OOP MAJOR PRAC DESIGN

Pokipet Pet

OOP Semester 2, 2020

Group-0029
Ziqi Zhang a1810054 - Yifeng Lyu a1751549 – Junguo Wang a1792147

The University of Adelaide

Project Specification - Major Practical Introduction

Our project designs a C++ game called ‘digital pets’, ‘Pokipet pets’ is a kind of artificial playmates. It allows players to select their own pet and play with it by choosing different functionality (e.g. feeding, petting, playing, talking, ignoring …). Players can also choose what types of food to feed, if they don’t feed or play with their pet, it might be dead then they need to re-create or load a pet again.

Design Description

Assessment Concepts

Memory allocation from stack and the heap

- **Arrays:** We will use a dynamic array using new to add in the words that the player wants to share with the pet, by calling the speak function.
- **Strings:** Things like player names, pet names, player input output and the behaviour of some functionality. Save file function uses strings.
- **Objects:** Game, Pet, Dog, Cat.

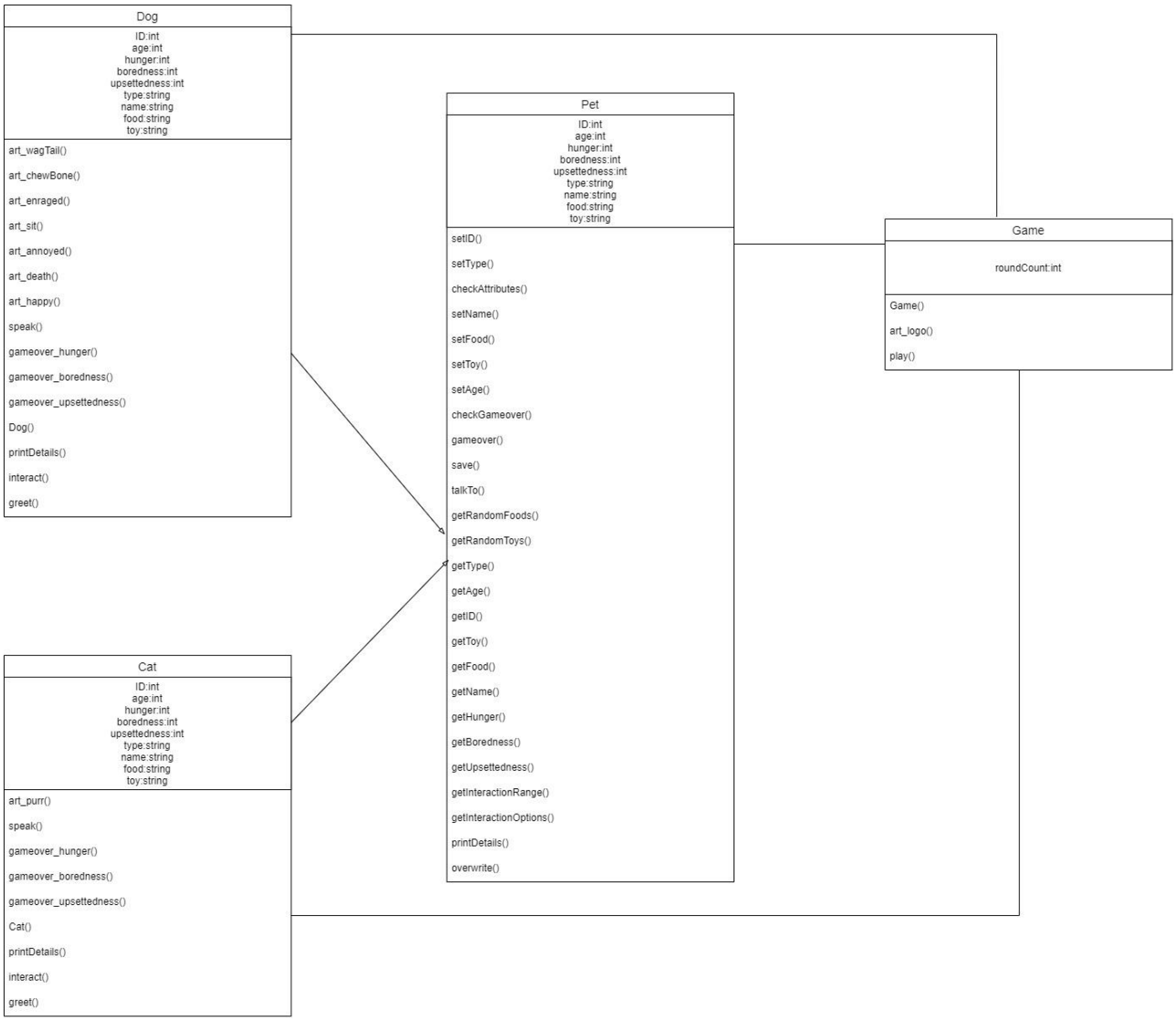
User Input and Output

- **I/O of different data types:** Command-line. Players enter names select pet by numbers. Console displays lists of available pets and different functionality. Additionally, player can also set their pets’ names and also speak with the pets.

Object-oriented programming and design

- **Inheritance:** There are different pet types which allow players to select, more different behavior depending on the type. Such as Dog which can wag it’s tail, sit,chew a bone and so on, Cat can make it’s special sound. In addition, there are variations in the randomness during interactions with different types of pets. Both of the sub classes (Cat and Dog) are inheritance form the parent class (Pet), thus both of the pets have the age, name, toy, food and some mood functions.
- **Polymorphism:** There are food, toy and name for each pet, once players select which pet they want, there are different types of food and toy for each pet. Such as Dog eats bones and plays ball, Cat eats fish and plays laser.
- **Abstract Classes:** Pet is our abstract class which covers common functionality of the sub classes,also there contains most of the pure virtual functions in the project.

Class Diagram



Class Descriptions

Pet

Parent class and Abstract class which covers common functionality of the sub classes. The behaviors in Pet allows players to choose the types of pet they want, also set a name for it.

Cat

Sub class. This class simulates a cat with some corresponding ‘functions’. As the players select the cat as their pet, there different functions are able to be called, For example, there are interacting functions like random food and toys contain different kinds of selections. Also if the player does not feed or look after the pet properly, it will be dead, the history will be saved. In order to specialize the Cat class, there is a function can create special sound from the cat.

Dog
Sub class. This class simulates a dog with some corresponding ‘functions’. As the players select the cat as their pet, there different functions are able to be called, For example, there are interacting functions like random food and toys contain different kinds of selections (same with the cat class). Also if the player does not feed or look after the pet properly, it will be dead, the history will be saved. In order to specialize the Dog class, the functionality of sit, wag tails, chew bones and so on are included into the class.

Game
The Game class is our user interface class, the player will follow the instructions in this class to select the pet they want and start playing the game.

User Interface
A user of this program is a player who wants to own a digital pet. They use the command-line. Users are presented with a list of options to choose from like:

```
Welcome to Pokipet!
Please enter your choice:
0. Load existing pet
1. Adopt a dog!
2. Adopt a cat!
3. Surprise me!
```

and enter the number for the corresponding option to select the pet.

Code Style
All code in the program will be properly indented using 4 space tabs.

Classes will begin with a capital letter. Class files (.h and .cpp) will be listed.

Function Comments will be given for each function that describes what the function does, what the return type represent and what any arguments are available for the function. Code Comments will be used to describe what each block of code performs if it is not obvious from just reading the code or the code is short. Comments will be written as the code is written.

Testing Plan

All our tests will run through our makefile each time we compile our code, help us significantly cut down on fixing bugs in the later development stages. Following testing methods will be included in our testing plan:

Unit Testing

Our unit tests will cover all public functions in our classes. Each class will have a corresponding test file like Pet_Tester.cpp, Game_Tester.cpp ..., which will include a main method that will exhaustively test each function with a set of inputs and test it matches an expected output.

This will help us significantly cut down on fixing bugs in the later development stages.

Integration Testing

Our integration tests will use bottom-up method to check the different classes work well and help to show progressively higher-level combinations of our units. In other words, there will be one Tester.cpp which includes players' different inputs and test if the program present what we expected. At this stage, we will consider different situations, such as what if the player types nothing or something unexpected.

Automated Testing

Using the Google testing.

Regression Testing

Our regression tests will re-run part of the unit tests and integration tests to ensure that previously developed and tested codes still perform after a change. This will be included in to another testing file.

Schedule Plan

Stretch Goals

Our goal is to outline all of the functionalities by week 9. If we have finished testing by this point then we will begin expanding our program to add more details into our functions.

Week 8 - Preparing for assessment in Week 9

Check-list:

Develop project plan and upload to svn.
Create SVN structure.
-Yifeng Lyu

Organise code sharing with group members.
- Ziqi Zhang

Write makefile.
- Junguo Wang

Finalise testing strategy with makefiles and input/output files outlined in Testing Plan.
-Yifeng Lyu

Break Week 1

Break Week 2

Week 9 - Preparing for assessment in Week 10

Check-list:

Complete the functions in our Pet header file.
- Junguo Wang

Think about more interesting functionality in our sub classes (Cat and Dog files).
- Ziqi Zhang

Outline the testing plan.
Unit testing, test all the small functions as far as we get.
-Yifeng Lyu

Week 10 - Preparing for assessment in Week 11

Check-list:

Fix bugs (the image of the pet is too large, the expected output of one of the functions has some issue), complete the program in more detail way (add more different functions into our sub classes, highlight the difference between the Cat and Dog).
- Junguo Wang

Unit testing new functions.
- Yifeng Lyu

Create a simple user interface, help to check if the project makes clear and easy to understand for the player.
- Junguo Wang

Integration and Automated testing to check our project.
Finish the final vision of the class diagram and description.
-Yifeng Lyu

Bring all group members works together and modify.

Week 11

Regression testing by limiting all the inputs and outputs.
Automated testing by using Google Test to double check the interacting functions in our sub classes (Cat and Dog).
- Junguo Wang

Adding the comments (state each function's functionality and the purpose).
Unify all of the code styles.
- Yifeng Lyu

Checking the rubric and fix the project in a more comprehensive way.
- Ziqi Zhang

Consummating the whole project and the document (e.g the class diagrams and the description).
Be ready to present the Project!
- All of the group members