What the assigment 1 was about?

The assigment 1 was about projectile, and simulating them through code. Gravity, velocity and accerlations to show the projectile's path. The code can be described in the biggest overview as :

A blue cannon sits on the left side of the screen . You aim by moving your mouse - the cannon barrel points wherever you click.Press SPACEBAR to fire the cannonball.

The program calculates four real forces acting on the cannonball:

1. **Gravity** (9.8 m/s²) - Pulls the ball downward
2. **Wind** - Randomly changes direction and strength each shot
3. **Air Resistance** - Slows the ball down as it flies
4. **Initial Velocity** - The power from the cannon blast

1. Multiplayer Gameplay System

Original Implementation:

- Single cannon with no competitive elements
- No turn-based mechanics
- Limited to solo projectile physics demonstration

```
# CHANGE 1: ADDED SECOND PLAYER
cannon2 = {
    "x": width - 200,  # Place on right side
    "y": 0 + cannon_height,
    "vx": -84.85,  # Aim left
    "vy": 84.85,
    "width": cannon_width,
    "height": cannon_height,
    "color": GREEN,
    'ball_radius': 2
}
```

This turn solo game into a multiplayer game.

2. Gameplay with Targets

Original Implementation:

- Projectiles fired without purpose
- No scoring or achievement system
- Physics demonstration without goals

```
# CHANGE 2: ADDED TARGETS TO HIT
targets = [
    {"x": 500, "y": 100, "width": 30, "height": 30, "color": RED},
    {"x": 1500, "y": 200, "width": 30, "height": 30, "color": RED}
]
```

Added clear gameplay objectives. Created scoring opportunities in the future.

What the assigment 2 was about?

This is a virtual ecosystem - like a digital aquarium or a computer game where we create a small world with plants, fish, and bears that interact with each other. We're trying to answer the question: Can we make a balanced digital world where all three species can live together for a long time?

This project started as a group assignment with:

- Magali Baulina
- Ignacio Lucero
- Agustin Marino Navarro
- Muhammad Ansar Mahmood

The code worked okay, but we had a issue of the plant quickly just being the dominant species. And literaly the whole grid was turned green. So, I have tried to improve it.

The code

Imagine a **grid of squares** (50 wide × 20 tall = 1000 squares total). Each square can hold:

- A plant (green)
- A fish (blue)
- A bear (brown)
- Or be empty (beige)

The rules are simply : Plants get older, change color as they grow, can spread to empty spots when mature, and eventually die from old age.

Fish need to eat plants to survive. If they're old enough and have enough food, they can have babies. When too many fish are crowded together, some die. They move to empty spaces.

Bears need to eat fish to survive. If they're old enough and have enough food, they can have babies. If they don't eat for too long, they starve to death. They move around looking for food

Food Chain Implementation of Fish

This report analyzes the key differences between the original ecosystem simulation and the improved version, focusing on three major changes that significantly impact ecosystem longevity and balance. While the improved version demonstrates better sustainability, both implementations ultimately struggle with plant dominance due to fundamental design constraints.

```
def fish_rules(cur,r,c,neighbour_fish, neighbour_empty):
    """ Given the current grid {cur}, a position (r,c) which contains a fish, and a list of grid-positions
    fish-neighbours  and a list of grid-positions of empty neighbour cells. Update the grid according to th
    # implement the fish rules
    # update age
    cur[r,c]['age'] += 1
```

In the improved version, I tried to but the original code the plants had no consumers, leading to exponential growth. Now the fish apply predation pressure on plants. Even with the new behaviour the fish consumption rate insufficient to control rapid plant reproduction. But still have the same issue with the food chain, being established but imbalance persists due to disproportionate reproduction rates.

```
def fish_rules(cur,r,c,neighbour_fish, neighbour_plant, neighbour_empty):
    """IMPROVED: Better energy management and movement"""
    cur[r,c]['age'] += 1

    # ENERGY MANAGEMENT - Only consume food when moving/breeding
    if len(neighbour_empty) > 0 or len(neighbour_plant) > 0:
        cur[r,c]['food'] -= 1

    # FISH EAT PLANTS - Smart eating behavior
    if len(neighbour_plant) > 0:
        if cur[r,c]['food'] < fish_starvation * 0.6:  # Eat when moderately hungry
            r_plant, c_plant = random.choice(neighbour_plant)
            cur[r_plant, c_plant] = empty()
            cur[r,c]['food'] = min(fish_starvation, cur[r,c]['food'] + 8)  # Partial refill
            neighbour_plant.remove((r_plant, c_plant))
            neighbour_empty.append((r_plant, c_plant))
```

Population Control Mechanisms - Fish

Populations crashed quickly from strict overcrowding rules**. A**nd thus not allowing for flucation to be able to maintain some chance for letting the population not sorely die after a couple rounds, but it depends on probability.

```
    # the fish dies of overcrowding if there are now 2 or more neighboring fish
    if len(neighbour_fish) >= fish_overcrowd:
        cur[r, c] = empty()
```

More forgiving system allows population fluctuations. Reduced death rates combined with high birth rates still favor plant dominance. It allows for a longer ecosystem duration but eventually the plant takes over.

Reproduction Balancing – Plant

Plants filled the grid in 20-30 turns due to 70% reproduction rate.

```
# Plant initial parameters
plant_growth_time = 4       # ticks to grow to next stage
plant_max_age = 20          # dies naturally after this
plant_breed_chance = 0.7    # probability to reproduce if mature
def new_plant():
    ID_plant = new_ID()
    plant = {'type': 'plant', 'id': ID_plant, 'age': 0, 'col': col_small_plant}
    return plant
```

It does slow the expansion but still outcompetes animals.Even reduced reproduction rates exceed animal consumption capacity. It delays the plant overtaking the grid but in the end the plant still dominates.

```
# Plant parameters
plant_growth_time = 4          # Balanced growth
plant_max_age = 15            # Reasonable lifespan
plant_breed_chance = 0.35     # Sustainable reproduction rate
plant_max_density = 6         # Prevents overpopulation but allows spread

def new_plant():
    ID_plant = new_ID()
    plant = {'type': 'plant', 'id': ID_plant, 'age': 0, 'col': col_small_plant}
    return plant
```

References

The initial code from **Assignment 1** was used as the foundation for this improved code. All subsequent development and improvements presented in **Assignment 2** were a group project:

- **Assignment 1:** Individual submission by Muhammad Ansar Mahmood.Which was further improved in this report.
- **Assignment 2:** Code developed by Magali Baulina, Ignacio Lucero, Agustin Marino Navarro, and Muhammad Ansar Mahmood. Which served as a further improvement for this report.