

CS 201 - Data Structures and Algorithms II:

Assignment #2

Muhammad Mobeen Movania (L1),
Syeda Saleha Raza (L2),
Faisal Alvi (L3, L4),
Abdullah Zafar (L5).

Due on February 10, 2024, 11.59pm

Student 1 Name, ID

Student 2 Name, ID

Instructions

This assignment document consists of two problems.

- Problem 1 is a theoretical question which requires analysis. It should be completed and submitted within this document as a pdf on Canvas. This problem is worth 20 points.
- Problem 2 is a programming based question which requires implementation. It must be submitted by pushing all your code files to the Github repository. This problem is worth 40 points.

Problem 1

(20 points) [**Analysis**] Suppose that, instead of promoting an element from L_{i-1} into L_i based on a coin toss, we promote it with some probability p , $0 < p < 1$.

- (a) (5 points) Show that, with this modification, the expected length of a search path is at most $\frac{1}{p} \log_{\frac{1}{p}}(n)$.
- (b) (5 points) What is the value of p that minimizes the preceding expression?
- (c) (5 points) What is the expected height of the skiplist?
- (d) (5 points) What is the expected number of nodes in the skiplist?

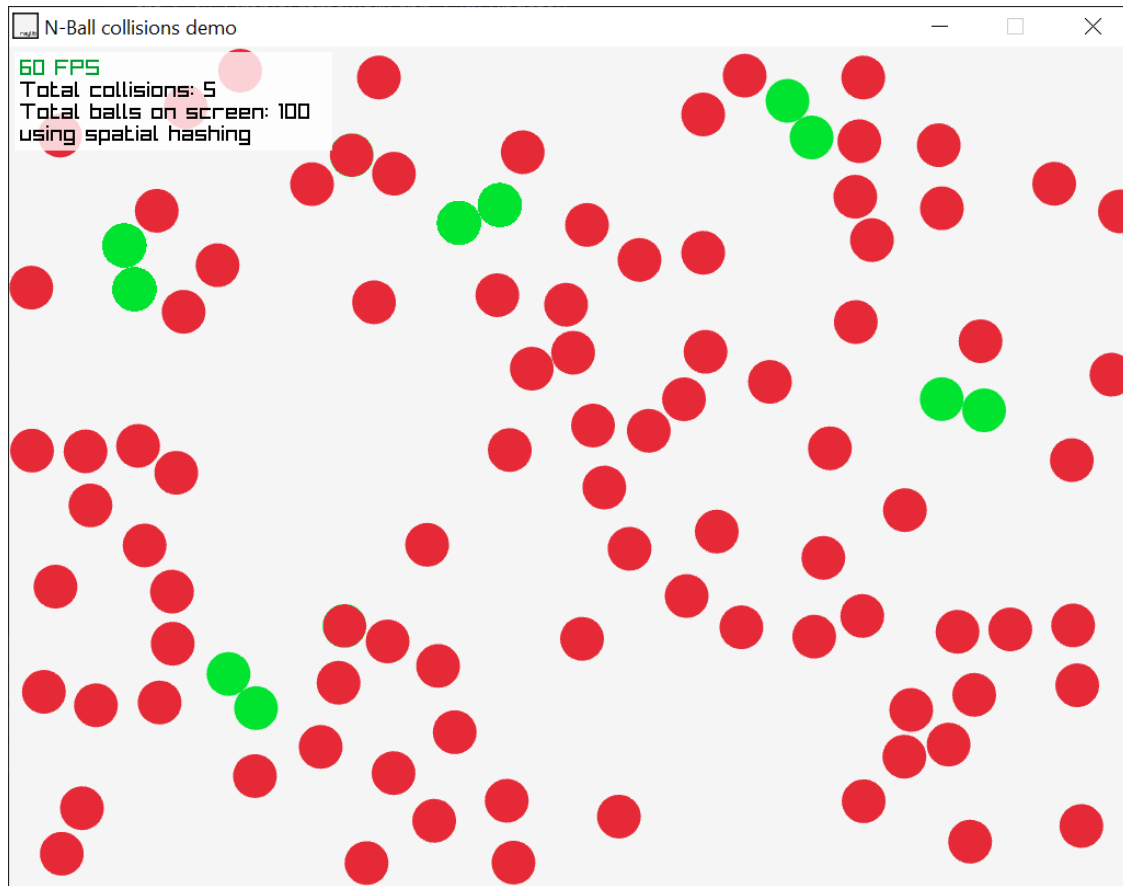


Figure 1: N-particles collision demo with 100 particles.

Problem 2

(40 points) [**Optimizing particle-particle collisions through spatial hashing.**]

You will apply the knowledge of hashing to implement spatial hashing to resolve particle-particle collisions [1] as shown in Figure 1. Template code has been shared with you which implements particle particle collisions through brute force method. However, the spatial hashing code does not implement spatial hashing for collision detection between particles. You can press 'h' key to toggle from using spatial hashing (default) to brute force method. As you would expect, the frame rate grinds down to a halt when brute force method is used.

For this assignment, you need to add collision detection in the shared template code using spatial hashing. You can get the technical overview of the method using reference [1] and a more implementation friendly description is shared in reference [2]. Relevant sections of the code are marked with TODOs so you can know where you need to add additional code.

If you have implemented spatial hashing based collision correctly, you should see collision detection and response happening at interactive framerates unlike the brute force method which makes the application unusable. Moreover, the UI should display the total collisions detected on screen which in the default template should be 0.

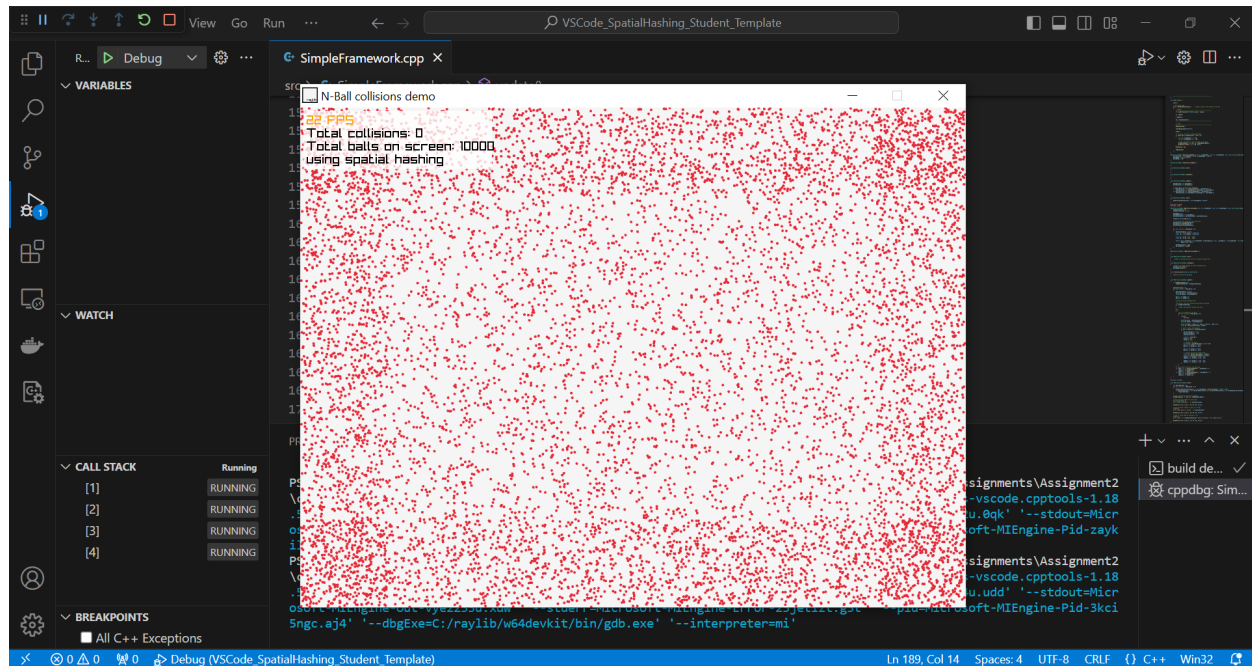


Figure 2: Running the template code.

Setting up raylib

The basic skeleton code is built on top of raylib [3]. You should be able to clone the assignment2 github repository and open the folder in VSCode. It should compile out of the box. If you want to get some details on the installation of raylib, the steps to install raylib are given for all platforms separately. For Windows OS, they are given here: <https://github.com/raysan5/raylib/wiki/Working-on-Windows>. For other OS, refer to <https://github.com/raysan5/raylib>. Note for the given code, there is no additional step required. All setup is done already for you. For those of you working on non-Windows OSes, you may refer to the raylib installation steps or get connected with one of the course TAs.

Running the template code

Go to VSCode and open the template code folder. Press F5 or go to **Run menu** and then select **Start Debugging**. If all goes well, you should see the application run as shown in Figure 2. By default, the code uses spatial hashing with boundary collisions but it does not implement particle-particle collision. The brute force method does the collision both with the boundaries as well as with all other particles. You can press 'h' key to toggle using spatial hashing to use brute force method. Note that the performance of the application will slow down considerably if you use the brute force method for collision detection.

Template code framework details

The C++ template code uses a fairly simple C++ framework. There is a base class called **Demo**. This class sets up raylib and the event loop so that you do not have to repeat the steps in your own application. There are four pure virtual functions in the Demo class which are detailed as in Listing 1.

```
1 virtual void init() = 0;
2 virtual void shutdown() = 0;
3 virtual void draw() = 0;
```

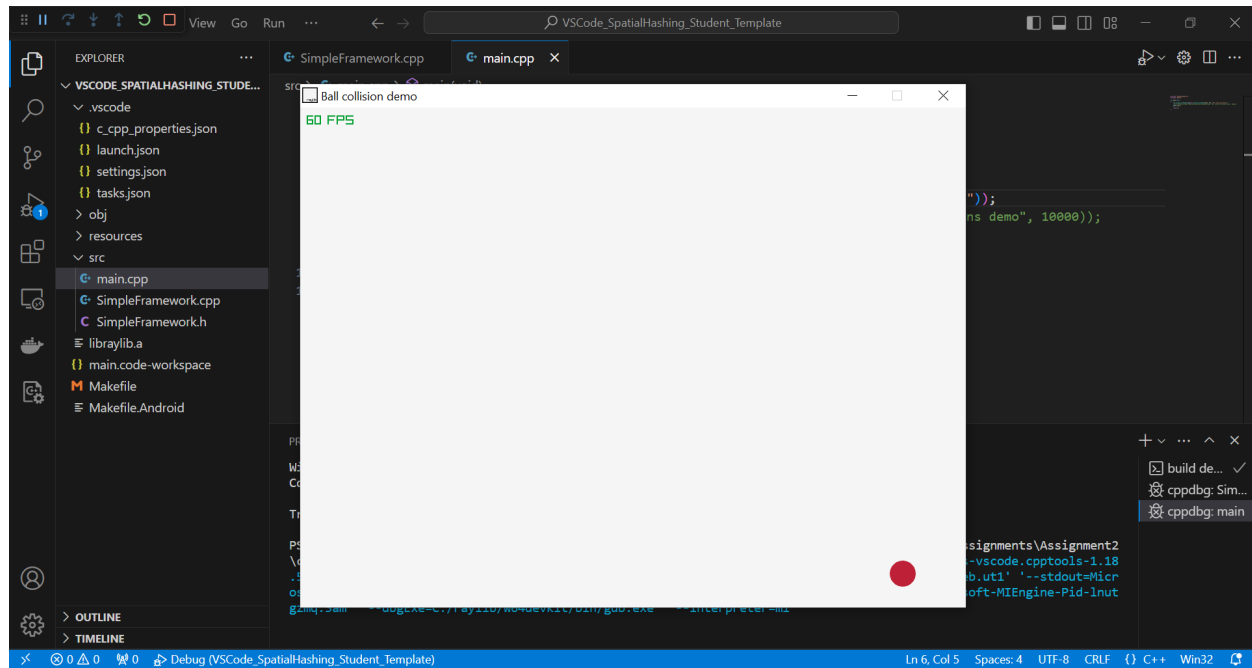


Figure 3: Simple particle demo example.

```
4 virtual void update() = 0;
```

Listing 1: Demo class pure virtual functions.

The init function is where you would do initialization of your application data structures and other variables. The shutdown method will do the de-initialization that is removing every data structure and data that you will use in your application. The draw function will be where the displaying of the objects will take place and finally, the update function is where the physics simulation will be updated.

Simple particle collision demo

Within the template framework code, we have given a simple example class inheriting from the `Demo` class called `BallCollisionDemo`. This class implements a single particle moving in the window and colliding with the screen bounds. You can see how we override the virtual methods of the `Demo` class. This is given as a simple example for you to help understand the given `Demo` framework structure. In order to run this demo, please uncomment line 6 and comment line 7 of `main.cpp` and then press 'F5' or go to **Run menu** and then **Start Debugging**. If all goes well, you should see a single particle moving in the screen and colliding with the screen bounds as shown in Figure 3. Note that the `BallCollisionDemo` is for your understanding only. You will implement spatial hashing in `NBallsCollisionDemo`.

N particles collision demo

Within the template framework code, we provide another example class inheriting from the `Demo` class called `NBallsCollisionDemo`. This class implements 10000 particles moving in the window and colliding with the screen bounds. All of the code for initialization and handling of rendering of the 10000 particles is already taken care of for you. In addition, the basic screen collision is also implemented. We provide the brute force inter particle collision code, see lines 187-228 `SimpleFramework.cpp` but the spatial hashing based optimized inter particle collision detection is missing and this is what you are required to implement in the given framework.

For your assistance, the relevant sections of the code are marked with TODOs comments so you can add relevant code there. Please note that you are free to add code any where you deem fit but please do not remove the code that is already provided. For testing, you can change the last parameter of the NBallsCollisionDemo constructor call in main.cpp to 100 to reduce the total number of particles on screen. This will help you debug the code faster. Once you are fine with the code, dont forget to change the last parameter back to 10000.

Required Tasks

You should implement a C++ class Hash that should contain all necessary data structures and functions. You should provide at least two functions: a function to repopulate the hash grid given all particle positions and another function to return the list of particles in the hash grid where the given particle position hashes to.

Rubric: Your submission will be evaluated on the following rubric. Your C++ Hash class should:

- (15 points) implement spatial hashing using appropriate data structures in the **Hash** class,
- (10 points) provide correct output for different configurations of particle count and particle radii (Note that all particles will have the same radii but you should set different values of radii to create smaller or larger particles.),
- (10 points) provide necessary functions to accelerate the neighbor search by using the spatial hash grid.
- (5 points) use proper naming convention with well commented code.

Compilation Guidelines: Before proceeding with submission, kindly verify that your code compiles utilizing the **C++17** standard. MacOS and Linux users must compile the Raylib library from its source in order to successfully compile the program. The instructions for compiling the raylib from the source can be accessed from [3].

Due date: Sunday, 11 March 2024, 9:00 AM.

References

- [1] Matthias Teschner, Bruno Heidelberger, Matthias Muller, Danat Pomeranets, and Markus Gross, "*Optimized Spatial Hashing for Collision Detection of Deformable Objects*", in Proceedings of the 8th Workshop on Vision, Modeling, and Visualization (VMV 2003), Munich, Germany, 2003.
- [2] Blogpost: The mind of Conkerjo, *Spatial hashing implementation for fast 2D collisions*, available online: <https://conkerjo.wordpress.com/2009/06/13/spatial-hashing-implementation-for-fast-2d-collisions/>
- [3] Ramon Santamaria (@raysan5), raylib: a simple and easy-to-use library to enjoy videogames programming, available online: <https://www.raylib.com/>