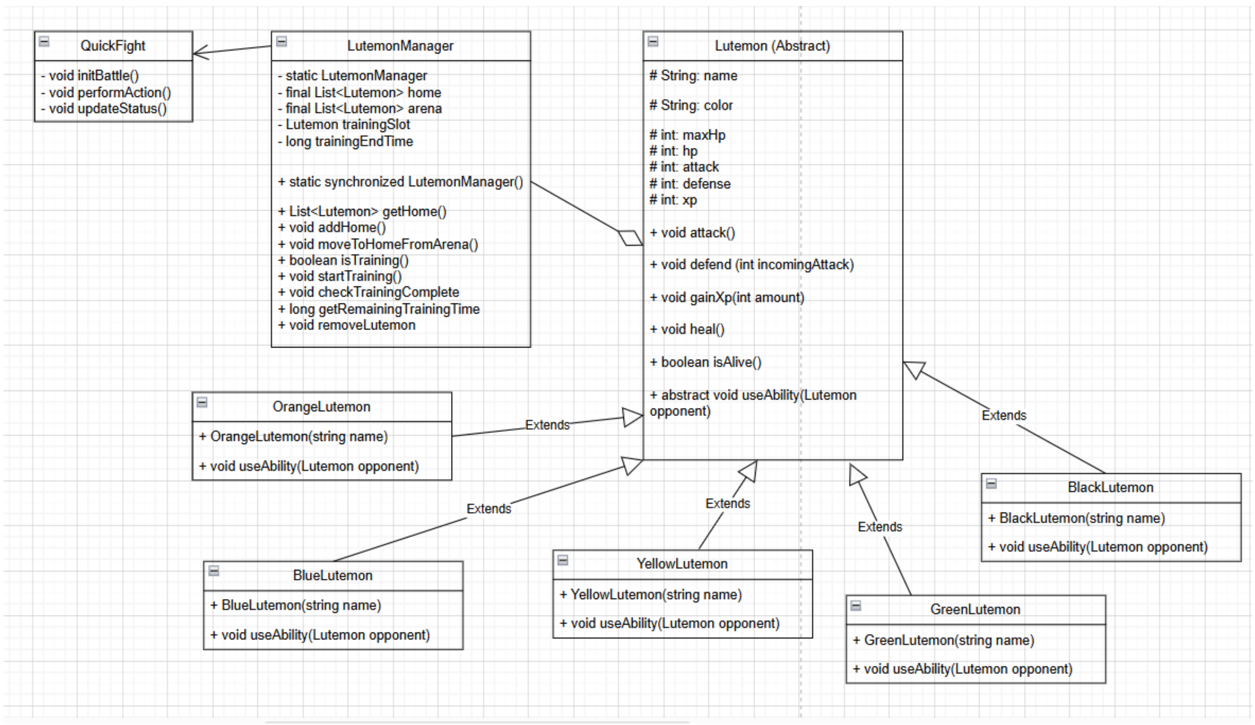
Object-Oriented Programming Course 2025 3rd-4th Period Project Description

Máté Havas – 001560882

Introduction

My project implements the basic functionalities outlined in the requirements. Additionally, I included some randomization mechanisms, recyclerView, used fragments meaningfully and also created and timer/countdown functionality in case the user would want to progress through the game without actually playing. The final product massively differs from what I explained in the plan, I had a lot of troubles with Android Studio that was not related to my OOP skills, but rather trying to navigate how it works and was spending a lot of hours trying to troubleshoot the gradle building of my program etc. Which in the end resulted in a massive loss of ambition, and while I planned on having a campaign mode and also displaying statistics, I had to scrape those functionalities to somehow hit the deadline.

UML Diagram

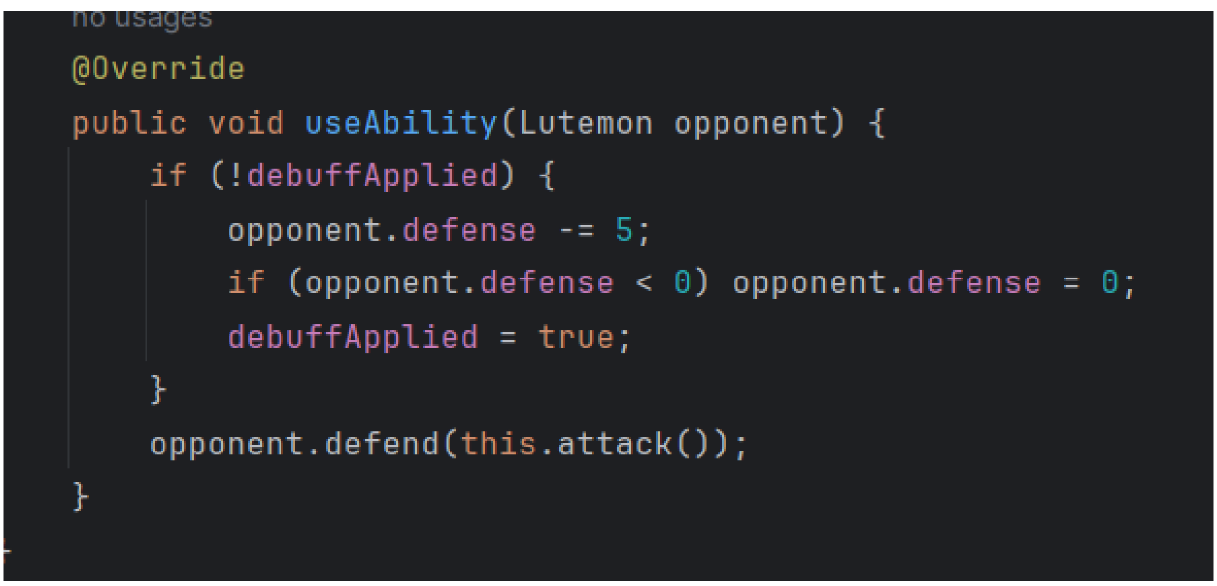

QuickFight
- void initBattle()
- void performAction()
- void updateStatus()
LutemonManager
- static LutemonManager
- final List<Lutemon> home
- final List<Lutemon> arena
- Lutemon trainingSlot
- long trainingEndTime
+ static synchronized LutemonManager()
+ List<Lutemon> getHome()
+ void addHome()
+ void moveToHomeFromArena()
+ boolean isTraining()
+ void startTraining()
+ void checkTrainingComplete
+ long getRemainingTrainingTime
+ void removeLutemon
Lutemon (Abstract)
# String: name
# String: color
# int: maxHp
# int: hp
# int: attack
# int: defense
# int: xp
+ void attack()
+ void defend (int incomingAttack)
+ void gainXp(int amount)
+ void heal()
+ boolean isAlive()
+ abstract void useAbility(Lutemon opponent)
OrangeLutemon
+ OrangeLutemon(string name)
+ void useAbility(Lutemon opponent)
Extends
BlueLutemon
+ BlueLutemon(string name)
+ void useAbility(Lutemon opponent)
Extends
YellowLutemon
+ YellowLutemon(string name)
+ void useAbility(Lutemon opponent)
Extends
GreenLutemon
+ GreenLutemon(string name)
+ void useAbility(Lutemon opponent)
Extends
BlackLutemon
+ BlackLutemon(string name)
+ void useAbility(Lutemon opponent)
Extends

Members

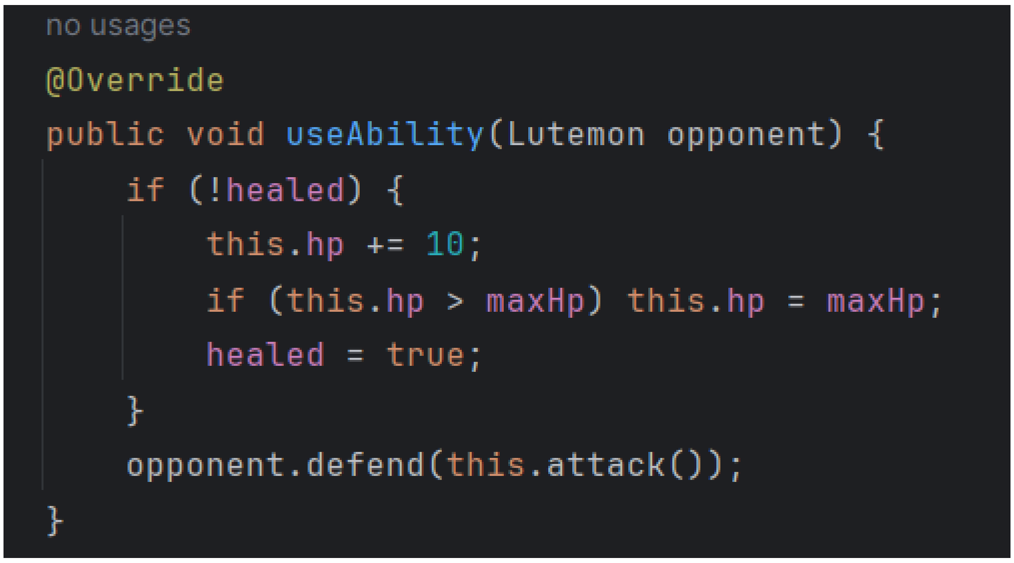
Máté Havas – was solo work, implemented everything.

Description and features

My program includes the outlined pokémon- like game. Users can create Lutemons, 5 types in total: black, orange, yellow, green and blue. The varying types of Lutemons have different base health points, defense and also attack at the very beginning. The most exciting version of this is the different abilities the Lutemons can use during battle, some of the heal, some of the deflect etc.:



```java
no usages
@Override
public void useAbility(Lutemon opponent) {
    if (!debuffApplied) {
        opponent.defense -= 5;
        if (opponent.defense < 0) opponent.defense = 0;
        debuffApplied = true;
    }
    opponent.defend(this.attack());
}
```

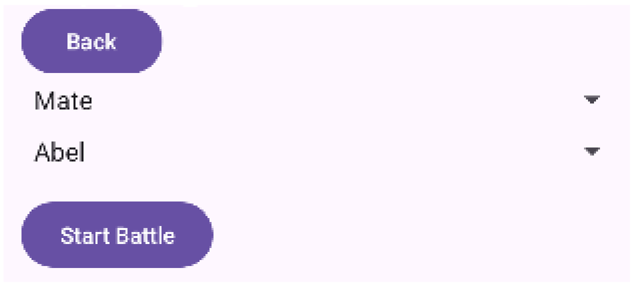


```java
no usages
@Override
public void useAbility(Lutemon opponent) {
    if (!healed) {
        this.hp += 10;
        if (this.hp > maxHp) this.hp = maxHp;
        healed = true;
    }
    opponent.defend(this.attack());
}
```

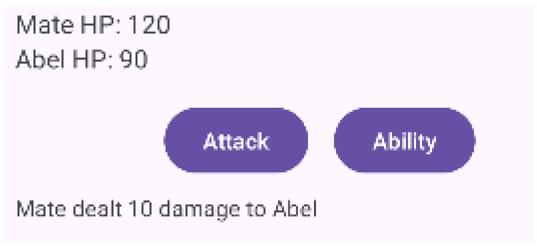
Healing ability!

The top-most lutemon in the home list can be moved to a training area, where after a 5 minute countdown, the Lutemon receives a certain amount of XP, this feature is for the lazy folks.

Furthermore, the program evidently includes the fighting mode. The user selects their Lutemons for the battle, after which they can press start:

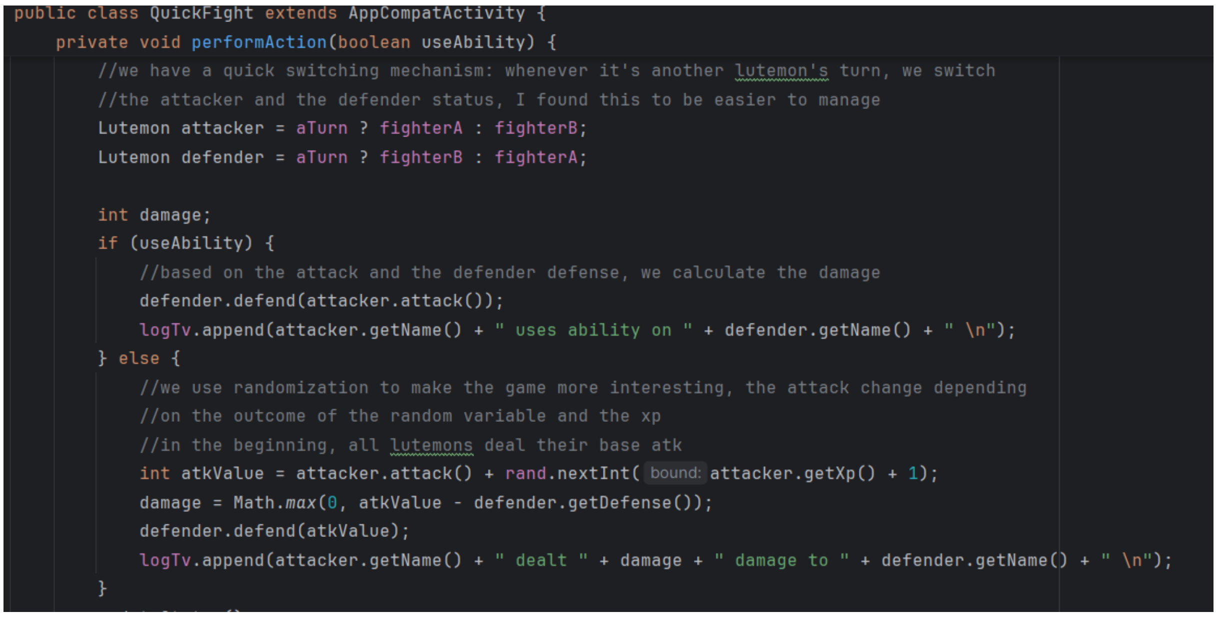


While the fight itself may not be visually appealing, there are plenty of things happening underneath!



```
Mate HP: 120
Abel HP: 90
```

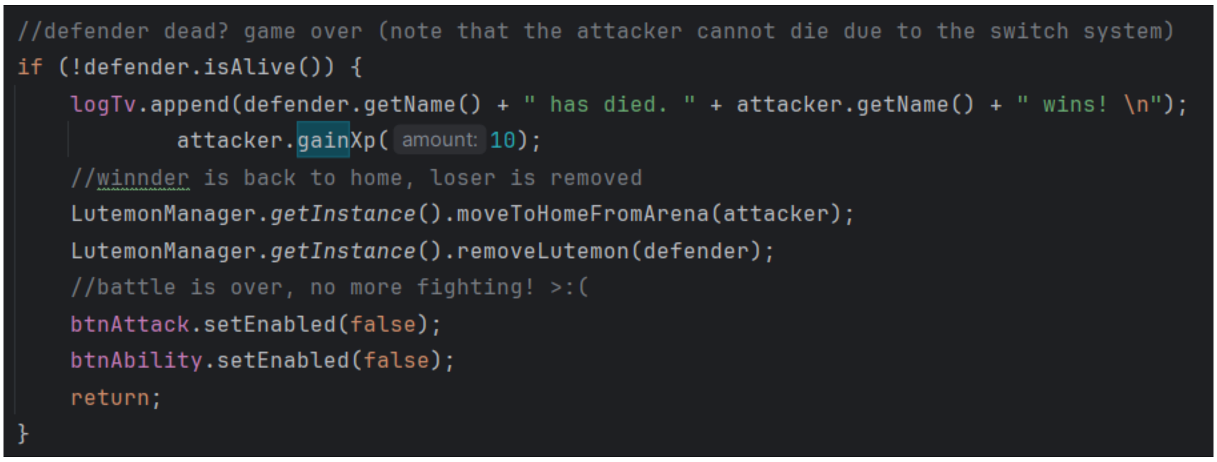
```
Mate dealt 10 damage to Abel
```



```java
public class QuickFight extends AppCompatActivity {
    private void performAction(boolean useAbility) {
        //we have a quick switching mechanism: whenever it's another lutemon's turn, we switch
        //the attacker and the defender status, I found this to be easier to manage
        Lutemon attacker = aTurn ? fighterA : fighterB;
        Lutemon defender = aTurn ? fighterB : fighterA;

        int damage;
        if (useAbility) {
            //based on the attack and the defender defense, we calculate the damage
            defender.defend(attacker.attack());
            logTv.append(attacker.getName() + " uses ability on " + defender.getName() + " \n");
        } else {
            //we use randomization to make the game more interesting, the attack change depending
            //on the outcome of the random variable and the xp
            //in the beginning, all lutemons deal their base atk
            int atkValue = attacker.attack() + rand.nextInt( bound: attacker.getXp() + 1);
            damage = Math.max(0, atkValue - defender.getDefense());
            defender.defend(atkValue);
            logTv.append(attacker.getName() + " dealt " + damage + " damage to " + defender.getName() + " \n");
        }
```

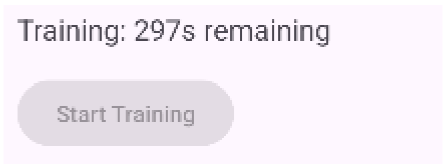
Using randomization and accounting for the xp the Lutemon has acquired in their lifetime, a certain number gets added on its base value, and that will be the final damage dealt tot the opponent.



```
//defender dead? game over (note that the attacker cannot die due to the switch system)
if (!defender.isAlive()) {
    logTv.append(defender.getName() + " has died. " + attacker.getName() + " wins! \n");
            attacker.gainXp( amount: 10);
    //winnder is back to home, loser is removed
    LutemonManager.getInstance().moveToHomeFromArena(attacker);
    LutemonManager.getInstance().removeLutemon(defender);
    //battle is over, no more fighting! >:(
    btnAttack.setEnabled(false);
    btnAbility.setEnabled(false);
    return;
}
```

If there is a winner, the winning Lutemon returns tot he home list, and the other one is removed from the program.

In the case of not wanting to kill our precious Lutemons, we have the option to train the top-most lutemon ont he recycleView list which receives a certain amount of XP after a countdown:



Training: 297s remaining