Faculty of Computers, Informatics and Microelectronics

Technical University of Moldova

WINDOWS PROGRAMMING

Laboratory work #4

Windows Timer. Animation.

Authors:
Irina Ungureanu

Supervisor:
Irina Cojanu

Laboratory work #4

1 Laboratory Work Requirements

- **Basic Level (grade 5 - 6) you should be able to:**
  a) Create an animation based on Windows timer which involves at least 5 different drawn objects
- **Normal Level (grade 7 - 8) you should be able to:**
  a) Realize the tasks from *Basic Level*.
  b) Increase and decrease animation speed using mouse wheel/from keyboard
  c) Solve flicking problem describe in your readme/report the way you had implemented this
- **Advanced Level (grade 9 - 10) you should be able to:**
  a) Realize the tasks from *Normal Level*.
  b) Add 2 animated objects which will interact with each other. Balls that have different velocity and moving angles. They should behave based on following rules:
    1) At the begining you should have 3 balls of different colors of the same size
    2) On interaction with each other, if they are of the same class (circle, square), they should change their color and be multiplied.
    3) On interaction with the right and left wall (the margins of the window), they should be transformed into squares.
    4) On interaction with the top and bottom of the window - the figures should increase their velocity.
    5) Please, take into consideration that the user can increase and decrease animation speed using mouse wheel/from keyboard
- **for Bonus Point Tasks :**
  a) For the task above, add balls with mouse.

2 Laboratory work implementation

2.1 Tasks and Points

- Create an animation based on Windows timer which involves at least 5 different drawn objects

In order to accomplish this, I created a class called Objects where I specified the coordinates of the top left corner and the right bottom corner. It also contains the speed at which the object will move. I also created a function that will move these objects. It takes the coordinates of the corners and adds to each of them the speed specified in the class variable. Afterwards, it checks if the ball is not touching the borders of the client zone and if it does it inverses the speed variable which makes the object move in the opposite direction. Then, I created several objects of this class and set a timer in WM_CREATE. Each time the timer elapses, it will send a WM_TIMER message to the window and we need this for the animation. Then, in the WM_PAINT message I called the Move() function for the objects declared earlier which now move thanks to the timer.

```
SetTimer(hwnd, ID_TIMER, timerSpeed, NULL);
```

- Increase and decrease animation speed using mouse wheel/from keyboard

I created two functions for this: one for increasing the speed ad the other for decreasing the speed. The acceleration one adds a delta to the speed variable declared in the class which makes the object move faster. The deceleration one does the opposite, it subtracts the delta from the speed variable. I use these two functions in the WM_KEYDOWN message for processing the speed control from keyboard and in the WM_MOUSEWHEEL message in order to process the messages sent from the mouse wheel.

- Solve flicking problem

In solved this problem using double buffering. It involves doing all the drawing in memory first, and then copying the completed drawing to the screen in a single BitBlt() so that the screen is updated directly from the old image, to the complete new image with none of the individual operations visible. To do this, I created a temporary HBITMAP in memory that is the exact size of the client area.

```
hdcMem = CreateCompatibleDC(hdc);
hbmMem = CreateCompatibleBitmap(hdc, rect.right, rect.bottom);
hOld = SelectObject(hdcMem,hbmMem);
```

Now that I have a place to draw to in memory, all of the drawing operations use hdcBuffer instead of hdc and the results are stored on the bitmap in memory until the drawing is complete. Then, I copy the whole thing over to the window using BitBlt().

```
BitBlt(hdc, 0, 0, rect.right, rect.bottom, hdcMem, 0, 0, SRCCOPY);
```

- Add 2 animated objects which will interact with each other. Balls that have different velocity and moving angles.

In order to see if the objects interact, I created a function where I calculate the center point of the shape according to its position in space. Then, I calculate the distance to the center of the other shape using a mathematical formula. If this distance is smaller than the sum of the radii of the shapes than I call the function changeColor() which randomly changes the color of the object. Then, I also create a new object and give it a random speed which gives the illusion f shape multiplication.

Afterwards, I check if the objects collide with the left or right wall and if they do I set a flag to 1. This flag will be processed in the Move() function and if it is set to 1 it will draw a rectangle instead of a circle. If the objects touch the top or bottom walls of the window, I just call the accelerate() function which will increase their speed.

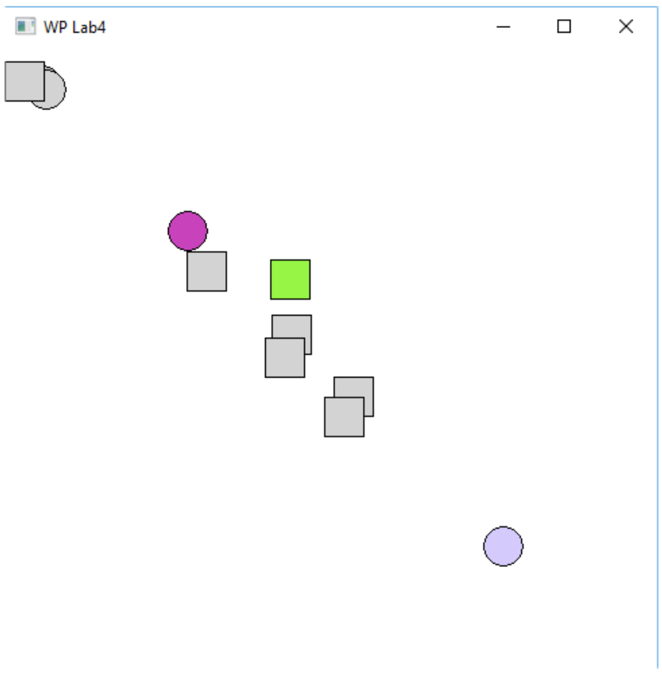
- For the task above, add balls with mouse.

For this, I firstly created a global array of type Objects. Then, each time the left mouse button is clicked (processed in the WM_LBUTTONDOWN message) I add to this array a new object. Then, this array is parsed in the WM_PAINT message where all the elements are animated using the same Move() function. Here, I also check them for interactions.

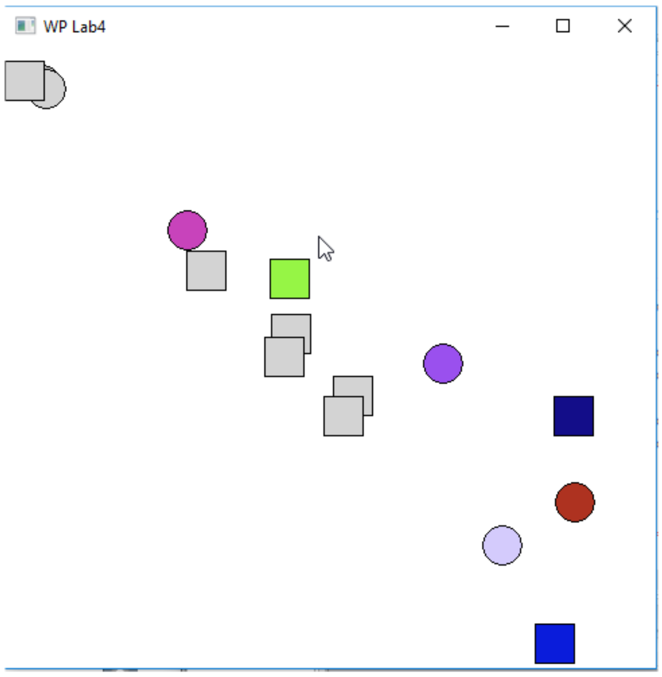
2.2 Laboratory work analysis

https://github.com/taurrielle/WP

2.3 Prove your work with screens



The basic window



Adding shapes with mouse click

Conclusions

During this laboratory work, I learned about how to work with the windows timer and how to make basic animation in Win32 API. In my application, I initially have 3 balls that move inside the client zone and bounce off the edges of the window. If the balls intersect, they change their color and multiply. If they touch the side edges, they transform into rectangles and if they touch the top and the bottom edges, they increase their velocity. This increase in velocity does not affect the user, since he can always decrease it using the mouse wheel or the UP and DOWN keys. All in all, this laboratory work provided a better understanding of what is a timer and what it is used for and gave a little insight in basic animations.

References

1 Animation, `http://www.winprog.org/tutorial/animation.html`

2 Charles Petzold, *Programming Windows, 5th Edition*, 1998