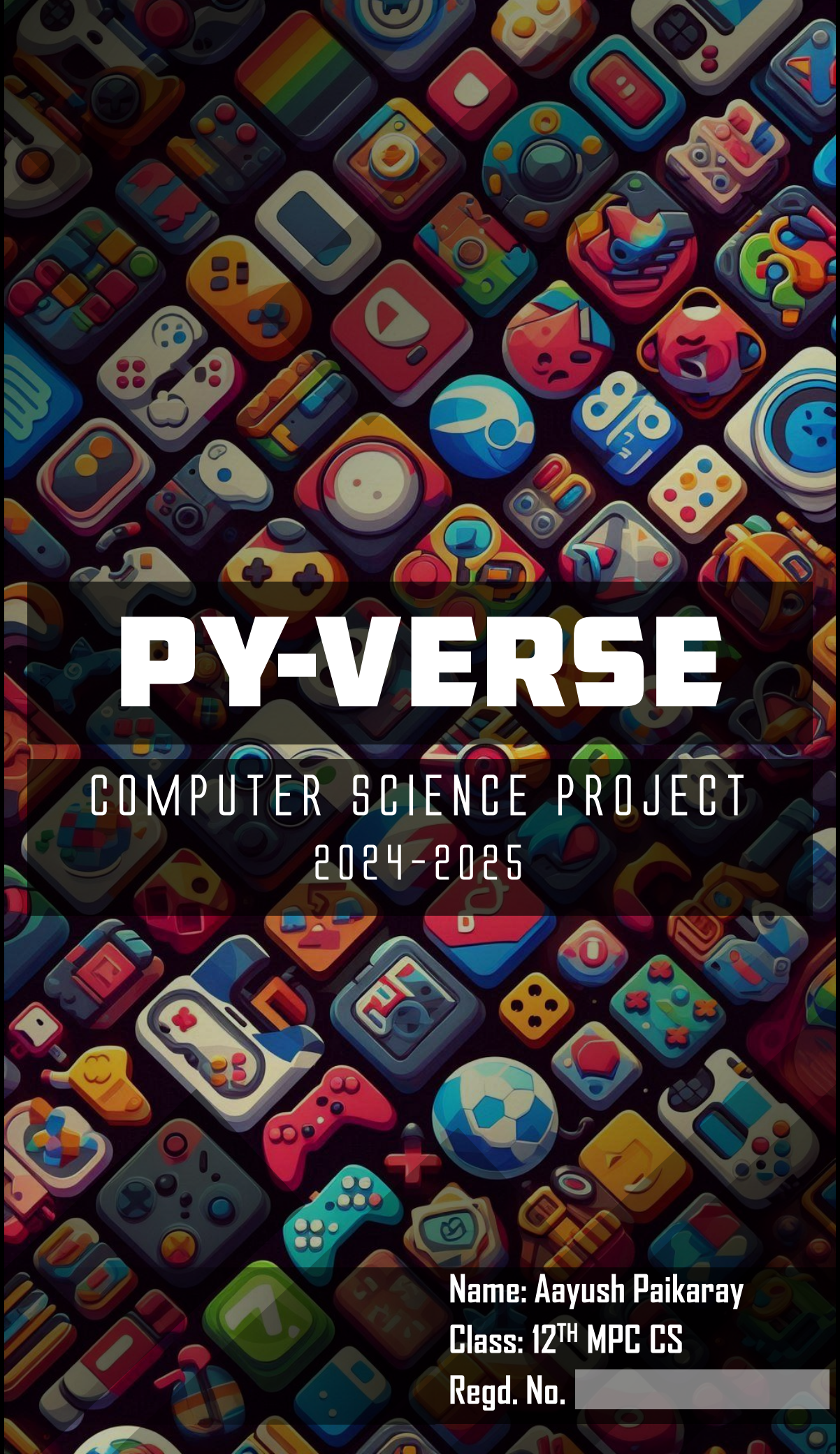


PY-VERSE

COMPUTER SCIENCE PROJECT

2024-2025

Name: Aayush Paikaray

Class: 12TH MPC CS

Regd. No.

SRI SATHYA SAI HIGHER SECONDARY SCHOOL

CERTIFICATE

Certified that this is a bonafide project done by

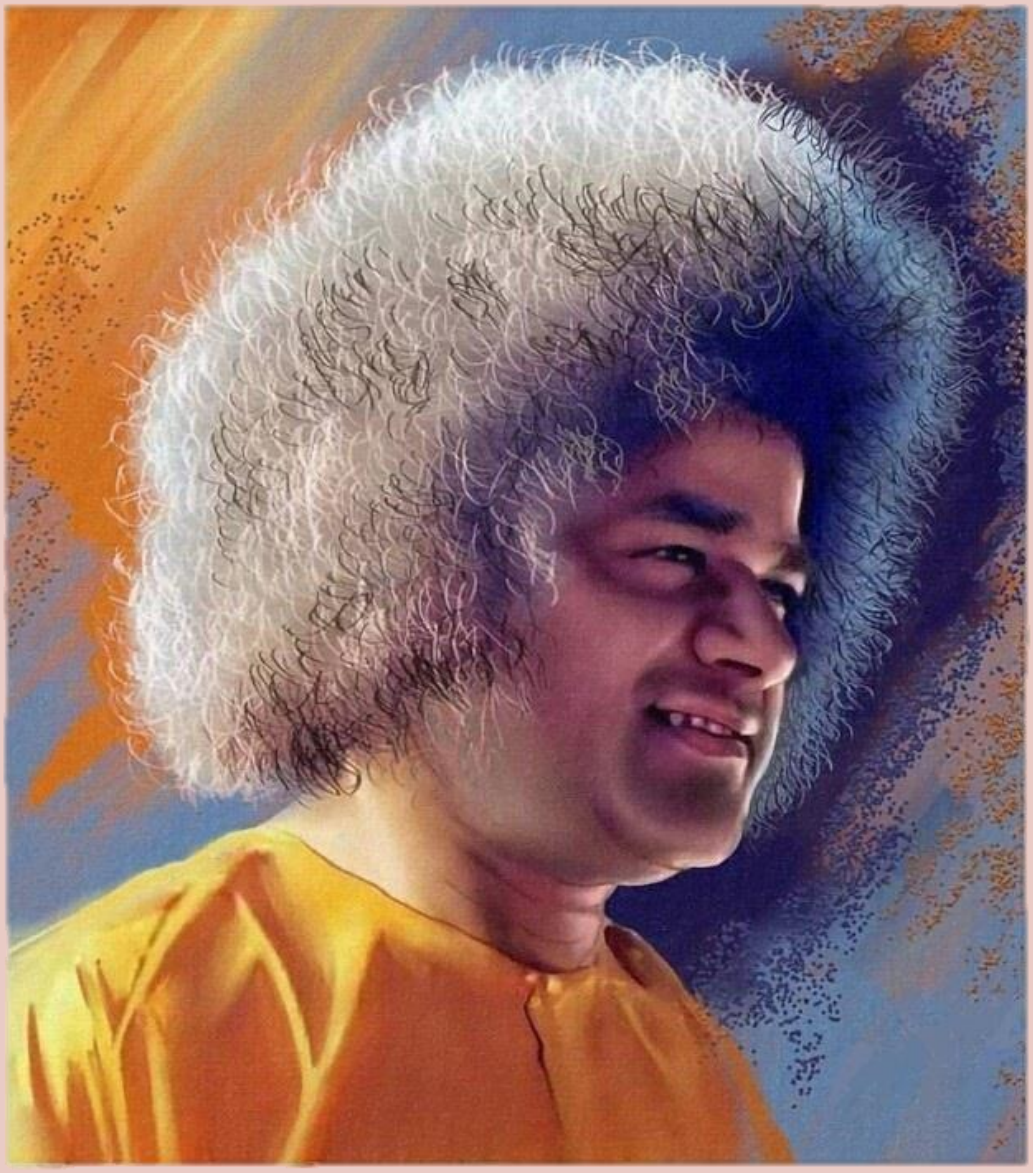
Aayush Paikaray

Regd. No._____

in AISSCE course in COMPUTER SCIENCE during the year 2024-25.

This Project work is submitted for the **Practical Examination** conducted by CBSE ,New Delhi.

PRINCIPAL	EXTERNAL EXAMINER	INTERNAL EXAMINER



Dedicated at the lotus feet of

BHAGAWAN

SRI SATHYA SAI BABA

ACKNOWLEDGEMENTS

Behind every act we do, there is an unseen hand I would like to gratefully acknowledge the hand of our beloved **Swami** without whose grace the project would not have become a success.

For being a constant source of inspiration, our guide, and always being by my side in the hour of need I would like to extend my heartfelt gratitude to **Sri Venkateswar Prusty** for teaching us with much love and patience.

I would like to thank our principal **Sri Sivaramakrishnaiah** who provided us with such excellent facilities at our lab.

I am most grateful to my **parents** for always encouraging and supporting me.

I am thankful to the collaborative environment wrought about by my **classmates** and **my seniors**.

Contents

Aim 1

Introduction 2

Theory.....3

Implementation 4

Program code 9

Future enhancements 45

Bibliography 46

Aim

The aim of this project is to develop a dynamic gaming platform called

PY-VERSE

to create a collaborative gaming ecosystem where users can not only play but also upload and share their own games.

Objectives

The primary objectives of the platform are:

- To create an intuitive, easy-to-use interface for both developers and gamers.
- To allow users to upload their game creations directly to the platform, making them accessible to others.
- To support a range of game types, from casual single-player games to networked multiplayer games.
- To ensure security and data integrity by using local and remote server-based data storage.

Introduction

Unlike traditional arcade systems, **Py-verse** allows any developer to upload their game, making it instantly accessible to all users on the platform. where users can create, upload, and play a variety of games.

The App features:

- **Games Tab**: Play games uploaded by other users via network.
- **Upload Tab**: Upload your games - make it available to all.
- **Groups Tab**: Chat with other users by creating groups.
- **Friends Tab**: One-on-One chatting feature for players to communicate... (future enhancements).
- **Settings Tab** : Offers a wide range of themes choose and other features.

Theory

The application is divided into three key layers:

Client Layer:

This includes the interface where users interact with the platform. It is designed using custom tkinter, a Python library for building GUIs, and uses socket for client-server communication.

Server Layer:

The app uses a central server hosted at **INFOLAB**, which acts as a repository for game files, icons, and user data. The system integrates with this central server to allow game uploads, downloads, and updates in real-time.

Data Management Layer:

This layer handles the storage and retrieval of user data, game files, and assets. The data is stored locally in a local_data directory on the user's machine and remotely on a shared INFOLAB Server Directory, ensuring that user games and data are accessible across devices.

Implementation

Components

The core components of **Py-verse** include:

Login & User Authentication:

This ensures only registered users can access and upload games. Users' credentials are verified by checking against stored data on the server.

Game Uploading System:

Users can upload their game files, including the main game file and icon, through a GUI interface. The application facilitates file selection and uploads to the infolab-server, maintaining the integrity of the game data. Users can choose to play, delete, or re-upload their games.

Game Display & Management:

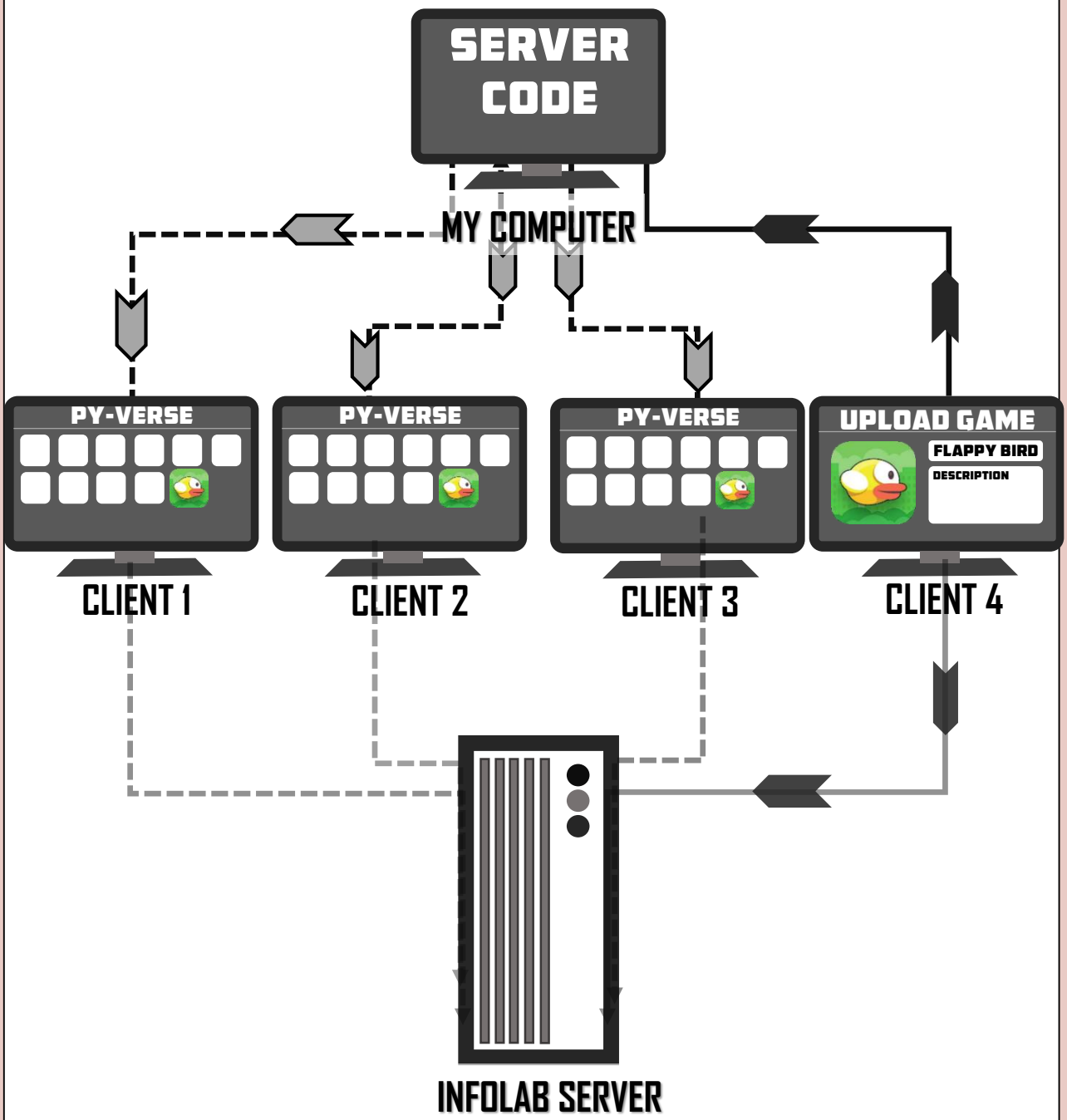
The platform displays all available games using a scrollable interface, with game thumbnails dynamically loaded.

Customization & Theming:

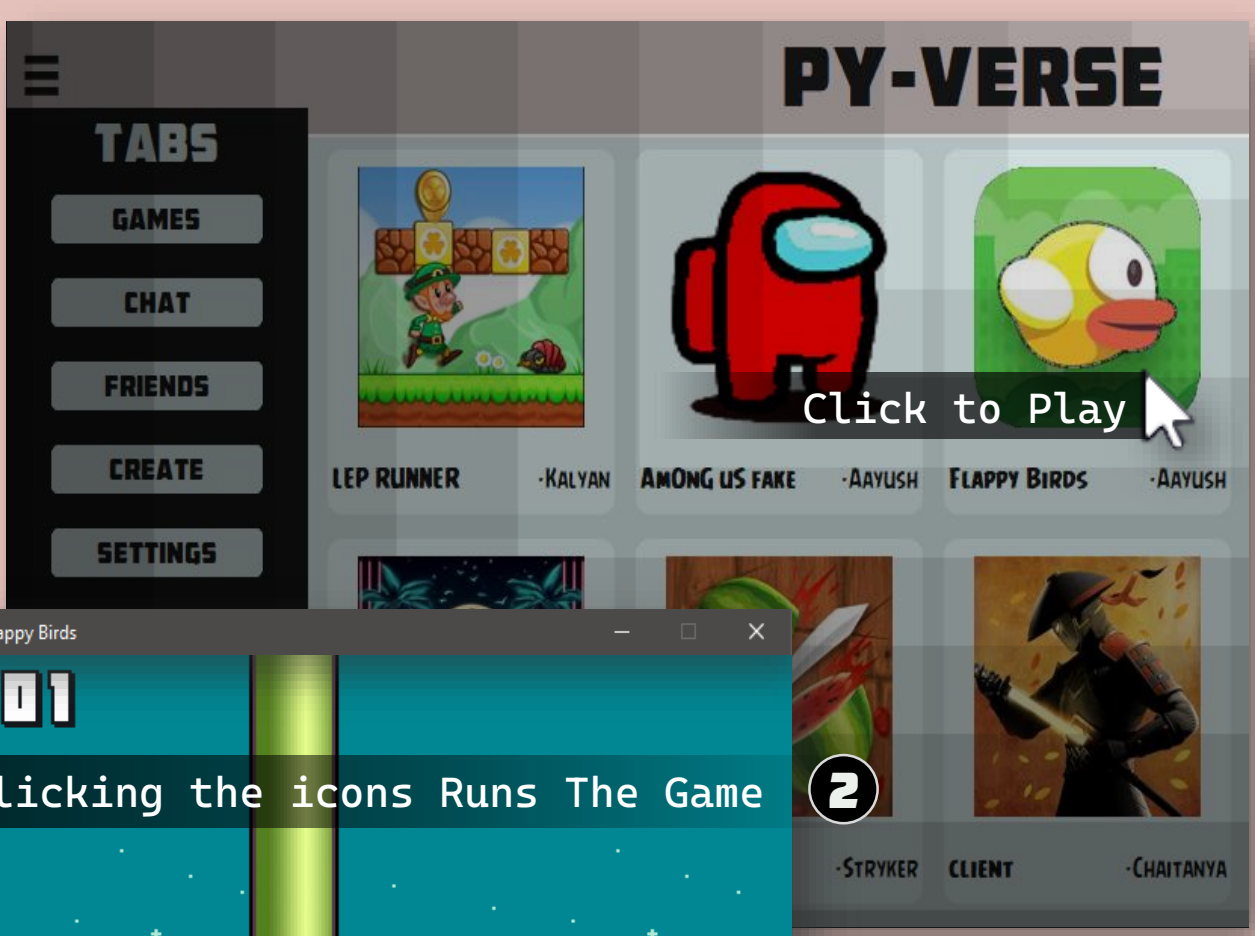
Users can personalize their experience by selecting from predefined color themes, such as 'Dark Theme' or 'Light Theme,' allowing for a more customizable interface.

Implementation

MECHANISM



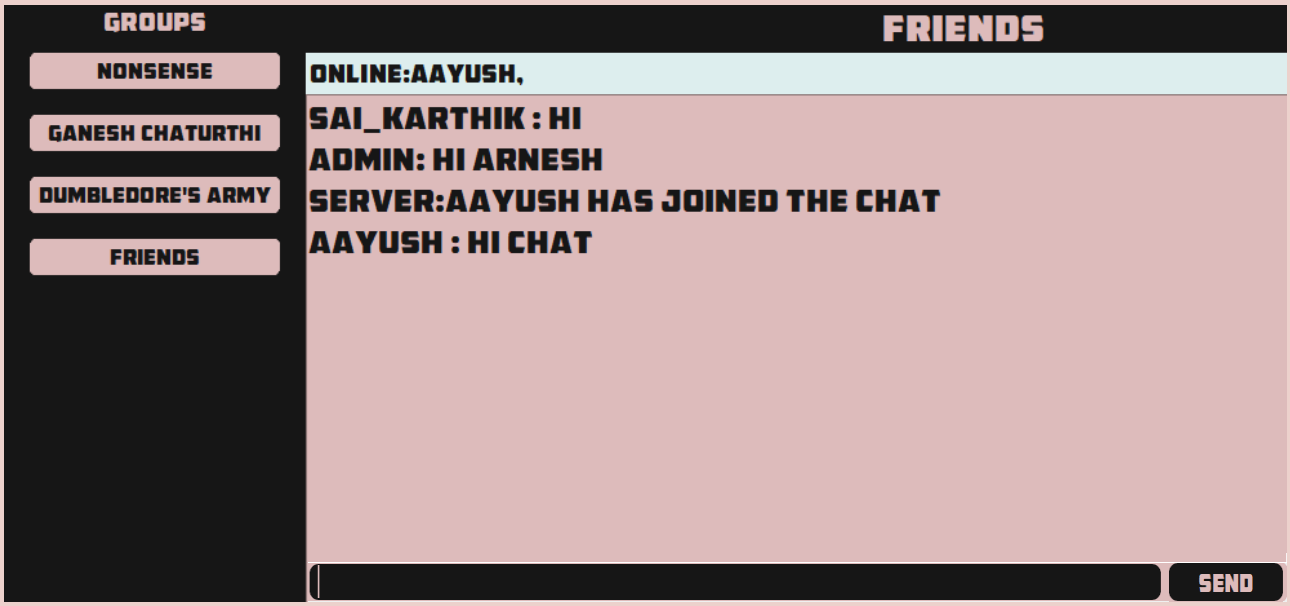
Implementation



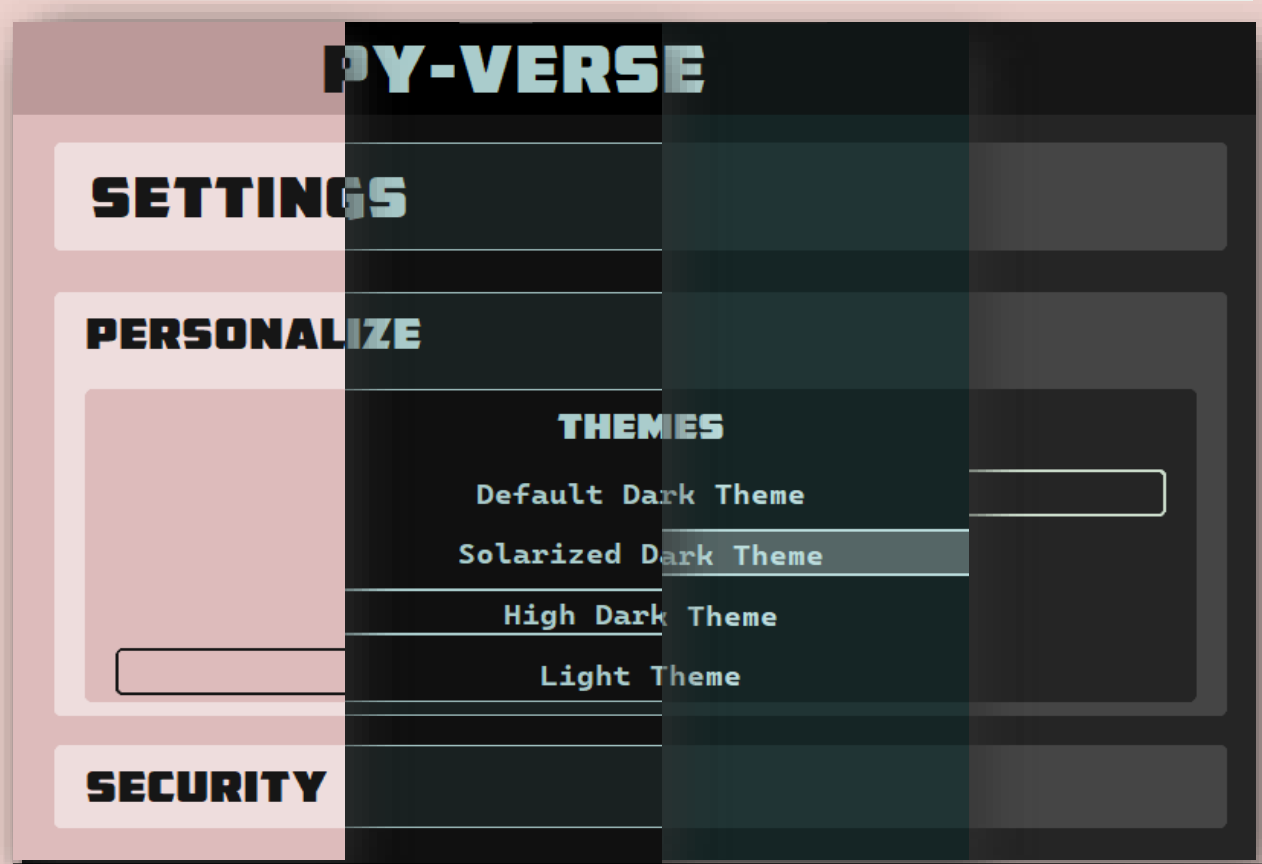
Games Uploaded by other users on their systems appear in the tab for everyone.

Implementation

Implementation



Chatting system achieved through multi-server communication



Variety of Themes that are changed with the help of color variables

PROGRAM CODE

CLIENT SIDE CODE

Importing built-in modules and making the code auto-updated

```
1   import socket, threading, pickle, os, re, subprocess, sys, shutil,
    time, random
2   if open(r'\\infolab-server\Student Share\Pycrium\Pycrium.py','r+').
    read() != open(__file__,'r+').read():
3       open(__file__,'w+').write(open(r'\\infolab-server\Student
        Share\Pycrium\Pycrium.py','r+').read())
4       subprocess.Popen([sys.executable,__file__])
5       print('Updated...')
6       sys.exit(1)
```

Downloading modules that are not installed

```
8   import tkinter as tk
9   import tkinter as tk
10  from tkinter import simpledialog, messagebox, filedialog, ttk
11  from tkinter.ttk import Progressbar
12  try:
13      from PIL import Image, ImageTk,ImageFont
14  except ImportError:
15      print("Pillow is not installed. Installing...")
16      try:
17          subprocess.check_call([sys.executable, "-m", "pip",
            "install", "Pillow"])
18          from PIL import Image, ImageTk
19      except subprocess.CalledProcessError:
20          print("Failed to install Pillow. Please log into internet...")
21          messagebox.showerror('"Pillow" module not found ','Log into
            internet and restart')
22
23  try:
24      import customtkinter as ctk
25  except ImportError:
26      print("customtkinter is not installed. Installing...")
27      try:
28          subprocess.check_call([sys.executable, "-m", "pip",
            "install", "customtkinter"])
29          import customtkinter as ctk
30      except subprocess.CalledProcessError:
31          print("Failed to install customtkinter. Please log into
            internet...")
32          sys.exit(1)
```

PROGRAM CODE

Selecting themes for the app

```
34  COLOUR_PALLETTE = {'Default Dark Theme':('#161616','#CCDDCC',
    '#252525','#353535','#454545','#665544','#CCDDCC','#252525','#FF5252',
    0),
35                     'Solarized Dark Theme':('#101616','#BBDDDD',
                       '#152525','#253535','#203434','#556666','#BBDDDD',
                       '#152525','#FF5252',0),
36                     'High Dark Theme':('#000000','#AACCCC','#101010',
                       '#172020','#192121','#556666','#AACCCC','#101010',
                       '#FF5252',1),
37                     'Light Theme':('#BB9999','#161616','#DDBBBB',
                       '#DDEEEE','#EEDDDD','#887777','#DDBBBB','#161616',
                       '#FF5252',0)}
38  # * DEFAULT COLOUR
39  THEME = 'Light Theme'
40  LALK,CREAM,DARG,LARG,GARG,BREW,MENT,TECK,DANG,BORR = COLOUR_PALLETTE
    ['Light Theme']
41  NAME = 'Py-verse'
42  CURR = os.getcwd()
43  INFOLAB_SERVER_DIRECTORY = r"\\infolab-server\Student
    Share\AayushPaikaray\storage"
```

Required functions for client-server communication

```
57  def load_local_data() -> dict:
59      try:
60          with open(os.path.join(LOCAL_DATA_DIRECTORY, "user.dat"),
            "rb") as f:
61              return pickle.load(f)
62      except FileNotFoundError:
63          return {}
64
65  def save_local_data(data):
66      os.chdir(CURR)
67      with open(os.path.join(LOCAL_DATA_DIRECTORY, "user.dat"), "wb")
        as f:
68          pickle.dump(data, f)
69
70  def load_settings_data() -> dict:
71      os.chdir(CURR)
72      try:
73          with open(os.path.join(LOCAL_DATA_DIRECTORY, "settings.dat"),
            "rb") as f:
74              return pickle.load(f)
75      except FileNotFoundError:
76          return {}
```

PROGRAM CODE

```
70  def load_settings_data() -> dict:
71      os.chdir(CURR)
72      try:
73          with open(os.path.join(LOCAL_DATA_DIRECTORY, "settings.dat"),
            "rb") as f:
74              return pickle.load(f)
75      except FileNotFoundError:
76          return {}
77
78  def connect_to_server():
79      client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
80      client.connect(SERVER_ADDR)
81      return client
82
83  def verify_credentials(username, password):
84      client = connect_to_server()
85      request = {"action": "login", "username": username, "password":
        password}
86      client.send(pickle.dumps(request))
87      response = pickle.loads(client.recv(4096))
88      client.close()
89      if response.get("status") == "success":
90          __USERNAME__ = username
91          return True
92      return False
93
94  def signup(username, password):
95      client = connect_to_server()
96      request = {"action": "signup", "username": username, "password":
        password}
97      client.send(pickle.dumps(request))
98      response = pickle.loads(client.recv(4096))
99      client.close()
100     return response.get("status") == "success"
```

```
102 def copy_icon_to_storage(icon_path):
104         raise FileNotFoundError(f"Icon file not found: {icon_path}")
105
106     icon_name = os.path.basename(icon_path)
107     dest_icon_path = os.path.join(ICONS_DIR, icon_name)
108     shutil.copyfile(icon_path, dest_icon_path)
109     return dest_icon_path
110
```

PROGRAM CODE

```
111  def upload_game(username, game_name, game_dir, game_main_file,
     game_icon, description, download, view):
112      client = connect_to_server()
113      dest_dir = os.path.join(INFOLAB_SERVER_DIRECTORY, os.path.basename
         (game_dir))
114      shutil.copytree(game_dir, dest_dir, dirs_exist_ok=True)
115      relative_main_file = os.path.relpath(game_main_file, game_dir[:len
         (game_dir)-game_dir[::-1].find('/')])
116      print(relative_main_file,game_dir, game_main_file,dest_dir,sep=',
         ')
117      dest_icon_path = copy_icon_to_storage(game_icon)
118      request = {
119          "action": "upload",
120          "username": username,
121          "game_name": game_name,
122          "game_main_file": relative_main_file,
123          "game_icon": os.path.relpath(dest_icon_path,
             INFOLAB_SERVER_DIRECTORY),
124          "description": description,
125          "download": download,
126          "view": view
127      }
128      client.send(pickle.dumps(request))
129      response = pickle.loads(client.recv(4096))
130      client.close()
131      return response.get("status") == "success"
132
133  def reupload_game(username, game_name, game_dir, game_main_file,
     game_icon):
134      client = connect_to_server()
135      dest_dir = os.path.join(INFOLAB_SERVER_DIRECTORY, os.path.basename
         (game_dir))
136      shutil.copytree(game_dir, dest_dir, dirs_exist_ok=True)
137      relative_main_file = os.path.relpath(game_main_file,
         INFOLAB_SERVER_DIRECTORY)
138      dest_icon_path = copy_icon_to_storage(game_icon)
139      request = {
140          "action": "reupload",
141          "username": username,
142          "game_name": game_name,
143          "game_main_file": relative_main_file,
144          "game_icon": os.path.relpath(dest_icon_path,
             INFOLAB_SERVER_DIRECTORY)
145      }
146      client.send(pickle.dumps(request))
147      response = pickle.loads(client.recv(4096))
148      client.close()
149      return response.get("status") == "success"
```

PROGRAM CODE

```
151  def delete_game(username, game_name):
152      client = connect_to_server()
153      request = {"action": "delete", "username": username, "game_name":
         game_name}
154      client.send(pickle.dumps(request))
155      response = pickle.l
156      client.close()
157      return response.get
158
159  def rename_game(usernam
160      client = connect_to
161      request = {"action"
         "old_game_name": ol
162      client.send(pickle.dumps(request))
163      response = pickle.loads(client.recv(4096))
164      client.close()
165      return response.get("status") == "success"
166
167  def leave_group(username, port):
168      client = connect_to_server()
169      request = {"action": "leave_group", "username": username, "PORT":
         port}
170      client.send(pickle.dumps(request))
171      client.close()
172
173  def delete_group(port):
174      client = connect_to_server()
175      request = {"action": "delete_group", "PORT": port}
176      client.send(pickle.dumps(request))
177      client.close()
178
179  def get_games():
180      client = connect_to_server()
181      request = {"action": "get_games"}
182      client.send(pickle.dumps(request))
183      response = client.recv(4096)
184      client.close()
185      if not response:
186          print("No data received from server")
187          return []
188      try:
189          games = pickle.loads(response)
190          if isinstance(games, list):
191              return games
192          else:
193              print("Invalid response format from server")
194              return []
```

```
(function) def dumps(
    obj: Any,
    protocol: int | None = None,
    *,
    fix_imports: bool = True,
    buffer_callback: _BufferCallback = None
) -> bytes
```

PROGRAM CODE

```
195         except Exception as e:
196             print(f"Error decoding server response: {e}")
197             return []
198 
199     def get_players():
200         client = connect_to_server()
201         request = {"action": "get_players"}
202         client.send(pickle.dumps(request))
203         response = client.recv(4096)
204         client.close()
205         if not response:
206             print("No data received from server")
207             return []
208         try:
209             games = pickle.loads(response)
210             if isinstance(games, list):
211                 return games
212             else:
213                 print("Invalid response format from server")
214                 return []
215         except Exception as e:
216             print(f"Error decoding server response: {e}")
217             return []
218 
219     def get_groups():
220         client = connect_to_server()
221         request = {"action": "get_groups",'username':load_local_data()
        ['username']}
222         client.send(pickle.dumps(request))
223         response = client.recv(4096)
224         client.close()
225         if not response:
226             print("No data received from server")
227             return []
228         try:
229             groups = pickle.loads(response)
230             if isinstance(groups, list):
231                 return groups
232             else:
233                 print("Invalid response format from server")
234                 return []
235         except Exception as e:
236             print(f"Error decoding server response: {e}")
237             return []
```

PROGRAM CODE

```
239  def get_playing_count(game_name):
240      client = connect_to_server()
241      request = {"action": "get_playing_count", "game_name": game_name}
242      client.send(pickle.dumps(request))
243      response = client.recv(4096)
244      client.close()
245
246      if not response:
247          print("No data received from server")
248          return 0
249
250      try:
251          playing_count = pickle.loads(response)
252          if isinstance(playing_count, int):
253              return playing_count
254          else:
255              print("Invalid response format from server")
256              return 0
257      except Exception as e:
258          print(f"Error decoding server response: {e}")
259          return 0
260
261  def create_group(group_name:str,group_members:list,group_mode:bool,
     admin:str) -> bool:
262      client = connect_to_server()
263      request = {"action": "create_group", "group_name": group_name,
         "group_members": group_members, 'group_mode':group_mode,
         'chat_history':'', 'admin':admin}
264      client.send(pickle.dumps(request))
265      response = pickle.loads(client.recv(4096))
266      client.close()
267      return response
```

PROGRAM CODE

Class for the loading screen

```
269  class ProgressScreen(tk.Tk):
270      def __init__(self,):
271          super().__init__()
272          self.start_time = time.time()
273          self.geometry('500x200+500+300')
274          self.resizable(False,False)
275          self.configure(bg=DARG)
276          self.overrideredirect(True)
277          self.value = 1
278
279          self.main_label=tk.Label(self,text=NAME,font=("AMCAP Eternal",
             40),bg=DARG,fg=CREAM)
280
281          self.main_label.pack(pady=30)
282
283          self.progress_label=tk.Label(self,text="Loading...",font=
             ("AMCAP Eternal",10),bg=DARG,fg=CREAM)
284          self.progress_label.pack(pady=10)
285
286          self.progress=Progressbar(self, orient=tk.HORIZONTAL,
             mode='determinate',style='Striped.Horizontal.TProgressbar')
287          self.progress.pack(side="bottom",fill='x')
288
289          self.load()
290          self.mainloop()
291
292      def loa  (parameter) self: Self@ProgressScreen
293          if self.progress['value']!=101:
294              if time.time()-self.start_time > 10:
295                  self.progress['value'] = 100
296              self.progress_label.config(text=(f'Loading...{int(self.
                 progress["value"])}'))
297              self.progress['value']+=self.value
298              self.randval = random.randint(1,100-int(self.progress
                 ["value"])//2)
299              if self.randval==1:
300                  self.progress_label.after(100+random.randint(2000,2000
                     +int(self.progress["value"])*40),self.load)
301              else:self.progress_label.after(50,self.load)
302          else:
303              try:time.sleep(14-(time.time()-self.start_time));self.
                 withdraw()
304              except:self.withdraw()
```

PROGRAM CODE

```
309  class Tabs():
310      alltabs=[]
311      def __init__(self,tetx,app,frame):
312          self.frame=frame
313          self.label=ctk.CTkButton(app.toggle_menu_fm,
314                                   text=f'{tetx}', fg_color=DARG,
315                                   text_color=CREAM,hover_color=BREW,font=('AM
316          self.label.pack(side=tk.TOP,pady=10)
317          Tabs.alltabs.append(self);self.app=app
318      def activate(self):
319          try:self.app.toggle_collapse()
320          except:pass
321          for tab in Tabs.alltabs:tab.frame.pack_forget()
322          self.frame.pack(expand=True, fill="both")
323
324  class CTabs():
325      alltabs=[]
326      def __init__(self,tetx,app,frame,add_button=True):
327          self.frame=frame;self.t=tetx;self.app=app
328          if add_button:
329              self.label=ctk.CTkButton(app.toggle_menu_fm,
330                                       text=f'{tetx}', fg_color=DARG,
331                                       text_color=CREAM,hover_color=BREW,font=
332              self.label.pack(side=tk.TOP,pady=10,padx=20,fill=tk.X)
333          CTabs.alltabs.append(self);self.app=app
334      def activate(self):
335          for tab in CTabs.alltabs:tab.frame.pack_forget()
336          self.frame.pack(expand=True, fill=tk.BOTH)
337          self.app.label['text'] = self.t
338
339  class Application(tk.Tk):
340
341      def __init__(self):
342          super().__init__()
343          self.title(NAME)
344          self.geometry("1550x1000")
345          self.configure(bg=LALK)
346          self.attributes('-fullscreen', True)
347          self.bind('<Escape>', lambda event:self.destroy())
348          self.local_data = load_local_data()
349          if self.local_data:
350              if verify_credentials(self.local_data["username"], self.loc
351                  threading.Thread(target=lambda:ProgressScreen()).start(
352                  self.show_menu()
353              else:
354                  self.show_login()
355          else:
356              self.show_login()
```

PROGRAM CODE

```
358     def show_login(self):
359         for widget in self.winfo_children():
360             widget.destroy()
361 
362         fm = tk.Frame(self,bg=LALK,width=1000,height=1000,)
363         tk.Label(master=fm, text=NAME, font=("AMCAP Eternal", 45,
            "bold"), foreground=CREAM, background=LALK).pack()
364         main_fm = tk.Frame(fm,bg=DARG,width=1000,height=1000,
            highlightbackground=CREAM,highlightcolor=CREAM,
            highlightthickness=BORR)
365         sub_fm = tk.Frame(main_fm,bg=DARG,width=900,height=800)
366         ss_fm=tk.Frame(main_fm,bg=DARG,width=900,height=800)
367         self.username_entry = ctk.CTkEntry(ss_fm,font=("AMCAP
            Eternal", 25, "bold"), text_color=CREAM, fg_color=GARG,
            bg_color=DARG,corner_radius=10,placeholder_text='USERNAME',
            placeholder_text_color=CREAM,border_width=BORR,width=600,
            height=50,border_color=CREAM)
368         self.username_entry.pack(padx=20,pady=30)
369         self.password_entry = ctk.CTkEntry(ss_fm,font=("AMCAP
            Eternal", 25, "bold"), text_color=CREAM, show='*',
            fg_color=GARG, bg_color=DARG,corner_radius=10,
            placeholder_text='PASSWORD',placeholder_text_color=CREAM,
            border_width=BORR,width=600,height=50,border_color=CREAM)
370         self.password_entry.pack(padx=20,pady=0)
371         ctk.CTkButton(sub_fm, text="Login", command=self.login, font=
            ("AMCAP Eternal", 30), text_color=CREAM, fg_color=LALK,
            bg_color=DARG, corner_radius=10, border_width=BORR,
            hover_color=BREW,border_color=CREAM).pack(pady=20)
372 
373         ss_fm.pack()
374         sub_fm.pack()
375         main_fm.pack()
376         fm.pack(pady=200)
377         ctk.CTkButton(fm, text="Don't have an account? Signup",
            command=self.show_signup, font=("Cascadia Code", 15),
            text_color=CREAM, fg_color=LALK, bg_color=LALK,
            corner_radius=10, border_width=0, hover=BREW,).pack(side=tk.
            RIGHT)
378 
379 
380     def show_signup(self):
381         for widget in self.winfo_children():
382             widget.destroy()
383 
384         self.new_username = tk.StringVar()
385         self.new_password = tk.StringVar()
386         self.confirm_password = tk.StringVar()
```

PROGRAM CODE

```
388         fm = tk.Frame(self,bg=LALK,width=1000,height=1000)
389         tk.Label(master=fm, text=NAME, font=("AMCAP Eternal", 45,
            "bold"), foreground=CREAM, background=LALK).pack()
390         main_fm = tk.Frame(fm,bg=DARG,width=1000,height=1000,
            highlightbackground=CREAM,highlightcolor=CREAM,
            highlightthickness=BORR)
391         sub_fm = tk.Frame(main_fm,bg=DARG,width=900,height=800)
392         ss_fm=tk.Frame(main_fm,bg=DARG,width=900,height=800)
393         self.signin_username_entry = ctk.CTkEntry(ss_fm,font=("AMCAP
            Eternal", 25, "bold"), text_color=CREAM, fg_color=GARG,
            bg_color=DARG,corner_radius=10,placeholder_text='USERNAME',
            placeholder_text_color=CREAM,border_width=BORR,width=600,
            height=50,border_color=CREAM)
394         self.signin_username_entry.pack(padx=20,pady=20)
395         self.signin_username_entry.bind("<KeyRelease>", self.
            validate_username)
396         temp_fm = tk.Frame(ss_fm,bg=DARG)
397         self.username_error_label = ctk.CTkLabel(temp_fm, text="",
            font=("Cascadia Code", 15), text_color=CREAM, fg_color=DARG,
            bg_color=DARG, )
398         self.username_error_label.pack(side=tk.RIGHT)
399         temp_fm.pack(expand=True,fill=tk.X,padx=30)
400         self.signin_password_entry = ctk.CTkEntry(ss_fm,font=("AMCAP
            Eternal", 25, "bold"), text_color=CREAM, show='*',
            fg_color=GARG, bg_color=DARG,corner_radius=10,
            placeholder_text='PASSWORD',placeholder_text_color=CREAM,
            border_width=BORR,width=600,height=50,border_color=CREAM)
401         self.signin_password_entry.pack(padx=20,pady=0)
402         self.signin_confirm_password_entry = ctk.CTkEntry(ss_fm,font=
            ("AMCAP Eternal", 25, "bold"), text_color=CREAM, show='*',
            fg_color=GARG, bg_color=DARG,corner_radius=10,
            placeholder_text='Confirm PASSWORD',
            placeholder_text_color=CREAM,border_width=BORR,width=600,
            height=50,border_color=CREAM)
403         self.signin_confirm_password_entry.pack(padx=20,pady=20)
404         ctk.CTkButton(sub_fm, text="Signup", command=self.signup,
            font=("AMCAP Eternal", 30), text_color=CREAM, fg_color=LALK,
            bg_color=DARG, corner_radius=10, border_width=BORR,
            hover_color=BREW,border_color=CREAM).pack(pady=20)
405
406         ss_fm.pack()
407         sub_fm.pack()
408         main_fm.pack()
409         fm.pack(pady=150)
410         ctk.CTkButton(fm, text="Already have an account? Login",
```

PROGRAM CODE

```
414     def validate_username(self, event):
415         username = self.signin_username_entry.get()
416         if re.match(r"^[a-zA-Z0-9_]{3,20}$", username):
417             self.username_error_label.configure(text="")
418         else:
419             self.username_error_label.configure(text="Invalid
                username")
420
421     def login(self):
422         username = self.username_entry.get()
423         password = self.password_entry.get()
424         if verify_credentials(username, password):
425             save_local_data({"username": username, "password":
                password})
426             self.show_menu()
427         else:
428             messagebox.showerror("Login Failed", "Invalid
                credentials")
429
430     def signup(self):
431         username = self.signin_username_entry.get()
432         password = self.signin_password_entry.get()
433         confirm_password = self.signin_confirm_password_entry.get()
434         if password != confirm_password:
435             self.username_error_label.configure(text="Passwords do
                not match")
436             return
437         if signup(username, password):
438             self.username_error_label.configure(text="Account created
                successfully")
439             save_local_data({"username": username, "password":
                password})
440             self.local_data=load_local_data()
441             if verify_credentials(self.local_data["username"], self.
                local_data["password"]):
442                 self.show_menu()
443         else:
444             self.username_error_label.configure(text="Username
                already exists")
445
446     def show_menu(self):
447         for widget in self.winfo_children():
448             widget.destroy()
449         self.head_frame=ctk.CTkFrame(master=self,bg_color=LALK,
            border_color=CREAM,border_width=0,fg_color=LALK)
450         self.toggle_btn=tk.Button(self.head_frame, text='☰', bg=LALK,
            fg=CREAM, background=LALK,
```

PROGRAM CODE

```
451                         font=('Bold', 20), bd=0,
                            activebackground=DARG, activeforeground=CREAM,
452                         command=self.toggle_menu)
453         self.toggle_btn.pack(side=tk.LEFT, fill=tk.Y, anchor=tk.W)
454         img = Image.open(r'\\infolab-server\Student
            Share\AayushPaikaray\storage\Icons\profile.png')
455         img = img.resize((38,38))
456         thumbnail = ImageTk.PhotoImage(img)
457         self.user_btn=ctk.CTkLabel(self.head_frame, text='',
            text_color=CREAM,bg_color=LALK, fg_color=LALK,
            image=thumbnail,)
458         self.user_btn.pack(side=tk.RIGHT, fill=tk.Y, anchor=tk.W,
            padx=15)
459         self.user_btn=ctk.CTkLabel(self.head_frame,
            text=load_local_data()['username'], text_color=CREAM,
            bg_color=LALK, fg_color=LALK,font=("AMCAP Eternal", 33,
            'bold') )
460         self.user_btn.pack(side=tk.RIGHT, fill=tk.Y, anchor=tk.W)
461         self.label=tk.Label(master=self.head_frame, text=NAME, font=
            ("AMCAP Eternal", 35,"bold"), foreground=CREAM,
            background=LALK)
462         self.label.pack(expand=True, fill="x",side=tk.TOP)
463
464         self.head_frame.pack(side="top", fill="x")
465         self.games_tab = ctk.CTkFrame(self,fg_color=DARG)
466         self.create_tab = ctk.CTkFrame(self,fg_color=DARG)
467         self.chat_tab = ctk.CTkFrame(self,fg_color=DARG)
468         self.settings_tab = ctk.CTkFrame(self,fg_color=DARG)
469         self.toggle_menu();self.toggle_collapse()
470         self.show_games()
471         self.show_create()
472         self.show_settings()
473         threading.Thread(target=self.show_chat).start()
474         self.deiconify()
475
476     def toggle_collapse(self):
477         self.toggle_menu_fm.destroy()
478         self.toggle_btn.config(text='☰',fg=CREAM)
479         self.toggle_btn.config(command=self.toggle_menu)
480     def toggle_menu(self):
481
482
483         self.toggle_menu_fm=tk.Frame(self,bg=CREAM)
484
485         Tabslabel=ctk.CTkLabel(self.toggle_menu_fm,text="Tabs",
            text_color=DARG,font=("AMCAP Eternal",30))
486         Tabslabel.pack(side="top")
```

PROGRAM CODE

```
489         window_height=self.winfo_height()
490
491         self.toggle_menu_fm.place(x=0,y=52,height=window_height,
            width=200)
492
493         self.toggle_btn.config(text="☰")
494         self.toggle_btn.config(command=self.toggle_collapse)
495         self.games=Tabs("Games",self,self.games_tab)
496         self.chat=Tabs("Chat",self,self.chat_tab)
497         self.friends=Tabs("Friends",self,self.games_tab)
498         self.create=Tabs("Create",self,self.create_tab)
499         self.settings=Tabs("Settings",self,self.settings_tab)
500
501     def show_games(self):
502         for widget in self.games_tab.winfo_children():
503             widget.destroy()
504
505         games = get_games()
506         self.num_games = len(games)
507         self.max_columns = 7
508         self.num_columns = min(self.max_columns, max(1, int(7)))
509         self.num_rows = (self.num_games + self.num_columns - 1) //
            self.num_columns
510
511         canvas = tk.Canvas(self.games_tab,bg=DARG,bd=0,borderwidth=0)
512         scrollbar = tk.Scrollbar(self.games_tab, orient="vertical",
            command=canvas.yview, bd=0,borderwidth=0)
513         scrollbar.pack(side="right", fill="y")
514         canvas.pack(side="left", fill="both", expand=True)
515         canvas.configure(yscrollcommand=scrollbar.set)
516         scrollbar.pack_forget()
517
518         self.main_game_frame = tk.Frame(canvas, bg=DARG)
519         canvas.create_window((0, 0), window=self.main_game_frame,
            anchor="nw")
520         self.main_game_frame.bind("<Configure>", lambda event,
            canvas=canvas: canvas.configure(scrollregion=canvas.bbox
            ("all")))
521
522         self.add_games(games)
523
524         def _on_mouse_wheel(event):
525             canvas.yview_scroll(int(-1 * (event.delta / 120)),
                "units")
526         canvas.bind_all("<MouseWheel>", _on_mouse_wheel)
527
528         self.games_tab.update_idletasks()
529         self.games.activate()
530
```

PROGRAM CODE

```
531     def add_games(self,games,load = True):
532         for widget in self.main_game_frame.winfo_children():
533             try:widget.destroy()
534             except:pass
535         for i, game in enumerate(games):
536             game_frame = ctk.CTkFrame(self.main_game_frame,bg_color=DARG, width=190,
                height=210, corner_radius=10,fg_color=LARG,border_color=CREAM,
                border_width=BORR)
537             game_frame.pack_propagate(False)
538 
539             game_name = game.get("game_name")
540             author = game.get("username")
541             game_main_file = game.get("main_file")
542             playing_count = get_playing_count(game_name)
543 
544             icon_path = os.path.join(INFOLAB_SERVER_DIRECTORY, game.get("icon", ""))
545             try:
546                 img = Image.open(icon_path)
547                 img = img.resize((150,150))
548 
549                 thumbnail = ImageTk.PhotoImage(img)
550                 thumbnail_label = ctk.CTkLabel(game_frame, image=thumbnail,
                    corner_radius=50, text='')
551                 thumbnail_label.pack(pady=(10, 10))
552                 name_label = tk.Label(game_frame, text=game_name, font=("Angrybirds",
                    11, "bold"),  fg=CREAM,background=LARG)
553                 name_label.pack(side='left')
554 
555                 author_label = tk.Label(game_frame, text=f"-{author}", font=
                    ("Angrybirds", 11), fg=CREAM,background=LARG)
556                 author_label.pack(side="right")
557 
558                 thumbnail_label.bind("<Button-1>", lambda event,
                    game_main_file=game_main_file: self.play_game(game_main_file))
559             except Exception as Error:print(Error)
560 
561 
562             row_num = i // self.num_columns
563             col_num = i % self.num_columns
564             game_frame.grid(row=row_num, column=col_num, padx=7, pady=7,
                sticky="nsew")
565         for i in range(self.num_rows):
566             self.main_game_frame.grid_rowconfigure(i, weight=1)
567         for j in range(self.num_columns):
568             self.main_game_frame.grid_columnconfigure(j, weight=1)
569         time.sleep(10)
570         threading.Thread(target=self.add_games,args=(get_games(),)).start()
571 
572     def set_theme(self,theme):
573         global LALK,CREAM,DARG,LARG,GARG,BREW,MENT,TECK,DANG,BORR,THEME;THEME = theme
574         LALK,CREAM,DARG,LARG,GARG,BREW,MENT,TECK,DANG,BORR = COLOUR_PALLETTE[THEME]
575         self.show_menu()
576         self.settings.activate()
```

PROGRAM CODE

```
582     def play_game(self, game_main_file):
583         game_main_file_path = os.path.join(INFOLAB_SERVER_DIRECTORY, game_main_file)
584         game_dir = os.path.dirname(game_main_file_path)
585 
586         self.update_playing_count(game_main_file, 1)
587         os.chdir(game_dir)
588         python_executable = sys.ex
589         try:subprocess.Popen([pyth                    (module) messagebox    ain_file_path])
590         except Exception as error:messagebox.showerror("Error Running Code", f"{error}
            ")
591 
592     def update_playing_count(self, game_name, count_change):
593         client = connect_to_server()
594         request = {"action": "update_playing_count", "game_name": game_name,
            "count_change": count_change}
595         client.send(pickle.dumps(request))
596         client.close()
597 
598     def show_create(self):
599         for widget in self.create_tab.winfo_children():
600             widget.destroy()
601 
602         def select_icon():
603             self.game_icon = filedialog.askopenfilename(title="Select Game Icon",
                filetypes=[("Image Files", "*.png;*.jpg;*.jpeg")])
604             img = Image.open(self.game_icon)
605             img = img.resize((300,300))
606             thumbnail = ImageTk.PhotoImage(img)
607             icon.configure(text='');icon.configure(image=thumbnail)
608         def enable(event=None,num=0):
609             if num==0:gfm.pack(padx=20,pady=10);self.after(0,lambda:enable(num=1))
610             if num==1:ggfm.pack(expand=True,fill=tk.Y,padx=5,pady=5);self.after(0,
                lambda:enable(num=2))
611             if num==2:sub_fm.pack(side="left",expand=True,fill=tk.Y,padx=5,pady=5);
                self.after(400,lambda:enable(num=3))
612             if num==3:main_fm.pack(side="left",expand=True,fill=tk.Y,padx=5,pady=5);
                self.after(400,lambda:enable(num=4))
613             if num==4:select_folder_fm.pack(expand=True,fill=tk.X,);self.after(400,
                lambda:enable(num=5))
614             if num==5:select_code_fm.pack(expand=True,fill=tk.X,)
615         def disable(event):gfm.pack_forget();sub_fm.pack_forget();main_fm.pack_forget
            ();select_folder_fm.pack_forget();select_code_fm.pack_forget()
616         def test():
617             game_dir = os.path.dirname(self.code._text)
618             os.chdir(game_dir)
619             python_executable = sys.executable
620             subprocess.Popen([python_executable, self.code._text])
621         create_fm = tk.Frame(self.create_tab,bg=DARG)
622         fm = tk.Frame(create_fm,bg=GARG,highlightbackground=CREAM,
            highlightcolor=CREAM,highlightthickness=BORR)
623         upload_fm = tk.Frame(fm,bg=GARG,width=400)
624         upload_button=ctk.CTkButton(upload_fm, text="UPLOAD", command=self.
            upload_game_window, font=("AMCAP Eternal", 25), text_color=CREAM,
            fg_color=GARG, bg_color=GARG, corner_radius=10, hover_color=BREW,hover=True,
            border_color=CREAM,border_width=BORR)
625         upload_button.bind('<Enter>',lambda event:enable(event))
626         create_fm.bind('<Enter>',lambda event:disable(event))
```

PROGRAM CODE

```
626         create_fm.bind('<Enter>',lambda event:disable(event))
627         create_label = tk.Label(master=upload_fm, text='Create Your Game', font=
            ("AMCAP Eternal", 25,"bold"), foreground=CREAM, background=GARG)
628
629         create_label.pack(side=tk.LEFT)
630         upload_button.pack(side=tk.RIGHT)
631         tk.Label(master=upload_fm, text='', font=("AMCAP Eternal", 25,"bold"),
            foreground=CREAM, background=GARG,).pack(side=tk.LEFT,padx=150)
632         upload_fm.pack(expand=True,fill=tk.X,padx=20,pady=10)
633         gfm = tk.Frame(fm,bg=DARG,highlightbackground=CREAM,highlightcolor=CREAM,
            highlightthickness=BORR)
634         ggfm = tk.Frame(gfm,bg=DARG)
635         main_fm = tk.Frame(ggfm,bg=DARG)
636         sub_fm = tk.Frame(ggfm,bg=DARG)
637         ss_fm=tk.Frame(main_fm,bg=DARG)
638         self.new_gamename = ctk.CTkEntry(ss_fm,font=("AMCAP Eternal", 25, "bold"),
            text_color=CREAM, fg_color=GARG, bg_color=DARG,corner_radius=10,
            placeholder_text='GAME NAME',placeholder_text_color=CREAM,border_color=CREAM,
            border_width=BORR,width=430,height=30)
639         self.new_gamename.pack(padx=10,pady=10)
640         self.description = ctk.CTkTextbox(ss_fm,font=("Cascadia Code", 16),
            text_color=CREAM, fg_color=GARG, bg_color=DARG,corner_radius=10,
            border_color=CREAM,border_width=BORR,width=430,height=230)
641         self.description.pack(padx=10)
642         self.description.insert('1.0','<description>')
643         temp_fm = ctk.CTkFrame(ss_fm,bg_color=DARG,fg_color=GARG,corner_radius=10,
            border_color=CREAM,border_width=BORR)
644         ctk.CTkLabel(master=temp_fm, text=f"Creator: {load_local_data()['username']}
            ", font=("Cascadia Code", 20,"bold"), text_color=CREAM, fg_color=GARG,
            bg_color=DARG,).pack(side=tk.RIGHT,padx=10,pady=3)
645         temp_fm.pack(expand=True,fill=tk.X,padx=10,pady=10)
646         temp2_fm = ctk.CTkFrame(ss_fm,bg_color=DARG,fg_color=GARG,corner_radius=10,
            border_color=CREAM,border_width=BORR)
647         ctk.CTkLabel(master=temp2_fm, text=f"Permissions:", font=("Cascadia Code",
            16,), text_color=CREAM, fg_color=GARG,bg_color=DARG,).pack(side=tk.LEFT,
            padx=10,pady=3)
648         img = Image.open(r'\\infolab-server\Student
            Share\AayushPaikaray\storage\Icons\download123.png')
649         img = img.resize((30,30))
650         thumbnail1 = ImageTk.PhotoImage(img)
651         img = Image.open(r'\\infolab-server\Student
            Share\AayushPaikaray\storage\Icons\download2.png')
652         img = img.resize((30,30))
653         thumbnail2 = ImageTk.PhotoImage(img)
654         imgs=[thumbnail1,thumbnail2]
655         mode_var = {'downloadable':'Not downloadable','Not
            downloadable':'downloadable'}
656         def change_img():self.downloadable.configure(image=imgs[1],text=mode_var[self.
            downloadable._text]);imgs.reverse()
657         self.downloadable = ctk.CTkButton(temp2_fm,text='downloadable', font=
            ("Cascadia Code", 14),command=change_img,text_color=CREAM, image = thumbnail1,
            fg_color=GARG, bg_color=GARG,hover_color=BREW,border_width=0,width=50)
658         self.downloadable.pack(side=tk.RIGHT,padx=4)
659         img = Image.open(r'\\infolab-server\Student
            Share\AayushPaikaray\storage\Icons\view.png').resize((30,30))
660         thumbnail1 = ImageTk.PhotoImage(img)
```

PROGRAM CODE

```
661         self.view = ctk.CTkButton(temp2_fm,text='view', font=("Cascadia Code", 14),
            text_color=CREAM, image = thumbnail1,fg_color=GARG, bg_color=GARG,
            hover_color=BREW,border_width=0,width=50)
662         self.view.pack(side=tk.RIGHT,padx=3,pady=2)
663         temp2_fm.pack(expand=True,fill=tk.X,padx=10,pady=2)
664         self.game_icon = r'\\infolab-server\Student
            Share\AayushPaikaray\storage\Icons\default.png'
665         img = Image.open(r'\\infolab-server\Student
            Share\AayushPaikaray\storage\Icons\default.png')
666         img = img.resize((300,300))
667         thumbnail = ImageTk.PhotoImage(img)
668         icon = ctk.CTkButton(sub_fm,text='', command=select_icon, font=("AMCAP
            Eternal", 30), text_color=DARG, fg_color=DARG, bg_color=DARG,
            image=thumbnail, hover_color=BREW,border_color=CREAM,border_width=BORR,
            height=320,width=320)
669         icon.pack(pady=0)
670         self.test = ctk.CTkButton(sub_fm,text='TEST', font=("AMCAP Eternal", 20),
            text_color=GARG, fg_color='#779966', bg_color=DARG,hover_color="#99AFAA",
            border_width=0,height=50,width=320,command=lambda:test())
671         self.test.pack(pady=10)
672         def select_folder():
673             self.folder.configure(text=filedialog.askdirectory(title="Select Folder
                containing CODE and ASSETS"))
674         select_folder_fm = tk.Frame(gfm,bg=DARG,)
675         ctk.CTkButton(select_folder_fm, text="select Folder", command=select_folder,
            font=("AMCAP Eternal", 16), text_color=CREAM, fg_color=GARG, bg_color=DARG,
            corner_radius=10,border_color=CREAM,border_width=BORR, hover_color=BREW,
            hover=True,).pack(side=tk.RIGHT,padx=20)
676         self.folder = ctk.CTkLabel(select_folder_fm,font=("Cascadia Code", 16),
            text_color=CREAM, fg_color=DARG, bg_color=DARG,corner_radius=10,text='Select
            Your Game Folder',anchor='w',width=400)
677         self.folder.pack(expand=True,padx=10,pady=10,side=tk.LEFT,fill=tk.X)
678         def select_code():
679             try:self.code.configure(text=filedialog.askopenfilename(title="Select
                Python code file", initialdir=self.folder._text))
680             except:self.code.configure(text="Select Folder first")
681         select_code_fm = tk.Frame(gfm,bg=DARG,)
682         ctk.CTkButton(select_code_fm, text="Select Code", command=select_code, font=
            ("AMCAP Eternal", 16), text_color=CREAM, fg_color=GARG, bg_color=DARG,
            corner_radius=10,border_color=CREAM,border_width=BORR, hover_color=BREW,
            hover=True,).pack(side=tk.RIGHT,padx=20)
683         self.code = ctk.CTkLabel(select_code_fm,font=("Cascadia Code", 16),
            text_color=CREAM, fg_color=DARG, bg_color=DARG,corner_radius=10,text='Select
            Code from the above folder',anchor='w',width=400)
684         self.code.pack(expand=True,padx=10,pady=10,side=tk.LEFT,fill=tk.X)
685         ss_fm.pack()
686         fm.pack(pady=30)
687         create_fm.pack()
688         mygames_label = tk.Label(master=create_fm, text='  My Games', font=("AMCAP
            Eternal", 25,"bold"), foreground=CREAM, background=GARG,anchor='w',
            highlightbackground=CREAM,highlightcolor=CREAM,highlightthickness=BORR)
689         mygames_label.pack(expand=True,fill=tk.X,pady=10)
690
691         games = get_games()
692         self.local_data=load_local_data()
693         for game in games:
694             if game.get("username") == self.local_data['username']:
```

PROGRAM CODE

```
695                 img = Image.open(os.path.join(INFOLAB_SERVER_DIRECTORY,game.get
                    ('icon')))
696                 img = img.resize((50,50))
697                 thumbnail1 = ImageTk.PhotoImage(img)
698                 game_frame = ctk.CTkFrame(create_fm,fg_color=LARG,border_color=CREAM,
                    border_width=BORR)
699                 game_frame.pack(fill="x", pady=3)
700                 game_name = game.get("game_name")
701                 game_label=ctk.CTkLabel(game_frame,font=("AMCAP Eternal", 25),
                    text='',image=thumbnail1, fg_color=LARG,text_color=CREAM,anchor='w')
702                 game_label.pack(side="left",padx=20,pady=5)
703                 game_label=ctk.CTkLabel(game_frame,font=("AMCAP Eternal", 25),
                    text=game_name, fg_color=LARG,text_color=CREAM,anchor='w')
704                 game_label.pack(side="left",padx=20,pady=5)
705                 reupload = ctk.CTkButton(game_frame, text="Reupload", command=lambda
                    g=game_name: self.reupload_game_window(g), font=("AMCAP Eternal",
                    21), text_color=CREAM, fg_color=GARG, bg_color=LARG, corner_radius=10,
                    border_color=CREAM,border_width=BORR, hover_color=BREW,hover=True,)
706                 delete = ctk.CTkButton(game_frame, text="Delete", command=lambda
                    g=game_name: self.delete_game(g),font=("AMCAP Eternal", 21),
                    text_color=DANG, fg_color=GARG, bg_color=LARG, corner_radius=10,
                    border_color=DANG,border_width=2, hover_color=BREW,hover=True,)
707                 rename = ctk.CTkButton(game_frame, text="Rename", command=lambda
                    g=game_name: self.rename_game_window(g), font=("AMCAP Eternal", 21),
                    text_color=CREAM, fg_color=GARG, bg_color=LARG, corner_radius=10,
                    border_color=CREAM,border_width=BORR, hover_color=BREW,hover=True,)
708                 delete.pack(side=tk.RIGHT,padx=3)
709                 reupload.pack(side=tk.RIGHT,padx=3)
710                 rename.pack(side=tk.RIGHT,padx=3)
711
712     def show_settings(self):
713         for widget in self.settings_tab.winfo_children():
714             widget.destroy()
715
716         settings_fm = ctk.CTkFrame(self.settings_tab,fg_color=DARG)
717         settings_fm.pack(expand=False,fill=tk.Y)
718         SET_fm = ctk.CTkFrame(settings_fm,fg_color=GARG,border_color=CREAM,
            border_width=BORR)
719         SET_label = tk.Label(master=SET_fm, text='SETTINGS', font=("AMCAP Eternal",
            30,"bold"), fg=CREAM, bg=GARG,anchor='w',width=25)
720         SET_label.pack(expand=True,fill=tk.X,padx=20,pady=10)
721
722         SET_fm.pack(expand=True,fill=tk.X,pady=20)
723
724         personalize_fm = ctk.CTkFrame(settings_fm,fg_color=GARG,border_color=CREAM,
            border_width=BORR)
725         personalize_label = ctk.CTkLabel(master=personalize_fm, text='PERSONALIZE',
            font=("AMCAP Eternal", 30,"bold"), text_color=CREAM, fg_color=GARG,anchor='w')
726         personalize_label.pack(expand=True,fill=tk.X,padx=20,pady=10)
727
728         theme_fm = ctk.CTkFrame(personalize_fm,fg_color=DARG,border_color=CREAM,
            border_width=BORR)
729         theme_label = ctk.CTkLabel(master=theme_fm, text='THEMES', font=("AMCAP
            Eternal", 25,"bold"), text_color=CREAM, fg_color=DARG)
730         theme_label.pack(expand=True,fill=tk.X,padx=30,pady=10)
731         themes_buttons = []
```

PROGRAM CODE

```
733             for theme in COLOUR_PALLETTE.keys():
734                 button = ctk.CTkButton(theme_fm,text=theme,font=("Cascadia Code", 20,
                    "bold"), text_color=CREAM, fg_color=DARG,border_color=CREAM,
                    border_width=0,hover=True,hover_color=BREW)
735                 if theme == THEME:button.configure(border_width=2)
736                 themes_buttons.append(button)
737                 def update_button_colors(button:ctk.CTkButton):
738                     for button_THEME in themes_buttons:
739                         button_THEME.configure(border_width=0,fg_color=DARG)
740                     button.configure(border_width=2,fg_color=GARG)
741                     self.set_theme(button._text)
742                 button.configure(command=lambda b=button: update_button_colors(b))
743                 button.pack(expand=True, fill=tk.X,pady=5,padx=20)
744 
745             theme_fm.pack(expand=True,fill=tk.X,padx=20,pady=10)
746 
747             personalize_fm.pack(expand=True,fill=tk.X,pady=10)
748 
749 
750             security_fm = ctk.CTkFrame(settings_fm,fg_color=GARG,border_color=CREAM,
                border_width=BORR)
751             security_label = ctk.CTkLabel(master=security_fm, text='SECURITY', font=
                ("AMCAP Eternal", 30,"bold"), text_color=CREAM, fg_color=GARG,anchor='w')
752             security_label.pack(expand=True,fill=tk.X,padx=20,pady=10)
753             security_fm.pack(expand=True,fill=tk.X,pady=10)
754 
755 
756 
757         def reupload_game_window(self, game_name):
758             game_dir = filedialog.askdirectory(title="Select Game Directory")
759             if game_dir:
760                 game_main_file = filedialog.askopenfilename(title="Select Main Game
                    File", initialdir=game_dir)
761                 game_icon = filedialog.askopenfilename(title="Select Game Icon",
                    filetypes=[("Image Files", "*.png;*.jpg;*.jpeg")])
762                 if game_main_file and game_icon:
763                     if reupload_game(self.local_data["username"], game_name, game_dir,
                        game_main_file, game_icon):
764                         messagebox.showinfo("Success", "Game reuploaded successfully")
765                         self.show_create()
766                     else:
767                         messagebox.showerror("Error", "Failed to reupload game")
768 
769         def delete_game(self, game_name):
770             if delete_game(load_local_data()["username"], game_name):
771                 messagebox.showinfo("Success", "Game deleted successfully")
772                 self.show_create()
773             else:
774                 messagebox.showerror("Error", "Failed to delete game")
775 
776         def rename_game_window(self, game_name):
777             new_name = simpledialog.askstring("Rename Game", "Enter new game name:")
778             if new_name:
779                 if rename_game(self.local_data["username"], game_name, new_name):
780                     messagebox.showinfo("Success", "Game renamed successfully")
781                     self.show_create()
```

PROGRAM CODE

```
782                 else:
783                     messagebox.showerror("Error", "Failed to rename game")
784 
785     def upload_game_window(self):
786         game_name = self.new_gamename.get()
787         game_dir = self.folder._text
788         game_main_file = self.code._text
789         game_icon = self.game_icon
790         description = self.description.get('1.0',ctk.END)
791         download = 'downloadable'== self.downloadable._text
792         view = True
793         if game_name and game_main_file and game_icon and game_dir:
794             if upload_game(self.local_data["username"], game_name, game_dir,
                game_main_file, game_icon, description, download, view):
795                 messagebox.showinfo("Success", "Game uploaded successfully")
796                 self.show_create()
797             else:
798                 messagebox.showerror("Error", "Failed to upload game")
799 
800     def show_chat(self):
801         for widget in self.chat_tab.winfo_children():
802             widget.destroy()
803         root = self.chat_tab
804         client = ChatClient(root)
805         root.mainloop()
806 
807 class ChatClient:
808     current = []
809     def __init__(self, root):
810         for i in ChatClient.current:
811             i.lobby.close()
812             for port in i.tabs.keys():i.tabs[port]["socket"].close()
813 
814         self.root = root
815         self.lobby = self.connect_to_lobby()
816 
817         self.main_frame = tk.Frame(root,)
818         self.head_frame=ctk.CTkFrame(master=self.main_frame,bg_color=TECK,
            border_color=CREAM,border_width=BORR, )
819 
820         self.toggle_menu_fm=tk.Frame(root,bg=TECK,highlightbackground=CREAM,
            highlightcolor=CREAM,highlightthickness=BORR)
821 
822         Tabslabel=ctk.CTkLabel(self.toggle_menu_fm,text="GROUPS",text_color=MENT,font=
            ("AMCAP Eternal",19))
823         Tabslabel.pack(side=tk.TOP)
824 
825         self.toggle_menu_fm.pack(side='left',fill=tk.Y, )
826         self.main_frame.pack(side='left',fill=tk.BOTH, expand=True)
827         self.tabs = {}
828         mode_var = {'PUBLIC':'PRIVATE','PRIVATE':'PUBLIC',LARG:CREAM,CREAM:LARG}
829         self.create_group_fm = ctk.CTkFrame(self.main_frame,bg_color=LARG,
            fg_color=LARG)
830         self.create_group_tab = CTabs('',self,self.create_group_fm,
            add_button=False)
831         ChatClient.current.append(self)
```

PROGRAM CODE

```
833         def check_group_create():
834             group_name = self.name_entry.get()
835             admin = load_local_data()['username']
836             group_members = []
837             def player_filter(button):
838                 if button['background']==CREAM:return True
839                 else:return False
840             for user in self.player_buttons.keys():
841                 if player_filter(self.player_buttons[user]):group_members.append(user)
842             mode_fun = lambda:self.mode_button['text']=='PUBLIC'
843             port = create_group(group_name=group_name,group_members=group_members,
                group_mode=mode_fun(),admin=admin)
844             if not port:self.name_entry['text'] = 'GROUP EXIST'
845             else:self.add_chat_tab({'group_name':group_name,
                'group_members':group_members,'group_mode':mode_fun(),'admin':admin,
                'PORT':port})
846
847         self.main_fm = ctk.CTkFrame(self.create_group_fm,bg_color=LARG,fg_color=LARG)
848         self.name_fm = ctk.CTkFrame(self.main_fm,bg_color=LARG,corner_radius=10,
            fg_color=LARG)
849         self.name_label = ctk.CTkLabel(self.name_fm, corner_radius=50,font=
            ("Angrybirds", 27, "bold"), text='Group Name ',bg_color=LARG,fg_color=LARG,
            text_color=CREAM)
850         self.name_entry = ctk.CTkEntry(self.name_fm, corner_radius=50,font=
            ("Angrybirds", 27, "bold"),bg_color=LARG,fg_color=CREAM,text_color=DARG,
            border_color=CREAM,border_width=BORR, )
851         self.conf_label = ctk.CTkLabel(self.main_fm, corner_radius=50,font=
            ("Angrybirds", 30, "bold"), text='Configure',bg_color=LARG,fg_color=LARG,
            text_color=CREAM)
852         self.mode_fm = ctk.CTkFrame(self.main_fm,bg_color=LARG,corner_radius=10,
            fg_color=LARG)
853         self.mode_label = ctk.CTkLabel(self.mode_fm, corner_radius=50,font=
            ("Angrybirds", 24, "bold"), text='Set Mode ',bg_color=LARG,fg_color=LARG,
            text_color=CREAM)
854         self.mode_button = tk.Button(self.mode_fm,font=("Angrybirds", 22, "bold"),
            text='PUBLIC',bg=CREAM,fg=DARG,command=lambda :self.mode_button.configure
            (text=mode_var[self.mode_button['text']]))
855         self.crt_button = tk.Button(self.mode_fm,font=("Angrybirds", 22, "bold"),
            text='CONFIRM CREATE',bg=CREAM,fg=DARG,command=check_group_create)
856
857         self.c_fm = ctk.CTkFrame(self.create_group_fm,bg_color=LARG,corner_radius=10,
            fg_color=LARG,)
858         canvas = tk.Canvas(self.c_fm,bg=DARG,bd=0,borderwidth=0)
859         scrollbar = tk.Scrollbar(self.c_fm, orient="vertical", command=canvas.yview,
            bd=0,borderwidth=0)
860         players = get_players()
861
862         self.main_game_frame = tk.Frame(canvas, bg=DARG)
863         canvas.create_window((0, 0), window=self.main_game_frame, anchor="nw")
864         self.main_game_frame.bind("<Configure>", lambda event, canvas=canvas: canvas.
            configure(scrollregion=canvas.bbox("all")))
865         num_columns = 1 ; num_rows = len(players); self.player_buttons={}
866         for user in players:
867             player_fm = ctk.CTkFrame(self.main_game_frame, bg_color=DARG,
                corner_radius=10, fg_color=LARG, height=25,border_color=CREAM,
                border_width=BORR, )
```

PROGRAM CODE

```
868             player_fm.pack_propagate(False)
869             button = tk.Button(player_fm,text=user,font=("Angrybirds", 20, "bold"),
                fg=CREAM,background=LARG)
870             def update_button_colors(button):
871                 current_bg = button['background']
872                 current_fg = button['fg']
873                 button.configure(
874                     background=mode_var[current_bg],
875                     fg=mode_var[current_fg]
876                 )
877             button.configure(command=lambda b=button: update_button_colors(b))
878 
879             self.player_buttons[user] = button
880 
881             button.pack(expand=True, fill=tk.X)
882             player_fm.pack()
883 
884         for i in range(num_rows):
885             self.main_game_frame.grid_rowconfigure(i, weight=1)
886         for j in range(num_columns):
887             self.main_game_frame.grid_columnconfigure(j, weight=1)
888 
889         self.name_label.pack()
890         self.name_entry.pack(expand=True,fill=tk.X,pady = 40,padx = 40)
891         self.name_fm.pack(expand=True,fill=tk.X)
892         self.conf_label.pack()
893         self.mode_label.pack(side='left',padx = 40)
894         self.mode_button.pack(side='left')
895         self.mode_fm.pack(side=tk.TOP,expand=True,fill=tk.X)
896         scrollbar.pack(side="right", fill="y")
897         canvas.pack(side="left", fill="both", expand=True)
898         canvas.configure(yscrollcommand=scrollbar.set)
899         self.c_fm.pack(side='right',fill=tk.Y,expand=True)
900         self.crt_button.pack()
901         self.main_fm.pack(side="left", fill="both", expand=True)
902 
903 
904 
905         self.create_btn=ctk.CTkButton(self.head_frame, text='create group +',
                bg_color=TECK, fg_color=TECK,text_color=MENT,
906                                       font=("AMCAP Eternal", 20,),command=self.
                                      create_group_tab.activate,hover_color=BREW,
                                      border_width=BORR,border_color=MENT)
907         self.create_btn.pack(side=tk.RIGHT, fill=tk.Y, anchor=tk.W)
908         self.label=tk.Label(master=self.head_frame, text="GROUP CHAT", font=("AMCAP
                Eternal", 20,"bold"), fg=MENT, bg=TECK,highlightbackground=MENT,
                highlightcolor=MENT,highlightthickness=BORR,borderwidth=0)
909         self.label.pack(expand=True, fill="x",side=tk.TOP)
910 
911         self.head_frame.pack(side="top", fill="x")
912 
913         for i in get_groups():self.add_chat_tab(i)
914 
915     def connect_to_lobby(self):
916         lobby = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

PROGRAM CODE

```
917             try:
918                 lobby.connect((HOST, __PORT__))
919             except Exception as e:
920                 messagebox.showerror("Connection Error", f"Failed to connect to the
                    lobby: {e}")
921                 self.root.destroy()
922             return lobby
923
924         def add_chat_tab(self,details):
925             port = details.get('PORT')
926             tab = ttk.Frame(self.main_frame)
927             self.temp = CTabs(details.get('group_name'),self,tab)
928
929             members = 'MEMBERS\n'+'\n'.join(details.get('group_members'))
930             def show_members(event):
931                 member_button['text'] = members
932             def hide_members(event):
933                 member_button['text'] = "MEMBERS"
934                 leave_button.pack_forget()
935                 if is_admin.get():delete_button.pack_forget()
936             def show_settings(event):
937                 leave_button.pack()
938                 if is_admin.get():delete_button.pack()
939
940             def leave():
941                 leave_group(load_local_data()['username'],port)
942                 time.sleep(2)
943                 self.tabs[port]["socket"].close()
944                 input_entry.configure(state=tk.DISABLED)
945
946             def delete_grp():
947                 delete_group(details.get('PORT'))
948                 time.sleep(2)
949                 self.tabs[port]["socket"].close()
950                 input_entry.configure(state=tk.DISABLED)
951
952             settings=tk.Frame(master=tab,background=LARG,highlightbackground=CREAM,
                highlightcolor=CREAM,highlightthickness=BORR,borderwidth=0)
953             setting_frame = tk.Frame(master=settings,background=LARG,
                highlightbackground=CREAM,highlightcolor=CREAM,highlightthickness=BORR)
954             setting_label = tk.Label(master=setting_frame, text='SETTINGS', font=("AMCAP
                Eternal", 14), foreground=CREAM, background=LARG,width=12)
955             setting_label.pack()
956             leave_button = ctk.CTkButton(setting_frame, text="Leave", command=leave, font=
                ("AMCAP Eternal", 20), text_color=CREAM, fg_color=DARG, bg_color=LARG,
                corner_radius=10, border_color=CREAM,border_width=BORR, hover_color=BREW,)
957             is_admin = tk.BooleanVar(value=False)
958             if load_local_data()['username']==details.get('admin') :
959                 is_admin.set(True)
960                 delete_button = ctk.CTkButton(setting_frame, text="DELETE",
                    command=delete_grp, font=("AMCAP Eternal", 20), text_color=DANG,
                    fg_color=DARG, bg_color=LARG, corner_radius=10,border_color=CREAM,
                    border_width=BORR,  hover_color=BREW,)
961             setting_frame.pack(side=tk.RIGHT)
962             member_button = tk.Label(master=settings, text='MEMBERS', font=("AMCAP
                Eternal", 14), foreground=CREAM, background=LARG,highlightbackground=CREAM,
                highlightcolor=CREAM,highlightthickness=BORR,width=12)
```

PROGRAM CODE

```
963             member_button.pack(side=tk.RIGHT)
964
965             online = tk.Label(master=settings, text='', font=("AMCAP Eternal", 14),
                foreground=CREAM, background=LARG)
966             online.pack(side=tk.LEFT)
967             settings.pack(expand=True, fill="x",side=tk.TOP)
968
969             member_button.bind('<Enter>',lambda event:show_members(event))
970             setting_label.bind('<Enter>',lambda event:show_settings(event))
971             chat_display = tk.Text(tab, state=tk.DISABLED, wrap=tk.WORD, font=("AMCAP
                Eternal", 18), fg=TECK, background=MENT,highlightbackground=CREAM,
                highlightcolor=CREAM,highlightthickness=BORR)
972
973             chat_display.bind('<Enter>',lambda event:hide_members(event))
974
975
976             input_frame = ctk.CTkFrame(tab, fg_color=MENT,border_color=CREAM,
                border_width=BORR, )
977             input_frame.pack(fill=tk.X, side=tk.BOTTOM, expand=True)
978
979             input_text = tk.StringVar()
980             input_entry = ctk.CTkEntry(input_frame, textvariable=input_text, font=("AMCAP
                Eternal", 25, "bold"), text_color=MENT, fg_color=TECK, bg_color=MENT,
                corner_radius=10,placeholder_text='chat with your friends',
                placeholder_text_color=MENT,border_color=CREAM,border_width=BORR, )
981
982             input_entry.pack(fill=tk.X, side=tk.LEFT, expand=True,padx=5)
983             chat_display.pack(side=tk.TOP, fill=tk.BOTH, expand=True)
984             input_entry.bind('<Return>', lambda event: self.send_message(port, input_text.
                get()))
985
986             send_button = ctk.CTkButton(input_frame, text="Send", command=lambda: self.
                send_message(port, input_text.get()), font=("AMCAP Eternal", 25),
                text_color=MENT, fg_color=TECK, bg_color=MENT, corner_radius=10,
                border_color=CREAM,border_width=BORR,  hover_color=BREW,)
987             send_button.pack(side=tk.RIGHT,padx=5)
988
989             self.tabs[port] = {"frame": tab, "display": chat_display, "input":
                input_text, "online": online, "socket": None}
990
991             self.connect_to_server(port)
992
993
994     def connect_to_server(self, port):
995
996             self.lobby.send(pickle.dumps(port))
997             client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
998             client.connect((HOST, port))
999             self.tabs[port]["socket"] = client
1000            threading.Thread(target=self.receive_message, args=(port,)).start()
1001
1002    def receive_message(self, port):
1003        client = self.tabs[port]["socket"]
1004        while True:
1005            try:
1006                message = pickle.loads(client.recv(2048))
1007                if message.split(':')[0].lower() == 'online':
```

PROGRAM CODE

```
                        self.tabs[port]["online"]['text']=message
                    else:self.update_chat_display(port, message)
                except:
                    break

    def send_message(self, port, message):
        client = self.tabs[port]["socket"]
        if client:
            try:
                client.sendall(pickle.dumps(message))
                self.tabs[port]["input"].set("")
            except:
                messagebox.showerror("Send Error", f"Failed to send message to port
                {port}")

    def update_chat_display(self, port, message):
        chat_display = self.tabs[port]["display"]
        chat_display.config(state=tk.NORMAL)
        chat_display.insert(tk.END, message + "\n")
        chat_display.config(state=tk.DISABLED)
        chat_display.see(tk.END)


Application().mainloop()
```

PROGRAM CODE

SERVER SIDE CODE

```
import socket
import threading
import pickle
import os
import shutil
import tkinter as tk
from tkinter import ttk, messagebox, simpledialog

HOST = socket.gethostbyname(socket.gethostname())
__PORT__ = 1010

INFOLAB_SERVER_DIRECTORY = r"\\infolab-server\Student Share\AayushPaikaray\storage"

SERVER_ADDR = ("0.0.0.0", 9998)

PC = {}
currently_playing_counts = {}

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(SERVER_ADDR)
server.listen(5)
print("Server started...")

def load_user_data():
    if os.path.exists("users.dat"):
        with open("users.dat", "rb") as f:
            return pickle.load(f)
    return {}

def save_user_data(data):
    with open("users.dat", "wb") as f:
        pickle.dump(data, f)


def load_games_data():
    if os.path.exists("games.dat"):
        with open("games.dat", "rb") as f:
            return pickle.load(f)
    return []

def load_groups_data() -> dict:
    if os.path.exists("groups.dat"):
        with open("groups.dat", "rb") as f:
            data = pickle.load(f)
            if data == None:return {}
            return data
    return {}

def save_groups_data(data):
    with open("groups.dat", "wb") as f:
        pickle.dump(data, f)
```

PROGRAM CODE

```
53  def save_games_data(data):
54      with open("games.dat", "wb") as f:
55          pickle.dump(data, f)
56
57  def handle_client(conn,addr):
58      try:
59          request = pickle.loads(conn.recv(4096))
60          action = request.get("action")
61
62          if action == "login":
63              handle_login(conn, request, addr)
64          elif action == "signup":
65              handle_signup(conn, request)
66          elif action == "upload":
67              handle_upload(conn, request)
68          elif action == "reupload":
69              handle_reupload(conn, request)
70          elif action == "delete":
71              handle_delete(conn, request)
72          elif action == "rename":
73              handle_rename(conn, request)
74          elif action == "get_games":
75              handle_get_games(conn)
76          elif action == "get_groups":
77              handle_get_groups(conn,request)
78          elif action == "get_players":
79              handle_get_players(conn)
80          elif action == "get_playing_count":
81              handle_get_playing_count(conn, request)
82          elif action == "update_playing_count":
83              handle_update_playing_count(conn, request)
84          elif action == 'create_group':
85              handle_create_group(conn,request)
86          elif action == 'leave_group':
87              handle_leave_group(conn,request)
88          elif action == 'delete_group':
89              handle_delete_group(conn,request)
90
91      except Exception as e:
92          print(f"Error handling client: {e}")
93      finally:
94          conn.close()
95
96  def handle_get_players(conn):
97      response = list(load_user_data())
98      conn.send(pickle.dumps(response))
99
100 def handle_create_group(conn,request):
101     response = 1; groups = list(load_groups_data().values())
102     for group in groups:
103         if group.get('group_name').lower() == request.get('group_name').lower():
104             response=0
105     if response:
106         request.pop("action")
107         grps=list(load_groups_data().values())
108         func = lambda grp:grp.get('PORT')
```

PROGRAM CODE

```
            try:request['PORT']=max(map(func,grps))+1
            except:request['PORT'] = 1234
            data = load_groups_data()
            data[request['PORT']] = request
            save_groups_data(data)
            conn.send(pickle.dumps(request.get('PORT')))
        else:conn.send(pickle.dumps(0))

    def handle_login(conn, request, addr):
        username = request.get("username")
        password = request.get("password")
        users = load_user_data()
        if users.get(username) == password:
            response = {"status": "success"}
            PC[addr[0]]=username

        else:
            response = {"status": "failure"}
        conn.send(pickle.dumps(response))

    def handle_signup(conn, request):
        username = request.get("username")
        password = request.get("password")
        users = load_user_data()
        if username in users:
            response = {"status": "failure"}
        else:
            users[username] = password
            save_user_data(users)
            response = {"status": "success"}
        conn.send(pickle.dumps(response))

    def handle_upload(conn, request):
        game_name = request.get("game_name")
        username = request.get("username")
        game_main_file = request.get("game_main_file")
        game_icon = request.get("game_icon")
        description = request.get("description")
        download = request.get('download')
        view = request.get('view')

        games = load_games_data()
        games.append({
            "game_name": game_name,
            "username": username,
            "main_file": game_main_file,
            "icon": game_icon,
            "description":description,
            'download':download,
            'view':view
        })
        save_games_data(games)
        response = {"status": "success"}
        conn.send(pickle.dumps(response))

    def handle_reupload(conn, request):
        game_name = request.get("game_name")
        username = request.get("username")
```

PROGRAM CODE

```
167         game_main_file = request.get("game_main_file")
168         game_icon = request.get("game_icon")
169 
170         games = load_games_data()
171         for game in games:
172             if game["game_name"] == game_name and game["username"] == username:
173                 game["main_file"] = game_main_file
174                 game["icon"] = game_icon
175                 save_games_data(games)
176                 response = {"status": "success"}
177                 conn.send(pickle.dumps(response))
178                 return
179         response = {"status": "failure"}
180         conn.send(pickle.dumps(response))
181 
182     def handle_delete(conn, request):
183         game_name = request.get("game_name")
184         username = request.get("username")
185 
186         games = load_games_data()
187         games = [game for game in games if not (game["game_name"] == game_name and game
        ["username"] == username)]
188         save_games_data(games)
189 
190         game_dir = os.path.join(INFOLAB_SERVER_DIRECTORY, game_name)
191         if os.path.exists(game_dir):
192             shutil.rmtree(game_dir)
193 
194         response = {"status": "success"}
195         conn.send(pickle.dumps(response))
196 
197     def handle_rename(conn, request):
198         old_game_name = request.get("old_game_name")
199         new_game_name = request.get("new_game_name")
200         username = request.get("username")
201 
202         games = load_games_data()
203         for game in games:
204             if game["game_name"] == old_game_name and game["username"] == username:
205                 game["game_name"] = new_game_name
206                 save_games_data(games)
207 
208                 old_game_dir = os.path.join(INFOLAB_SERVER_DIRECTORY, old_game_name)
209                 new_game_dir = os.path.join(INFOLAB_SERVER_DIRECTORY, new_game_name)
210                 if os.path.exists(old_game_dir):
211                     os.rename(old_game_dir, new_game_dir)
212 
213                 response = {"status": "success"}
214                 conn.send(pickle.dumps(response))
215                 return
216         response = {"status": "failure"}
217         conn.send(pickle.dumps(response))
218 
219     def handle_get_games(conn):
220         games = load_games_data()
221         conn.send(pickle.dumps(games))
```

PROGRAM CODE

```
223  def handle_leave_group(conn, request):
224      Room.allrooms[request.get('PORT')].SEND(f"SERVER:{request.get('username')} exited
     the group...")
225      groups = load_groups_data()
226      groups[request.get('PORT')]['group_members'].remove(request.get('username'))
227      save_groups_data(groups)
228
229  def handle_delete_group(conn, request):
230      Room.allrooms[request.get('PORT')].END()
231      groups = load_groups_data()
232      r=groups.pop(request.get('PORT'))
233      save_groups_data(groups)
234      print('GROUP',r)
235
236  def handle_get_groups(conn,request):
237      groups = list(load_groups_data().values()) ; user = request.get('username')
238      def filter_by(group):
239          if user in group.get('group_members'):return True
240          return False
241
242      groups = list(filter(filter_by,groups))
243      print(groups)
244      conn.send(pickle.dumps(groups))
245
246  def handle_get_playing_count(conn, request):
247      game_name = request.get("game_name")
248      count = currently_playing_counts.get(game_name, 0)
249      conn.send(pickle.dumps(count))
250
251  def handle_update_playing_count(conn, request):
252      game_name = request.get("game_name")
253      count_change = request.get("count_change", 0)
254      if game_name in currently_playing_counts:
255          currently_playing_counts[game_name] += count_change
256      else:
257          currently_playing_counts[game_name] = count_change
258      conn.send(pickle.dumps({"status": "success"}))
259
260  def rmain():
261      while True:
262          conn, addr = server.accept()
263          print(f"Connection from {addr}")
264          threading.Thread(target=handle_client, args=(conn,addr)).start()
265
266  if __name__ == "__main__":
267      threading.Thread(target=rmain).start()
268
269  lobby = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
270  random_user = 1
271  try:
272      lobby.bind((HOST, __PORT__))
273      print(f'Created Lobby {HOST} {__PORT__}')
274  except Exception as e:
275      print('Error creating Lobby:', e)
276      exit(1)
277  lobby.listen()
```

PROGRAM CODE

```
279  class Room:
280      codes = []
281      allrooms = {}
282
283      def __init__(self, PORT):
284          global HOST
285          self.PORT = PORT
286          self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
287          try:
288              self.server.bind((HOST, PORT))
289              print(f'Created server {HOST} {PORT}')
290          except Exception as e:
291              print(f'Error Creating Server {HOST} {PORT}:', e)
292          self.server.listen()
293          self.run = True
294          self.allclients = []
295          self.alladdress = {}
296          self.chathistory = load_groups_data()[PORT]['chat_history']
297          Room.allrooms[PORT] = self
298          Room.codes.append(PORT)
299          threading.Thread(target=self.getclient).start()
300
301      def getclient(self):
302          while self.run:
303              try:
304                  client, address = self.server.accept()
305                  self.alladdress[client] = address
306                  client.sendall(pickle.dumps(self.chathistory))
307                  print(f'Connected {PC.get(address[0], address[0])} to {self.PORT}')
308                  self.allclients.append(client)
309                  self.update_online_players()
310                  self.SEND(f'server:{PC.get(address[0], address[0])} has joined the
                     chat',False)
311                  threading.Thread(target=self.RECIEVE, args=(client,)).start()
312              except OSError:
313                  break
314              except Exception as e:
315                  print(f'Error accepting client: {e}')
316
317      def update_online_players(self):
318          onlineplayers = 'online:'
319          for clts in self.allclients:
320              onlineplayers = f'{onlineplayers}{PC.get(self.alladdress[clts][0], self.
                 alladdress[clts][0])},'
321          self.SEND(onlineplayers,False)
322
323      def SEND(self, message:str,b=True):
324          if b:
325              if self.chathistory:self.chathistory = f'{self.chathistory}\n{message}'
326              else:self.chathistory = message
327          groups_dat = load_groups_data();groups_dat[self.PORT]['chat_history'] = self.
             chathistory
328          save_groups_data(groups_dat)
329          for client in self.allclients:
330              try:
331                  client.sendall(pickle.dumps(f'{message}'))
332              except:
333                  self.allclients.remove(client)
```

PROGRAM CODE

```
335     def RECIEVE(self, client):
336         while self.run:
337             try:
338                 message = pickle.loads(client.recv(2048))
339                 if message:
340                     self.SEND(f'{PC.get(self.alladdress[client][0], self.alladdress
                        [client][0])} : {message}')
341             except:
342                 self.allclients.remove(client)
343                 self.SEND(f'server:{PC.get(self.alladdress[client][0], self.alladdress
                    [client][0])} went offline.',False)
344                 self.update_online_players()
345                 break
346 
347     def END(self):
348         self.run = False
349         self.SEND('SERVER: Group deleted by admin...')
350         self.allclients.clear()
351         self.server.close()
352         del self
353 
354     def history(self):
355         return self.chathistory
356 
357 def commands(client, address):
358     global random_user;run=True
359     while run:
360         try:
361             while True:
362                 try:
363                     code = pickle.loads(client.recv(2048))
364                     print(f'Request {PC.get(address[0], address[0])} to PORT {code}')
365                     if code in Room.codes:
366                         print(f'Connecting {PC.get(address[0], address[0])} to {code}')
367                         break
368                     else:Room(code)
369 
370                 except:
371                     print(f'Disconnected {address[0]}')
372                     client.close();run=False;break
373         except:client.close()
374 def RunLobby():
375     while True:
376         try:
377             client, address = lobby.accept()
378             threading.Thread(target=commands, args=(client, address)).start()
379         except Exception as e:
380             print(f'Error in lobby: {e}')
381             break
382 
383 def chat(message:str):
384     for code in Room.codes:
385         Room.allrooms[code].SEND(message)
386 
387 class AdminUI:
388     def __init__(self, root):
389         self.root = root
```

PROGRAM CODE

```
387  class AdminUI:
388      def __init__(self, root):
389          self.root = root
390          self.root.title("Admin Control Panel")
391
392          self.main_frame = ttk.Frame(root)
393          self.main_frame.pack(fill=tk.BOTH, expand=True)
394
395          self.tab_control = ttk.Notebook(self.main_frame)
396          self.tab_control.pack(fill=tk.BOTH, expand=True)
397
398          self.server_tabs = {}
399          self.user_listbox = tk.Listbox(self.main_frame, selectmode=tk.SINGLE)
400
401          self.create_command_frame()
402          self.update_server_list()
403
404          threading.Thread(target=self.update_server_tabs).start()
405
406      def create_command_frame(self):
407          command_frame = ttk.Frame(self.main_frame)
408          command_frame.pack(fill=tk.X, side=tk.BOTTOM)
409
410          kick_button = ttk.Button(command_frame, text="Kick User", command=self.
             kick_user)
411          kick_button.pack(side=tk.LEFT)
412
413          broadcast_button = ttk.Button(command_frame, text="Broadcast", command=self.
             broadcast_message)
414          broadcast_button.pack(side=tk.LEFT)
415
416          refresh_button = ttk.Button(command_frame, text="Refresh Users", command=self.
             refresh_user_list)
417          refresh_button.pack(side=tk.LEFT)
418
419          self.user_listbox.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
420
421      def update_server_list(self):
422          for port in Room.codes:
423              if port not in self.server_tabs:
424                  tab = ttk.Frame(self.tab_control)
425                  self.tab_control.add(tab, text=f"Port {port}")
426                  chat_display = tk.Text(tab, state=tk.DISABLED, wrap=tk.WORD)
427                  chat_display.pack(fill=tk.BOTH, expand=True)
428                  self.server_tabs[port] = chat_display
429
430      def update_server_tabs(self):
431          while True:
432              try:
433                  self.update_server_list()
434                  for port, chat_display in self.server_tabs.items():
435                      chat_display.config(state=tk.NORMAL)
436                      chat_display.delete(1.0, tk.END)
437                      chat_display.insert(tk.END, Room.allrooms[port].history())
438                      chat_display.config(state=tk.DISABLED)
439                  self.root.after(1000, self.update_server_list)
```

PROGRAM CODE

```
430     def update_server_tabs(self):
440             except: pass
441     def kick_user(self):
442         port = self.get_selected_port()
443         if port is None:
444             messagebox.showinfo("Error", "No server selected")
445             return
446 
447         selected_user = self.user_listbox.get(tk.ACTIVE)
448         if not selected_user:
449             messagebox.showinfo("Error", "No user selected")
450             return
451 
452         user_ip = self.get_ip_from_name(selected_user)
453         if user_ip is None:
454             messagebox.showinfo("Error", "User not found")
455             return
456 
457         room = Room.allrooms.get(port)
458         if room:
459             for client, address in room.alladdress.items():
460                 if address[0] == user_ip:
461                     room.SEND(f'{PC[user_ip]} was kicked from the server.')
462                     client.close()
463                     room.allclients.remove(client)
464                     self.refresh_user_list()
465                     break
466 
467     def broadcast_message(self):
468         message = simpledialog.askstring("Input", "Enter message to broadcast:")
469         if message:
470             chat(f'ADMIN: {message}')
471 
472     def refresh_user_list(self):
473         self.user_listbox.delete(0, tk.END)
474         port = self.get_selected_port()
475         if port is not None:
476             room = Room.allrooms.get(port)
477             if room:
478                 for client in room.allclients:
479                     user_name = PC[room.alladdress[client][0]]
480                     self.user_listbox.insert(tk.END, user_name)
481 
482     def get_ip_from_name(self, name):
483         for ip, user_name in PC.items():
484             if user_name == name:
485                 return ip
486         return None
487 
488     def get_selected_port(self):
489         tab_index = self.tab_control.index(self.tab_control.select())
490         tab_text = self.tab_control.tab(tab_index, "text")
491         if tab_text.startswith("Port "):
492             return int(tab_text.split(" ")[1])
493         return None
```

PROGRAM CODE

```
495    def main():
496        threading.Thread(target=RunLobby).start()
497
498        root = tk.Tk()
499        admin_ui = AdminUI(root)
500        root.mainloop()
501
502    if __name__ == "__main__":
503        main()
504
505
```

FUTURE ENHANCEMENTS

Expanding from Local Network to Global Server Integration:

Currently, the application '**Py-verse**', operates on a local network, which limits its accessibility to users within the same network (i.e., a lab environment connected via Ethernet).

Global Server Hosting:

Moving from a local server (**INFOLAB**) to a global cloud-based server (e.g., **AWS**, **Google Cloud**, or **Azure**) will allow users to upload, download, and play games from anywhere. This transition would also facilitate the management of high volumes of users and data, ensuring better scalability.

One-on-One Chat System:

At present, the application offers a **group chat system** where multiple users can engage in real-time discussions within lobbies. A key enhancement would be adding a **1-on-1 chat system**, which will allow direct, private communication between two users.

Notifications:

The system can notify users when they receive a new message. This could involve real-time push notifications and unread message indicators.

File Sharing & Attachments:

The private chat system could also be extended to support file sharing (**images**, **screenshots**, **code snippets**, etc.) within 1-on-1 conversations. Integrating with cloud storage for efficient file handling will be necessary to maintain performance.

Game Discovery and Rating System:

With the expansion of the platform, a game discovery engine and rating system will help users find the **most popular** or **highest-rated games**, improving overall user experience.

BIBLIOGRAPHY

Computer Science with Python :

– Author: Sumita Arora

Pillow (PIL - Python Imaging Library - Image, ImageTk):

- Authors: Alex Clark and Contributors

CustomTkinter :

- Author: Tom Schimansky

Visual Studio Code:

- Author: Microsoft

GeeksforGeeks

- Author: GeeksforGeeksSource:
- URL: https://www.geeksforgeeks.org

Stack Overflow

- Author: Stack Exchange, Inc.
- URL: https://stackoverflow.com

ChatGPT

- Author: OpenAI
- URL: https//openai.com

GitHub

- Author: GitHub, Inc.
- URL: https://github.com

Thank You