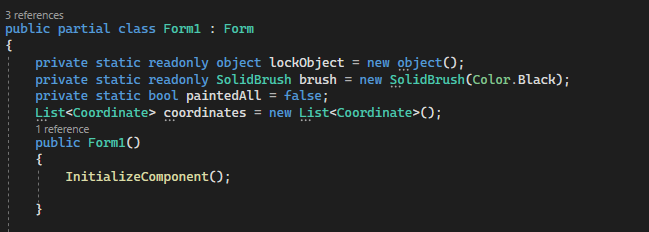
CIS 6007 Parallel Programming and Distributed Systems Section B

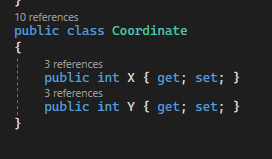
Artem Meshcheryakov SE3G9, 2117065, St20283944

## **Program overview and structure**

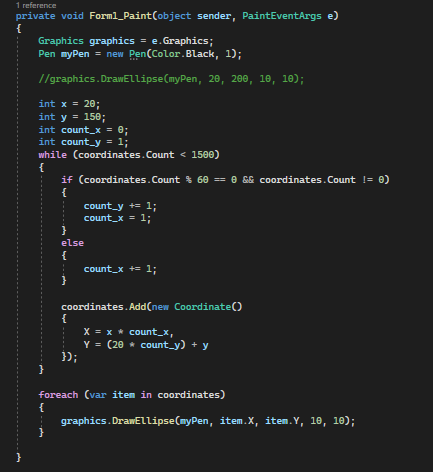
The program is a windows form application based on a C# because visualization is required. That is why all code has been written in a Form1.cs file. The static and global variables which has been used throughout the entire code:



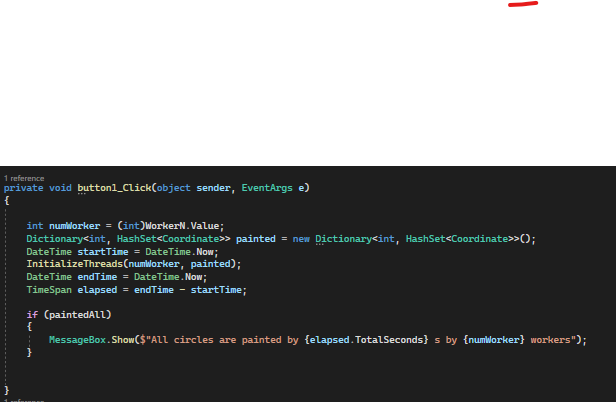
The first method which will be described is generating 1500 circles in the form in order to see how they are painted during the execution. A circle will have a 10 pixel radius hardcoded. In order to generate 1500 units the class “coordinate” has been created for storing x and y coordinates. There is a structure of the following class:



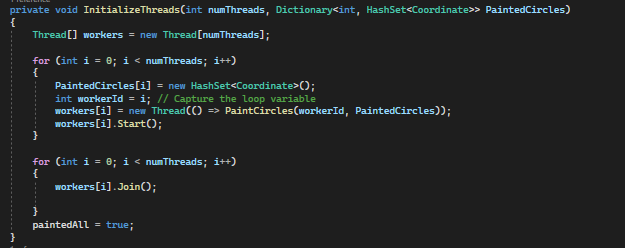
That is the reason why the program has a global list of the coordinates. Now that is the time to generate coordinates of a circle center. Starting x = 20, starting y = 150. The program will display a 60X25 matrix of the circles. In order to do that we need to change Y to the next coordinate for each 60 circles. Also we need to paint circles in a row by the formula x = x \* count\_x where x is 20 and count\_x increase every next step if x is not the 60th circle in a row. The formula for the y is the next one: y = (20 \* count\_y) + y where y is 150 and count\_y increasing every 60th circle. After generating the coordinates we can finally place the circles. Algorithm implementation:



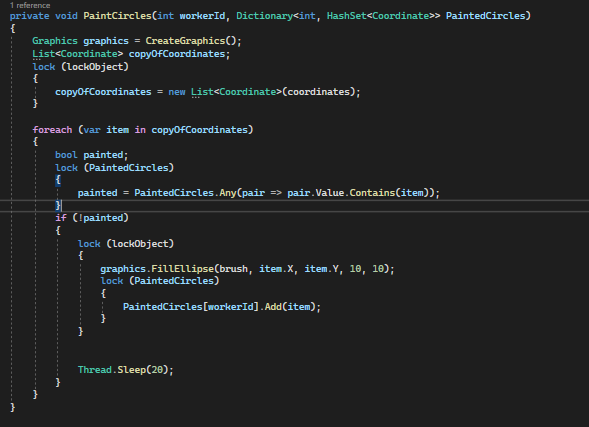
Let’s address the main function which will be executed after hitting the form.



The function took the number of the threads provided by the user input and initialized threads execution with time measurement. The dictionary is needed to keep track of the painted circles for each worker ( thread). Something like a common buffer of communicating between workers to let each of them know which circle has already been painted by which worker. Hash set has been implemented for less time consumption. The threads initialization function:



Threads are stored in the array where they are called synchronously with the same function for circle painting. When all threads have finished the job, the flag has been changed to the true and the program returns results of execution. The paint function:



The copy of the coordinates has been locked and created in order to not overlap several threads on each other. painted = PaintedCircles.Any(pair => pair.Value.Contains(item)); checks if the circle has been already painted. This piece is locked in order to not be executed by several threads. That is why the painting method and adding to the painted array has been also locked. The logic is simple: check if the circle with specific coordinates is in the array (does not matter which worker) if yes => go to the next one, if no => paint => sleep for 20 milliseconds (M).

## **Evaluation**

**Is this problem able to be parallelized?**

Since workers are painting circles on their own and the sequence in which they paint them has no bearing on the outcome, the answer to the question is yes: this issue can be parallelized.

**How would the problem be partitioned?**

The collection of circles can be divided among several worker threads to split the issue. Each worker is in charge of painting a portion of the circles.

**Are communications needed?**

Coordination of the efforts of several threads requires communication, particularly to prevent numerous workers painting the same circle twice.

**Are there any data dependencies?**

The coordinate list is the primary source of data reliance. To determine which circles to paint, employees must have access to this list.

**Are there synchronization needs?**

Yes, synchronization is required to stop different threads from editing common resources at the same time, such as the painted circle dictionary and the coordinate list. Locks are used to do this.

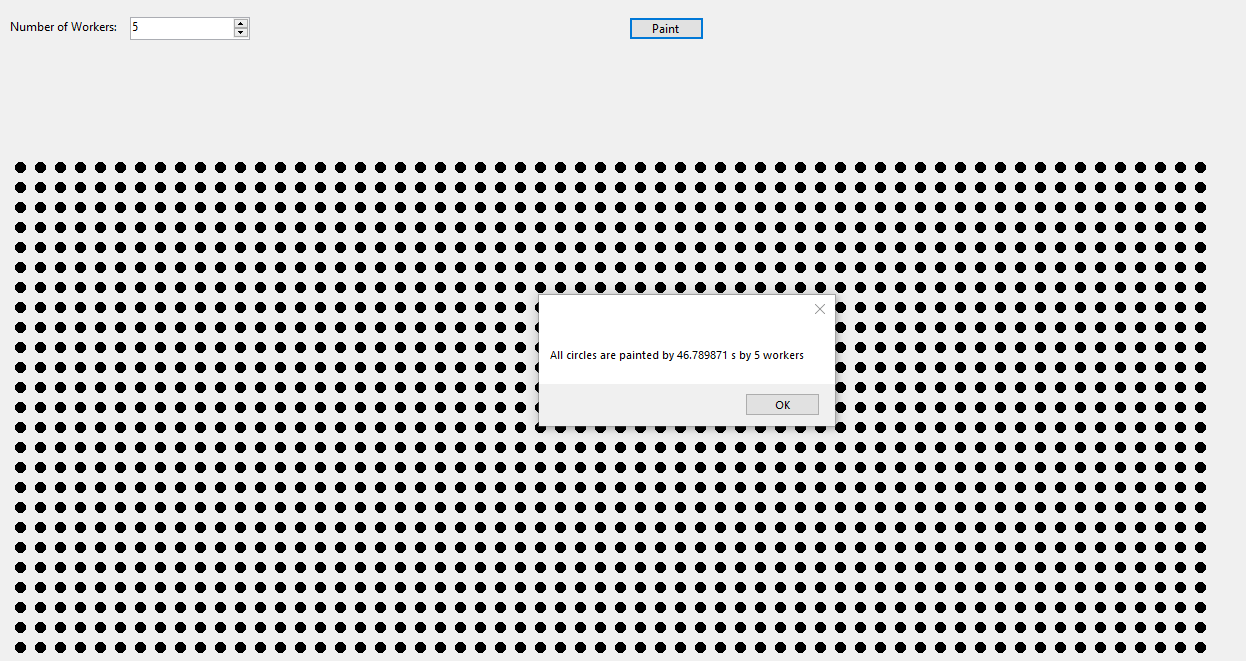
**Will load balancing be a concern?**

Depending on how the circles are distributed in the first list, load balancing may be an issue. There may be an imbalance in the workload if some employees complete their jobs ahead of others in the list if certain sections have more circles than others.Depending on how the circles are distributed in the first list, load balancing may be an issue. There may be an imbalance in the workload if some employees complete their jobs ahead of others in the list if certain sections have more circles than others.

## **Results**

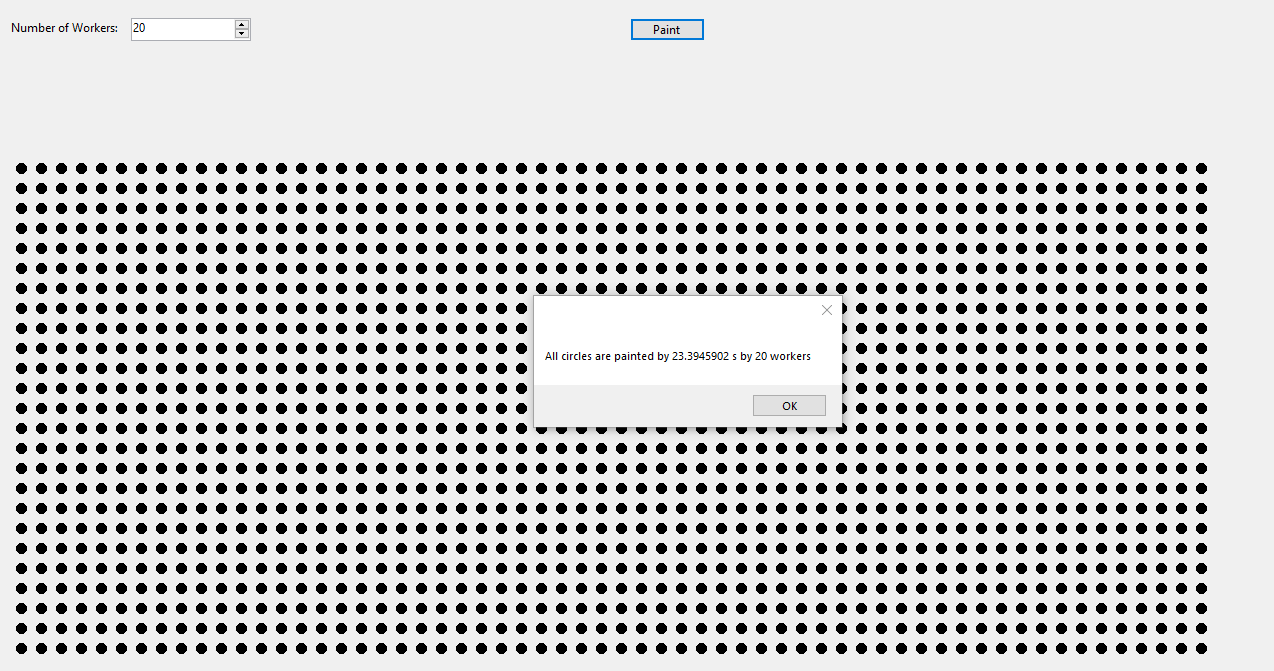
Sadly the machine was not powerful enough to execute the program for 100 threads but the progress still remains significantly different for the amount of the threads. In order to execute the program, we should open the form for the full screen. If we want to change the number of threads we should relaunch the program.

The result for K = 5:



The time is 46.798….. Seconds

The result for K = 20:



The time is: 23.394…. seconds.