컴퓨터과학과 코딩 게임 만들기

컴퓨터 인공지능 학부 202312789 이채원
컴퓨터 인공지능 학부 202312807 정사임
컴퓨터 인공지능 학부 202312821 조서린

게임을 위해 필요한 사진

player: 

veges:   

noodle:   

skewer:    

trash:   

(목차)

서론

1-1. 아이디어 논의 과정

과제를 시작하기 앞서 브레인스토밍 과정을 거쳐 의논을 하였다. 아이디어 도출을 위해 팀원들과 정해진 시간 내에 최대한 많은 아이디어를 작성한 다음 아이디어를 논의하고 평가하는 과정을 통해 주제를 정하였다.

여러 아이디어 중 하나는 컴퓨터 과학과 코딩 수업시간에 배웠던 내용 중 for문을 이용하여 샌드위치 만들기 과제를 떠올리며 샌드위치 대신 팀원들이 모두 좋아하는 마라탕을 적용하여 게임의 틀을 구성하자는 아이디어였다.

처음에는 마라탕 식당처럼 요리사가 손님에게 주문을 받은 후 마라탕을 만들어 결과물을 보여주는 것까지를 게임의 틀로 잡았다. 게임에 재미요소를 추가하고자 랜덤 모듈을 사용하여 사장님이 마라탕 재료를 주문받고, 기존에 있던 레이싱게임의 변형을 통해 정해진 시간 내에 마라탕 재료를 획득하되, 레이싱게임 중 폭탄을 내려오게 만들어 폭탄에 맞으면 초기화되는 방식으로 게임을 구성하려고 하였다.

그러나 재료를 획득하여 요리를 하고, 결과물을 보여주는 것까지의 과정을 코드로 구현하는 것에 한계를 느꼈다. 결과적으로 랜덤으로 획득해야 하는 재료들의 목표개수를 정해두고 제한 시간 내에 목표에 도달하면 '성공', 도달하지 못하면 '실패'라는 문구를 띄우는 것까지를 코드로 만들었다.

1-2 팀원과의 협력(어떻게 협동/분업하였는가?)

각자의 개인 시간을 너무 뺏기 보다는 컴퓨터 과학과 코딩 수업, 그리고 다른 과목 시간의 휴강으로 인해 생긴 공강 시간을 최대한 활용하여 시간적인 효율을 높이려고 했으며, 시간이 맞지 않는 경우에는 화상 회의를 통해 꾸준히 의견을 공유했다.

회의 시간에 특정 사람만 대화를 독점하거나, 준비가 되지 않아 침묵이 흐르지 않도록 모두가 각자 맡은 부분에 대해 사전에 충분한 정보 탐색 및 숙지를 했다. 브레인스토밍을 하며 아이디어가 신속하게 산출되었고 이 과정에서 모든 사람이 아이디어를 낼 수 있었다.

코드에 이미지를 추가하여 랜덤으로 재료가 내려오게 하는 파트, 일정 점수에 도달하면 레벨을 올리는(내려오는 재료의 속도를 높이는) 파트, 제한 시간 내에 목표 점수에 도달하면 성공 문구를, 도달하지 못하면 실패 문구를 띄우는 파트 총 세가지로 나누어 팀원들이 각자 코드를 짠 후 공유하여 서로 피드백을 해주는 방식으로 조별 과제를 진행하였다. 각자의 코드를 공유한 후 디버깅 했을 때 발생하는 오류를 해결하기 위해 해당 오류에 대해 다른 방향으로 코드를 하나씩 짜서 비교하는 방법으로 오류를 해결했다.

레포트를 작성하는 과정에서는 게임을 만들면서 의논했던 부분과 아이디어 도출 과정 등을 떠올리며 서론을 작성했고, 게임 만들기를 하면서 각자 느꼈던 부분을 공유하여 결론을 도출해냈다.

본론

2-1. 코드 설명

1. 게임을 실행하면 화면에 마라탕 재료 종류(veges, noodle, skewer)를 각각 몇 개 얻어야하는지 랜덤으로 나온다.
2. 제한시간 20초안에 방향키(상하좌우)로 플레이어를 움직이며 얻어야하는 개수 이상의 마라탕 재료를 얻는다.
3. 내려오는 사진과 플레이어가 충돌하면 사진에 해당하는 종류가 +1된다.
4. 제한시간이 끝나기 전에 음식물 쓰레기에 충돌하면 게임이 종료된다.
5. 제한시간동안 음식물 쓰레기에 충돌하지 않아도 목표 이상의 재료를 얻지 못하면 게임은 실패한다.
6. 제한시간동안 음식물 쓰레기에 충돌하지 않고 목표 이상의 재료를 얻으면 게임은 성공한다.
7. 모든 재료를 획득할 때마다 점수가 10점씩 증가하며 50점, 100점, 150점에 도달할 때마다 레벨이 1씩 증가하여 내려오는 재료의 속도가 빨라진다.

코드 설명

```
import pygame
import random
import time
import sys

# 게임 초기화
pygame.init()

# 게임 화면 설정
screen_width = 800
screen_height = 600
screen = pygame.display.set_mode((screen_width, screen_height))
pygame.display.set_caption("마라탕 게임")

#폰트 설정
small_font = pygame.font.SysFont('Malgun Gothic', 20)
font = pygame.font.SysFont('Tahoma', 60)
game_over = font.render("you fail!",True,"black")
game_clear = font.render("clear!",True,"black")

# 획득 목표 랜덤 선택
a=random.randint(1,5)
b=random.randint(1,5)
c=random.randint(1,5)

#목표 개수를 화면에 나타내기
aima = small_font.render("veges:"+str(a), True,"black")
aimb = small_font.render("noodle:"+str(b), True,"black")
aimc = small_font.render("skewer:"+str(c), True,"black")

# 플레이어 설정
player_width = 50
player_height = 100
playerx = screen_width // 2 - player_width // 2
playery = screen_height - player_height - 20
player_speed = 13
player_image                                                    =
```

```
pygame.image.load('C:/Users/user/Downloads/player.png').convert_alpha()



# 재료 설정(채소)
vege_width = 50
vege_height = 100
vege_speed = random.randint(3, 5)
```
(랜덤 모듈을 활용해서 채소 종류 사진이 내려오는 속도를 랜덤으로 설정한다.)
```
vege_images =
[
    pygame.image.load('C:/Users/user/Downloads/cabbage.png').convert_alpha(),
    pygame.image.load('C:/Users/user/Downloads/mushroom.png').convert_alpha(),
    pygame.image.load('C:/Users/user/Downloads/sukju.png').convert_alpha()
]
```
리스트에 채소 종류에 해당하는 사진 3개를 추가한다.

```
vege_image = random.choice(vege_images)
```
랜덤 모듈로 채소 리스트에 추가된 사진이 랜덤으로 내려온다.

```
vege_x = random.randint(0, screen_width - vege_width)
```
랜덤 모듈을 활용해 사진의 x 좌표를 설정한다.
```
vege_y = -vege_height

# 재료 설정(면)
noo_width = 50
noo_height = 100
noo_speed = random.randint(3, 5)
noo_images = [
    pygame.image.load('C:/Users/user/Downloads/oksusu.png').convert_alpha(),
    pygame.image.load('C:/Users/user/Downloads/dang.png').convert_alpha(),
    pygame.image.load('C:/Users/user/Downloads/bunmoja.png').convert_alpha()
]

noo_image = random.choice(noo_images)
noo_x = random.randint(0, screen_width - noo_width)
noo_y = -noo_height

# 재료 설정(꼬치류)
ske_width = 50
```

```
ske_height = 100
ske_speed = random.randint(3,5)
ske_images = [
    pygame.image.load('C:/Users/user/Downloads/haem.png').convert_alpha(),
    pygame.image.load('C:/Users/user/Downloads/saewu.png').convert_alpha(),
    pygame.image.load('C:/Users/user/Downloads/ubu.png').convert_alpha()
]

ske_image = random.choice(ske_images)
ske_x = random.randint(0, screen_width - ske_width)
ske_y = -ske_height

# 방해물 설정(음식물 쓰레기)
trash_width = 50
trash_height = 100
trash_speed = random.randint(6, 10)
trash_images = [
    pygame.image.load('C:/Users/user/Downloads/watermelon.png').convert_alpha(),
    pygame.image.load('C:/Users/user/Downloads/apple.png').convert_alpha(),
    pygame.image.load('C:/Users/user/Downloads/pizza.png').convert_alpha()
]

trash_image = random.choice(trash_images)
trash_x = random.randint(0, screen_width - trash_width)
trash_y = -trash_height

# 제한 시간
time_limit = 15
start_time = time.time()

# 레벨, 점수
level = 1
score = 0

#획득 재료 개수
noodle = 0
skewer = 0
veges = 0

# 점수 50씩 도달시 레벨 상승
```

```
def increase_level(score):
    global level
    if score >= 50:
        level += 1
    if score >= 100:
        level += 1
    if score >= 150:
        level += 1
```

score라는 매개변수와 level이라는 전역변수를 사용하는 increase_level라는 함수를 정의한다. 이 함수는 score 값(50,100,150)에 따라 level의 값을 증가시킨다.

```
# 레벨 상승에 따라 난이도 증가
def change_difficulty(level):
    global vege_speed, noo_speed, ske_speed
    if level == 1:
        # Easy mode
        vege_speed = random.randint(7,9)
        noo_speed = random.randint(7,9)
        ske_speed = random.randint(7,9)
    elif level == 2:
        # Medium mode
        vege_speed = random.randint(10,12)
        noo_speed = random.randint(10,12)
        ske_speed = random.randint(10,12)
    elif level == 3:
        # Hard mode
        vege_speed = random.randint(13,15)
        noo_speed = random.randint(13,15)
        ske_speed = random.randint(13,15)
```

level이라는 매개변수와 vege_speed, noo_speed, ske_speed라는
전역변수들을 참조하는 change_difficulty라는 함수를 정의한다. 이 함수는 level 값에 따라 게임의 난이도를 변경한다. 함수의 level이 1이면 7에서 9사이의 숫자로 재료의 속도를 무작위로 설정한다.(나머지 레벨도 동일한 구조)

```
# 카운트다운 함수
def countdown():
    for i in range(3, 0, -1):
        screen.fill((255, 255, 255))
        countdown_text = font.render(str(i), True, "black")
        screen.blit(countdown_text, (screen_width // 2 - countdown_text.get_width() //
2, screen_height // 2 - countdown_text.get_height() // 2))
```

```
        screen.blit(countdown_text, (screen_width // 2 - countdown_text.get_width() //
2, screen_height // 2 - countdown_text.get_height() // 2))
        screen.blit(aima, (10, 10))
        screen.blit(aimb, (200, 10))
        screen.blit(aimc, (390, 10))
        pygame.display.update()
        time.sleep(1)

# 게임 시작 전 카운트다운 실행
countdown()

# 게임 루프
running = True
clock = pygame.time.Clock()

while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False




# 플레이어 이동
    keys = pygame.key.get_pressed() -10
    if keys[pygame.K_UP] and playery > 0:
        playery -= player_speed
    if keys[pygame.K_DOWN] and playery < screen_height - player_height:
        playery += player_speed
    if keys[pygame.K_LEFT] and playerx > 0:
        playerx -= player_speed
    if keys[pygame.K_RIGHT] and playerx < screen_width - player_width:
        playerx += player_speed
```

플레이어의 움직임을 처리하는 부분이다. 플레이어의 키 입력을 감지하여 플레이어를 움직인다.
1. 'keys' 변수는 'pygame.key.get_pressed()' 함수를 통해 현재 눌려진 모든 키의 상태를 나타내는 배열을 받는다.
2. 'keys[pygame.K_UP]'은 플레이어가 위쪽 화살표 키를 눌렀는지를 확인한다. 만약 눌려진 상태이고, 플레이어의 y좌표('playery')가 0보다 크다면, 플레이어의 y좌표를 'player_speed'만큼 감소시킨다=플레이어가 위로 움직인다.
3. keys[pygame.K_DOWN]는 플레이어가 아래쪽 화살표 키를 눌렀는지를 확인한다. 만약 눌려진 상태이고, 플레이어의 y 좌표 (playery)가 화면의 높이에서 플레이어의 높이를 뺀 값보다 작다면,

플레이어의 y 좌표를 player_speed만큼 증가시킨다.=플레이어가 아래로 움직인다.
4. ‘keys[pygame.K_LEFT]는 플레이어가 왼쪽 화살표 키를 눌렀는지를 확인한다. 만약 눌려진 상태이고, 플레이어의 x 좌표 (playerx)가 0보다 크다면, 플레이어의 x 좌표를 player_speed만큼 감소시킨다.= 플레이어가 왼쪽으로 움직인다.
5.’keys[pygame.K_RIGHT]는 플레이어가 오른쪽 화살표 키를 눌렀는지를 확인한다. 만약 눌려진 상태이고, 플레이어의 x 좌표 (playerx)가 화면의 너비에서 플레이어의 너비를 뺀 값보다 작다면, 플레이어의 x 좌표를 player_speed만큼 증가시킨다. 이는 플레이어가 오른쪽으로 움직인다

```
# 재료 이동(채소)
vege_y += vege_speed -11
if vege_y > screen_height:
    vege_x = random.randint(0, screen_width - vege_width)
    vege_y = -vege_height
    vege_image = random.choice(vege_images)
```

재료(채소)의 움직임을 처리하는 부분이다. 재료를 아래로 움직이게 하고, 화면 아래로 벗어나면 다시 무작위 위치에서 등장하도록 한다.
1. if vege_y > screen_height:는 채소의 y 좌표가 화면의 높이(screen_height)를 넘어섰는지 확인한다.
2. 채소가 화면 아래로 벗어났다면
vege_x = random.randint(0, screen_width - vege_width)는 채소의 x 좌표(vege_x)를 화면의 너비 내에서 무작위로 선택하고, 채소가 다시 나타날 위치가 결정된다. vege_y = -vege_height는 채소의 y 좌표(vege_y)를 음수로 설정하고, vege_image = random.choice(vege_images)는 리스트에서 랜덤으로 채소 이미지를 선택한다. 채소 이미지가 다시 화면에 나타난다.

```
# 재료 이동(면) -12
noo_y += noo_speed
if noo_y > screen_height:
    noo_x = random.randint(0, screen_width - noo_width)
    noo_y = -noo_height
    noo_image = random.choice(noo_images)

# 재료 이동(꼬치류) -13
ske_y += ske_speed
if ske_y > screen_height:
    ske_x = random.randint(0, screen_width - ske_width)
    ske_y = -ske_height
    ske_image = random.choice(ske_images)

# 방해물 이동(음식물 쓰레기) -14
trash_y += trash_speed
if trash_y > screen_height:
    trash_x = random.randint(0, screen_width - trash_width)
```

```
        trash_y = -trash_height
        trash_image = random.choice(trash_images)

    # 충돌 검사
    player_rect = pygame.Rect(playerx, playery, player_width, player_height)
    vege_rect = pygame.Rect(vege_x, vege_y, vege_width, vege_height)
    noo_rect = pygame.Rect(noo_x, noo_y, noo_width, noo_height)
    ske_rect = pygame.Rect(ske_x, ske_y, ske_width, ske_height)
    trash_rect = pygame.Rect(trash_x, trash_y, trash_width, trash_height)
```

'pygame.Rect()' 함수를 사용하여 플레이어, 채소, 면, 꼬치류, 방해물의 충돌을 감지하기 위한 충돌 박스(Rectangle)를 생성한다. 각각의 충돌박스는 위치, 크기를 가지고 있다.

```
    if player_rect.colliderect(noo_rect): -16
        noodle += 1
        noo_x = random.randint(0, screen_width - noo_width)
        noo_y = -noo_height
        noo_image = random.choice(noo_images)
        score += 10
```

player_rect.colliderect(noo_rect)는 플레이어의 충돌 박스와 면류의 충돌 박스가 서로 겹치는지를 확인한다.
만약 충돌이 발생하면 noo_x = random.randint(0, screen_width - noo_width)와 noo_y = -noo_height는 면류의 위치를 화면 내의 무작위 좌표로 재설정하여 면류가 다시 나타날 위치를 결정한다.

noo_image=random.choice(noo_images)는 리스트에서 면류 이미지를 선택하여 noo_image 변수에 할당한다. 또한 score가 10만큼 증가한다.

```
    if player_rect.colliderect(vege_rect):
        veges += 1
        vege_x = random.randint(0, screen_width - vege_width)
        vege_y = -vege_height
        vege_image = random.choice(vege_images)
        score += 10
    if player_rect.colliderect(ske_rect):
        skewer += 1
        ske_x = random.randint(0, screen_width - ske_width)
        ske_y = -ske_height
        ske_image = random.choice(ske_images)
        score += 10

    if player_rect.colliderect(trash_rect):
        trash_x = random.randint(0, screen_width - trash_width)
        trash_y = -trash_height
```

```
        running = False
        screen.fill("pink")
        final_scores = font.render("Your Score: " + str(score), True, "black")
        screen.blit(final_scores, (200, 200))
        screen.blit(game_over, (300, 300))
        time.sleep(1)
        pygame.display.update()
        time.sleep(5)
        pygame.quit()
        sys.exit()
```

screen.fill("pink"는 화면을 분홍색으로 채워준다.
final_scores = font.render("Your Score: " + str(score), True, "black")는 최종 점수를 알려주는 텍스트를 나타낸다.
screen.blit(final_scores, (200, 200))과 screen.blit(game_over, (300, 300))는 최종 점수와 게임 오버 메시지를 화면에 표시한다.
time.sleep(1)은 1초 동안 대기한다.
pygame.display.update()은 화면을 업데이트하여 변경 사항을 표시한다.
time.sleep(5)는 5초 동안 대기한다.
pygame.quit()과 sys.exit()은 pygame을 종료하고 프로그램을 완전히 종료한다.

```
    #점수 증가, 레벨 증가
    increase_level(score)
    change_difficulty(level)
```

increase_level(score) 함수는 주어진 점수를 기반으로 레벨을 증가시킨다. 함수 내부에서는 점수에 따라 레벨이 증가하는 조건문이 사용되고, 조건에 부합할 경우 전역 변수인 level을 증가시킨다.
change_difficulty(level) 함수는 주어진 레벨에 따라 게임의 난이도를 변경한다. 함수 내부의 조건문을 통해 레벨에 따른 난이도를 설정한다.

```
 #화면을 하얗게 하기
    screen.fill((255, 255, 255))
```

screen.fill((255, 255, 255))는 화면을 흰색으로 채운다.
screen.blit(player_image, (playerx, playery))는 플레이어 이미지를 화면에 그린다. player_image는 플레이어 이미지 객체를 나타내며, (playerx, playery)는 플레이어의 위치를 나타낸다. vege, noo, ske, trash 모두 같은 형태이다.

```
    #화면에 이미지 그리기
    screen.blit(player_image, (playerx, playery))
    screen.blit(vege_image, (vege_x, vege_y))
    screen.blit(noo_image, (noo_x, noo_y))
    screen.blit(ske_image, (ske_x, ske_y))
```

```
    screen.blit(trash_image, (trash_x, trash_y))

    #목표 개수를 화면에 나타내기
    aima = small_font.render("veges:"+str(a), True,"black")
    screen.blit(aima, (10, 10))
    aimb = small_font.render("noodle:"+str(b), True,"black")
    screen.blit(aimb, (200, 10))
    aimc = small_font.render("skewer:"+str(c), True,"black")
    screen.blit(aimc, (390, 10))
```

screen.blit(aima,(10, 10))는 'aima' 텍스트를 화면 (10, 10)위치에 나타낸다. aimb, aimc도 같은 형태이다. 또한 초반에 형성한 변수 a, b, c를 문자열로 변환한다. 각 텍스트 객체는 render() 함수를 사용하여 생성되고, blit() 함수를 사용하여 화면에 그려진다.

```
    #획득한 점수와 재료 개수를 화면에 나타내기
    scores = small_font.render("Score: " + str(score), True,"black")
    screen.blit(scores, (10, 560))
    veges_text = small_font.render("Veges: " + str(veges), True, "black")
    screen.blit(veges_text, (10, 40))
    noodle_text = small_font.render("Noodle: " + str(noodle), True, "black")
    screen.blit(noodle_text, (200, 40))
    skewer_text = small_font.render("Skewer: " + str(skewer), True, "black")
    screen.blit(skewer_text, (390, 40))
```

위의 코드와 같은 형태로 현재 점수와 획득한 재료의 개수를 나타낸다.

```
    pygame.display.update()

    # 게임 종료 시간 검사
    current_time = time.time()
    elapsed_time = current_time - start_time
    if elapsed_time >= time_limit:
        running = False
```

current_time = time.time()은 현재 시간을 측정하여 current_time 변수에 저장한다.
elapsed_time = current_time - start_time은 현재 시간과 게임 시작 시간(start_time)의 차이를 계산하여 경과 시간(elapsed_time)을 구한다.
if elapsed_time >= time_limit:은 경과 시간이 지정된 시간 제한(time_limit) 이상인지 확인한다.
running = False은 running 변수를 False로 설정하여 게임 실행을 중지한.

```
        #성공시 화면
        if ((veges>=a)and(noodle>=b)and(skewer>=c)):
```

만약 제한시간이 끝났을 때 모든 재료가 목표 이상의 개수라면 점수와 'clear!'라는 문구가 있는 화면이 뜬다.

```
screen.fill("pink")
final_scores = font.render("Your Score: " + str(score), True, "black")
screen.blit(final_scores, (200, 200))
screen.blit(game_clear, (350, 300))
time.sleep(1)
pygame.display.update()
time.sleep(5)
pygame.quit()
sys.exit()
```

#실패시 화면

만약 제한시간이 끝났을 때 재료 중 하나라도 목표 개수를 채우지 못했다면, 점수와 'you fail!'이라는 문구가 있는 화면이 뜬다.

```
else:
    screen.fill("pink")
    final_scores = font.render("Your Score: " + str(score), True, "black")
    screen.blit(final_scores, (200, 200))
    screen.blit(game_over, (300, 300))
    time.sleep(1)
    pygame.display.update()
    time.sleep(5)
    pygame.quit()
    sys.exit()
```

clock.tick(60)

초당 60프레임으로 게임을 실행하도록 설정한다.

```
# 게임 종료
pygame.quit()
```

-게임 간단 설명-

1. 코드를 실행하면 게임 창의 상단에 플레이어가 획득해야 하는 여러 마라탕 재료들의 개수가 랜덤으로 주어진다.

2. 3, 2, 1 카운트다운을 하는 동안 플레이어는 획득해야 하는 마라탕 재료들이 무엇인지, 각 몇 개씩인지 확인한다.

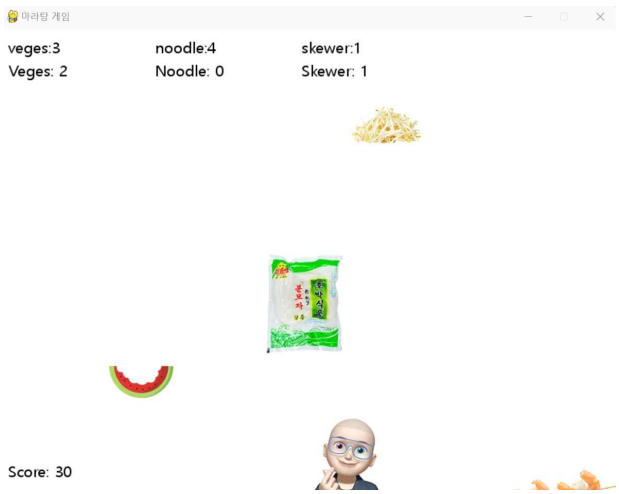
3. 방향키들을 사용하여 캐릭터를 이동시키며 재료들을 제한 시간 내에 획득해야 성공한다.

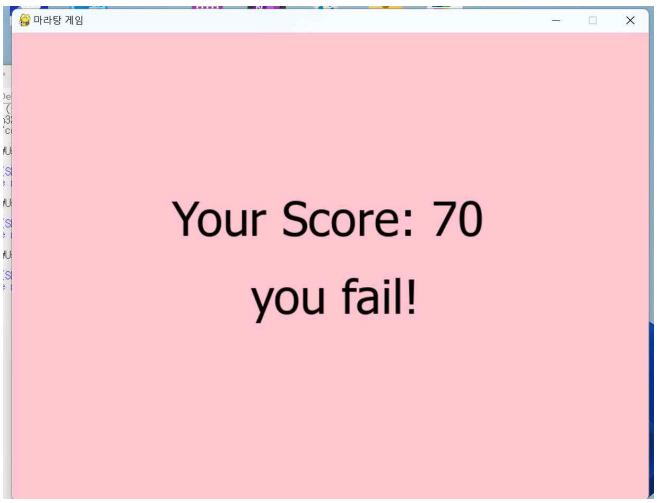
4. 제한 시간 내에 50점, 100점, 150점처럼 50점씩 증가할 때마다 레벨이 올라가는 동시에 위에서 떨어지는 마라탕 재료들의 속도들이 빨라진다.

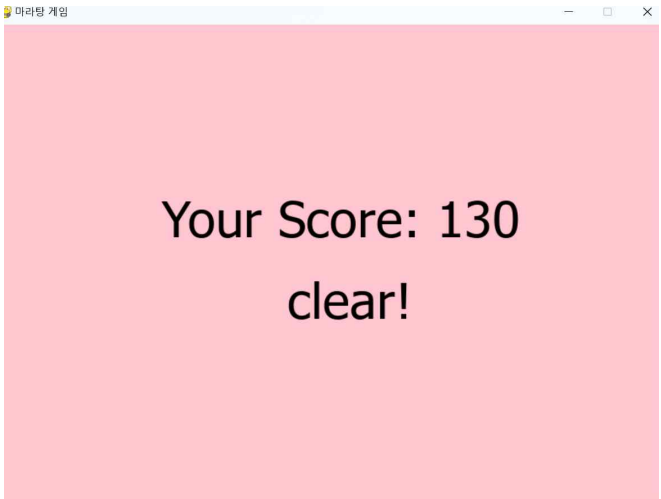
5. 점수가 높더라도 제한 시간 내에 게임 창 상단에 표시되어 있는 마라탕 재료들을 목표 개수만큼 획득하지 못하면 플레이어의 점수와 'fail'이라는 문구가 나타나며 게임이 종료된다.

6. 제한 시간 내에 마라탕 재료들을 목표 개수만큼 획득하면 마찬가지로 플레이어의 점수와 'clear'라는 문구가 나타나며 게임이 종료된다.

게임 스크린샷







1. 게임 시작 전 카운트다운 하는 모습

2.게임 실행 후 마라탕 재료와 관련없는 음식물 쓰레기들이 랜덤으로 내려오는 모습
(스코어가 50 씩 증가할 때마다 레벨이 올라가며 음식이 내려오는 속도가 빨라진다.)

3. 게임 창 상단에 적힌 목표에 제한 시간 내에 도달하지 못하면 점수와 실패했다는 문구가 뜨며 게임이 종료되는 모습

4. 게임 창 상단에 적힌 목표에 제한 시간 내에 도달하면 점수와 성공했다는 문구가 뜨며 게임이 종료되는 모습

결론

3-1. AI도구 활용 방법

AI 도구는 'chat gpt'와 'bard'를 사용했다. 'chat gpt'만을 사용하다가 정보량이 부족하다고 생각하였고, 보다 더 정확한 결과를 위해 구글의 'bard'를 추가로 사용하였다.

AI가 내놓은 결과를 맹신하지 않고 검토하고 비판하려고 했다.

수업 시간에 배운 내용을 토대로 교재를 참고하여 최대한 직접 코드를 짜보되 막히는 부분에 대해서는 AI를 통해 확인하고 참고하여 코드를 수정하였다.

AI에 100% 의존하기보단 우리의 상상력 및 코드 작성에 대한 아이디어를 얻도록 돕는 역할로 사용하려 했고 AI가 내놓은 결과를 그대로 붙여넣는 것이 아닌 반드시 직접 실행해보고 다른 방향으로의 수정을 의논하고 검토하였다.

3-2. 느낀점

게임 만들기 조별 과제를 하는 동안 그동안 수업시간에 배웠던 내용들을 활용할 수 있어 보람을 느꼈다. 아직은 아는 것이 많지 않아 간단한 게임을 만드는 것도 쉽지 않았지만, 더 많은 것을 배우면서 더 복잡하고 재밌는 게임과 어플리케이션을 만들 수도 있겠다는 생각에 앞으로 배울 내용에 대해 기대감을 가질 수 있었다. 또한 팀원들과 브레인스토밍을 통해 의견을 조율하며 게임을 만드는, 과거에는 해본 경험이 없는 색다른 과제를 통해 협력하는 것과 실력적인 면에서 성장할 수 있음을 느꼈다.

chat gpt와 같은 채팅형 인공지능이 발전함에 따라 AI가 인간의 영역을 침범하는 것이 아닌가 하는 의구심이 있었다.

게임을 만드는 과정에서 다양한 채팅형 인공지능을 사용해보면서 AI가 알려주는 코드와 내용들 중 정확하지 않은 부분들이 많았기에 AI가 알려준 내용들을 참고용으로 사용하며 팀원들의 지식들을 모아 살을 붙이는 방식으로 진행하였다. 그것이 AI를 사용하는 올바른 방식이라고 생각하였고, 이 과정을 통해 AI를전적으로 믿으면 안된다는 생각을 하였다. 아직은 AI가 인간의 일을 돕는 차원에서는 널리 사용될 수 있겠으나, 인간이 하는 일의 전체를 대체할 수는 없을 것 같다.