Object Relations

Composition & Inheritance

1. Content

2. Objective

At the end of these exercises you should be able to:

- Use inheritance to accomplish **generalization**
- Create projects with classes that cooperate with each other using composition, aggregation, association, inheritance.

We advise you to **make your own summary of topics** that are new to you.

3. Exercises

Give your **projects** the same **name** as mentioned in the title of the exercise, e.g. **InheritanceShapes**. Other names will be rejected.

Always adapt the **window title**: it should mention the name of the project, your name, first name and group.

Create a blank solution with name **W03** in your **1DAExx_03_name_firstname** folder

3.1. InheritanceShapes

3.1.1. Create the project

Add a new **project** with name **InheritanceShapes** to the W03 solution and **add the framework files**.

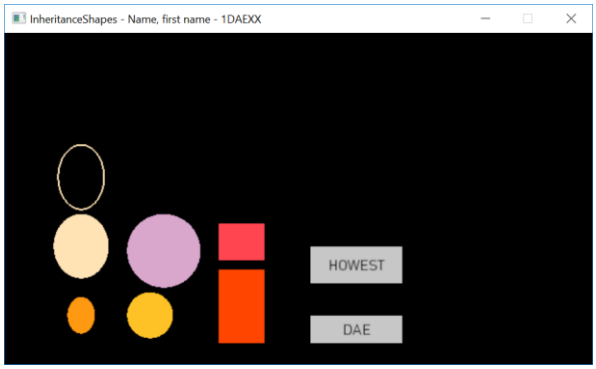
Delete the generated file **InheritanceShapes.cpp**.

Overwrite the **Game** class files by the given ones in the folder CodeAndResources/01_InheritanceShapes.

Copy and add the given files of the class definitions **DaeEllipse**, **DaeRectangle**, **DaeCircle** and **DaeRectLabel** to the project.

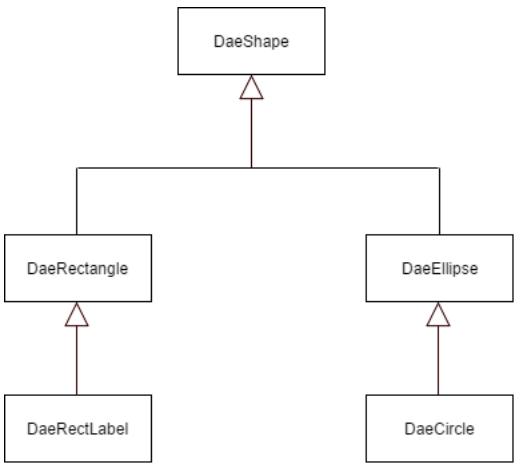
Also copy the **Resources** folder.

Build and run the application. This shouldn't give any problems. Using the arrow-keys you are able to translate the shapes.



3.1.2. Add inheritance

Have a look at these shape-classes. Notice that they have some data members and a method in common.



Add a new class **DaeShape** and make the given shape-classes inherit from it. Make sure you obtain the inheritance tree as depicted here.

Complete this class with the common data member(s) and/or method(s) and adapt the **derived classes**.

Provide the **base class** DaeShape only with **one** (not default) **constructor** that initializes these data members.

Changing the Game class is not needed.

Build, run and test.

A working solution does not necessarily mean that you have implemented inheritance correctly. Before continuing with the next exercise, **show your solution to the lecturer**, to be sure that you implemented inheritance correctly.

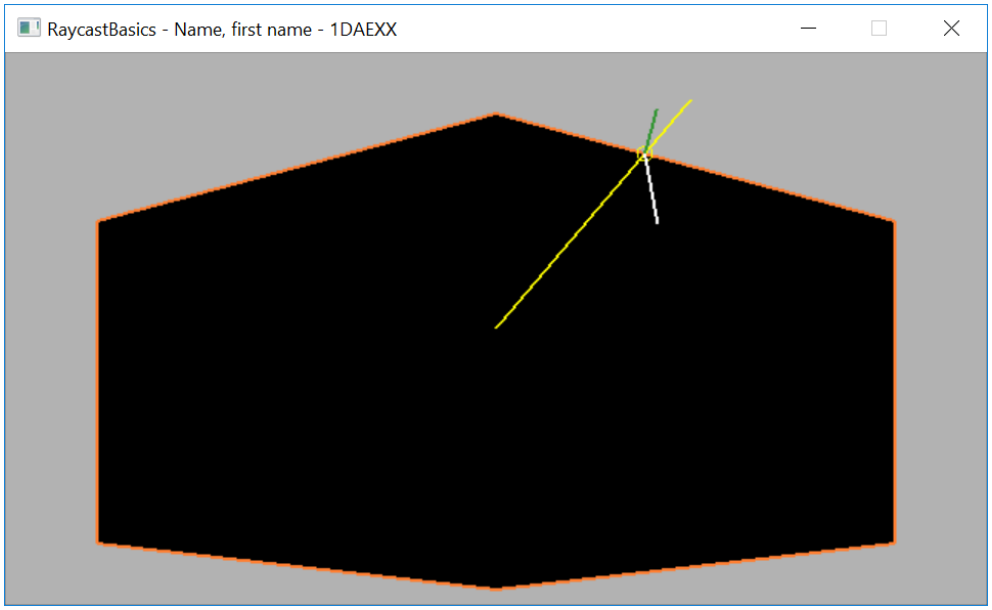
3.2. RaycastBasics

3.2.1. Create the project

Add a new project with name **RaycastBasics** to the W03 solution and **add the framework files**.

Remove and delete the generated file **RaycastBasics**.cpp.

At the end of this exercise you have a window that looks like this.



3.2.2. General

The utils namespace contains collision functionality, some of which you already used, e.g.:

- IsOverlapping in the PowerUp class;
- IsPointIn… in the Diamond class.

For more information have a look at utils.h. Most of the functions are obvious and don't require extra information except for the **Raycast** function.

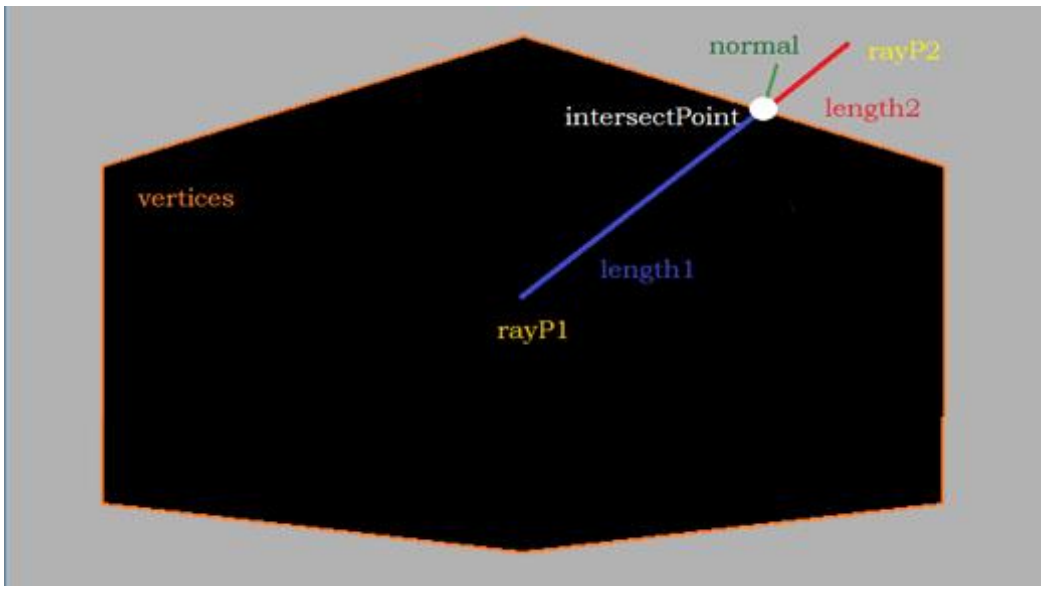
The RayCast function verifies whether a ray (indicated by the parameters *rayP1* and *rayP2*) hits a surface (parameter *vertices*). When this is the case the function returns *true* and the parameter *hitInfo* is filled with more information. It contains following members:

intersectPoint: point where the ray hits the surface

lambda: length1/ total length of ray

normal: normal on the surface segment that is hit by the ray.

The picture below illustrates this.



The lambda member allows us to calculate both lengths:

length1 = length of the ray * lambda.

length2 = Length of the ray * (1 – lambda).

Knowing this, make the following exercise.

3.2.3. Use Raycast to determine reflection on a surface

a. The surface

Define a vector container of Point2f type elements and fill it with the vertices of the surface.

Draw the surface. We've drawn it as a filled black polygon with an orange border.

b. The ray

Define 2 variables of type Point2f, they contain the end point values of the ray. The start point doesn't change and is located in the center of the window. The end point follows the mouse position.

Draw the ray in yellow.

c. Raycast

Call the Raycast function and when there is a hit with the surface then:

- Draw a small circle with a center that corresponds with the intersection point.
- Draw the **normal** in green. Give it a length of e.g. 30 pixels. Let the line start at the intersection point
- Draw the **reflection** of the ray on the surface in white.
  - You can use the **Reflect** function of Vector2f
  - Give the reflection the same length as the length of the part of the ray after the intersection point (length2).

3.3. MiniGame

Now that you know how to perform a raycast, we will now use it in the MiniGame.

3.3.1. MiniGame Part 1

Go to the MiniGame assignment and implement part 3.3

4. Game project

By now you should have made a choice and have had approval.:

- Start working on the sprite sheets, align them on to a grid. Use the provided spritesheet tester.

5. Submission instructions

You have to upload the compressed folder *1DAExx_03_name_firstname.* This folder contains 2 solution folders:

- W03
- Name_firstname_GameName.

Don't forget to clean the solution before closing Visual Studio. Remove the x64 folder and hidden .vs

6. References