Daniel Fletcher(1613444)

Tamagotchi Final Report

For Project 2, I chose the Tamagotchi project, a virtual pet day care simulator, based on three common household pets, a Dog, Cat, and a Snake. I started my project by designing the functions because it was important to create the foundation of the game first before developing the game simulation logic. I first chose to create a base class where the rest of my classes will be derived from, so for this assignment, I chose to have Pet be the base class, and Snake, Cat, and Dog be the derived classes that inherit the common attributes and interactions of a pet and for my game, some extras such as the levels of its bodily mechanisms like hunger, sleepiness, and happiness. The inherited functions from the base class were setName, playWith, makeNap, Train, Feed, speak, and nextHour. These functions were to be virtual so my derived classes could override and have their own unique implementations. The only function that wasn’t virtual and shared the same implementation among the classes was setName.

The instructions for running this code is, when the program is executed you will be presented with a screen of options, each option is represented by an integer. Enter the integer for the desired menu option.

In-depth view of the Pet base class header:

class Pets {

public:

Pets(); // for assigning random values to the instantiated object

void setName(string); // assigns name to the current Pet through user input

string getName(); // returns name of Pet

virtual void playWith(); //

virtual void makeNap(); // to make the Pet sleep and recover energy

virtual void Train(); // training function for boosting the pet’s stats

virtual void Feed(); // interactive function for feeding the pet

virtual void nextHour(); // time lapse by an hour

virtual void save(); // to save the game attributes into a file

virtual void load(); // to load the attributes and continue with the file

protected:

string name; // name variable of the pet file

int age; // age variable to describe the age of the pet

int hunger\_lvl; // the level of the pet’s hunger

int sleepy\_lvl; // the level of the pet’s sleepiness

int happy\_lvl; // the level of the pet’s happiness

};

The first class I designed after the Pet base class, was the Dog derived class. It shared the same functions as the Pet class, but it also had some of its own like Fetch, takeForWalk, etc. I also included an attribute known as a breed to specify what type of Dog the user desires.

How my final implementation differed from my original plan, was that I had to cut out a lot of features, I had to weigh the benefits and disadvantages, and while I could’ve had an overwhelming pet simulator I decided to choose requirements over creative endeavors to also have a project delivered in time. I also strayed away from using pointers, and a menu which in the end I wish I did because I ended up reusing a lot of code in my codebase which overall made my program very large and inefficient. I originally wanted a menu function to be assigned to each pet and whenever the user was to see it, the program would call the function, but I ended up using various print statements and conditional statements to give a similar look. I also considered using switch statements to limit the amount of if statements I used.

video overview: <https://drive.google.com/file/d/1o6ffbcGVyMCpFprbZImRij-NmIcKYf6g/view?usp=sharing>

Example of redundant/repetitive code:

if (choice == 1){

pet\_cat.playWith();

cout << endl;

}

if (choice == 2){

pet\_cat.makeNap();

cout << endl;

}

if (choice == 3){

pet\_cat.Train();

cout << endl;

}

if (choice == 4){

pet\_cat.Feed();

cout << endl;

}

if (choice == 5){

pet\_cat.yoyo();

cout << endl;

}

if (choice == 6){

pet\_cat.Scratch();

cout << endl;

}