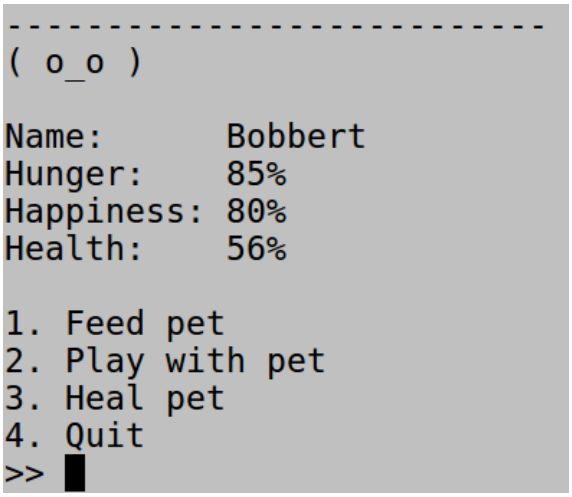
Introduction

For this first assignment, we will make a simple “virtual pet” game. The pet will have three stats: health, hunger, and happiness, and those stats will be adjusted based on the interactions that the player chooses.



You will be working with two source files, *main.cpp* and *pa_virtual_pet.hpp*. The header file (.hpp file) will contain functions that deal with the pet, while main.cpp will just contain the main() function and game loop.

Make sure that you’ve downloaded the starter files off Desire2Learn!

---

pa_virtual_pet.hpp

DisplayPetStats

Parameter name	Data type	Description
petName	string	The name of the pet
hungerPercent	int	The hunger amount, between 0 - 100%
happinessPercent	int	The happiness amount, between 0 - 100%
healthPercent	int	The health amount, between 0 - 100%

Displays all the stats of the pet, with labels to make it user-friendly.

DisplayMainMenu

Displays the main menu options:
1. Feed pet, 2. Play with pet, 3. Heal pet, 4. Quit

DrawPet

Parameter name	Data type	Description
healthPercent	int	The current health of the pet

Draws different faces depending on the health of the pet. You can make up your own, or use mine.

75% and above	(^_^)
50 – 75%	(o_o)
25 – 50%	(._.)
Below 25%	(x_x)

GetChoice

Parameter name	Data type	Description
min	int	The minimum value of the input
max	int	The maximum value of the input

Return: *int*, the validated input

This is responsible for getting the user's input via *cin* and validating the input. If the user's input is invalid, it will continuously ask the user to re-enter their selection until the user finally enters something valid.

This function should contain a **while loop** that continues looping until we get valid input from the user.

UpdatePet

Parameter name	Data type	Description
hungerPercent	int& (reference)	The hunger amount, between 0 - 100%
happinessPercent	int& (reference)	The happiness amount, between 0 - 100%
healthPercent	int& (reference)	The health amount, between 0 - 100%

This function is called every game-cycle. Each cycle, the pet's hunger goes up. Depending on how hungry the pet it, it also affects the health/happiness stats.

- Each cycle: Hunger + 5
- If Hunger is above 75: Health - 10, Happines - 10
- If Hunger is between 50 and 75: Health - 5, Happines - 5
- If Hunger is between 25 and 50: Health - 2, Happines - 2

Make sure to fix any percentages;

- If hunger is above 100, set it to 100.
- If happiness is below 0,then set it to 0.
- If health is below 0, then set it to 0.

Feed

Parameter name	Data type	Description
hungerPercent	int& (reference)	The hunger amount, between 0 - 100%

This updates the hunger percent; subtract 8 units from hunger. If hunger is below 0 afterward, then set it to 0.

Play

Parameter name	Data type	Description
happinessPercent	int& (reference)	The happiness amount, between 0 - 100%

This updates the happiness percent; adds 5 to happiness. If happiness is above 100 afterward, then set it to 100.

Heal

Parameter name	Data type	Description
healthPercent	int& (reference)	The health amount, between 0 - 100%

This updates the health percent; add 5 units to health. If health is above 100 afterward, then set it to 100.

---

main.cpp

Within your **main()** function, do the following:

1. Create four variables:
   a) `petName`, a string
   b) `hungerPercent`, an integer
   c) `happinessPercent`, an integer
   d) `healthPercent`, an integer
2. Initialize `hungerPercent` to 0, `happinessPercent` to 100, and `healthPercent` to 100.

3. Display a message asking the user to enter the pet's name.
4. Get the user's input and store it in the `petName` variable.

5. Create a boolean variable called `quit`, and initialize it to false.
6. Create a while loop. It will continue looping while `quit` is false:
   a) Call the `UpdatePet` function and pass in the appropriate arguments.
   b) Call the `DrawPet` function and pass in the appropriate arguments.
   c) Call the `DisplayPetStats` function and pass in the appropriate arguments.

   d) Check to see if `healthPercent` is 0. If it is 0, then display a message that your pet died (☹) and exit the loop with a `break` command.

   e) Call the `DisplayMainMenu` function.
   f) Declare an integer variable called `choice`.
   g) Call the `GetChoice` function, with minimum as 1 and maximum as 4, and store the result in the `choice` variable.

   h) Create a set of if / else if statements, or a switch statement to decide how to handle the user's input:

1. If the user's choice was 1, then call the `Feed` function.
2. If the user's choice was 2, then call the `Play` function.
3. If the user's choice was 3, then call the `Heal` function.
4. If the user's choice was 4, then set `quit` to true.

7. Once the while loop is over, `return 0;` should be called.

Grading Breakdown

Breakdown				
Score				
Item	**Score (0-5)**	**Task weight**	**Weighted score**	**Notes**
Creating the pet stat variables	5	5.00%	**5.00%**	
Creating a game loop in main()	5	10.00%	**10.00%**	
Calling the Display and Update functions properly	5	10.00%	**10.00%**	
Getting the user's choice and handling it with a branch	5	10.00%	**10.00%**	
DisplayPetStats	5	5.00%	**5.00%**	
DisplayMainMenu	5	5.00%	**5.00%**	
DrawPet	5	5.00%	**5.00%**	
GetChoice	5	20.00%	**20.00%**	
UpdatePet	5	15.00%	**15.00%**	
Feed	5	5.00%	**5.00%**	
Play	5	5.00%	**5.00%**	
Heal	5	5.00%	**5.00%**	
Score totals		100.00%	**100.00%**	
Penalties				
Item	**Score (0-4)**	**Max penalty**	**Weighted penalty**	**Notes**
Syntax errors (doesn't build)	0	-50.00%	0.00%	
Logic errors	0	-10.00%	0.00%	
Run-time errors	0	-10.00%	0.00%	
Memory errors (leaks, bad memory access)	0	-10.00%	0.00%	
Ugly code (bad indentation, no whitespacing)	0	-5.00%	0.00%	
Ugly UI (no whitespacing, no prompts, hard to use)	0	-5.00%	0.00%	
Not citing code from other sources	0	-100.00%	0.00%	
Penalty totals			**0.00%**	
Totals		**Final score:**	**100.00%**	