1 SDL Project Short Intro

The overall goal of the project is to create a *game* in which multiple species of animals exist and proliferate. For the final version, we also want to add a controllable character. For instance one could go for sheep, wolves, shepherd dogs and a shepherd as playable character. These animals will then move around, interact with one another and also be influenced by the controlled character. For visualization we will use the SDL2 libreary, see `https://www.libsdl.org`.

2 SDL Project Part 2

The goal of the second part of the project is to have an actually playable game with a controllable character.

The following characters should exist in your game:

- sheep
- wolf
- shepherd dog
- shepherd.

2.1 Side characters

The side characters correspond to the sheep, wolves and shepherd dogs. These identities are not controlled by the player and should behave in the following way:

Sheep

Sheep move randomly around the map except when getting too close to a wolf. In this case, they get a temporary speed boost in the opposite direction of the wolf. When two sheep meet they can produce an offspring (a new sheep).

BONUS: They only produce an offspring under the following preconditions:

1. They are of different sex
2. The female has not created an offspring for some time

Note the bonus is only given if the code layout is "proper". That is the interactions remain generic, and rely only on the stored properties.

Wolves

Wolves will actively hunt for sheep. That is they will direct themselves towards the nearest sheep. If a wolf does not catch a sheep after a certain period, it will starve and die.

Wolves fear shepherd dogs. Therefore they will try to keep a minimal distance to the closest dog, this is of higher importance to them than hunting.

Wolves do not produce offsprings.

Shepherd dogs

Shepherd should circle around the shepherd, wherever he goes.

BONUS (hard): Make them ”clickable” so that they can receive orders. That is if first a dog is clicked and then some position on the screen, the dog should go to this position and then automatically return to circling around the shepherd.

2.2 Playable character

The shepherd corresponds to the playable character. The player should be able to move him on the map using either Q,Z,S and D or the arrows on the keyboard.

2.3 Gameplay

The game should end after a predefined number of seconds. The score of the game corresponds to the average number of sheep on the field.

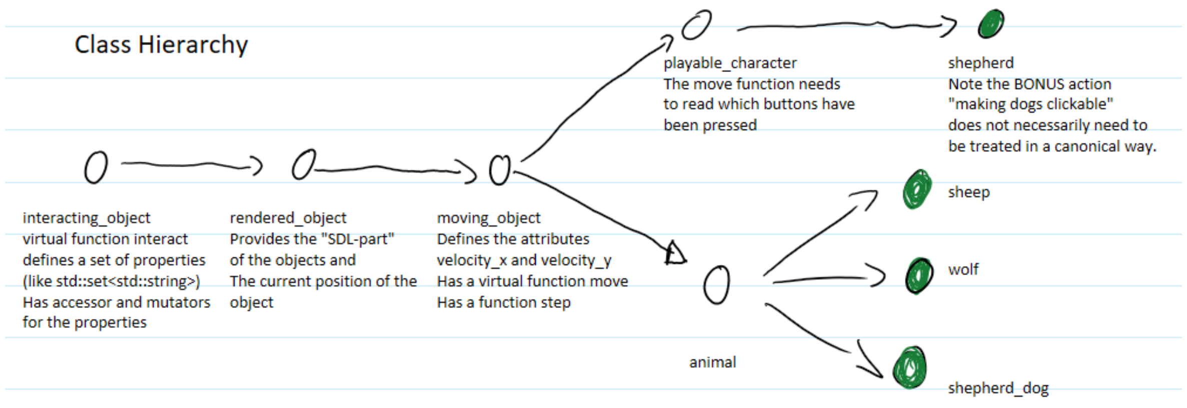
2.4 Code layout

We now have several different characters in our game which can even interact. In order to have “clean” code at the end, you should first reflect on the data structures needed, how we can create useful class hierarchies and how to construct the interaction.

Moreover, when writing your code you should also think about those who might want to reuse/extend it later on. You should therefore seek to conceive your classes such that it is “easy” for someone else to integrate new animals or a multiplayer version later on. This corresponds to the “Open/Closed-principle” (look it up!) of object oriented programming.

Proposed layout

I will “detail” an example layout here. You do not need to use it if you dislike, but if you use your own, I want you to come up with something “clean” and a clearly recognizable structure. “It works” is not a reason for choosing a certain approach or layout.


Class Hierarchy
playable_character
The move function needs to read which buttons have been pressed
shepherd
Note the BONUS action "making dogs clickable" does not necessarily need to be treated in a canonical way.
sheep
interacting_object
virtual function interact
defines a set of properties (like std::set<std::string>)
Has accessor and mutators for the properties
rendered_object
Provides the "SDL-part" of the objects and
The current position of the object
moving_object
Defines the attributes velocity_x and velocity_y
Has a virtual function move
Has a function step
wolf
animal
shepherd_dog

The class “animal” and all of its derived classes change in two ways. First, the final classes (shown in green), need to implement the *interact* function (inherited from interacting_object).

This function should only rely on the properties of the two interacting instances, so do not try to *dynamic_cast* them into some other type or use type_info etc. to gain additional information.

For instance, you can use the properties {"sheep", "male", "prey", "alive", *eventually others*} for sheep. This way you can have generic interactions:

A wolf will hunt everything that is tagged with "prey", no need to only hunt sheep, maybe someone wants to extend your game later on.

Proposed new version for "update" of ground

Make all animals interact with each other. This will create new animals, tag some as dead etc. Erase all dead animals from the vector, then move and draw the alive animals like before.

The classes "ground" and "application" mostly remain the same. I want that "ground" only has a single vector storing all currently existing characters as (smart)-pointers to **moving_objects** or **interacting_objects**.

This allows us to have a unique way of handling all characters the same way and therefore provides flexibility for future changes.

Other important things

Modularity:
The code needs to be as modular as possible. That means in particular that the update function should not be more than (about) 20 lines.

All the logic of how animals move and behave is "hidden" in the corresponding virtual functions. This way your code is very easy to extend (by introducing a new kind of animal for example).

Passing information:
If you need more precise information in a member function (for instance, in order to flee from a wolf a sheep needs to know its position), two solutions present themselves (maybe more, but write clean code!):

1. You move the position attributes from rendered_object to interacting_object[1].
2. You add additional generic information. For instance interacting_object could also have another member: valued_propoerties with type std::set¡string, int¿ or similar.

[1]You deviate from the design given above, but you have a reason so its no problem