Ankara University
COM102B – Spring 2016-17

PA#4: The Intersection of Shapes

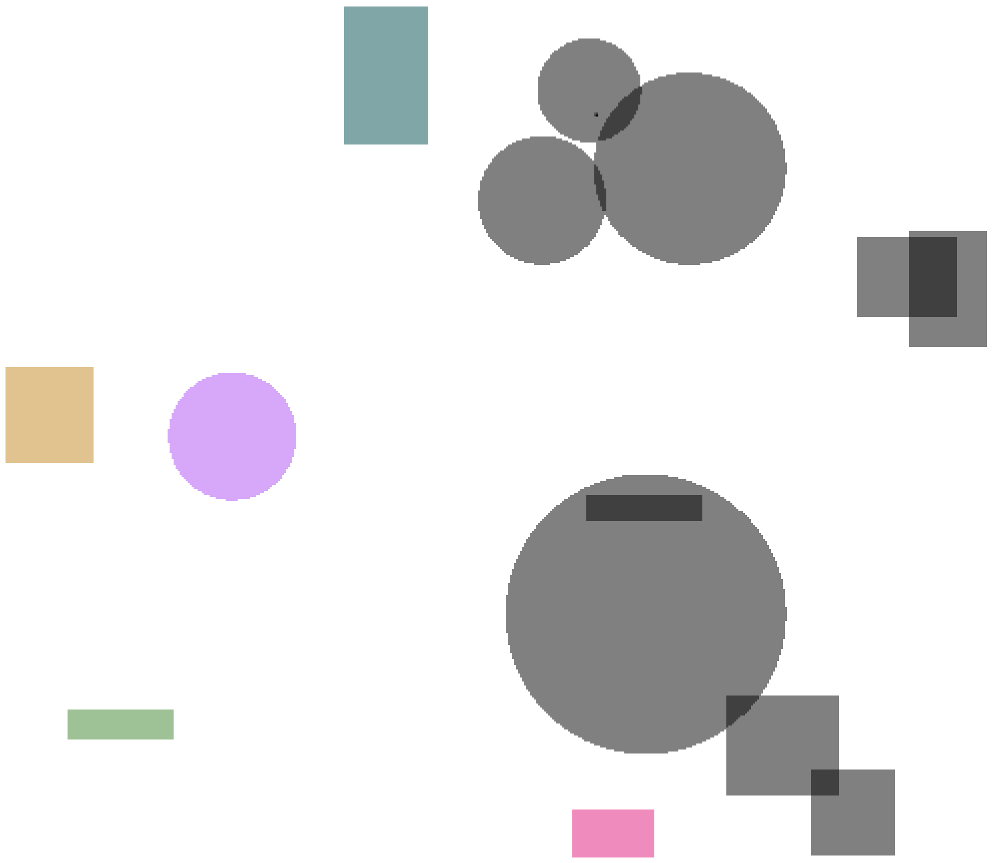


Figure 1: A sample run of the application

Task: Determining the intersection of different shape objects

Aim: Practicing with polymorphic types

In this PA, you will write the implementations of the virtual functions for the two derived classes: Rectangle and Circle. You are given a running but incomplete C++ code that depicts a set of moving shapes on a graphics window. Your job is to define the intersection function, *i.e. intersects*, for both classes. The function prototype is given in the corresponding class header and CPP files. In its default implementation, the function simply returns false, meaning that the object does not intersect with any shapes. Therefore, at the beginning of the application the moving objects are in color and escape from the viewport. You need to implement the virtual intersects function for Rect and Circ classes so that all the shape objects are preserved within the viewport and intersecting objects are colored gray as shown in Figure 1.

In this context, you are provided 3 header files: Shapes.h, Rectangle.h, Circle.h; 3 cpp files: ShapesMain.cpp, Rectangle.cpp, Circle.cpp. Please work these files carefully to understand the structure of the implementation. There are informative comments for you in the ShapesMain.cpp file. You can change some of the parameters in that file freely while developing your algorithm. The graphics interfaces for the Rectangle and Circle shapes are implemented for you. Do not modify these functions; your implementation will be compiled with the original versions of the files.

ShapesMain.cpp file contains some parameters to enable you to test your applications during development. For example, if you do not want the shapes move while testing your intersects function, simple uncomment the define macro in ShapesMain.cpp file:

```
// UNCOMMENT THE LINE BELOW TO STOP MOVING SHAPES
//#define NO_MOTION
```

NO_MOTION macro will direct your application to set the velocities of each shape to 0.0. If you want the shapes move faster or slower, you can press UP or DOWN arrow keys, respectively, many times to tune the velocities to a level that you like. Moreover, if you want to visualize randomly generated different shape scenarios, simply press ‘r’ to reset your shape initialization. Every time you press ‘r’, the application is initialized with a random set of rectangles and circles, with different sizes, colors and velocities.

So, your task is to implement the following two functions:

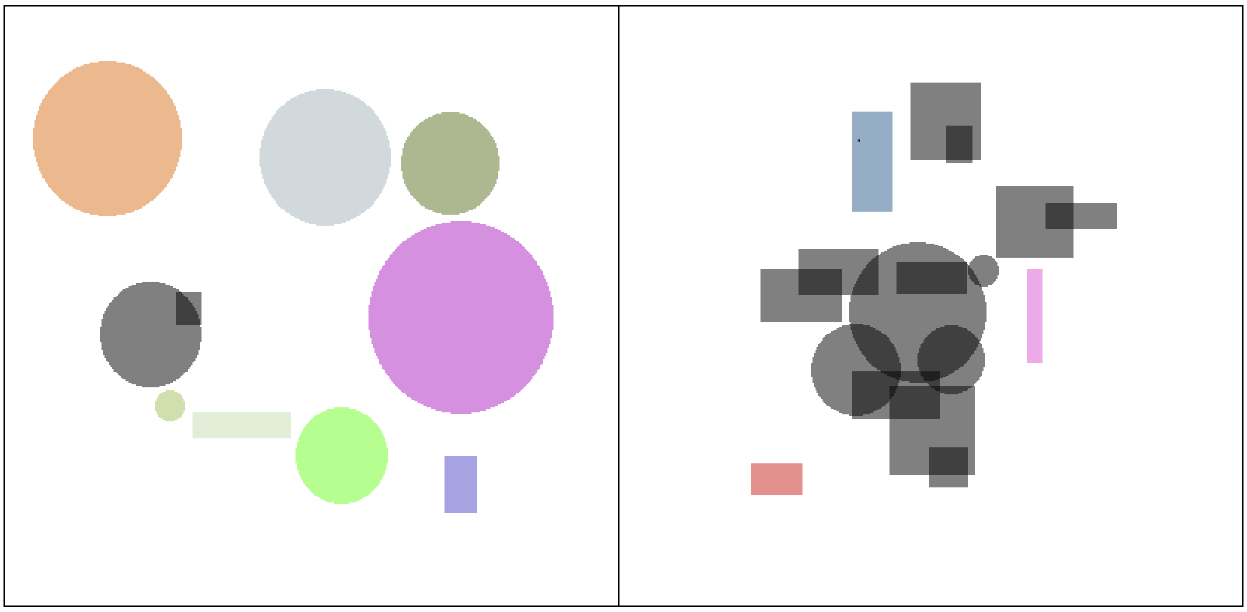
```
bool Rect::intersects(Shape* pshape)
```

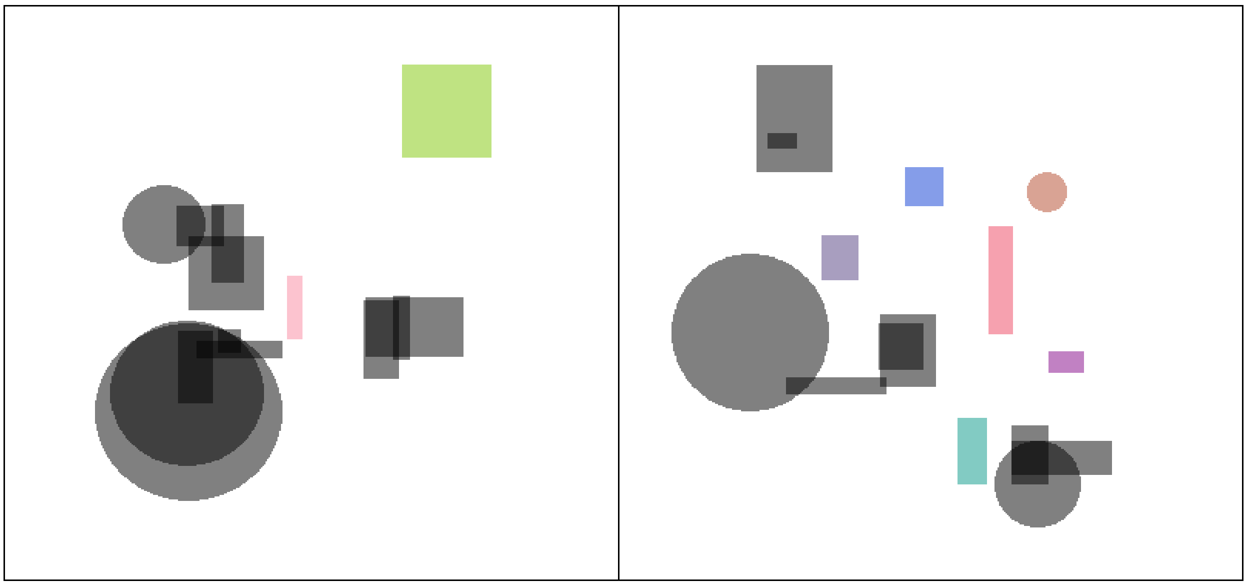
```
bool Circ::intersects(Shape* pshape)
```

As you see in the provided cpp files, these functions simply return false. You need to change these with a proper implementation.

Four exemplary cases are depicted below. As it is seen from these sample images, when shapes intersect, their color turn into gray. If you correctly implement the intersections of shapes, your application is expected to draw the shape objects in a similar fashion.

Please post any PA related questions to the Moodle news forum.





Necessary Installations:

The preparation of your environment for running graphics functions requires the same steps that you completed in PA3. Please follow the instructions below if you did not prepare your environment before.

1- Install glut packages to your linux environment. You can do this using the following command from a terminal (console)

```
sudo apt-get install freeglut3 freeglut3-dev
```

2- Then, update your packages using:

```
sudo apt-get update
```

if your Ubuntu version is higher than 11.10, this update will install the true versions of the glut library.

3- Test your installations by writing this simple code in a test.cpp file:

```
#include <GL/glut.h>
//Drawing funciton
void draw(void)
{   //Background color
  glClearColor(0,1,0,1);
  glClear(GL_COLOR_BUFFER_BIT );
  //Draw order
  glFlush();
}
//Main program
int main(int argc, char **argv)
{
  glutInit(&argc, argv);
  //Simple buffer
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB );
  glutInitWindowPosition(50,25);
  glutInitWindowSize(500,250);
  glutCreateWindow("Green window");
  //Call to the drawing function
  glutDisplayFunc(draw);
  glutMainLoop();
  return 0;
}
```

4- Compile test.cpp file, linking the OpenGL/Glut libraries (be careful to write the g++ command with the same order as shown below):

```
>g++ -o test test.cpp -lGL -lglut
```

5-Run your test application:

```
>./test
```

6- See a green window.

Now your working environment is ready to begin.

Compilation Command for the Pong Game:

```
> g++ ShapesMain.cpp Rectangle.cpp Circle.cpp -lGL -lglut -o IntersectingShapes
```

Run your game using:

```
>./ IntersectingShapes
```

Submission: Follow the announcements for valid filename before submission.

Have fun ☺