Matching Pairs Game (memory game, 짝 맞추기 게임)

1. 구현한 기능

핵심 기능	세부 기능	사용한 함수
로그인 (Login 클래스)	회원가입을 통해 유저 계정을 만들어 DB에 저장	signIn_button
	회원가입시 이미 아이디, 닉네임이 존재하고 있는지 중복 확인	signIn_button
	아이디, 비밀번호를 입력해 로그인	signIn, loginCheck
	로그인 시 아이디가 존재하지 않거나 비밀번호가 틀렸을 경우 경고창	loginCheck
게임 플레이 (Card, Game 클래스)	게임 level 1~3 선택 (레벨마다 카드 이미지가 다름)	level_select
	게임시작 시 처음 3초간 카드 앞면을 전부 보여주는 기능	initial_open
	키보드 w(위),a(왼쪽),s(아래),d(오른쪽), space(카드 선택) 로 게임 플레이	push_A, push_D, push_W, push_S, push_space
	카드 한쌍을 맞췄을 시 점수가 올라가고, 앞면으로 고정	push_space, score_up
	카드 한쌍을 틀렸을 시 점수가 차감되고, 뒷면으로 다시 뒤집기	push_space, score_down, incorrect_action
	콤보 기능 (연속으로 카드쌍을 맞췄을시 추가 점수)	score_up, score_down
	지정된 시간이 다 되면 게임 오버	timer
	점수가 0점 이하가 되면 게임 오버	timer
	점수, 시간이 10 이하가 되면 빨간색으로 표시하여 위험한다는걸 알려주기	timer
	게임이 종료된 후 다시하기, 레벨 재선택, 종료하기 중 고를 수 있음	timer, game_end
게임 기록, 랭킹 보드 (Ranking 클래스)	게임 종료 후 최종점수(점수+남은 시간)을 계산하여 표시	__init__
	전체 유저의 기록과 내 기록을 나누어 랭킹보드 조회	show_rank, page1_pack, page2_pack
	level1, level2, level3 게임 기록을 각각 선택하여 조회 가능	input_data, input_mydata
	최고 기록 달성 여부를 알려주는 기능	new_record

2. 각 기능에 대한 설명 (게임 실행 흐름 순서대로 함수와 함께 설명)

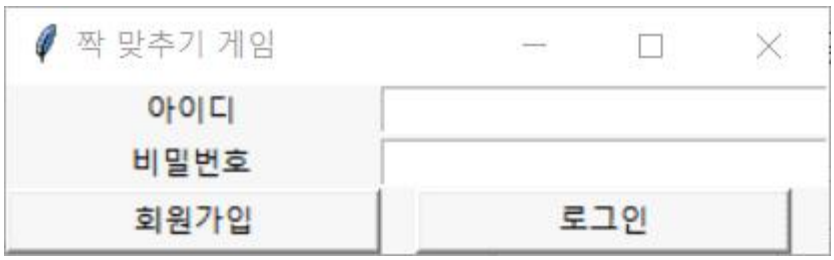
1) 메인함수 실행

```
if __name__ == '__main__':
    root = Tk()
    root.title('짝 맞추기 게임')

    login = Login(root)
    root.mainloop()
```

제일 먼저 메인함수를 실행하면 root창이 생성되며, root창은 앞으로 게임 화면의 제일 최상위에 위치한 부모(parent)창이 될 것이다. root를 생성한 후 Login 클래스의 인스턴스를 생성하여 login 객체를 만든다.

2) Login 클래스의 생성자 실행

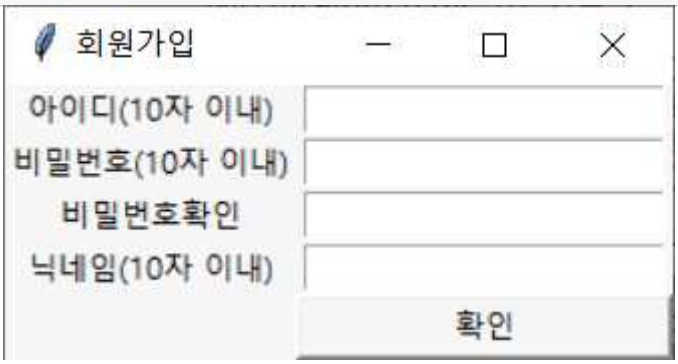
```
class Login:
    def __init__(self, parent):
        self.Parent = parent

        self.login = Frame(parent)
        self.login.pack()

        Label(self.login, text="아이디").grid(row=0, column=0)
        self.user_id = Entry(self.login, width=20)
        Label(self.login, text="비밀번호").grid(row=1, column=0)
        self.user_pw = Entry(self.login, width=20, show="●")
        Button(self.login, text="회원가입", width=20, command=self.signIn).grid(row = 2, column=0)
        Button(self.login, text="로그인", width=20, command=self.loginCheck).grid(row = 2, column=1)

        self.user_id.grid(row=0, column=1)
        self.user_pw.grid(row=1, column=1)
```



로그인 클래스의 생성자가 실행되어 로그인창의 frame가 label, entry, button 등 기본 위젯이 생성되고 배치된다. 새로운 계정을 만들기 위해 회원가입 버튼을 누르면 signIn 함수가 실행된다.

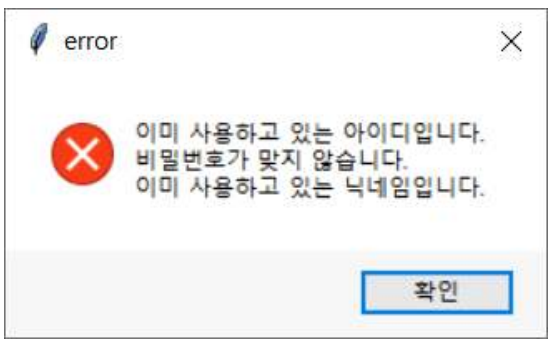
3) 회원가입 실시

```
def signIn(self):
    self.signin = Toplevel(self.Parent)
    self.signin.title("회원가입")

    Label(self.signin, text="아이디(10자 이내)").grid(row= 0, column= 0)
    Label(self.signin, text="비밀번호(10자 이내)").grid(row= 1, column= 0)
    Label(self.signin, text="비밀번호확인").grid(row= 2, column= 0)
    Label(self.signin, text="닉네임(10자 이내)").grid(row= 3, column= 0)

    self.signin_id = Entry(self.signin, width=20)
    self.signin_password = Entry(self.signin, width=20, show="●")
    self.signin_password_check = Entry(self.signin, width=20, show="●")
    self.signin_nickname = Entry(self.signin, width=20)

    self.signin_id.grid(row= 0, column= 1)
    self.signin_password.grid(row= 1, column= 1)
    self.signin_password_check.grid(row= 2, column= 1)
    self.signin_nickname.grid(row= 3, column= 1)

    Button(self.signin, width = 20, text = "확인", command = self.signIn_button).grid(row = 4, column=1)

    self.signin.mainloop()
```

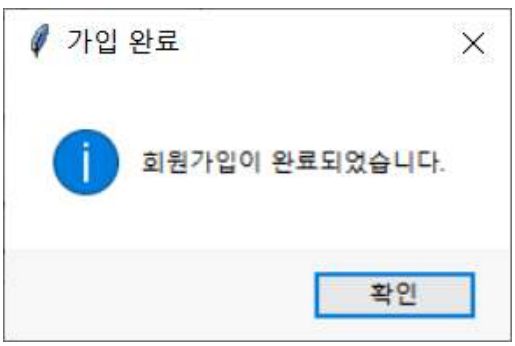


signIn 함수에서 각각 위젯을 생성하고 배치 후, 각 정보를 입력한다.비밀번호는 '●' 기호로 표시되어 비밀이 보장되게 하였다. 확인 버튼을 누르면 signIn_button 함수가 실행된다.

4) 아이디, 닉네임 중복 여부 및 비밀번호 확인 후 회원가입 완료



아이디, 닉네임이 중복되고 비밀번호와 비밀번호확인 값이 다를 경우 오류창



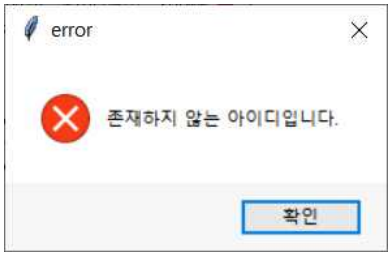
중복X, 비밀번호를 올바르게 입력했을 시 회원가입 완료

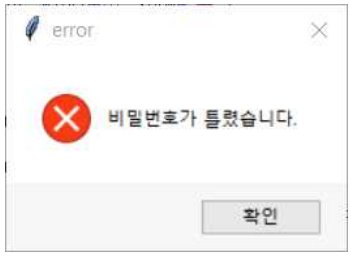
```
def signIn_button(self):

    con, cur = None, None
    con = sqlite3.connect("C:/Users/dudtj/Desktop/짝맞추기 게임/sqlite/gameDB")
    cur = con.cursor()

    cur.execute("SELECT * FROM userTable")

    id_list = []
    nickname_list = []

    while(True):
        row = cur.fetchone()
        if row == None:
            break
        id_list.append(row[0])
        nickname_list.append(row[2])

    message = ""

    ID = self.signin_id.get()
    password = self.signin_password.get()
    nickname = self.signin_nickname.get()

    if ID in id_list:
        message += "이미 사용하고 있는 아이디입니다.₩n"

    if password != self.signin_password_check.get():
        message += "비밀번호가 맞지 않습니다.₩n"

    if nickname in nickname_list:
        message += "이미 사용하고 있는 닉네임입니다.₩n"

    if message == "":
        params = (ID, password, nickname)
        #sql = "INSERT INTO userTable VALUES('"+ID+"','"+password+"',"+nickname+")"
        cur.execute("INSERT INTO userTable VALUES (?, ?, ?)", params)
        #cur.execute(sql)
        con.commit()
        con.close()
        messagebox.showinfo("가입 완료","회원가입이 완료되었습니다.")
        self.signin.destroy()
    else:
        messagebox.showerror("error",message)
```

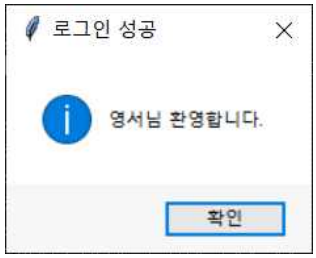
확인 버튼을 누르면 gameDB를 불러와 중복된 아이디와, 닉네임이 있는지 확인하고 각각 해당되는 오류 메시지를 message 변수에 저장한다. 또한 비밀번호 entry와 비밀번호확인 entry에 쓰인 값이 다를 경우 비밀번호가 맞지 않다는 메시지도 message 변수에 추가한다. message 변수를 오류창에 띄워 다시 정보를 입력하도록 하고, 만약 message 변수가 비어있으면 회원가입을 완료하는 창을 띄운 후 유저 정보를 gameDB에 새롭게 집어넣는다.

** userTabel 정보: userTable(id char(10), password char(10), nickname char(10))*

5) 로그인 실시





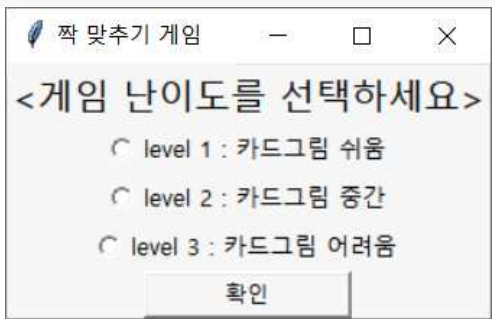



```
def loginCheck(self):
    con, cur = None, None
    con = sqlite3.connect("C:/Users/dudtj/Desktop/짝맞추기 게임/sqlite/gameDB")
    cur = con.cursor()

    cur.execute("SELECT * FROM userTable")

    ID = self.user_id.get()
    password = self.user_pw.get()

    while(True):
        row = cur.fetchone()
        if row == None:
            messagebox.showerror("error","존재하지 않는 아이디입니다.")
            return
        elif row[0] == ID:
            if row[1] == password:
                messagebox.showinfo("로그인 성공",row[2]+"님 환영합니다.")
                self.user = row[2]
                break
            else:
                messagebox.showerror("error","비밀번호가 틀렸습니다.")
                return
        else:
            continue

    self.login.destroy()
    game = Game(self.user,self.Parent)
```

아이디와 비밀번호를 입력한 후 로그인 버튼을 누르면 loginCheck 함수가 실행되고, gameDB를 불러와 아이디 존재 여부와 비밀번호 일치 여부를 조회한다. 아이디가 존재하지 않을 시 에러창이 뜨며, 아이디는 존재하지만 비밀번호가 틀렸을 시 역시 에러창이 뜬다. 아이디와 비밀번호를 모두 잘 입력했으면 닉네임을 부르며 환영한다는 메시지창을 띄우고, login Frame을 destroy하고 Game 클래스의 인스턴스를 생성하여 game 객체를 만든다.

6) Game 클래스의 생성자 실행

```
class Game:
    def __init__(self, user, parent):
        self.Parent = parent
        self.user = user

        self.level_select()

    def level_select(self):
        self.level_frame = Frame(self.Parent)
        self.level_frame.pack()

        Label(self.level_frame,text = "<게임 난이도를 선택하세요>").pack()
        self.var = IntVar()
        level_rb1 = Radiobutton(self.level_frame, text = "level 1 : 카드그림 쉬움", variable = self.var, value = 1)
        level_rb2 = Radiobutton(self.level_frame, text = "level 2 : 카드그림 중간", variable = self.var, value = 2)
        level_rb3 = Radiobutton(self.level_frame, text = "level 3 : 카드그림 어려움", variable = self.var, value = 3)
        level_rb1.pack(); level_rb2.pack(); level_rb3.pack();

        Button(self.level_frame, text = "확인", command = self.place).pack()
```



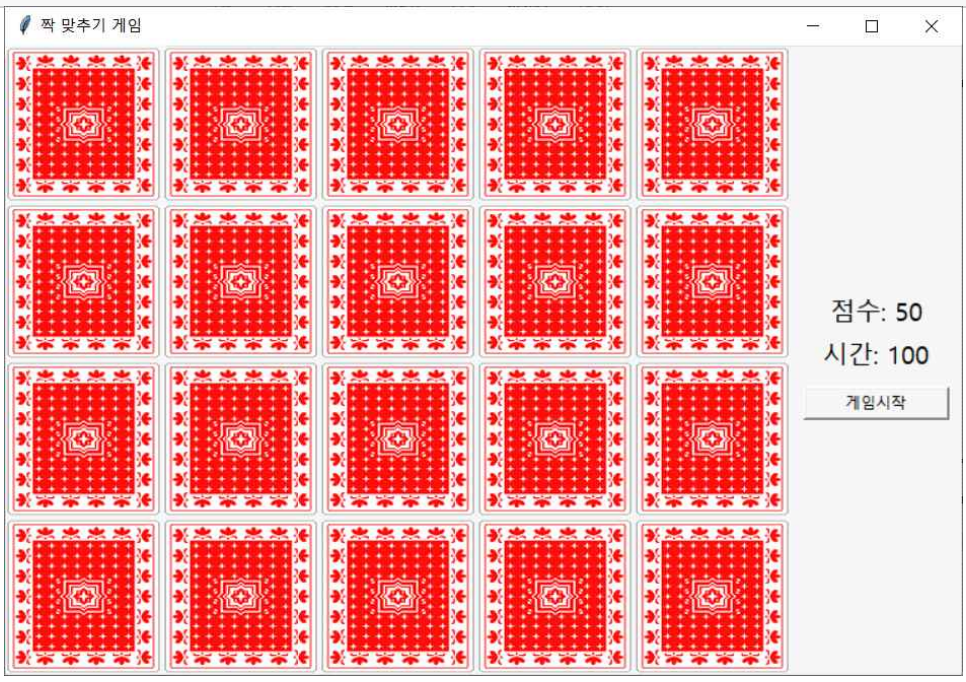
Game 클래스의 생성자가 실행되어 level_select 함수를 자동으로 실행한다. level_select 함수는 게임 레벨을 선택할 수 있도록 라디오버튼을 이용하여 각각 레벨에 해당하는 value 값을 부여했다. 레벨을 선택하고 확인 버튼을 누르면 place 함수가 실행된다.

7) 게임 플레이 창 구현 및 카드 배치

```
def place(self):
    self.level = self.var.get()

    if self.level == 0: #난이도를 선택하지 않았을때 에러창 띄움
        messagebox.showerror("error","난이도를 선택해주세요.")
        return
    elif self.level == 1:
        self.score = 30
        self.time = 60
    elif self.level == 2:
        self.score = 40
        self.time = 80
    else:
        self.score = 50
        self.time = 100

    self.level_frame.destroy()
    self.card_frame = Frame(self.Parent)
    self.card_frame.pack(side=LEFT)

    self.create_card(self.level)

    self.sub_frame = Frame(self.Parent)
    self.sub_frame.pack(side=RIGHT)

    self.label_score = Label(self.sub_frame, text = "점수: " + str(self.score))
    self.label_time = Label(self.sub_frame, text = "시간: " + str(self.time))

    self.label_score.pack(); self.label_time.pack()

    self.start_button = Button(self.sub_frame, text = "게임시작", command = self.game_start)
    self.start_button.pack()
```

```
def create_card(self,level):
    self.card_list = []
    back_name = "back.png"

    for i in range(1,11):
        front_name = "level"+str(level)+" ("+ str(i) + ").png"
        card1 = Card(back_name, front_name, self.card_frame)
        card2 = Card(back_name, front_name, self.card_frame)
        card1.pair = i
        card2.pair = i
        self.card_list.append(card1)
        self.card_list.append(card2)

    random.shuffle(self.card_list)

    for i in range(4):
        for j in range(5):
            self.card_list[i*5 + j].label.grid(row = i, column = j)
```

```
class Card:
    def __init__(self, back, front, parent): #parent는 카드 놓는 frame
            self.Parent = parent

            self.back_image = PhotoImage(file = back)
            self.front_image = PhotoImage(file = front)
            self.label = Label(parent, image = self.back_image)
            self.label.image = self.back_image

            self.pair = ''
            self.front = False
```



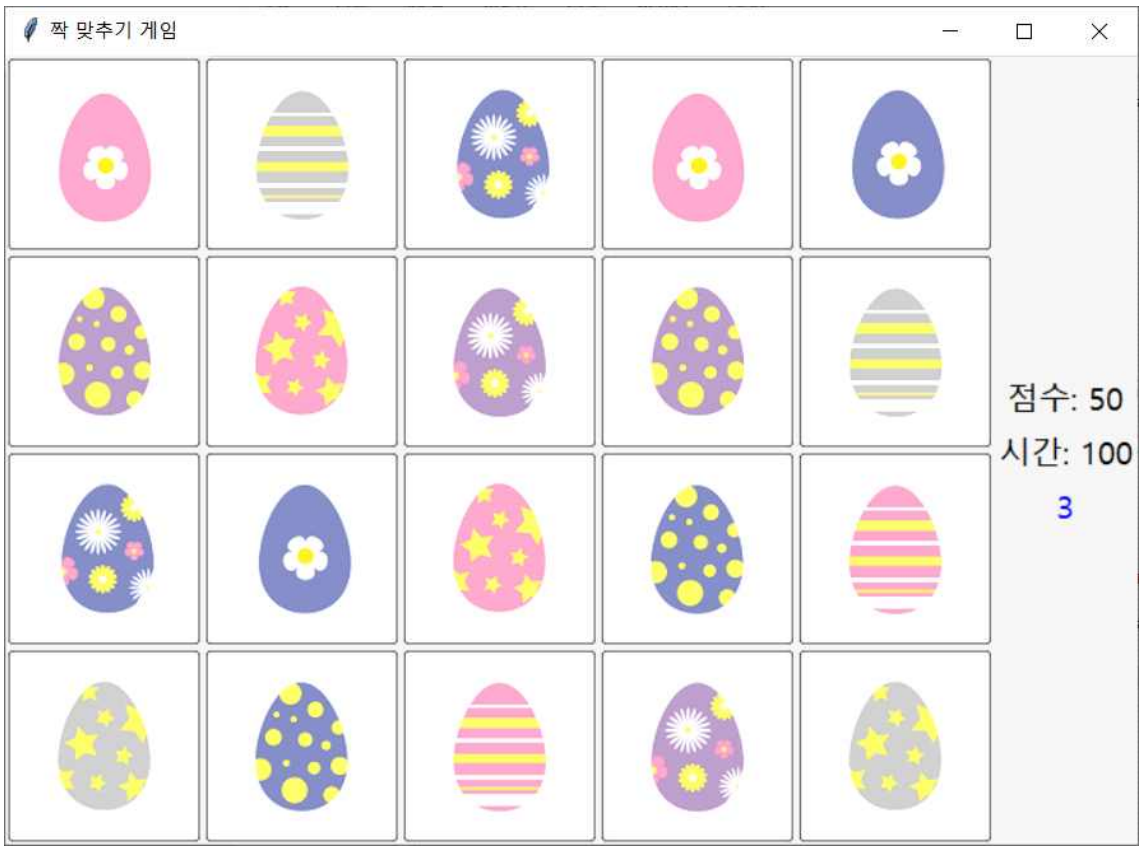
라디오버튼에 부여했던 value 값을 level 변수에 저장하고 level 변수의 값에 따라 초기 점수와 시간을 부여한다. (각 레벨마다 초기 점수와 시간이 다름, 라디오 버튼 중 아무것도 선택하지 않고 확인 버튼을 눌렀을 시 에러창 띄우고 다시 레벨 선택화면으로 돌아감) 그리고 level 선택 frame destroy하고 게임 플레이 창에 필요한 프레임을 생성하고 배치한 뒤 카드를 생성하고 배치하는 create_card 함수를 실행한다. create_card 함수에서는 card_list 라는 카드 리스트 변수를 생성하고 이 리스트 안에 Card 클래스의 객체 20개(10쌍의 카드 객체)를 저장한다. Card 클래스의 생성자는 카드 뒷면 이미지와 앞면 이미지를 불러와 라벨에 부여해 이미지를 저장하고, 현재 카드가 뒷면임을 표시하도록 front 변수를 False로 설정한다. 이때 서로 같은 카드 쌍이면 Card 객체의 pair 변수에 서로 같은 값을 부여한다. 카드 객체를 모두 생성했으면 card_list를 셔플하여 랜덤으로 뒤섞고 4x5 형태로 배치하여 보여지도록 한다. create_card 함수가 종료되면 다시 place 함수로 돌아가 점수와 시간을 표시하는 라벨과 게임시작 버튼을 생성하여 배치하고, 게임시작 버튼을 누르면 game_start 함수가 실행된다.

8) 게임시작 - 카드 전체 앞면을 3초간 보여주기

```
def game_start(self):
    t = threading.Thread(target = self.initial_open) # 카드 처음에 3초 뒤집기
    t.start()
```

```
def initial_open(self):

        self.start_button.destroy()

        for i in range(len(self.card_list)):
            self.card_list[i].turn_front()

        self.text_label = Label(self.sub_frame, text = "3")
        self.text_label.pack()
        time.sleep(1)
        self.text_label.configure(text = "2")
        time.sleep(1)
        self.text_label.configure(text = "1")
        time.sleep(1)
        self.text_label.configure(text = "0")
        time.sleep(0.3)
        self.text_label.configure(text = "Game Start!!")

        for i in range(len(self.card_list)):
            self.card_list[i].turn_back()

        self.t2 = threading.Thread(target = self.timer) # 타이머 함수 실행
        self.t2.start()

        self.game_playing()
```

```
def turn_front(self):
    self.label.configure(image = self.front_image)
    self.front = True

def turn_back(self):
    self.label.configure(image = self.back_image)
    self.front = False
```



game_start 함수가 실행되면 initial_open 함수가 스레드로 실행된다. 스레드로 실행하는 이유는 root 창의 mainloop가 계속 실행되고 있으므로 스레드로 함수를 실행하지 않으면 initial_open 함수의 3초간 앞면을 보여주는 기능이 제대로 보여지지 않는다. initial_open 함수가 실행되면 card_list에 담긴 모든 카드 객체의 turn_front 함수 동작이 실행되어 모든 카드가 앞면으로 뒤집어진다. turn_front 함수와 turn_back 함수는 말 그대로 카드의 이미지를 앞, 뒤 이미지로 바꾸고 front 변수의 True, Flase 여부를 바꾸는 함수이다. 카드가 전부 앞면으로 바뀌었으면 time.sleep을 1초마다 실행되어 3, 2, 1 카운트 하고 0이 되면 카드를 전부 다시 뒷면으로 뒤집는다. 그 후 time 함수가 스레드로 실행되고, game_playing 함수가 실행된다.

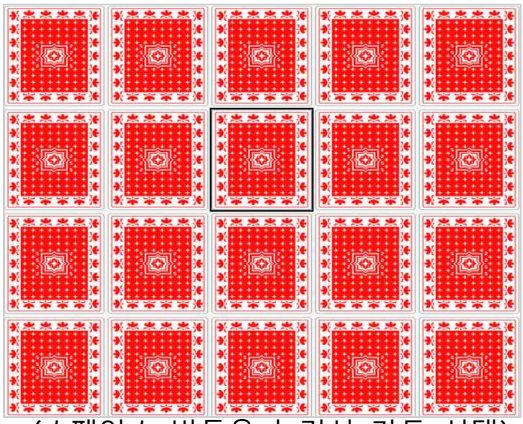
9) 게임시작 - timer 함수의 기능

```
def timer(self):

    self.time_record = 0 # 걸린시간 0초로 초기화

    while(True):
        time.sleep(1)
        self.time -= 1
        self.time_record += 1
        self.label_time.configure(text = "시간: " + str(self.time))

        if self.time == 0 or self.score <= 0: # 시간이 0이 됐거나 점수가 0이하일때 게임 오버
            self.text_label.configure(text = "Game Over!!")
            Button(self.sub_frame, text = "다시하기", command = self.regame).pack(padx = 10, pady = 10)
            Button(self.sub_frame, text = "레벨선택", command = self.reselect).pack(padx = 10, pady = 10)
            Button(self.sub_frame, text = "끝내기", command = self.quit).pack(padx = 10, pady = 10)
            self.Parent.focus_set()
            return
        elif self.correct_pair == 10: # 10개의 모든 카드 쌍을 뒤집으면 시간 멈추기
            return
        elif self.time <= 10:
            self.label_time.configure(fg = 'red')
        elif self.score <= 10:
            self.label_score.configure(fg = 'red')
        elif self.score > 10:
            self.label_score.configure(fg = 'black')
```

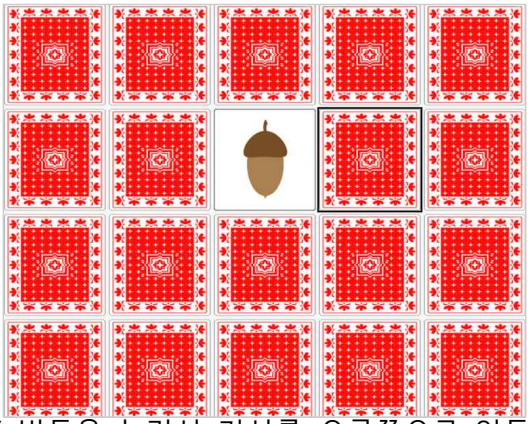
스레드로 timer 함수가 실행되면 1초마다 게임 시간이 감소되도록 무한반복문이 돌아간다. timer 함수는 중요한 기능이 많이 정의되어있는데, 1초가 감소될 때마다 점수 또는 시간이 0 이하이면 게임오버가 되도록, 카드쌍 10개를 모두 맞추면 시간 감소가 종료되도록, 시간 또는 점수가 10초 이하이면 빨간색으로 글씨를 표현하여 위험한 상태라는걸 알려주도록 반복문을 이용하여 구현하였다.

10) a,d,w,s 키보드 버튼을 눌러 카드 커서 이동하기

```
def game_playing(self):
    self.card_index = 0
    self.turn_over = [] # 카드를 뒤집었을때 뒤집은 카드 리스트에 추가 (2개가 채워지면 서로 맞는지 비교)
    self.correct_pair = 0 #지금까지 맞춘 카드 쌍 총 갯수 (10이면 모든 쌍 다 맞추고 게임 종료)
    self.combo = 0

    #self.combo_label = Label(self.sub_frame, text = "Combo " + str(self.combo))
    self.card_list[self.card_index].label.configure(bg = 'black')

    self.card_frame.focus_set() # 카드 프레임으로 포커스 고정
    self.card_frame.bind("<a>", self.push_A)
    self.card_frame.bind("<d>", self.push_D)
    self.card_frame.bind("<w>", self.push_W)
    self.card_frame.bind("<s>", self.push_S)
    self.card_frame.bind("<space>", self.push_space)
```



(스페이스 버튼을 눌러서 카드 선택)



('d' 버튼을 눌러서 커서를 오른쪽으로 이동)

timer 함수가 스레드로 실행되고 game_playing 함수도 그 이후 실행된다고 하였는데, game_playing 함수에서는 게임 동작에 필요한 변수들이 초기화 되고, 키보드로 게임을 플레이할 수 있도록 각 키보드 버튼에 bind 동작을 부여하였다. 맨 첫 번째 카드 라벨에 bg = 'black' 이라는 옵션을 부여하여 검은색 커서가 표시되게 하였는데 'a'를 누르면 왼쪽으로, 'd'를 누르면 오른쪽으로, 'w'를 누르면 위쪽으로, 's'를 누르면 아래쪽으로 검은색 커서가 이동된다.

```
def push_A(self,event):

    if self.card_index in [0, 5, 10, 15]:
        return
    else:
        self.card_list[self.card_index].label.configure(bg = 'SystemButtonFace')
        self.card_index -= 1
        self.card_list[self.card_index].label.configure(bg = 'black')

def push_D(self,event):

    if self.card_index in [4, 9, 14, 19]:
        return
    else:
        self.card_list[self.card_index].label.configure(bg = 'SystemButtonFace')
        self.card_index += 1
        self.card_list[self.card_index].label.configure(bg = 'black')

def push_W(self,event):

    if self.card_index in [0, 1, 2, 3, 4]:
        return
    else:
        self.card_list[self.card_index].label.configure(bg = 'SystemButtonFace')
        self.card_index -= 5
        self.card_list[self.card_index].label.configure(bg = 'black')

def push_S(self,event):

    if self.card_index in [15 ,16, 17, 18, 19]:
        return
    else:
        self.card_list[self.card_index].label.configure(bg = 'SystemButtonFace')
        self.card_index += 5
        self.card_list[self.card_index].label.configure(bg = 'black')
```

각 방향키 버튼에 부여한 함수는 위와 같은데 card_index는 커서가 위치할 카드 리스트의 인덱스 번호를 의미하고 'd'를 누르면 card_index 번호에 +1 을하여 오른쪽으로 커서가 이동하도록, 'a'를 누르면 card_index 번호에 -1을하여 왼쪽으로 커서가 이동하도록, 'w'를 누르면 바로 -5를 하여 바로 윗쪽 행으로 커서가 이동하도록, 's'를 누르면 +5를 하여 바로 아래쪽 행으로 커서가 이동하도록 구현하였다. 이때 중요한 점이 card_frame.focus_set()을 하여 꼭 카드 위젯이 배치되어있는 프레임에 포커스를 고정시켜놔야 bind 동작이 실행된다.

11) 스페이스 버튼을 눌러 커서가 위치한 카드 오픈 -> 짝 일치 여부 검사

```
def push_space(self, event):

    if self.card_list[self.card_index].front: # 이미 선택한 카드를 또 선택했을 때 아무 액션 없이 바로 함수 종료
        return

    self.card_list[self.card_index].turn_front()
    self.turn_over.append(self.card_list[self.card_index])

    if len(self.turn_over) == 2:
        self.a = self.turn_over[0]
        self.b = self.turn_over[1]

        if self.a.pair == self.b.pair: # 짝을 맞췄을때
            self.turn_over = []
            self.correct_pair += 1
            self.score_up()
        else: # 짝을 틀렸을때
            t1 = threading.Thread(target = self.incorrect_action)
            t1.start()
            self.turn_over = []
            self.score_down()
    if self.correct_pair == 10: # 10개의 모든 카드 쌍을 뒤집으면 게임 종료
        self.text_label.configure(text = "Game End!!")
        self.game_end()
```

커서가 위치한 카드 위에서 스페이스 버튼을 누르면 카드가 앞면으로 뒤집어진다. 이 bind 동작은 push_space 함수로 실행되는데, 이 함수가 실행되면 우선 card_list[card_index]에 해당하는 카드 객체의 앞면으로 뒤집는 함수를 실행하여 카드가 앞면으로 뒤집어진다. 앞면으로 뒤집어진 카드는 turn_over 리스트에 저장되고, turn_over 리스트에 두 개의 카드 객체가 저장되었으면 각각 이 두 카드가 같은 쌍인지 확인한다. 확인하는 방법은 간단하다. 각 카드 객체의 pair 변수가 서로 같으면 같은 쌍이라 판단하고 서로 다르면 다른 쌍이라 판단한다. 카드 쌍을 맞추면 correct_pair 변수 값이 +1이 되고 이 변수 값이 10이 되면 총 10쌍의 카드를 모두 맞췄다는 뜻이므로 게임을 종료하는 game_end 함수를 실행한다.

11-1) 카드가 같은 쌍일 때

```
def score_up(self): # 점수를 올리는 함수

    if self.level == 1:
        up_score = 5
    elif self.level == 2:
        up_score = 7
    else:
        up_score = 10

    self.combo += 1
    self.text_label.configure(text = str(self.combo) + " Combo!!")

    self.score += up_score * self.combo
    self.label_score.configure(text = "점수: " + str(self.score))
```

카드가 같은 쌍이면 점수를 올리는 함수인 score_up이 실행된다. 게임 레벨마다 올라가는 점수가 다르며, 연속으로 카드쌍을 맞출때마다 콤보기능이 발동되어 (combo횟수*기본점수) 만큼 점수가 올라간다. 콤보는 카드 쌍을 맞출때마다 +1이 되고 화면에 표시한다.

11-2) 카드가 다른 쌍일 때

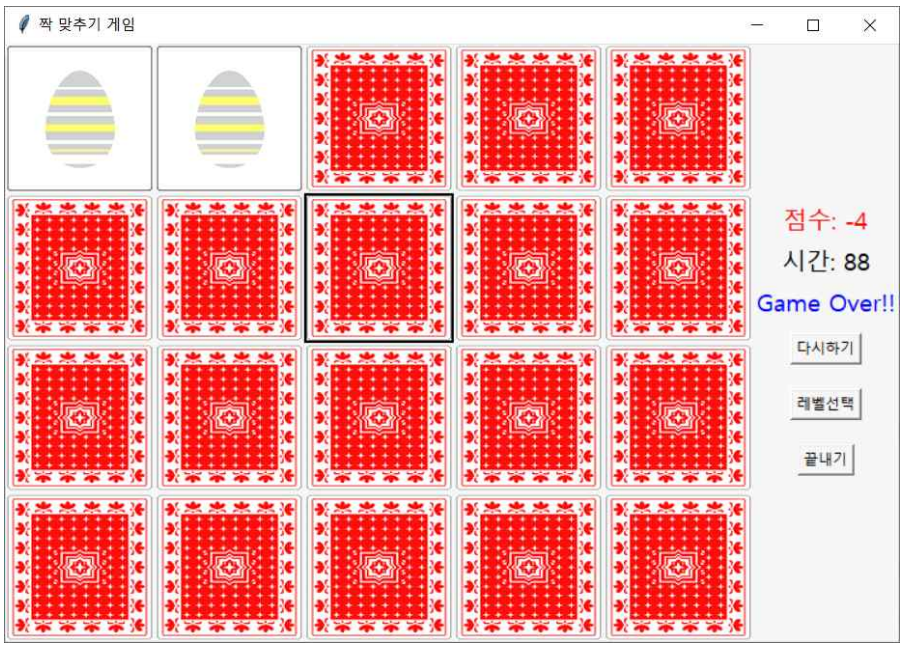
```
def score_down(self):

    if self.level == 1:
        down_score = 3
    elif self.level == 2:
        down_score = 5
    else:
        down_score = 8

    self.combo = 0
    self.text_label.configure(text = str(self.combo) + " Combo!!")

    self.score -= down_score
    self.label_score.configure(text = "점수: " + str(self.score))
```

```
def incorrect_action(self): # 선택한 카드가 서로 다를때 뒤집는 함수 (틀렸을때 액션)
    time.sleep(0.7)
    self.a.turn_back()
    self.b.turn_back()
```

카드가 다른 쌍이면 turn_over 리스트가 비워지고, 점수를 내리는 함수인 score_down이 틀렸을 때 액션을 구현한 incorrect_action 함수가 실행된다. score_down에서는 각 레벨에 따라 정해진만큼 점수를 내리고 combo 변수를 0으로 초기화한다. incorrect_action은 0.7초동안 sleep 했다가 앞면으로 뒤집어진 카드객체의 turn_back 함수를 실행하여 카드를 뒷면으로 뒤집는다.

12) 게임 종료

12-1) 게임 오버 - timer 함수



```
def timer(self):

    self.time_record = 0 # 걸린시간 0초로 초기화

    while(True):
        time.sleep(1)
        self.time -= 1
        self.time_record += 1
        self.label_time.configure(text = "시간: " + str(self.time))

        if self.time == 0 or self.score <= 0: # 시간이 0이 됐거나 점수가 0이하일때 게임 오버
            self.text_label.configure(text = "Game Over!!")
            Button(self.sub_frame, text = "다시하기", command = self.regame).pack(padx = 10, pady = 10)
            Button(self.sub_frame, text = "레벨선택", command = self.reselect).pack(padx = 10, pady = 10)
            Button(self.sub_frame, text = "끝내기", command = self.quit).pack(padx = 10, pady = 10)
            self.Parent.focus_set()
            return
        elif self.correct_pair == 10: # 10개의 모든 카드 쌍을 뒤집으면 시간 멈추기
            return
        elif self.time <= 10:
            self.label_time.configure(fg = 'red')
        elif self.score <= 10:
            self.label_score.configure(fg = 'red')
        elif self.score > 10:
            self.label_score.configure(fg = 'black')
```

앞서 9번에서 설명한 것처럼 timer 함수는 무한반복문이 계속 실행되어 1초가 감소할때마다 게임오버 여부를 계속 체크한다. 시간이 0이 됐거나 점수가 0이하이면 게임오버가 되고, card_frame에 포커스가 맞춰져서 bind 동작이 실행되었는데 이 포커스를 root로 변경하여 게임오버후 더 이상 키보드를 눌러서 게임을 할 수 없도록 만든다. 그리고 다시하기, 레벨선택, 끝내기 버튼을 생성하여 플레이어에게 선택하게 한다.

```
def regame(self):
    self.card_frame.destroy()
    self.sub_frame.destroy()
    self.place()

def reselect(self):
    self.card_frame.destroy()
    self.sub_frame.destroy()
    self.level_select()

def quit(self):
    self.Parent.destroy()
```

다시하기 버튼을 누르면 regame 함수가 실행되며, 게임화면을 구현한 frame을 모두 destroy 시키고 다시 처음부터 화면이 생성되도록 place 함수를 실행시켜 게임을 재시작 하도록 한다. 레벨선택 버튼을 누르면 reselect 함수가 실행되며, 게임화면을 구현한 frame을 모두 destroy 시키고 레벨 선택부터 다시 할 수 있도록 level_select 함수를 실행한다. 끝내기 버튼을 누르면 quit 함수가 실행되며 아예 root 창이 destroy이 되어 모든 창이 꺼진다.

12-2) 게임 클리어 - game_end 함수

```
def game_end(self):
    self.Parent.focus_set()
    self.record = self.score + self.time

    Button(self.sub_frame, text = "다시하기", command = self.regame).pack(padx = 10, pady = 10)
    Button(self.sub_frame, text = "레벨선택", command = self.reselect).pack(padx = 10, pady = 10)
    Button(self.sub_frame, text = "끝내기", command = self.quit).pack(padx = 10, pady = 10)

    rank = Ranking(self.user, self.record, self.level, self.time_record, self.Parent)
```

게임 클리어는 11번에서 설명했던 것처럼 스페이스 버튼을 눌러 카드 쌍을 뒤집었을 때 총 10쌍의 카드 쌍을 모두 뒤집었을 때 game_end 함수가 실행된다. 게임 오버때와 마찬가지로 포커스를 root로 변경하여 더 이상 키보드드 동작이 실행되지 않도록 만들고 최종 점수(점수+남은시간)을 record 변수에 저장하고 랭킹보드를 보여주도록 Ranking 클래스의 객체를 만든다. (다시하기, 레벨선택, 끝내기 버튼의 기능은 게임 오버 때와 같다.)

13) Ranking 클래스의 생성자 실행, 최고기록 달성 여부 검사

```
class Ranking:
        def __init__(self, user, record, level, time, parent):
            self.Parent = parent
            self.user = user

            self.ranking = Toplevel(self.Parent)
            self.result_frame = Frame(self.ranking)
            self.result_frame.pack(side = TOP)
            self.rankboard = Frame(self.ranking);
            self.rankboard.pack(side = BOTTOM)

            font=tkinter.font.Font(family="맑은 고딕", size=15)

            self.result_label = Label(self.result_frame, text = "최종 점수₩n(점수 + 남은시간)₩n" + str(record), font = font)
            self.result_label.pack()
            self.new_record(user, record, level)

            self.insertData(user, record, level, time)
            self.loadData()
            self.show_rank(self.date)
```

```
def new_record(self, user, record, level):
    self.loadData()
    text = ""

    record1 = True
    record2 = True
    record3 = True

    for row in self.data:
        if row[0] == user:
            if row[2] > record:
                record1 = False
                break

    for row in self.data:
        if row[1] == "level" + str(level):
            if row[2] > record:
                record2 = False
                break

    for row in self.data:
        if row[2] > record:
            record3 = False
            break

    if record1:
        text += "내 기록에서 최고점수를 갱신했습니다!!₩n"
    if record2:
        text += "level" + str(level) + " 에서 최고점수를 달성했습니다!!₩n"
    if record3:
        text += "전체에서 최고점수를 달성했습니다!!₩n"

    font=tkinter.font.Font(family="맑은 고딕", size=10)

    self.record_label = Label(self.result_frame, text = text, font = font, fg = 'blue')
    self.record_label.pack()
```

Toplevel로 새로운 창을 만들어 이 창에 최종점수를 표현하는 라벨을 생성한다. 최종점수는 이전 game_end 함수에서 record 값을 인자로 넘겨받아 record 변수를 text로 보여준다. 그리고 게임 최종점수가 최고기록을 달성했는지의 여부를 검사하는 new_record 함수를 실행한다. 최고기록 여부로는 3가지 종류가 있는데 1. 내 기록(같은 유저) 중 최고점수를 갱신했는지, 2. 전체 사용자 중 같은 level에서 최고점수를 달성했는지 3. 전체 사용자, 전체 레벨에서 최고점수를 달성했는지 여부 등의 종류가 있다. 각각 이 종류에 해당하는지를 loadData 함수를 실행시켜 gameDB에서 게임기록 데이터를 불러와 모두 조회하여 종류에 맞는 최고기록 달성 여부를 text에 저장하여 보여준다. 이 과정 중에서 실행되는 loadData 함수에 대해 설명하겠다.

14) 게임 기록 DB에 저장하고 불러오기 - insertData 함수, loadData 함수

```
def insertData(self, user, record, level, time):
    con, cur = None, None

    con = sqlite3.connect("C:/Users/dudtj/Desktop/짝맞추기 게임/sqlite/gameDB")
    cur = con.cursor()

    now = datetime.now()
    self.date = str(now.year) + '/' + str(now.month) + '/' + str(now.day) + ' ' + str(now.hour) + ':' + str(now.minute)
    time = str(time) + '초'
    level = 'level'+ str(level)

    params = (user, level, record, time, self.date)
    cur.execute("INSERT INTO gameRecord VALUES (?, ?, ?, ?, ?)", params)
    con.commit()
    con.close()

def loadData(self):
    con, cur = None, None
    con = sqlite3.connect("C:/Users/dudtj/Desktop/짝맞추기 게임/sqlite/gameDB")
    cur = con.cursor()
    cur.execute("SELECT * FROM gameRecord")

    self.data = []

    while(True):
        row = cur.fetchone()
        if row == None:
            break
        self.data.append([row[0],row[1],row[2],row[3],row[4]])

    con.close()
```

loadData 함수는 gameDB를 불러와 게임기록 정보가 저장된 gameRecord 테이블을 조회하여 data 라는 리스트에 모든 행을 전부 저장한다. 게임 기록을 불러왔으면 마찬가지로 저장도 할 수 있어야 하는데 insertData 함수는 플레이어의 게임 기록을 gameDB의 gameRecord 함수에 저장한다. 저장하는 항목으로는 플레이어 닉네임, 걸린 시간, 게임 레벨, 날짜와 시간, 최종 점수가 있다.

**gameRecord 테이블의 정보: gameRecord(user char(10), level char(6), score int, time char(4), date char(20))*

15) 전체 유저와 내 기록으로 페이지를 나누어 랭킹보드 구현

```
def show_rank(self, date):
    notebook=tkinter.ttk.Notebook(self.rankboard)
    notebook.pack()

    self.page1 = Frame(self.rankboard)
    notebook.add(self.page1, text = '전체')
    self.page2 = Frame(self.rankboard)
    notebook.add(self.page2, text = '내 기록')

    self.view_frame1 = Frame(self.page1)
    self.view_frame2 = Frame(self.page2)
    self.view_frame1.pack()
    self.view_frame2.pack()

    self.button_frame1 = Frame(self.page1)
    self.button_frame2 = Frame(self.page2)
    self.button_frame1.pack()
    self.button_frame2.pack()

    self.page1_pack()
    self.page2_pack()
```

```
def page1_pack(self):
    self.treeview1=tkinter.ttk.Treeview(self.view_frame1, columns=["user", "level", 'score', 'time', 'date'], displaycolumns = ["us
    self.treeview1.pack(side = LEFT)

    scrollbar1=tkinter.ttk.Scrollbar(self.view_frame1, orient="vertical",command=self.treeview1.yview)
    scrollbar1.pack(side=RIGHT, fill= 'y')
    self.treeview1.configure(yscrollcommand=scrollbar1.set)

    self.var1 = IntVar()
    self.var2 = IntVar()
    self.var3 = IntVar()

    ckb1 = Checkbutton(self.button_frame1, text = 'level1', variable = self.var1, command = self.input_data)
    ckb2 = Checkbutton(self.button_frame1, text = 'level2', variable = self.var2, command = self.input_data)
    ckb3 = Checkbutton(self.button_frame1, text = 'level3', variable = self.var3, command = self.input_data)

    ckb1.select()
    ckb2.select()
    ckb3.select()

    ckb1.grid(row = 0, column = 0)
    ckb2.grid(row = 0, column = 1)
    ckb3.grid(row = 0, column = 2)


    #columns=["닉네임", "난이도", '점수', '걸린시간', '날짜']
    self.treeview1.column("#0", width=80, anchor="center")
    self.treeview1.column("#1", width=80, anchor="center")
    self.treeview1.column("#2", width=80, anchor="center")
    self.treeview1.column("#3", width=80, anchor="center")
    self.treeview1.column("#4", width=80, anchor="center")
    self.treeview1.column("#5", width=120, anchor="center")
    self.treeview1.heading("#0", text="순위")
    self.treeview1.heading("#1", text="닉네임")
    self.treeview1.heading("#2", text="난이도")
    self.treeview1.heading("#3", text="점수")
    self.treeview1.heading("#4", text="소요시간")
    self.treeview1.heading("#5", text="날짜")

    self.data.sort(key = itemgetter(2), reverse = True)

    self.input_data()
```

insertData 함수, loadData 함수를 각각 실행하여 gameDB를 갱신하고 다시 로드하여 data 리스트에 게임 랭킹 기록을 집어넣었으면 show_rank 함수가 실행된다. 이 함수에서는 랭킹 보드가 구현되도록 각종 프레임과 위젯이 정의된다. 이때 전체기록과 내 기록을 나누어 열람할 수 있도록 notebook을 사용해 '전체', '내 기록'으로 페이지를 나누었다. 그리고 각 노트북 항목에 트리뷰를 생성하고 데이터를 집어넣을 수 있도록 (전체 -> page1_pack 함수, 내 기록 -> page2_pack) 를 실행한다. page1_pack과 page2_pack 함수의 코드와 기능은 거의 똑같으므로 전체기록을 보여주는 page1_pack 함수를 대표로 설명하겠다. page1_pack 함수에서는 트리뷰에 각각 열 정보를 부여하고, 스크롤바를 삽입하고, 원하는 level의 기록만을 열람할 수 있도록 체크버튼을 삽입한다. 그리고 랭킹 데이터가 담긴 data 리스트를 최종 점수가 높은 순서로 내림차순하여 정렬 한 후 트리뷰에 이 data 리스트의 정보를 삽입하는 input_data 함수를 실행한다.(page2_pack 함수는 input_mydata 함수 실행)

16) 원하는 게임레벨만 선택하여 랭킹 기록 조회하기

순위	닉네임	난이도	점수	소요시간	날짜
1위(new)	영서	level3	678	22초	2019/12/3 23:16
2위	실험용2	level3	313	25초	2019/11/30 19:36
3위	실험용2	level3	304	26초	2019/11/30 19:13
4위	실험용2	level3	257	25초	2019/11/30 19:45
5위	실험용2	level2	250	16초	2019/11/30 19:49
6위	실험용2	level2	249	19초	2019/11/30 23:38
7위	영서	level3	245	25초	2019/12/1 15:41
8위	실험용2	level2	239	15초	2019/11/30 21:7
9위	실험용1	level2	227	22초	2019/11/30 17:47
10위	실험용1	level2	220	24초	2019/11/30 22:57

```
def input_data(self):
    self.treeview1.delete(*(self.treeview1).get_children())
    check_list = []

    if self.var1.get():
        check_list.append("level1")
    if self.var2.get():
        check_list.append("level2")
    if self.var3.get():
        check_list.append("level3")

    tmp = []

    for row in self.data:
        if row[1] in check_list:
            tmp.append(row)

    for i in range(len(tmp)):
        if tmp[i][-1] == self.date:
            self.treeview1.insert('', 'end', text=str(i+1) + '위'+'(new)', values=tmp[i], iid=str(i)+"번")
        else:
            self.treeview1.insert('', 'end', text=str(i+1) + '위', values=tmp[i], iid=str(i)+"번")
```

input_data 함수가 실행되면 우선 체크 리스트에 체크된 level을 check_list 리스트에 저장하고 이 체크한 level에 해당하는 랭킹기록만을 data 리스트에서 tmp 리스트로 옮긴다. 그리고 트리뷰에 tmp 리스트에 있는 랭킹 기록을 집어넣어 체크한 레벨에 해당하는 랭킹기록을 조회할 수 있도록 한다. 이 input_data 함수는 체크리스트가 체크, 체크해제 될 때마다 실행되어 데이터를 넣는다. 이 때 내 최종기록에 해당하는 랭킹 데이터는 새롭게 랭킹에 들어왔다는 의미에서 순위 옆에 (new)가 붙는데, 이 구분은 게임을 마친 시각(date 변수)와 랭킹 기록의 날짜 열과 비교하여 같으면 방금 플레이한 게임의 기록이 되는 것이다.