Screensaver Application Documentation

Saurabh Bhadada 2012MT50619

Sanjay Karela 2012MT50616

1 Overall Design

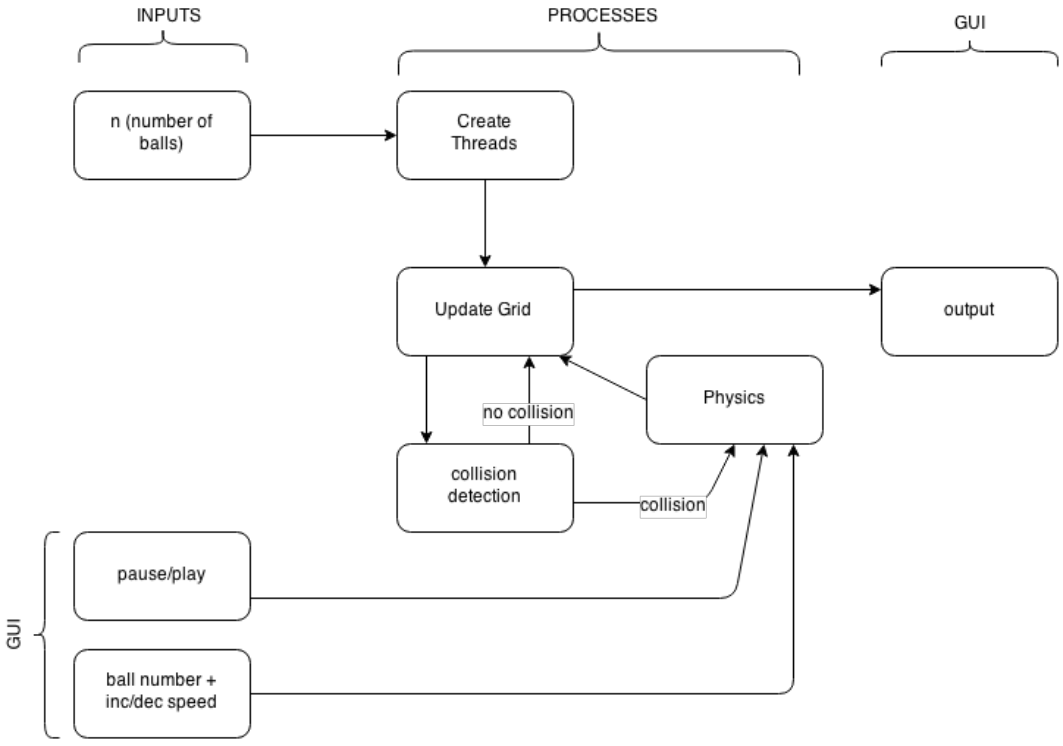

Figure 1.1: overall design of screen saver application

Input ‘n‘ mentioned in flow chart is number of balls user want in screensaver. upon recieving input n threads are created which furthur calls routine function. Routine function in each

thread makes object of class balls which is stored in file shapes.cpp (covered in next section). each thread updates position of corresponding ball on grid. Each pair is feeded to collision detection if collision is detected physics controller then updates positions after collision if collision is not detected positions are updated normally. Every updated position is feeded to output stream which updates position of ball in GUI.
Pause/Play button toggles state of system between play mode (i.e. balls are moving) and pause mode (i.e. balls are stationary and have same position at time of button pressing). Increase/Decrease speed increases/decreses speed of a particular ball with given thread number.

2 SUB COMPONENTS

The application directory consists of following files:

- shapes.cpp
- physics.cpp
- collisionDetection.cpp
- main.cpp

Entire screen where ball can be present is divided into small boxes and each box is assigned a 2-tuple corresponding to its position in two-dimensional array called *Grid*. This 2-tuple is called Gridboxnumbers.
Shapes.cpp contains a class called *balls* which have following instances

```
private int ball_no /* unique number associated with each ball*/
private int x  /* x coordinate of center of ball */
private int y  /* y coordinate of center of ball*/
private double dx /* change in x-coordinate per dt time */
private double dy /* change in y-coordinate per dt time */
private double dt /* time change  */
private int radius /*radius of ball*/
public int grid_box_no1
public int grid_box_no2
private int dx_store
private int dy_store
```

It clear from above that if velocity is known we can initilize value of dx,dy and dt. Grid box numbers are used to detect collisions between any two balls.
collisionDetection.cpp contains function checkCollision() which has following structure. It takes each ball's grid box numbers, radius and their coordinates and return true if there is a collision and false if there is not.

```
boolean checkCollision(balls ball1,balls ball2)
```

physics.cpp contains a function called duringCollision() with following structure. It take each ball's velocity before collision and their radius and apply laws of momentum and energy conservation and update dx,dy and dt of each ball.

```
void duringCollision (balls ball1, balls ball2)
```

All inputs and outputs from both GUI and command line are handled by main.cpp. And it is also ground for intraction of all different units i.e. physics, collision detection, GUI. A function (checkCollision()) of collisionDetection class is called in order to check collisions between two threads or balls if collision detected then a function (duringCollision()) from physics class is then called to determine and update post-collision velocities. Each thread with updated grid box number updates position of ball on screen. In this way Collision Detection, physics and GUI intracts with each other.

Class Name	**Test Cases**	**Output**
Ball	1.Negative co-ordinates of center.	No shape renders.
	2.Co-ordinates beyond, the Window size.	Depends on the co-ordinate e.g if y is beyond the window size but x in range than some part of ball will be drawn.
Physics	1.If ball to ball collision is not considered.	Ball will remain in Window size and reflects when it touches the wall.
	2.If ball to ball collision is considered.	Ball will reflect according to the laws of physics.

3 VARIABLE BALL SPEED

INCREASING AND DECREASING BALL'S SPEED: To increase speed of a particular ball(given ball number) we have to just search a ball with given ball number and increase dx and dy instances of *balls* object keeping dt instance same.

```
ball_ = balls.search(ball_no)
ball_.dx+=a  /* where a is an integer having same sign as of ball_.x */
ball_.dy+=a  /* where a is an integer having same sign as of ball_.y */
```

PLAY/PAUSE: Play and pause are similar to increase and decrease balls speed.To bring system to pause state curent dx,dy are stored in dx_store , dy_store instances of *balls* class and then dx,dy are set to zero.

```
ball_ = balls.search(ball_no)
ball_.dx_store= ball_.dx;
ball_.dy_store =ball_.dy
ball_.dx=0;
ball_.dy=0;
```

To achieve play state values of dx_store , dy_store instances are moved to dx,dy.

```
ball_ = balls.search(ball_no)
ball_.dx=ball_.dx_store;
ball_.dy=ball_.dy_store;
```

4 THREAD INTRACTION

In this application, one to one communication approach is used for inter thread synchronization. First the position of a ball is maintained inside a thread. The position of the balls are exchanged using message queues, because they exploit parallel programming to its fullest. We don't have to wait for the other balls to update their position into global array like that in case where one is using barrier.
It needs to be ensured that a thread is receiving a message only from one other thread at any given time. This is called synchronization. We will achieve synchronization using semaphores.