Assignment 1: Design Document
COP 290: Design Practices in Computer Science

Faran Ahmad(2013CS10220), Kartikeya Gupta(2013CS10231), Prateek Verma(2013CS10246)

January 2015

1 Overall Design

Our overall design of the screensaver would consist of a board or a background screen with N balls on it, where N can be provided as an input by the user. Initially, each ball would get a random color, start from a random position and would start moving in a random direction with a random velocity. Further, the application would be designed so that the collisions between the balls and between the ball and the wall follow laws of physics. Also, the complete motion of the balls can be paused by the user using a GUI button. The speed of any of the balls could be changed at that time so that when the motion is again set to resume, the balls move with their new velocities.
According to the requirements of the assignment

- The application would be programmed in C++.
- GUI part would be done using OpenGL.
- We will use POSIX thread for simulating the motion of the balls.
- Inter-thread synchronization would be done using mutex lock.
- GUI options will be available to the user to make changes to the balls and the board.

2 Sub Components

2.1 Design Of The Ball

Ball class consist of all the information about the balls i.e. its velocity, position, radius and color.

Listing 1: Class Parameters for Ball

```
1  class Ball
2  {
3      private:
4          double radius;                  //Radius of the ball
5          double coord_x;                 //X coordinate of the center of ball
6          double coord_y;                 //Y coordinate of the center of ball
7          double velocity_x;              //X component of the velocity of ball
8          double velocity_y;              //Y component of the velocity of ball
9          std::vector<float> color;       //Color in RGB format of ball
10 };
```

Functions will be made for accessing all the data parameters of the ball and updating them as desired. Functions for obtaining a random ball based on the desired range will also be created.

2.2 Design Of The Board

The background screen would be implemented using a Board class. It would consist of all the information about the background i.e. its dimensions, color, number of balls present and a vector containing the information of the balls.

Listing 2: Class Parameters for Board

```
1  class Board
2  {
3      private:
4          float dimension_x;                    //x dimension of board
5          float dimension_y;                    //y dimension of board
6          int number_balls;                     //number of balls on board
7          std::vector<Ball> vector_of_balls;    //vector of balls on board
8  };
```

Functions for accessing and updating the parameters of the board will be created. Apart from this, functions to add or remove particular balls will be provided.

2.3 Graphic User Interface

GUI displays the board and the balls on it at any instant of time. Balls will be modelled as solid spheres and an orthographic projection of them will be taken. The Z coordinates of the spheres will be set to 0 which will make all the balls be on the same plane.
For the 2-dimensional display, we will use the *GLUT* and *OpenGL Utility* libraries. At every instant of time, data from the balls is extracted and is displayed on the screen. All the balls have different color and radius which are randomly given when the balls are created. The setup also contains ambient lighting to give a shine to the balls. This lighting could be done by more than one light source of different colors. Specular and diffuse component of the light sources will also be controlled.
The background of the window will be given a 2-dimensional texture by rendering a bitmap image file. This bitmap image file is rendered on a rectangle of size same as that of the window and displayed on the screen. A frame will also be made to mark the boundaries of the region in which the balls will be confined. The size of the window can be changed but the background will adjust itself accordingly. The window will exit on pressing the escape key. However, size of the balls will remain unchanged. Six buttons will be added to the window for the user which will carry out the following functions:

- *Pause button*: This button will stop all the balls at their respective positions.
- *Play button*: This button will resume the motion of all the balls after pressing the pause button.
- *Up button*: This button will increase the speed of the selected ball. It can be done only after pressing the pause button.
- *Down button*: This button will decrease the speed of the selected ball. It can be done only after pressing the pause button.
- *Add button*: This button will add a new ball to the board at a random position having a random velocity vector.
- *Remove button*: This button will remove the selected ball from the board. It can be done only after pressing the pause button.

The Projection Matrix mode will be used so that the spheres towards the corners do not appear distorted. The speed of the ball can be altered by pressing the pause button first and then selecting the particular ball with a mouse click. To increase or decrease the speed of the selected ball, the user must press the up and down button respectively and then press the resume button to see the changes made.

2.4 Physics

We will consider that the collisions between the balls and board and amongst the balls would be perfectly elastic and the final velocities of the balls involved in collision would be according to the laws of conservation of momentum and energy. Further, the mass of the balls are also considered to be different and depend on their radii, assuming all balls to have same density. On collisions with the walls the velocity in the direction normal to the surface is inverted keeping the velocity along the wall same.

3 Testing

3.1 Individual Testing of Sub-Components

Firstly, we would do the individual testing of the various sub-components using a class for testing functions.

Listing 3: Class Parameters for Test

```
1 class Test
2 {
3     private:
4         bool verbose;                  //Variable if test is to be conducted
5         std::string description;       //String description of the test
6         bool isPass;                   //Boolean if the test has passed
7         void PrintPassFail(bool);      //Prints the status of the test
8 };
```

- ***Testing of the Ball class:*** Testing of ball class would be done by setting the specifications of the balls (using the set functions) and then checking that the balls acquired those specification (using the get functions). All functions are checked this way through multiple unit tests. The random ball generation tests will verify if the ball is being generated within the desired limits. The constructors are also verified by obtaining information.
- ***Testing of the Board class:*** Testing of board class was done using the same way as the ball class i.e. setting the specifications of the board class(using the set functions) and then checking that the board acquired those configurations(using the get functions). The constructors are verified by obtaining information from the functions. Ball adding and removal functions will be tested to see if the changes are getting passed in the vector and the other parameters.

3.2 Graphics Testing

Testing of graphics would be done in various stages.

- A *blank window* would be created of a given width and height. To make sure it is of the given dimensions, a point would be made on coordinate (width,height) and then checking that the point is drawn at the top corner of the window.
- A *colored sphere* would be created at origin of a given radius. To make sure it is of the given dimension, a point would be made on the coordinate (radius,0) and then checking that the point is drawn on the circumference of the sphere.
- A *colored sphere* would be created at a given X and Y coordinates of fixed radius. This would be tested by make a point at the same X and Y coordinates and checking that the point is drawn at the centre of the sphere.
- A *light source* would be created at a given X and Y coordinates with a given light color. This would be tested visually.

- A *test image* would be rendered as the background image. This would be tested visually.
- A *test button* would be created at a given X and Y coordinates. This would be tested by clicking the button at the created point and checking the X and Y coordinates of the mouse click function.

3.3 Combined Testing

The combined testing would be done in various stages.

- A *board* of given width and height would be created and then displayed on the screen using the graphics sub-component. This would be tested visually.
- A *vector of balls* would be created of given radii and colors (having zero speeds) and then displayed on the screen using the graphics sub-component.This would be tested visually.
- A *vector of balls* would be created of given radii and colors and having random speeds and then displayed on the screen using the graphics sub-component. Using this configuration, the following things will be tested.
  - The *collisions* between ball-ball and ball-wall would be taken into account.For ball-ball collisions, a larger ball would be considered to have a larger mass and the equations are applied accordingly. The collisions should follow the laws of physics. For ball-wall collisions, the angle of incidence should be equal to the angle of reflection. This would be tested visually.
  - The *play/pause buttons* would be pressed to check their functionality
  - The *up/down buttons* would be checked by pressing the pause button. Then a random ball would be selected using the mouse click and pressing the up or down buttons to increase or decrease the speed of the selected ball. Finally, the play button would show the desired change.
  - The *add ball button* would be pressed to check its functionality. The ball would be added at a random position such that it does not overlap with another ball or the frame. This would be tested visually.
  - The *remove ball button* would be checked by pressing the pause button. Then a random ball would be selected using the mouse click to achieve the desired change.

4 Interaction Amongst Various Sub-Components

Each of the balls on the screen is simulated using one thread. The board class consists of a variable that would store the information about all the balls using a vector(vector is used for random access of the balls and to allow removal and addition of balls). Now each of the thread controlling balls calls a function "updatefunction" that updates the state of the balls in the vector and thus the overall state of the board. In this case there are three possibilities.

- *When there is a Ball-Wall collision* In this case, the position and velocity of the ball is modified depending on the direction of the wall and it is updated in the Board.
- *When there is no Ball-Ball collision* In this case, the update function called by any thread would only modify the position of that particular ball in the vector.
- *When this is Ball-Ball collision* In this case, the update function, if called by any of the balls involved in collision would modify the positions and velocities of both the balls in the vector after applying the physics of elastic collisions.

The screen will display the current state of the board at all times. The threads update the position of the balls on the board and hence the desired motion will become visible.

5 Thread Interactions

Initially, each of the thread controlling the balls are in active state and try to pick up the `mutex_lock` by calling the updatefunction so as to modify the positions and velocity of the ball simulated by it. The threads are synchronized so that in each cycle the position and velocity of all the balls are updated i.e. in each cycle each of the thread gets the chance to pick up the lock and update the state of the ball which is simulated by it. This design is like a barrier method of communication. Once this is implemented correctly, we will attempt a one to one communication in which there will be no stored collection of the data of balls at one location.

6 Maintaining Variable Ball Speeds

To maintain the variable ball speeds, we will keep the velocity of the ball in both directions as a parameter in the Ball class. This eliminates the need for changing the sleep time for different threads. We will provide a GUI interface for the user to pause the screen, select a particular ball and increase or decrease its speed based on the buttons clicked.

7 Extra GUI Components

- Adding a GUI interface for the user to pause and play the screen.
- Adding GUI buttons for the user to select a particular ball and modify its speed.
- Adding GUI buttons to add a ball to the screen or remove a ball after selecting it.

8 Future endeavours

- Making the interaction area for the balls in 3 dimensions and providing the user an interface to view the cuboid containing the balls from different angles.
- Adding sound effects when collisions between balls takes place.
- Making individual threads control multiple balls.