​​HW 3: Tamagotchi

​​Overview

​​In this assignment you will implement a simplified version of [Tamagotchi](https://en.wikipedia.org/wiki/Tamagotchi). You will be able to play and feed pets (dogs, cats and poodles) as well as ask them to make a sound. They will leave if they are bored or hungry for too long. This will end the program. You will start with a partially implemented game that has some functions, parts of the player menu and the pet class implemented and documented. Building well designed sub classes for dogs, cats and poodles is up to you.

​​

​​The basic gameplay is as follows:

1. ​​The user is prompted for what kind of pet to adopt.
2. ​​The user is prompted for the pet’s name.
3. ​​The user is prompted for what sound the pet makes (e.g., “woof”, “meow”, “yo”)
4. ​​Game play begins.
5. ​​At each turn, the user issues a command (“speak”, “play”, “feed”, or “wait”). Depending upon the command, different things will happen, and the pet’s hunger​ and boredom​ attributes may be lessened.
6. ​​Regardless of the command, however, the pet’s hunger​ and boredom​ will *increase*, simply because of the passage of time.
7. ​​If a pet becomes too hungry, it will leave to find food elsewhere and the game will be over.
8. ​​If a pet becomes too bored, it will leave to find entertainment elsewhere and the game will be over.

​​

​​In this exercise you will gain experience designing classes, reflecting on encapsulation and working with class instances. You will deal with associated data in the form of attributes (or member variables) and methods that add unique functionalities to sub classes. You will become familiar with implementing inheritance, constructors and overriding methods.

​​

​​Instructions

​​For all assignment sections,

* ​​Make sure that your code adheres to the [507 Assignment Guide](https://paper.dropbox.com/doc/507-Assignment-Guide--AshR98EaGu3X1IlUreueQi4fAQ-RwRuP1S6RzY21Z04PhJDr).
* ​​See the Grading section for the specific point breakdown that will be used to calculate your grade.

​​

​​Main Assignment (100 points)

1. ​​Copy the file [hw3-tamagotchi.py](https://drive.google.com/file/d/1M2l6gR9IopBDmU1QffDK_mh_A52TBcmY/view?usp=sharing) to your computer and open it in your code editor. An example output of one run can be found [here](https://docs.google.com/document/d/197R7UyHSyVsQOVrcJhvwPIvRXkB3NwZi19BpwGp_s1E/edit?usp=sharing).

​​

​​**Part 1: Implement the Pet class**

1. ​​Add a parameter sound​ to the constructor of Pet. Assign the parameter to an *instance attribute* of Pet.
2. ​​Extend the Pet class to include following methods:
3. ​​clock\_tick( ):

* ​​This runs on every turn.  Add 2 to the pet’s hunger​ and boredom​

1. ​​speak( )

* ​​This runs when the “speak” command is issued.  Print "I say: "​ and then the pet’s unique sound. (e.g., "I say: woof!"​)

1. ​​play( )

* ​​This runs when the “play” command is issued. The user tries to guess which way the pet will look up to 3 times. If they guess correctly, the play is done and the pet’s boredom is decreased by 5 (recall that clock\_tick()​ will always increase boredom by 2, so the end result will be a decrease of 3). If they guess incorrectly all 3 times, the boredom​ is not affected (other than by clock\_tick()​). boredom​ cannot be negative, so if a change would result in a negative value for boredom​, then the boredom​ should be set to zero instead.a

​​

* ​​In each of the 3 rounds, the play()​ function randomly chooses which way the pet will look and then prompts the user to guess “left” or “right” (these are the only valid choices).  If the player guesses correctly print “Correct!” and exit play()​. If they guess wrong, print “I look to the right/left. Try again." In case invalid input was provided: "Only 'left' and 'right' are valid guesses. Try again.". Invalid input does not count as an attempt.
* ​​

1. ​​feed( )

* ​​This runs when the “feed” command is issued. The pet’s hunger​ is decreased by 5. As with boredom​, if a change to hunger​ would result in a negative value, instead set hunger​ to 0.

​​

​​**Part 2: Implement Dog and Cat as subclasses of Pet**

1. ​​Implement the Dog and *Cat* subclasses of Pet.
2. ​​Dogs express themselves a bit differently. They always add "arrrf!" at the end when they speak()​.  For example, if a pet’s sound​ is "Hello there!"​ her speak()​ would print "Hello there! arrrf!"​
3. ​​Cats will repeat their sound​ more than once. The number of repeats will be stored in an instance variable called meow\_count​, which is passed into the Cat constructor as a parameter. For example, while a Dog, like all Pets, will be instantiated with a name​  and a sound​, like so: d = Dog("Fido", "woof")​, cats will take another parameter like so: c = Cat("Milo", "purrrrr", 3)​.  When a cat speaks (i.e., when speak()​ is called on a Cat), they will repeat their sound meow\_count times. For example, if meow\_count​was 3 and the sound word was "meow"​, it should print "I say: meowmeowmeow"​.
4. ​​Cats have 5 attempts while playing, rather than the default 3 for Pets.

​​

​​

1. ​​At the top of the file you will find variables to which ASCII art dogs/cats are assigned as strings. Think of how you can refactor the play()​ method of the Pet​ class so that either a left or right looking cat or dog are displayed after a player has guessed a direction wrongly (i.e. first the failure message is displayed and then the ASCII image). *Hint:* The so far unused ascii\_art\_left​ and ascii\_art\_right​ attributes of Pet​ may be helpful. Think about how you could use inheritance and overriding in the Cat​ and Dog​ classes here to adapt them.

​​

​​**Part 3. Implement a subclass of *Dog called* *Poodle*.**

1. ​​*Poodle*
2. ​​Also has a method called *dance*, which returns "Dancing in circles like poodles do!".
3. ​​When the speak method is executed, it will print the dance method first and then the speak method from the superclass.

​​

​​**Part 4: Game Loop**

​​Create a menu to that allows player to adopt and care for a pet.

1. ​​When starting the program the players can choose through the prompt if they want to adopt a dog, cat or poodle and what its name and sound will be (in the case of the cat also a meow\_count should be given). We have provided functions (get\_name()​, get\_sound()​, get\_meow\_count()​) for prompting the player and while​ loop below them, you need to instantiate the pets.
2. ​​Then, the status of the pet is printed and the player can through console input use the methods available for each pet (except mood()​, \_\_init\_\_()​, has\_left()​, clock\_tick()​, status()​) after being prompted: "What should I do?"​. We have provided most of this functionality already for you, but you need take care of the subclasses of Pet​.
3. ​​*Poodles* have additionally the method dance​, which should be available in the menu if your pet is a poodle.
4. ​​If "wait"​ is provided in the console, then "Nothing to do..."​ is printed. Dogs can not wait, so instead an error message ("Please provide a valid command."​) is displayed. Afterwards a clock\_tick()​ is executed.
5. ​​Each time after the player has typed a command, clock\_tick()​ is executed. We have implemented this already for you.
6. ​​If a pet is bored or hungry (i.e. hunger​ great than max\_hunger​ or boredom​ greater than max\_boredom​) for 3 consecutive turns, then it leaves (i.e. has\_left()​ returns True​) and the program ends. We have implemented this already for you.

​​

​​Play-test your game as you develop to make sure that it works properly, and that it handles invalid inputs gracefully (i.e., the program does not crash).

​​

​​Extra Credit #1 (2 points)

1. ​​Create a new file called **hw2-ec1.py.**
2. ​​Add new instance variable: age
3. ​​A pet will initially be of age 0 and live up to 18  years (int)
4. ​​The age of pets increments by 1 for every clock\_tick
5. ​​*Dog* - increment of 2 for every clock\_tick
6. ​​*Cat* - increment of 3 for every clock\_tick
7. ​​*Poodle* - increment of 2.5 for every clock\_tick
8. ​​Once either a cat, poodle or a dog gets older than 18 years, they will leave and the program terminates.
9. ​​The message returned by the status()​ method now also contains the age of the pet.
10. ​​Players can adopt multiple pets now. Extend the menu to include the following commands:
11. ​​adopt - The player adds an additional pet to game in the same way as when the program was started. The new pet must get a new unique name (i.e. a name no other pet in the game has). The player is prompted for the name until they provide a valid one. Afterwards the standard menu is available again, but commands will be directed at the newly created pet.
12. ​​list - The names of all pets are listed. Each name in a separate line.
13. ​​choose - The names of all pets are listed. The player is then asked to provide a valid name. Afterwards the standard menu is available again, but commands will be directed at the pet with the name just provided by the player.
14. ​​Time keeps moving forward for all pets. This means clock\_tick()​ should be issued for all pets currently part of the game. If a pet leaves, you should print the name of the pet and that it “has left.”. If the currently active pet (the one that is receptive for commands at the moment) leaves, the player has to choose either another pet or adopt a new one. Pets that leave are no longer part of the game, so they should not be displayed in the list​ and available in the choose​ command.
15. ​​When the game ends (i.e. no pets are part of the game), it should provide a player two options to make: play it again​ or quit​.

​​

​​Extra Credit #2 (2 points)

1. ​​Create a new file called **hw2-ec2.py.**
2. ​​Implement a world for the pets to live in:
3. ​​When the program starts, create an array representing a matrix (similar to the board in HW 2) of size 8x8. This matrix or board is our world.
4. ​​Then the player is asked how many cats and dogs should be added to the world. They are then randomly placed in different cells on the board. Reuse the “display\_board” function from HW 2 to display the dogs as O’s and the cats as X’s on the board. Don’t display cell numbers. Cells without a pet should be left blank.
5. ​​The player can press enter to update the board, which means clock\_tick() is called for each pet on the board. Pets may leave the world now due to age (hunger and boredom are no longer supported). Then the pets move in a randomly chosen direction - dogs 1 step and cats randomly either 1 or 2 steps. If a pet would move on a cell with another pet on it, then it does not move this round. If a pet would move beyond the board, it instead does not move this round.
6. ​​If after the movement phase, the majority + 1 of the neighboring cells (including diagonal) of a cat are dogs, then the cat leaves the board. For a dog to leave, the cats need to be in the majority + 2. If a pet is next to a wall or in a corner, then it has less than 8 neighbors since the board is bounded by walls.
7. ​​If after the movement phase, the majority + 1 of the neighboring cells (including diagonal) of a cat are cats, then a new cat is placed on a free neighboring cell. For a new dog to be placed, the dogs need to be in the majority + 2. If a pet is next to a wall or in a corner, then it has less than 8 neighbors since the board is bounded by walls.
8. ​​When all pets have left, then the game ends.

​​

​​Grading

​​Main Assignment

​​Refer to [sample output](https://docs.google.com/document/d/197R7UyHSyVsQOVrcJhvwPIvRXkB3NwZi19BpwGp_s1E/edit?usp=sharing) for specific examples of how to meet requirements.

|  |  |  |  |
| --- | --- | --- | --- |
| ​​Req | ​​Description | ​​Category | ​​Point Value |
| ​​1 | ​​The program prompts users to chose a type of pet and does not crash after an input has been provided. | ​​Behavior | ​​0 |
| ​​2 | ​​The correct prompts are displayed for each type of pet during the adoption process, i.e. players are asked to provide name, sounds and possibly meow\_count. | ​​Behavior | ​​5 |
| ​​3 | ​​The speak()​ , clock\_tick()​ and feed()​ methods behave according to the description in the Pet class. | ​​Code | ​​5 |
| ​​4 | ​​All classes were defined and the class structure is correct, i.e. Dog and Cat are subclasses of Pet and Poodle is a subclass of Dog. | ​​Code | ​​5 |
| ​​5 | ​​The feed​ command behaves according to the description in the cat, dog and poodle class. | ​​Behavior | ​​5 |
| ​​6 | ​​The speak​ command behaves according to the description in the cat, dog and poodle class and inheritance+overriding were used to achieve this. | ​​Behavior | ​​5 |
| ​​7 | ​​The game menu can be used to issue all available commands for the respective kinds of pets and it has been adapted so that poodles can dance and dogs can not wait utilizing the class structure and inheritance. | ​​Behavior | ​​5 |
| ​​8 | ​​It is possible to create all three kinds of pets und run various commands until they leave without the program crashing. | ​​Behavior | ​​5 |
| ​​9 | ​​The Poodle​ class is implemented to match functionality specified in in the description above. | ​​Code | ​​5 |
| ​​10 | ​​The Dog​ class is implemented to match functionality specified in in the description above. | ​​Code | ​​10 |
| ​​11 | ​​The Cat​ class is implemented to match functionality specified in in the description above. | ​​Code | ​​10 |
| ​​12 | ​​The play()​ function has been implemented to match the functionality described in Part 1 of the assignment description. | ​​Code | ​​10 |
| ​​13 | ​​Inheritance and overriding was utilized to extend the play()​ function so that Cats have 5 attempts and the correct ASCII images are displayed. Note: you should *not* override play() in subclasses as this would result in unnecessary duplication of code. You should instead modify the behavior by overriding one or more attributes that are used by play(). | ​​Code | ​​5 |
| ​​14 | ​​The Pet​ class is implemented to match functionality specified in in the description above. | ​​Code | ​​10 |
| ​​15 | ​​docstring is correctly added to classes and methods | ​​Code | ​​5 |
| ​​16 | ​​Code style is good (variable and function names, if added, are clear; code layout, indentation, whitespace, etc. complies with [507 Assignment Guide](https://paper.dropbox.com/doc/507-Assignment-Guide--AshR98EaGu3X1IlUreueQi4fAQ-RwRuP1S6RzY21Z04PhJDr)) | ​​Code | ​​10 |
| ​​ | ​​**Total** | ​​ | ​​**100** |

​​

​​

​​**We will release the rubric later.**

​​Extra Credit #1

​​We are not providing sample output, so you are encouraged to exercise reasonable judgment in following the instructions above to meet the requirements listed here.

|  |  |  |  |
| --- | --- | --- | --- |
| ​​Req | ​​Description | ​​Category | ​​Point Value |
| ​​1 | ​​Pets can age and leave after they are older than 18 years. The status message now also reports the age. | ​​Behavior | ​​1 |
| ​​2 | ​​Players can adopt multiple pets now and the new commands behave according to the assignment description. | ​​Behavior | ​​0.5 |
| ​​3 | ​​The clock ticks should be run for all pets and the behaviour for leaving adatpted according to the assignment description. Players are also now able to either quit or play again. | ​​Behavior | ​​0.5 |
| ​​ | ​​**Total** | ​​ | ​​**2** |

​​

​​Extra Credit #2

​​We are not providing sample output, so you are encouraged to exercise reasonable judgment in following the instructions above to meet the requirements listed here. For EC2 credit, the game must at least meet requirement 1 and run without crashing. You can get EC2 credit without getting full credit for EC1.

|  |  |  |  |
| --- | --- | --- | --- |
| ​​Req | ​​Description | ​​Category | ​​Point Value |
| ​​1 | ​​Program prompts players for the amount of cats and dogs and displays the board with the pets accordingly. The clock ticks have been adapted to only account for age and the pets move randomly on the board according to the assignment description. | ​​Behavior | ​​1 |
| ​​2 | ​​The rules for adding and removing dogs and cats from the board has been implemented according to the assignment description (4. and 5.). The game ends when all pets have left. (Note: There are certain configurations of cat and dog placements on the board that result basically in endless games. This is not a problem for us this time.) | ​​Behavior | ​​1 |
| ​​ | ​​**Total** | ​​ | ​​**2** |

​​How to submit

​​Solutions to each component of the assignment must be **submitted to Canvas** as separate files with the following names. Failure to follow the naming convention will result in no credit for the offending component.

​​

​​Main Assignment: **hw3-tamagotchi.py**

​​Extra Credit #1: **hw3-ec1.py**

​​Extra Credit #2: **hw3-ec2.py**

​​

​​Assignments submitted after the deadline will be assessed a late penalty as stipulated in the syllabus.

​​