerges as a standout feature, allowing users to define specific criteria and reference individuals, thereby exerting influence over the selection and creation of compositions. This personalized touch enhances the automated music generation process, contributing to a diverse array of musical outputs that cater to individual creative preferences.

In this symbiotic marriage of art and technology, the project showcases not only a profound understanding of musical intricacies but also an acute awareness of the emotional and perceptual dimensions that contribute to a captivating musical experience. As a promising tool for musicians seeking inspiration and composers navigating the evolving landscape of automated composition, the "Automatic Music Synthesizer Using AI" exemplifies the harmonious convergence of human creativity and computational prowess.

MOTIVATION

The motivation behind the development of the "Music Synthesizer Using AI" stems from a desire to explore the intersection of music theory and cutting-edge computational intelligence. Traditional approaches to music composition often rely on the expertise and creativity of human composers. However, with the advancements in artificial intelligence, there is a unique opportunity to leverage algorithms to automate and enhance the creative process. This project is motivated by the belief that the marriage of music and AI can open up new possibilities for artistic expression, offering musicians and composers a tool that not only accelerates the composition process but also introduces innovative and inspiring musical outcomes.

The use of genetic algorithms in the synthesis of music introduces an intriguing dimension to the project's motivation. Genetic algorithms, inspired by natural selection, bring an evolutionary aspect to the generation of musical compositions. The motivation here lies in exploring whether an algorithmic approach can not only meet objective criteria such as beauty, harmonics, and rhythmic structure but also capture the subjective nuances that make music a deeply personal and emotional experience. This exploration reflects a curiosity to understand how computational methods can contribute to the richness and diversity of musical expression.

Additionally, the motivation extends to providing musicians and composers with a flexible and customizable tool. The adaptability of the algorithm, allowing users to define specific criteria and reference individuals, adds a layer of personalization to the automated music generation process. This motivation arises from a recognition that each artist has a unique style and preference, and the synthesizer seeks to empower them to infuse their individuality into the compositions. Ultimately, the "Music Synthesizer Using AI" is motivated by the prospect of pushing the boundaries of creativity, offering a bridge between the realms of artistic intuition and computational innovation in the realm of music composition.

PROBLEM STATEMENT

Develop an innovative Music Synthesizer that employs Genetic Algorithm (GA) methodologies to autonomously generate novel and captivating sound compositions. The challenge involves implementing genetic operators such as crossover and mutation for effective evolution. A technology that can automate parts of music production, speed the composition process, and motivate musicians by giving new musical ideas is required.

OBJECTIVES

- Broaden Access to Music Composition: Place powerful and intuitive tools in the hands of users, with a particular emphasis on composers, to streamline and accelerate the creative journey, allowing artists to focus more on artistic expression than the intricacies of musical theory.
- Streamline Creative Processes through Automated Composition: Employ automated processes for generating unique compositions, facilitating a smoother and faster creative journey.
- Empower Users with User-Friendly Platform: Empower users with a user-friendly platform that seamlessly integrates automated composition capabilities, enabling them to effortlessly create, assess, and refine melodies.
- Enhance Composition Experience with Real-Time Feedback: Integrate real-time feedback mechanisms, fostering an environment of continuous exploration and innovation in musical creation.

DELIVERABLES

Executable Application or Module:

Package the code into an executable application or a modular library that can be easily installed and run on various systems. This ensures accessibility for users without extensive programming knowledge.

Graphical User Interface (GUI):

Develop a user-friendly GUI to provide an intuitive interface for users to interact with the synthesizer. This GUI should allow users to input parameters such as the number of bars, notes per bar, key, scale, BPM, etc., and visualize the generated compositions

Integration with MIDI and Audio Libraries:

Ensure seamless integration with external libraries for MIDI and audio processing. This allows users to export compositions in standard MIDI file format or play them in real-time using audio output devices.

User Feedback Mechanism:

Integrate a user feedback mechanism within the application, allowing users to rate the generated compositions. This feedback can be used to continuously improve the genetic algorithm and enhance the user experience.

3. REQUIREMENTS ENGINEERING

CLIENT DETAILS

NAME	**:** Dr. V.Durgadevi B.E., M.Tech., Ph.D.,
DESIGNATION	**:** Assistant Professor, Department of Information Technology
E mail ID	**:** durgadeviv@ssn.edu.in

Functional Requirements:

Sound Generation:

The synthesizer should be capable of generating a diverse range of sounds, including traditional instrument sounds, electronic tones, and innovative textures.

Customizable Parameters:

Provide adjustable parameters for sound customization, including waveform, frequency, amplitude, filter settings, and modulation options.

MIDI Compatibility:

Support MIDI input and output for integration with external devices, software, and controllers.

Non-Functional Requirements:

Reliability:

Build a stable and reliable synthesizer, minimizing the occurrence of crashes, errors, and unexpected behavior.

Security:

Implement security measures to protect user data and prevent unauthorized access to the synthesizer's settings and presets.

Accessibility:

Ensure accessibility features for users with disabilities, making the synthesizer usable by a diverse audience.

List of all functional modules

Sprint	**Epic**	**Requirement / User Story**	**Essential or Desirable**	**Description of the Requirement**
1	Project Kick off and Genetic algorithm	Create initial project files and folder	Essential	Choosing problem statement for project development that uses AI algorithms to be implemented
		Research and install necessary dependencies and implement GA	Essential	implement genetic algorithm .
		Test GA code for dummy	Essential	Testing and documenting the

		data and document it		basic GA for future use.
		Prepare for the integration of the genetic algorithm code with the Flask app	Essential	Develop the integration code of GA with flask
2	Project setup and basic flask	Set up the Flask project structure	Essential	Flask structure has been setup for development.
		Create HTML templates for the main page and form and Define Flask routes for the main page and form	Essential	HTML templates, main page, forms created, and flask is defined for these.
		Implement basic input validation on the form and Test the basic Flask app locally	Essential	Validation and testing basics locally
3	Genetic Algorithm integration	Modify the genetic algorithm code to accept parameters from a form	Essential	GA accepting the parameters from the form.
		Create a new route in Flask to handle genetic algorithm execution and integrate the genetic algorithm code with the Flask app	Essential	Creating new flask integration code for genetic algorithm.
		Handle errors and edge cases in the integration	Essential	Errors are handled
4	Result page and styling	Create an HTML template for	Essential	Displaying result by creating HTML

		displaying the results		
		Modify the Flask route to pass the genetic algorithm results to the results template and Style the web pages using CSS for a better user experience	Essential	Styling the webpages and modifying the flask route
		Test the complete flow from form submission to displaying results and handle any additional styling or layout adjustments	Essential	Complete flow is tested and handling layout and styling adjustments.
5	Additional Features and Refinement	Implement user authentication	Essential	user authentications are being implemented
		Conduct thorough testing and fix any bugs and Finalize documentation	Essential	Testing implemented

4. Implementation and Risk Management

Name : Harsh Bansal

Register Number: 3122215002036

Role: Developer

A. Implementation

Sprint #	Epic	User Story	Requirement	Remarks
1	Project Kick off and Genetic Algorithm Code Creation	AMG-3,5	Research and install necessary dependencies and Test GA code for dummy data and document it	Testing and documenting the basic GA for future use.
2	Project Setup and Basic Flask App	AMG-11	Define Flask routes for the main page and form	flask is defined for these.
3	Genetic Algorithm Integration	AMG-16	Create a new route in Flask to handle genetic algorithm execution	Created new flask integration code for genetic algorithm.
4	Results Page and Styling	AMG-23,24	Test the complete flow from form submission to displaying results and Handle any additional styling or layout adjustments	Complete flow is tested and handling layout and styling adjustments.
5	Additional Features and Refinement	AMG-27	Conduct thorough testing	Testing implemented

B. Risk Management

Risk#	Risk Description	Probability	Impact	Mitigation Plan
1		High		Various constrains are

	Applying the constraints to the input		Users can give invalid input by mistake.	given and each input is tested for validity.

Name: Harsha Nandhini K M

Register Number: 3122215002037

Role: Developer

A. Implementation

Sprint #	Epic	User Story	Requirement	Remarks
1	Project Kick off and Genetic Algorithm Code Creation	AMG-4	Implement the basic structure of the genetic algorithm code	Adding functions to introduce mutation, crossover and selection and generate population and genome
		AMG-6	Document the genetic algorithm code for future reference	Documenting the genetic algorithm code
2	Project Setup and Basic Flask App	AMG- 12	Implement basic input validation on the form	Implementing basic input validation
		AMG- 13	Test the basic Flask app locally	Testing Flask locally
3	Genetic Algorithm Integration	AMG- 17	Integrate the genetic algorithm code with the Flask app	Code integration with flask
		AMG-18	Handle errors and edge cases in the integration	Handling errors and edge cases

4	Results Page and Styling	AMG- 21	Modify the Flask route to pass the genetic algorithm results to the results template	Modifying the Flask route
5	Additional Features and Refinement	AMG- 28	Finalize documentation	Finalizing documentation

B. Risk Management

Risk#	Risk Description	Probability	Impact	Mitigation Plan
User authentication	The occurrence of an existing customer being unable to log in arises.	Low	Moderate	Ensure user-friendly design and implement secure password policy

Name : Harshini D

Register Number: 3122215002038

Role: Developer

A. Implementation

Sprint #	Epic	User Story	Requirement	Remarks
1	Project Kick off and Genetic algorithm	AMG-1,2	Create initial project files and folder	Chosen the problem statement for project development that uses AI algorithms to be implemented
		AMG-7	Prepare for the integration of the genetic	Developed the integration

			algorithm code with the Flask app	code of GA with flask
2	Project setup and basic flask	AMG -8,9	Set up the Flask project structure	Flask structure is been setup for development.
3	Genetic Algorithm integration	AMG-14,15	Modify the genetic algorithm code to accept parameters from a form	GA accepting the parameters from the form.
4	Result page and styling	AMG-21,22	Modify the Flask route to pass the genetic algorithm results to the results template and Style the web pages using CSS for a better user experience	Styling the webpages and modifying the flask route
5	Additional Features and Refinement	AMG-25,26	Implement user authentication	user authentications are being implemented

B. Risk Management

Risk#	**Risk Description**	**Probability**	**Impact**	**Mitigation Plan**
Integration Challenges	Difficulties in integrating the developed models into the overall system may arise.	Moderate	High	Conduct thorough testing during development

Test log Report

TC id	Test Case	Description	Test Case Input	Expected Output	Result (PASS/FAIL)
1	Login and registration check	The credentials should be in the valid format	Invalid input for the credentials	Show error if input is invalid.	PASS
2	Filling of input fields in the forms	All the input fields should be filled	Unfilled input for submission	Shows warning for unfilled fields	PASS

5. PROJECT MANAGEMENT

Sprint 1: [11-09-2023 to 20-09-2023]

Project Kickoff and Genetic Algorithm Code Creation
Set up the version control repository, establish the initial project structure, install required dependencies, and create the basic structure of the genetic algorithm code. Conduct a brief code review, document coding conventions, and prepare for integration with the Flask app.

Sprint 2: [21-09-2023 to 03-10-2023]

Project Setup and Basic Flask App
Organize the Flask project structure, design HTML templates for the main page and form, and implement Flask routes. Ensure basic input validation on the form and test the Flask app locally.

Sprint 3: [04-10-2023 to 25-10-2023]

Genetic Algorithm Integration
Modify the genetic algorithm code to accept parameters from the form, create a new Flask route for genetic algorithm execution, and integrate the genetic algorithm with the Flask app. Test the complete flow from form submission to displaying results

Sprint 4: [26-10-2023 to 13-11-2023]

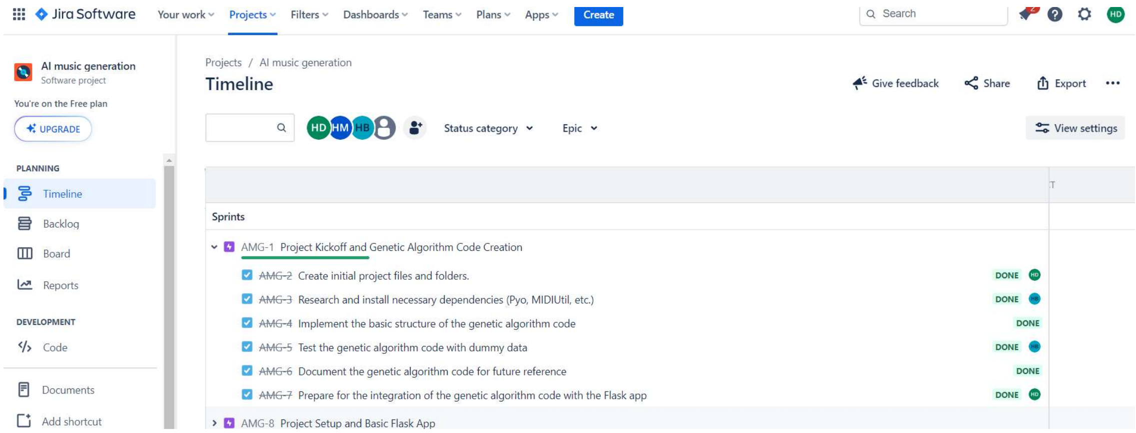
Results Page and Styling
Create an HTML template for displaying the genetic algorithm results, modify the Flask route to pass results to the template, and style the web pages using CSS for an improved user experience. Test the application thoroughly and handle any necessary adjustments.

Sprint 5: [14-11-2023 to 26-11-2023]

Additional Features and Refinement

Implement user authentication (if required), add error handling and validation for various scenarios, and incorporate additional features or improvements based on user feedback. Conduct thorough testing, fix any bugs, and finalize documentation for deployment.

Jira Screenshots

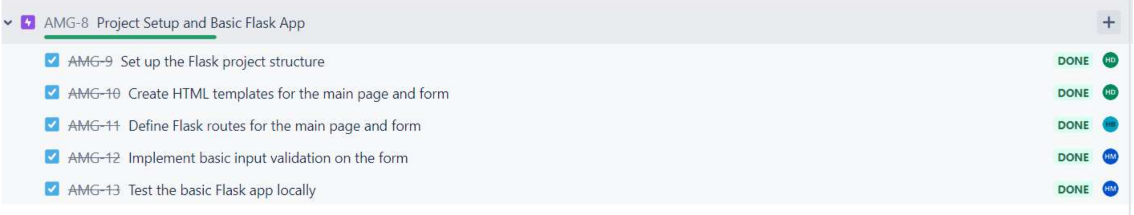


Sprints

AMG-1 Project Kickoff and Genetic Algorithm Code Creation

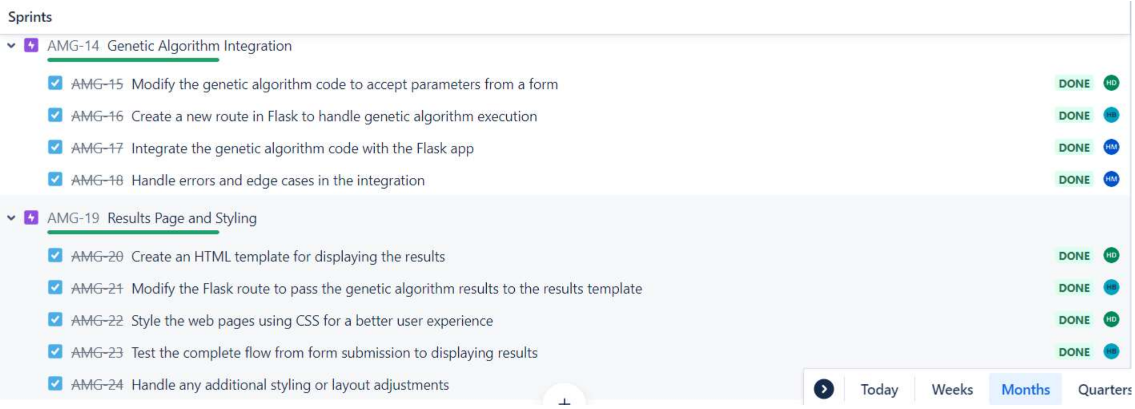
- ☑ ~~AMG-2~~ Create initial project files and folders. DONE
- ☑ ~~AMG-3~~ Research and install necessary dependencies (Pyo, MIDIUtil, etc.) DONE
- ☑ ~~AMG-4~~ Implement the basic structure of the genetic algorithm code DONE
- ☑ ~~AMG-5~~ Test the genetic algorithm code with dummy data DONE
- ☑ ~~AMG-6~~ Document the genetic algorithm code for future reference DONE
- ☑ ~~AMG-7~~ Prepare for the integration of the genetic algorithm code with the Flask app DONE

AMG-8 Project Setup and Basic Flask App



AMG-8 Project Setup and Basic Flask App

- ☑ ~~AMG-9~~ Set up the Flask project structure DONE
- ☑ ~~AMG-10~~ Create HTML templates for the main page and form DONE
- ☑ ~~AMG-11~~ Define Flask routes for the main page and form DONE
- ☑ ~~AMG-12~~ Implement basic input validation on the form DONE
- ☑ ~~AMG-13~~ Test the basic Flask app locally DONE



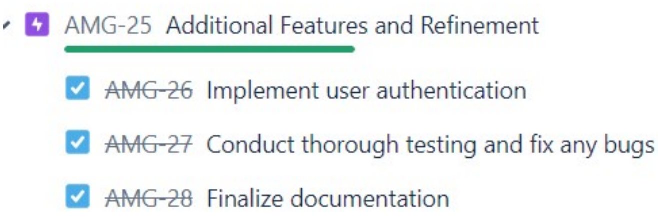
Sprints

AMG-14 Genetic Algorithm Integration

- ☑ ~~AMG-15~~ Modify the genetic algorithm code to accept parameters from a form DONE
- ☑ ~~AMG-16~~ Create a new route in Flask to handle genetic algorithm execution DONE
- ☑ ~~AMG-17~~ Integrate the genetic algorithm code with the Flask app DONE
- ☑ ~~AMG-18~~ Handle errors and edge cases in the integration DONE

AMG-19 Results Page and Styling

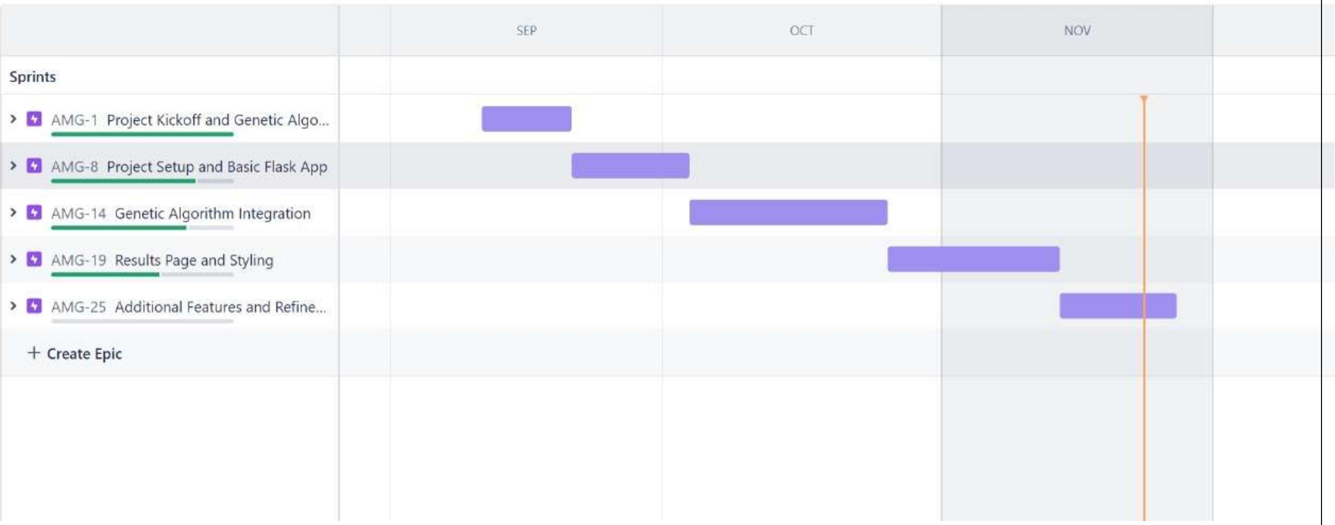
- ☑ ~~AMG-20~~ Create an HTML template for displaying the results DONE
- ☑ ~~AMG-21~~ Modify the Flask route to pass the genetic algorithm results to the results template DONE
- ☑ ~~AMG-22~~ Style the web pages using CSS for a better user experience DONE
- ☑ ~~AMG-23~~ Test the complete flow from form submission to displaying results DONE
- ☑ ~~AMG-24~~ Handle any additional styling or layout adjustments



AMG-25 Additional Features and Refinement

- ☑ ~~AMG-26~~ Implement user authentication
- ☑ ~~AMG-27~~ Conduct thorough testing and fix any bugs
- ☑ ~~AMG-28~~ Finalize documentation

JIRA TIMELINE



Sprints

AMG-1 Project Kickoff and Genetic Algo...

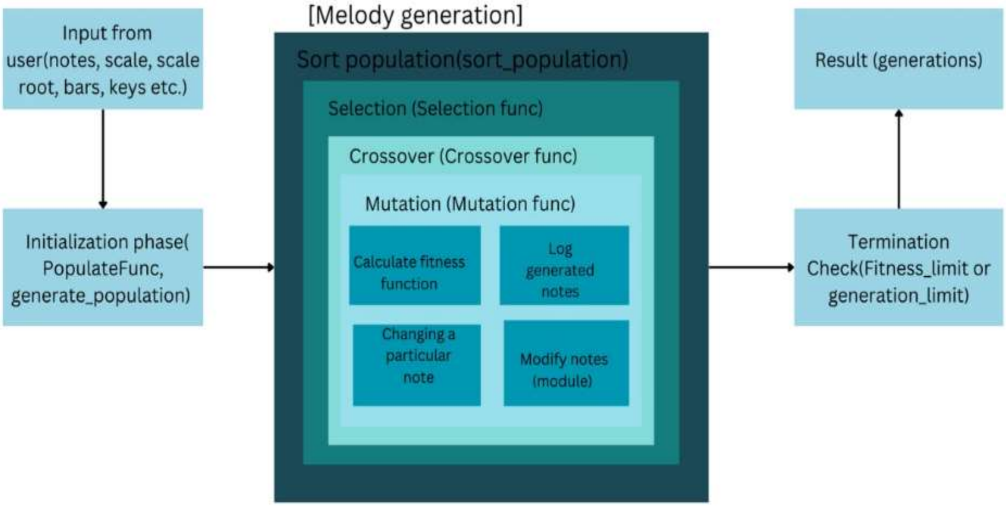
AMG-8 Project Setup and Basic Flask App

AMG-14 Genetic Algorithm Integration

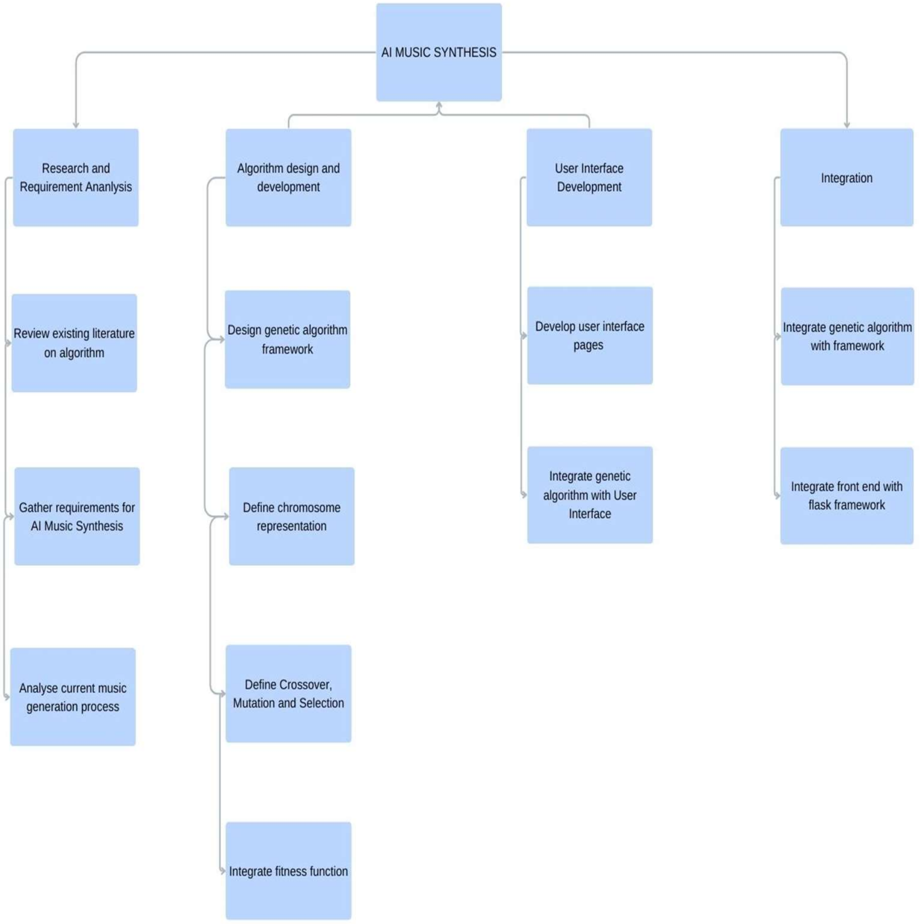
AMG-19 Results Page and Styling

AMG-25 Additional Features and Refine...

BLOCK DIAGRAM:


Input from user(notes, scale, scale root, bars, keys etc.)
Initialization phase(PopulateFunc, generate_population)
[Melody generation]
Sort population(sort_population)
Selection (Selection func)
Crossover (Crossover func)
Mutation (Mutation func)
Calculate fitness function
Log generated notes
Changing a particular note
Modify notes (module)
Termination Check(Fitness_limit or generation_limit)
Result (generations)

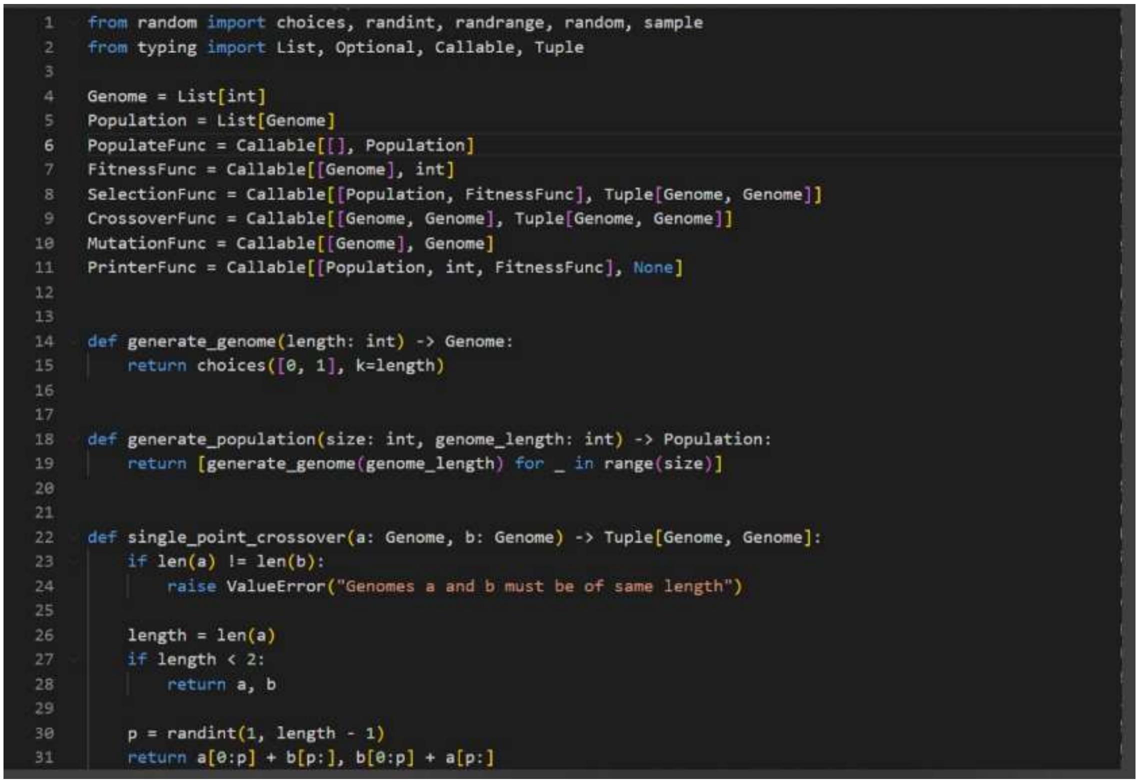
WORK BREAKDOWN CHART:


AI MUSIC SYNTHESIS
Research and Requirement Ananlysis
Review existing literature on algorithm
Gather requirements for AI Music Synthesis
Analyse current music generation process
Algorithm design and development
Design genetic algorithm framework
Define chromosome representation
Define Crossover, Mutation and Selection
Integrate fitness function
User Interface Development
Develop user interface pages
Integrate genetic algorithm with User Interface
Integration
Integrate genetic algorithm with framework
Integrate front end with flask framework

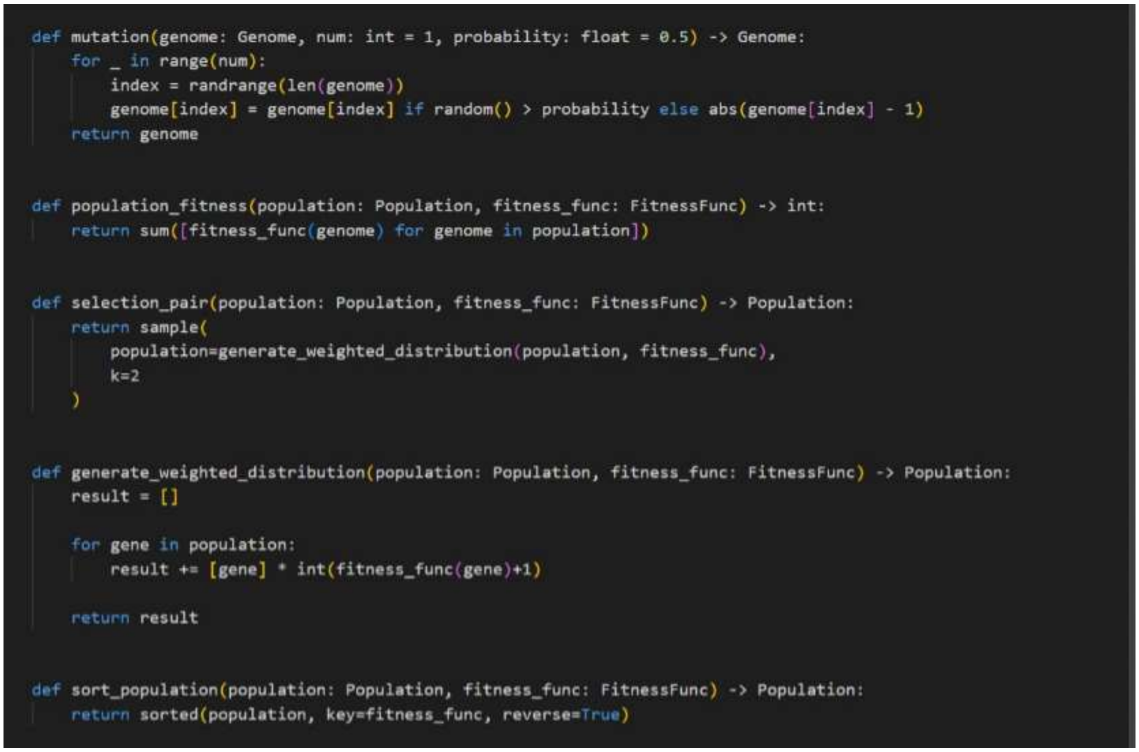
6. PROJECT OUTCOMES

CODE SNIPPETS

Genetic algorithm

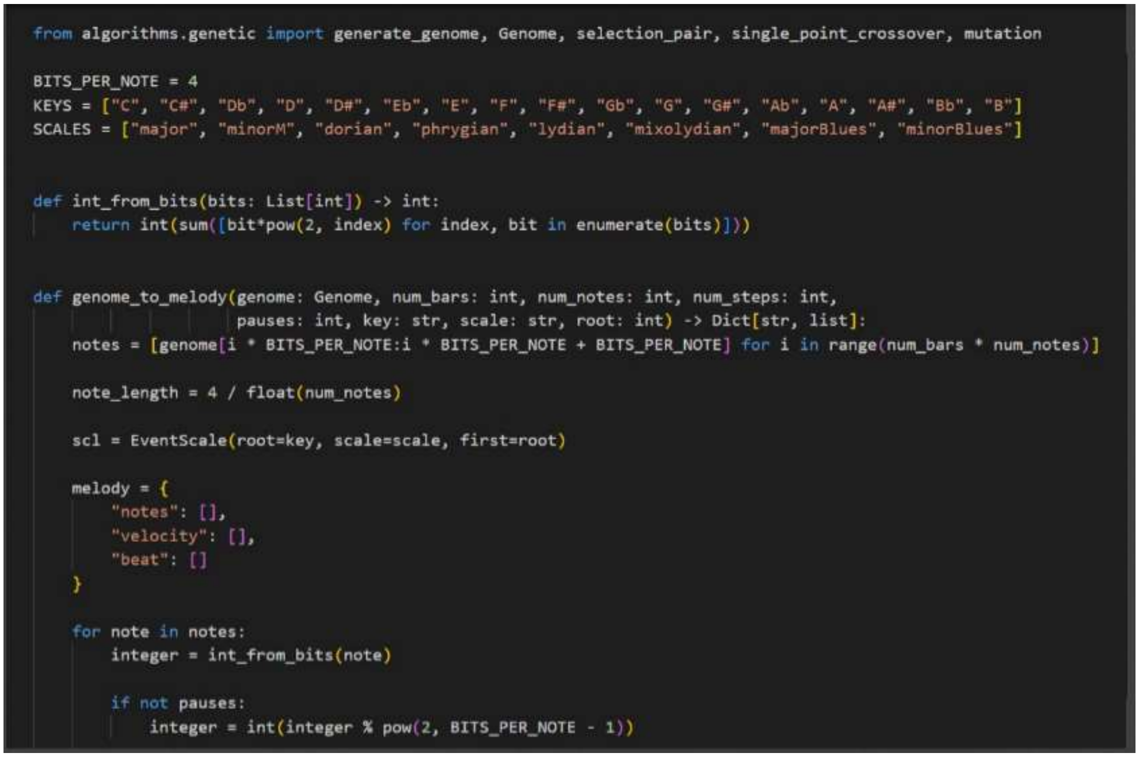


```python
from random import choices, randint, randrange, random, sample
from typing import List, Optional, Callable, Tuple

Genome = List[int]
Population = List[Genome]
PopulateFunc = Callable[[], Population]
FitnessFunc = Callable[[Genome], int]
SelectionFunc = Callable[[Population, FitnessFunc], Tuple[Genome, Genome]]
CrossoverFunc = Callable[[Genome, Genome], Tuple[Genome, Genome]]
MutationFunc = Callable[[Genome], Genome]
PrinterFunc = Callable[[Population, int, FitnessFunc], None]


def generate_genome(length: int) -> Genome:
    return choices([0, 1], k=length)


def generate_population(size: int, genome_length: int) -> Population:
    return [generate_genome(genome_length) for _ in range(size)]


def single_point_crossover(a: Genome, b: Genome) -> Tuple[Genome, Genome]:
    if len(a) != len(b):
        raise ValueError("Genomes a and b must be of same length")

    length = len(a)
    if length < 2:
        return a, b

    p = randint(1, length - 1)
    return a[0:p] + b[p:], b[0:p] + a[p:]
```

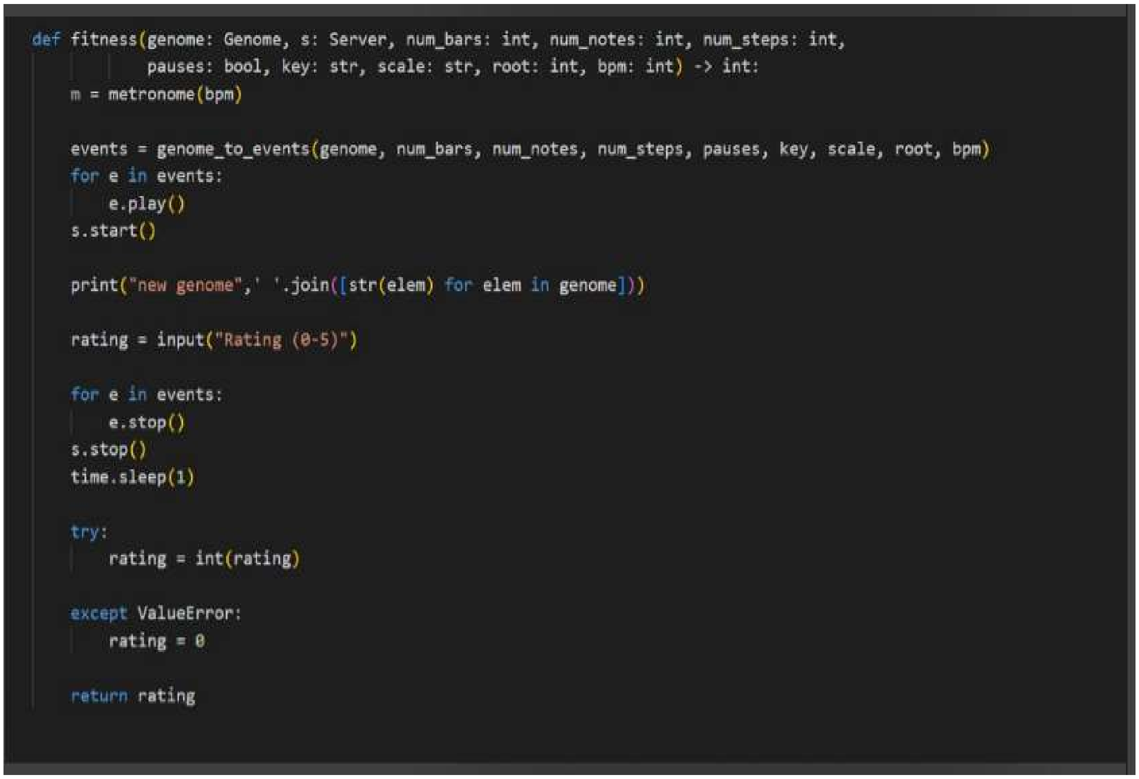


```python
def mutation(genome: Genome, num: int = 1, probability: float = 0.5) -> Genome:
    for _ in range(num):
        index = randrange(len(genome))
        genome[index] = genome[index] if random() > probability else abs(genome[index] - 1)
    return genome


def population_fitness(population: Population, fitness_func: FitnessFunc) -> int:
    return sum([fitness_func(genome) for genome in population])


def selection_pair(population: Population, fitness_func: FitnessFunc) -> Population:
    return sample(
        population=generate_weighted_distribution(population, fitness_func),
        k=2
    )


def generate_weighted_distribution(population: Population, fitness_func: FitnessFunc) -> Population:
    result = []

    for gene in population:
        result += [gene] * int(fitness_func(gene)+1)

    return result


def sort_population(population: Population, fitness_func: FitnessFunc) -> Population:
    return sorted(population, key=fitness_func, reverse=True)
```

Music generation

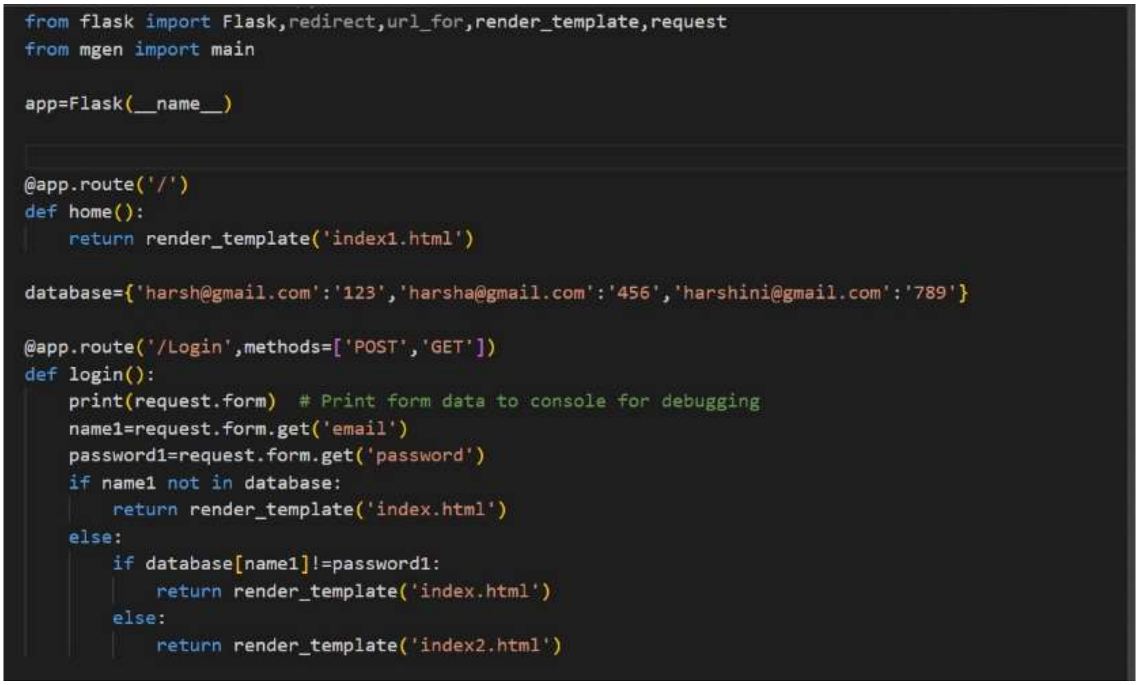


```python
from algorithms.genetic import generate_genome, Genome, selection_pair, single_point_crossover, mutation

BITS_PER_NOTE = 4
KEYS = ["C", "C#", "Db", "D", "D#", "Eb", "E", "F", "F#", "Gb", "G", "G#", "Ab", "A", "A#", "Bb", "B"]
SCALES = ["major", "minorM", "dorian", "phrygian", "lydian", "mixolydian", "majorBlues", "minorBlues"]


def int_from_bits(bits: List[int]) -> int:
    return int(sum([bit*pow(2, index) for index, bit in enumerate(bits)]))


def genome_to_melody(genome: Genome, num_bars: int, num_notes: int, num_steps: int,
                     pauses: int, key: str, scale: str, root: int) -> Dict[str, list]:
    notes = [genome[i * BITS_PER_NOTE:i * BITS_PER_NOTE + BITS_PER_NOTE] for i in range(num_bars * num_notes)]

    note_length = 4 / float(num_notes)

    scl = EventScale(root=key, scale=scale, first=root)

    melody = {
        "notes": [],
        "velocity": [],
        "beat": []
    }

    for note in notes:
        integer = int_from_bits(note)

        if not pauses:
            integer = int(integer % pow(2, BITS_PER_NOTE - 1))
```



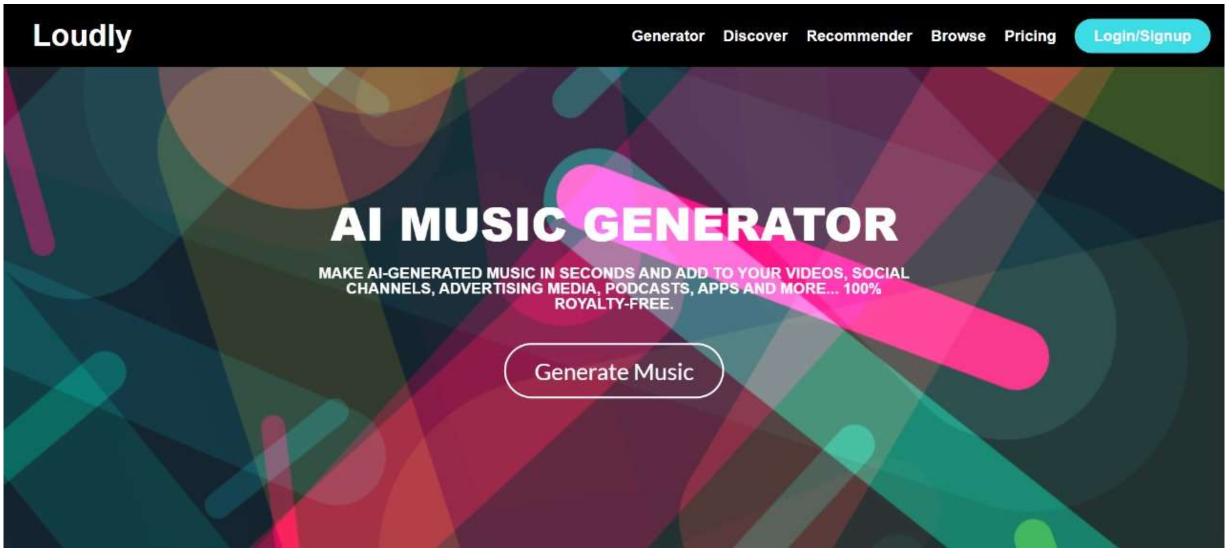
```python
def fitness(genome: Genome, s: Server, num_bars: int, num_notes: int, num_steps: int,
            pauses: bool, key: str, scale: str, root: int, bpm: int) -> int:
    m = metronome(bpm)

    events = genome_to_events(genome, num_bars, num_notes, num_steps, pauses, key, scale, root, bpm)
    for e in events:
        e.play()
    s.start()

    print("new genome",' '.join([str(elem) for elem in genome]))

    rating = input("Rating (0-5)")

    for e in events:
        e.stop()
    s.stop()
    time.sleep(1)

    try:
        rating = int(rating)

    except ValueError:
        rating = 0

    return rating
```

Flask integration



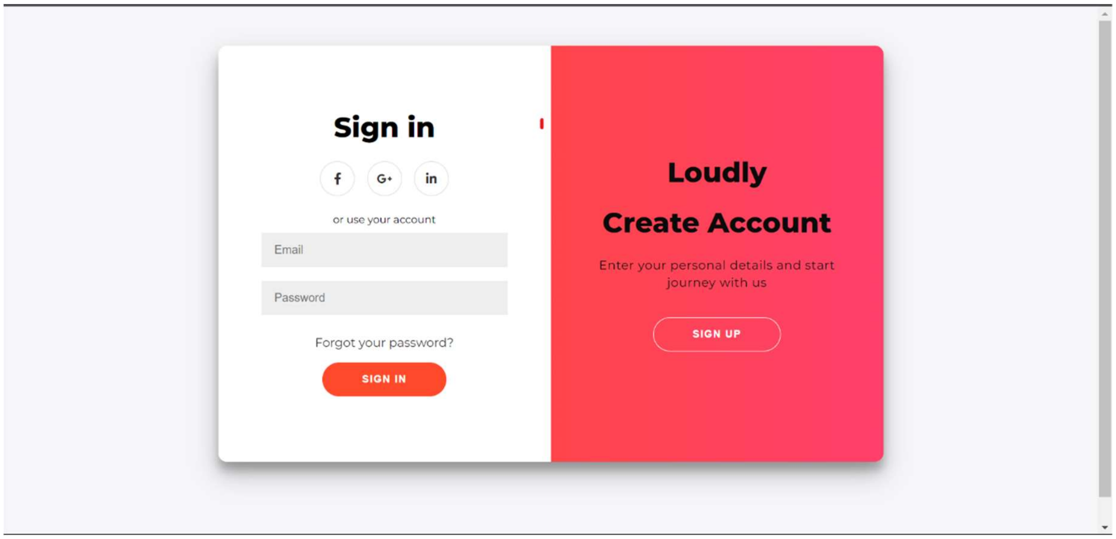
```
from flask import Flask,redirect,url_for,render_template,request
from mgen import main

app=Flask(__name__)


@app.route('/')
def home():
    return render_template('index1.html')

database={'harsh@gmail.com':'123','harsha@gmail.com':'456','harshini@gmail.com':'789'}

@app.route('/Login',methods=['POST','GET'])
def login():
    print(request.form)  # Print form data to console for debugging
    name1=request.form.get('email')
    password1=request.form.get('password')
    if name1 not in database:
        return render_template('index.html')
    else:
        if database[name1]!=password1:
            return render_template('index.html')
        else:
            return render_template('index2.html')
```

OUTPUT SCREENSHOTS



AI MUSIC GENERATOR

MAKE AI-GENERATED MUSIC IN SECONDS AND ADD TO YOUR VIDEOS, SOCIAL CHANNELS, ADVERTISING MEDIA, PODCASTS, APPS AND MORE... 100% ROYALTY-FREE.



Sign in

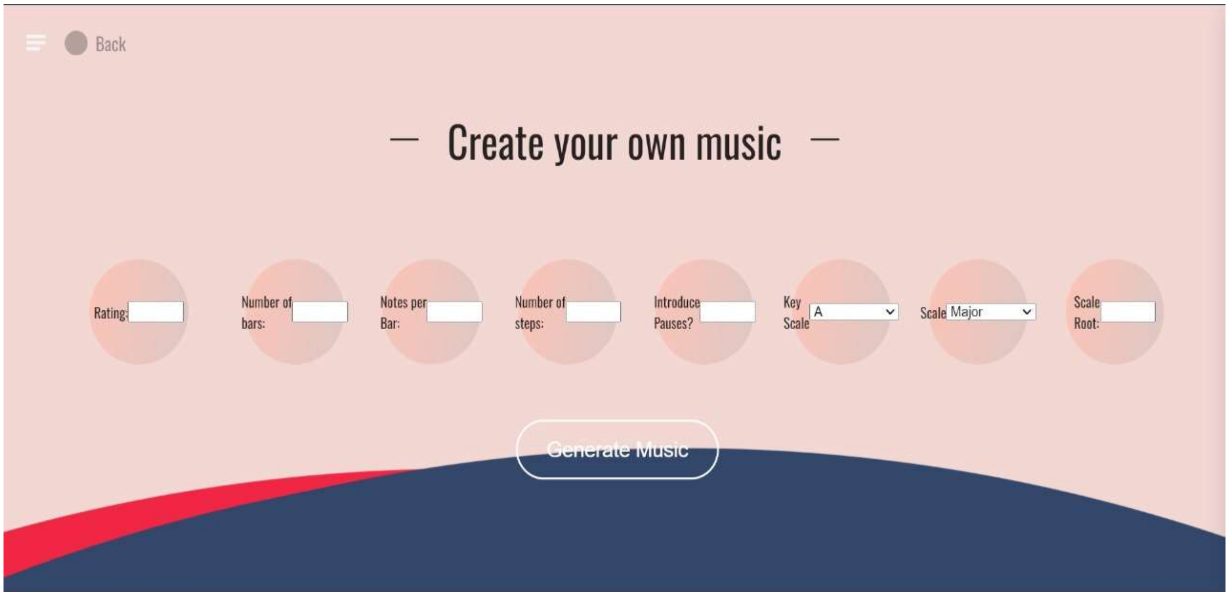
or use your account

Forgot your password?

Loudly

Create Account

Enter your personal details and start journey with us



— Create your own music —

Rating:

Number of bars:

Notes per Bar:

Number of steps:

Introduce Pauses?

Key Scale A

Scale Major

Scale Root:

Audio Visualisation

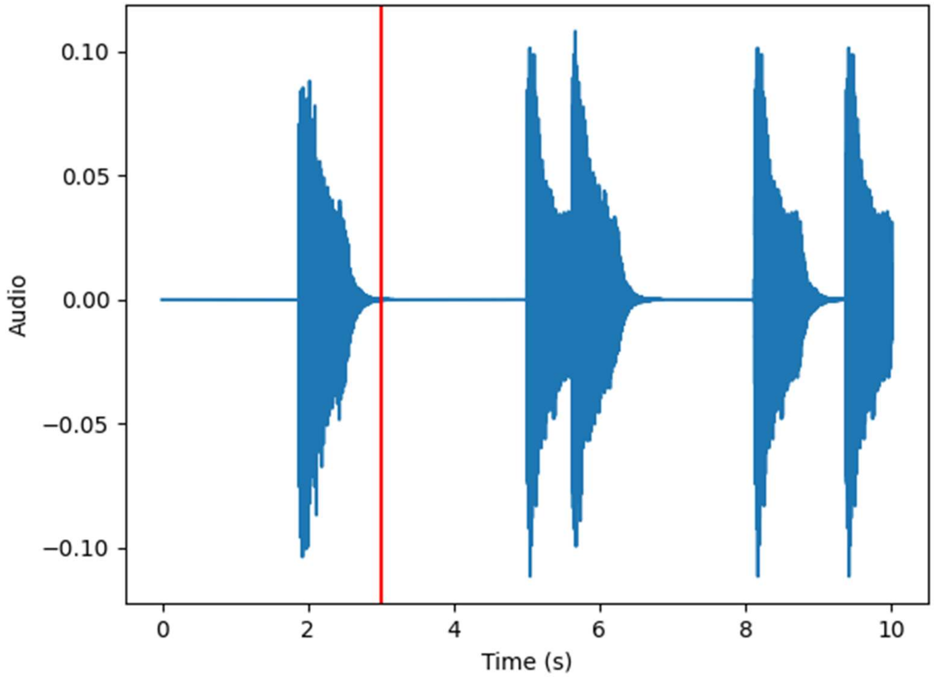


Figure I for Audio I

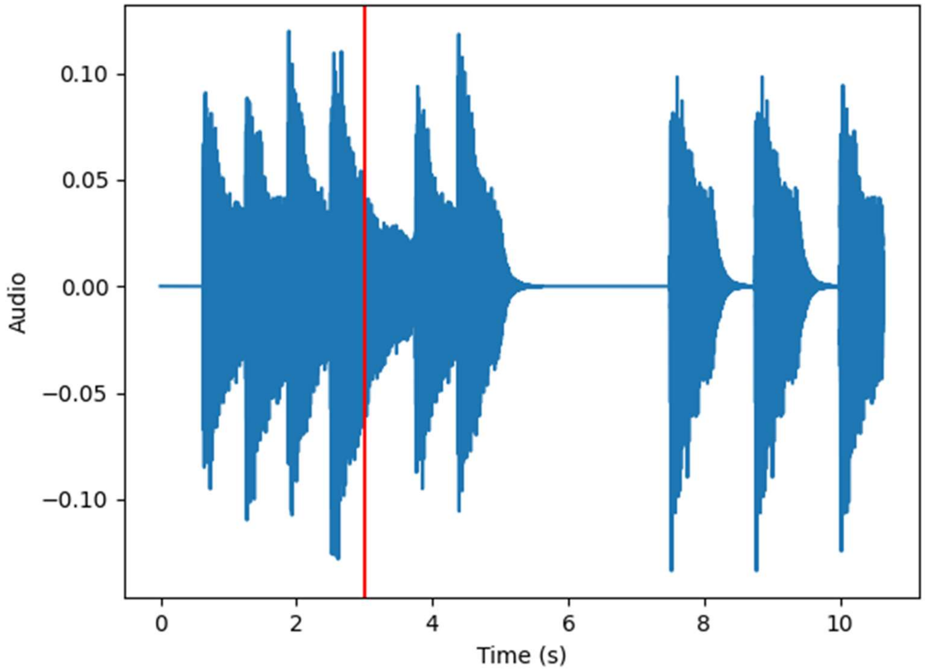


Figure II for Audio II

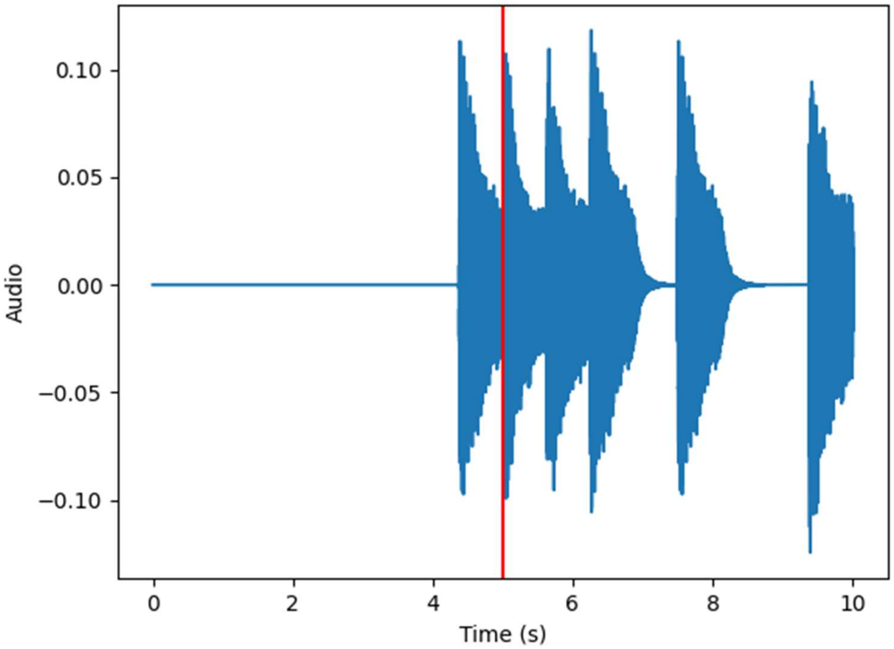


Figure III for Audio III

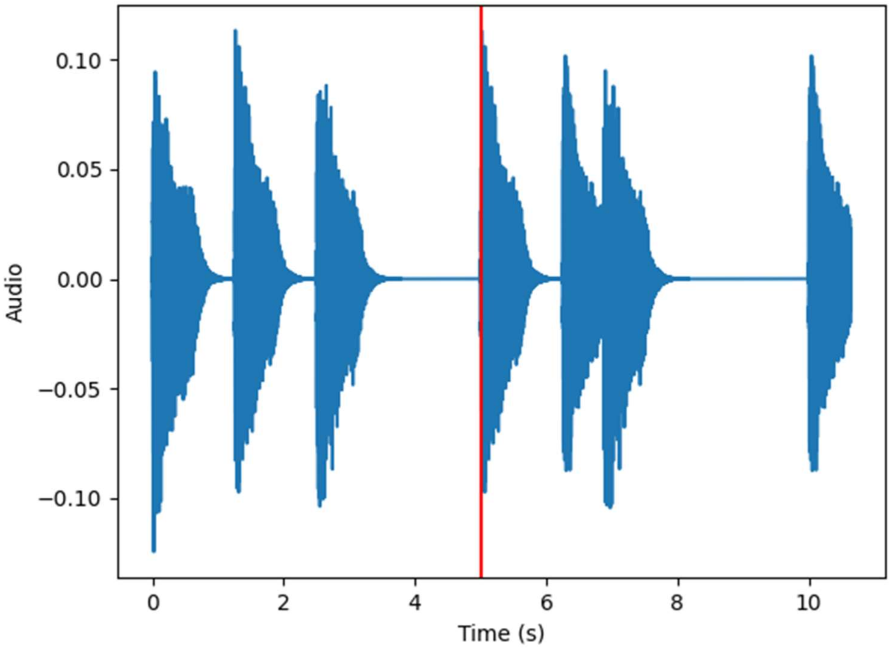


Figure IV for Audio IV

7. CONCLUSION AND FUTURE DIRECTIONS

This project introduces an innovative application of genetic algorithms to compose aesthetically pleasing music, focusing on desirable intervals and comprehensible rhythm. By leveraging coding techniques with arrays and mathematical functions, the approach offers swift control over composition elements. Moreover, there is a keen interest in adapting the algorithm for generating compositions across diverse music genres through parameter adjustments. In summary, this project presents a versatile and fine-tuned framework for musical composition, prompting further investigation into its applications and potential refinements in the context of the discussed genetic algorithm and Flask integration.

Future directions for this project could involve the incorporation of machine learning techniques to enhance the adaptability and creativity of the music composition process. By training models on vast musical datasets, the system could learn more complex patterns, styles, and nuances, enabling it to generate compositions that exhibit a deeper understanding of musical structures and preferences. Additionally, exploring real-time collaboration features or integration with interactive platforms could open avenues for user engagement and participation in the music creation process. This extension could transform the project into a dynamic and collaborative tool that empowers both AI-driven and user-driven musical composition experiences.

8. REFERENCES

[1] Feng, B., Jiang, Z., Fan, Z., Fu, N. (2010). "A Method for Member Selection of Cross-Functional Teams Using the Individual and Collaborative Performances." European Journal of Operational Research, 203(3), 652-661. Available from Business Source Complete, Ipswich, MA.

[2] Goldberg, D. "Genetic Algorithms in Search, Optimization and Machine Learning." Describes genetic algorithms as probabilistic search procedures designed for large spaces involving states represented by strings. Focuses on classifier systems and their derivatives.

[3] Tokui, N., Iba, H. "Music Composition with Interactive Evolutionary Computation." In Proc. Generative Art2000, the 3rd International Conference on Generative Art. Describes an approach to music composition using interactive evolutionary computation, specifically combining genetic algorithms and genetic programming.

[4] Johanson, B. E. (1997). "The GP-Music System: Interactive Genetic Programming for Music Composition." Created automatic fitness raters based on neural networks with shared weights trained using the backpropagation algorithm.

[5] Biles, J. A. Proposed a genetic algorithm-based model of a novice jazz musician learning to improvise, maintaining hierarchically related populations of melodic ideas.

[6] Gibson, P. M., Byrne, J. A. "Neurogen, Musical Composition Using Genetic Algorithms and Cooperating Neural Networks." Aims to produce coherent music using

neural networks to capture conceptual ideas and genetic algorithms to evolve musical fragments.

[7] Rosa, A.C. (1999). "Sample MIDI files." Addresses the problem of identifying the melodic track of a MIDI file in imbalanced scenarios.

[8] Pelchat, C., Craig M. Conducted research on the GTZAN dataset, categorizing songs into seven genres and using convolutional neural networks for classification.

[9] Meenakshi, K. Conducted a survey using ConvNet architectures for music genre classification, using features extracted with MFCC.

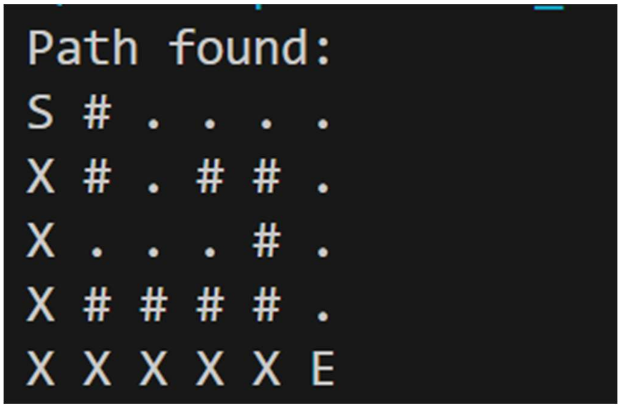
1) ROBOT – MAZE PROBLEM

CODE:

```
from collections import deque

def bfs_maze_solver(maze, start, end):
    def is_valid_move(x, y):
        return 0 <= x < len(maze) and 0 <= y < len(maze[0]) and maze[x][y] == 0

    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]
    queue = deque([(start[0], start[1], [])])
    visited = set()

    while queue:
        x, y, path = queue.popleft()
        if (x, y) == end:
            return path + [(x, y)]
        if (x, y) in visited:
            continue
        visited.add((x, y))
        for dx, dy in directions:
            new_x, new_y = x + dx, y + dy
            if is_valid_move(new_x, new_y):
                new_path = path + [(x, y)]
                queue.append((new_x, new_y, new_path))
    return None

def print_maze_with_path(maze, path):
    for i in range(len(maze)):
        for j in range(len(maze[0])):
            if (i, j) == path[0]:
                print("S", end=" ")
            elif (i, j) == path[-1]:
                print("E", end=" ")
            elif (i, j) in path:
                print("X", end=" ")
            elif maze[i][j] == 0:
                print(".", end=" ")
            else:
                print("#", end=" ")
        print()

if __name__ == "__main__":

    maze = [
        [0, 1, 0, 0, 0, 0],
        [0, 1, 0, 1, 1, 0],
        [0, 0, 0, 0, 1, 0],
        [0, 1, 1, 1, 1, 0],
        [0, 0, 0, 0, 0, 0]]
```

```
start = (0, 0)
end = (4, 5)
path = bfs_maze_solver(maze, start, end)

if path:
    print("Path found:")
    print_maze_with_path(maze, path)
else:
    print("No path found.")
```

OUTPUT:



```
Path found:
S # . . . .
X # . # # .
X . . . # .
X # # # # .
X X X X X E
```

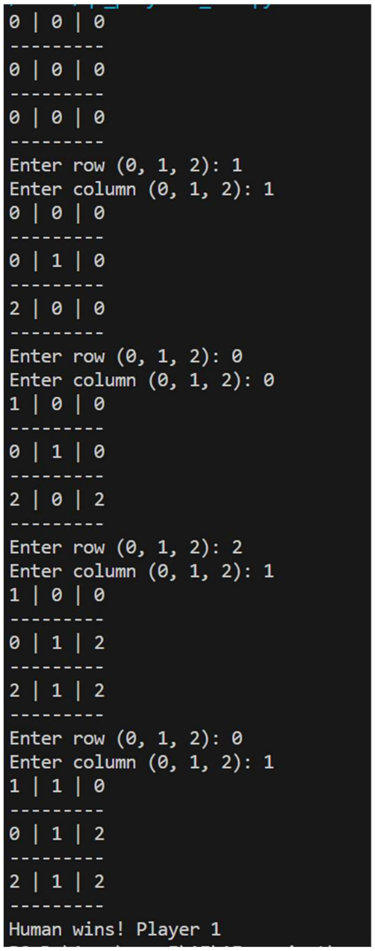
2) TIC TAC TOE PROBLEM

CODE:

```
import random
from collections import deque

class Game:
    def __init__(self):
        self.board = [[0, 0, 0],
                      [0, 0, 0],
                      [0, 0, 0]]

    def display_board(self):
        for row in self.board:
            print(" | ".join(map(str, row)))
            print("-" * 9)

    def is_valid_move(self, row, col):
        return 0 <= row < 3 and 0 <= col < 3 and self.board[row][col] == 0

    def make_move(self, player, row, col):
        if self.is_valid_move(row, col):
            self.board[row][col] = player
            return True
        return False

    def machine_move(self, player):
        # Implement the machine's move here using DFS or BFS.
        # For example, you can use random moves for demonstration purposes.
        available_moves = [(i, j) for i in range(3) for j in range(3) if self.board[i][j] == 0]
        if available_moves:
            return random.choice(available_moves)
        else:
            return None

def check_winner(board):
    for row in board:
        if row[0] == row[1] == row[2] != 0:
            return row[0]
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != 0:
            return board[0][col]
    if board[0][0] == board[1][1] == board[2][2] != 0 or board[0][2] == board[1][1] ==
board[2][0] != 0:
        return board[1][1]
    return 0

def is_board_full(board):
    return all(all(cell != 0 for cell in row) for row in board)
```

```
def play_game():
    game = Game()
    human_player = 1
    machine_player = 2

    while True:
        game.display_board()

        # Human's move
        while True:
            try:
                row = int(input("Enter row (0, 1, 2): "))
                col = int(input("Enter column (0, 1, 2): "))
                if game.make_move(human_player, row, col):
                    break
                else:
                    print("Invalid move. Try again.")
            except ValueError:
                print("Invalid input. Enter a number between 0 and 2.")
        winner = check_winner(game.board)
        if winner:
            game.display_board()
            print(f"Human wins! Player {winner}")
            break

        if is_board_full(game.board):
            game.display_board()
            print("It's a draw!")
            break

        # Machine's move
        machine_move = game.machine_move(machine_player)
        if machine_move:
            game.make_move(machine_player, *machine_move)
        winner = check_winner(game.board)
        if winner:
            game.display_board()
            print(f"Machine wins! Player {winner}")
            break
        if is_board_full(game.board):
            game.display_board()
            print("It's a draw!")
            break

if __name__ == "__main__":
    play_game()
```

OUTPUT:



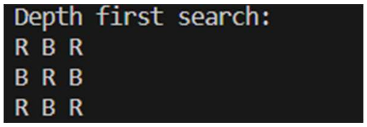
```
0 | 0 | 0
---------
0 | 0 | 0
---------
0 | 0 | 0
---------
Enter row (0, 1, 2): 1
Enter column (0, 1, 2): 1
0 | 0 | 0
---------
0 | 1 | 0
---------
2 | 0 | 0
---------
Enter row (0, 1, 2): 0
Enter column (0, 1, 2): 0
1 | 0 | 0
---------
0 | 1 | 0
---------
2 | 0 | 2
---------
Enter row (0, 1, 2): 2
Enter column (0, 1, 2): 1
1 | 0 | 0
---------
0 | 1 | 2
---------
2 | 1 | 2
---------
Enter row (0, 1, 2): 0
Enter column (0, 1, 2): 1
1 | 1 | 0
---------
0 | 1 | 2
---------
2 | 1 | 2
---------
Human wins! Player 1
```

3) GRID COLORING PROBLEM

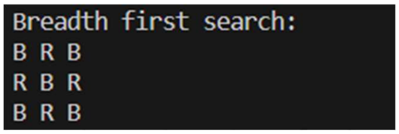
CODE:

```
class Grid:
    def __init__(self, n):
        self.grid = [[None for _ in range(n // 3)] for _ in range(n // 3)]
        self.goal_patterns = [[[1, 0, 1], [0, 1, 0], [1, 0, 1]], [[0, 1, 0], [1, 0, 1], [0, 1, 0]]]
        self.stack = None
        self.size = 3

    def is_valid(self, row, col, color):
        if row - 1 >= 0 and self.grid[row - 1][col] == color:
            return False
        if col - 1 >= 0 and self.grid[row][col - 1] == color:
            return False
        if row + 1 < len(self.grid) and col<self.size:
            if self.grid[row + 1][col] == color:
                return False
        if col + 1 < len(self.grid) and self.grid[row][col + 1] == color:
            return False
        return True

    def dfs(self, depth):
        if self.grid in self.goal_patterns:
            return self.grid
        for i in range(self.size):
            for j in range(self.size):
                if self.grid[i][j] is None:
                    for col in [0, 1]:
                        if self.is_valid(i, j, col):
                            self.grid[i][j] = col
                        if self.dfs(depth-1):
                            return self.grid
                        self.grid[i][j] = None
                return None
def bfs(self):
    queue=[(0,0)]
    while queue:
        # print(self.grid)
        if self.grid in self.goal_patterns:
            for row in self.grid:
                # print(row)
                for i in row:
                    if i==1:
                        print("B",end=" ")
                    else:
                        print("R",end=" ")
                print()
            return self.grid
        current=queue.pop(0)
        if 0<=current[0]<self.size and 0<=current[1]+1<self.size:
            queue.append((current[0],current[1]+1))
        if 0<=current[0]<self.size and 0<=current[1]-1<self.size:
            queue.append((current[0],current[1]-1))
        if 0<=current[0]+1<self.size and 0<=current[1]<self.size:
            queue.append((current[0]+1,current[1]))
        if 0<=current[0]-1<self.size and 0<=current[1]<self.size:
            queue.append((current[0]-1,current[1]))

        for color in 0,1:
            if self.is_valid(current[0],current[1],color):
                    self.grid[current[0]][current[1]]=color

    def solve_colouring(self):
        if not self.stack:
            self.stack = [self.grid]
        for i in range(1, (self.size * self.size) + 1):
            colored_grid = self.dfs(i)
            if colored_grid:
                for row in colored_grid:
                    # print(row)
                    for i in row:
                        if i==1:
                            print("B",end=" ")
                        else:
                            print("R",end=" ")
                    print()
                return
        return False
if __name__ == "__main__":
    grid_instance = Grid(9)
    print("Depth first search:")
    grid_instance.solve_colouring()
    print("Breadth first search:")
    grid_instance.bfs()
```

OUTPUT:



```
Depth first search:
R B R
B R B
R B R
```



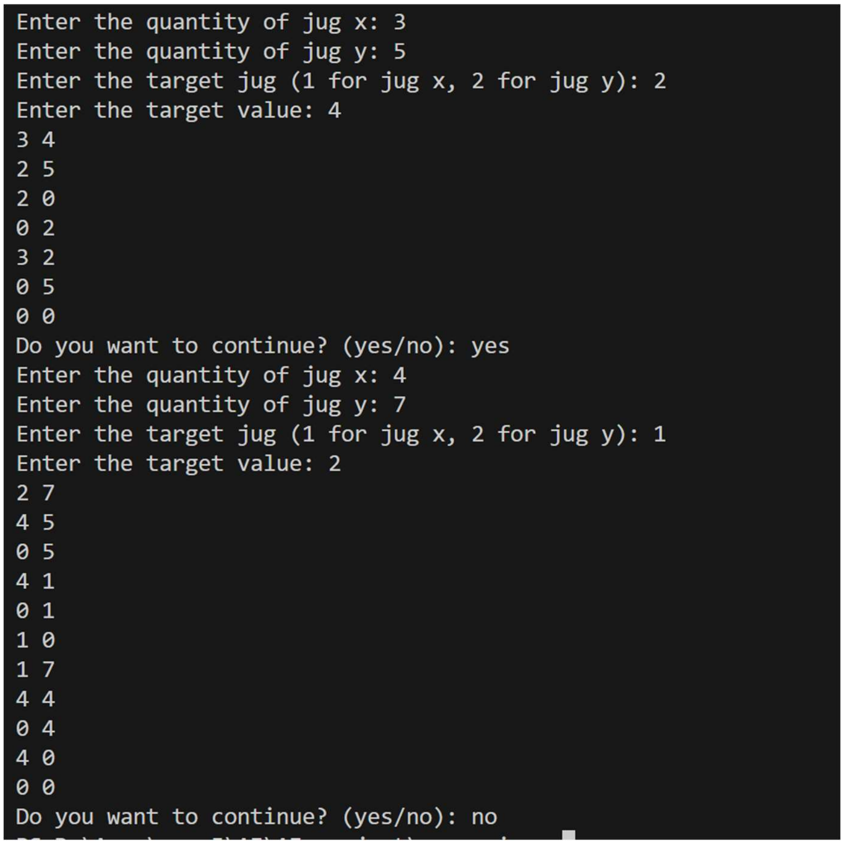
```
Breadth first search:
B R B
R B R
B R B
```

4) WATER JUG PROBLEM:

CODE:

```
class node:
    def __init__(self, x, y, prev):
        self.x = x
        self.y = y
        self.prev = prev
def conditions(x, y, cur):
    prev = cur

    # filling jug1
    if x < jug_x:
        queue.append(node(jug_x, y, prev))

    # filling jug2
    if y < jug_y:
        queue.append(node(x, jug_y, prev))

    # transferring contents of jug2 to jug1
    if x < jug_x:
        if x + y >= jug_x:
            d = jug_x - x
            queue.append(node(jug_x, y - d, prev))
        else:
            if x + y != 0:
                queue.append(node(x + y, 0, prev))

    # transferring contents of jug1 to jug2
    if y < jug_y:
        if x + y >= jug_y:
            d = jug_y - y
            queue.append(node(x - d, jug_y, prev))
        else:
            if x + y != 0:
                queue.append(node(0, x + y, prev))

    # emptying jug1
    if x > 1:
        queue.append(node(0, y, prev))
    # emptying jug2
    if y > 1:
        queue.append(node(x, 0, prev))

def solve_jug_problem():
    queue = [node(0, 0, None)]

    while queue:
        cur = queue.pop(0)
        x = cur.x
        y = cur.y

        if y == target and choice == 2:
            return cur
        elif x == target and choice == 1:
            return cur
        else:
            conditions(x, y, cur)
if __name__ == "__main__":
    while True:
        jug_x = int(input("Enter the quantity of jug x: "))
        jug_y = int(input("Enter the quantity of jug y: "))
        choice = int(input("Enter the target jug (1 for jug x, 2 for jug y): "))
        target = int(input("Enter the target value: "))

        ansnode = solve_jug_problem()

        while ansnode is not None:
            print(ansnode.x, ansnode.y)
            ansnode = ansnode.prev

        user_input = input("Do you want to continue? (yes/no): ")
        if user_input.lower() != "yes":
            break
```

OUTPUT:



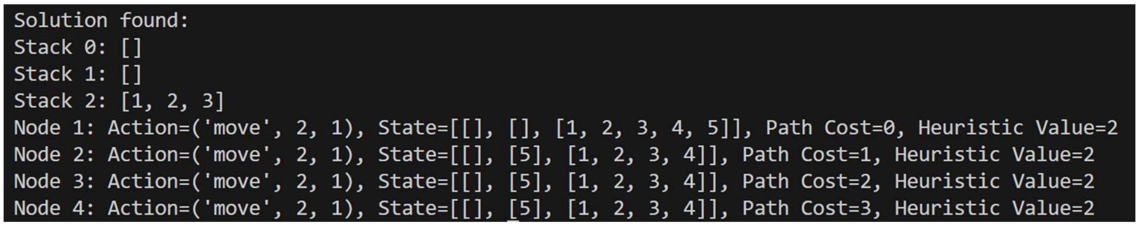
```
Enter the quantity of jug x: 3
Enter the quantity of jug y: 5
Enter the target jug (1 for jug x, 2 for jug y): 2
Enter the target value: 4
3 4
2 5
2 0
0 2
3 2
0 5
0 0
Do you want to continue? (yes/no): yes
Enter the quantity of jug x: 4
Enter the quantity of jug y: 7
Enter the target jug (1 for jug x, 2 for jug y): 1
Enter the target value: 2
2 7
4 5
0 5
4 1
0 1
1 0
1 7
4 4
0 4
4 0
0 0
Do you want to continue? (yes/no): no
```

5) HEURISTIC SEARCH

CODE:

```
from heapq import heappop, heappush


initial_state = [[], [], [1,2,3,4,5]]
goal_state = [[1,2,3,4,5], [], []]


def heuristic(state):
    max_height = max(len(stack) for stack in state)
    return sum(len(stack) != max_height for stack in state)

actions = [('move', 0, 1), ('move', 0, 2), ('move', 1, 0), ('move', 1, 2), ('move', 2, 0), ('move', 2,
1)]

def apply_action(state, action):
    new_state = [list(stack) for stack in state]
    move_type, source, target = action

    if move_type == 'move':
        if len(new_state[source]) > 0:
            block = new_state[source].pop()
            new_state[target].append(block)

    return new_state

def a_star_search(initial_state, goal_state, heuristic):
    frontier = [(0 + heuristic(initial_state), 0, initial_state, [])]
    explored = set()

    while frontier:
        _, path_cost, current_state, path = heappop(frontier)

        if current_state == goal_state:
            return path

        explored.add(tuple(map(tuple, current_state)))

        for action in actions:
            new_state = apply_action(current_state, action)

            if tuple(map(tuple, new_state)) not in explored:
                new_path = path + [action]
                g = path_cost + 1  # Uniform cost

                # f(n) = g(n) + h(n)
                f = g + heuristic(new_state)
                heappush(frontier, (f, g, new_state, new_path))

    return None

# Perform A* search
solution_path = a_star_search(initial_state, goal_state, heuristic)

# Print the first four expanded nodes
for i in range(4):
    action = solution_path[i] if i < len(solution_path) else None
    state = initial_state if i == 0 else apply_action(initial_state, action)
    path_cost = i
    heuristic_value = heuristic(state)
    print(f"Node {i + 1}: Action={action}, State={state}, Path Cost={path_cost}, Heuristic Value={heuristic_value}")


def heuristic_sum_of_distances(state, goal_state):
    total_distance = 0

    for block in set(block for stack in goal_state for block in stack):
        current_position = find_block_position(state, block)
        goal_position = find_block_position(goal_state, block)
        total_distance += manhattan_distance(current_position, goal_position)

    return total_distance

def find_block_position(state, block):
    for i, stack in enumerate(state):
        if block in stack:
            return i, stack.index(block)

def manhattan_distance(position1, position2):
    return abs(position1[0] - position2[0]) + abs(position1[1] - position2[1])
```

OUTPUT:



```
Solution found:
Stack 0: []
Stack 1: []
Stack 2: [1, 2, 3]
Node 1: Action=('move', 2, 1), State=[[], [], [1, 2, 3, 4, 5]], Path Cost=0, Heuristic Value=2
Node 2: Action=('move', 2, 1), State=[[], [5], [1, 2, 3, 4]], Path Cost=1, Heuristic Value=2
Node 3: Action=('move', 2, 1), State=[[], [5], [1, 2, 3, 4]], Path Cost=2, Heuristic Value=2
Node 4: Action=('move', 2, 1), State=[[], [5], [1, 2, 3, 4]], Path Cost=3, Heuristic Value=2
```