PROJECT FINAL REPORT

DX BALL RELOADED

INTRODUCTION:

The purpose of this document is to present a detailed description of the DX Ball Reloaded game that the team has developed. DX ball game is a well known brick breaking game. ‘Dx Ball Reloaded’ is our attempt to make a different version of the game based on the original idea. Well the game is about a ball which is bouncing around and breaking bricks. You bounce it back up to break more and if you let the ball pass your paddle, then you lose.

STATUS OF THE PROJECT:

A command-line interface (for input of the game) of the game stands completed.

WORK INITIALLY ALLOCATED:

* **Arkopal Dutt:** Generation of bricks, latency of bricks and the compilation of all the functions and menus into one integrated product
* **Arpit Agarwal:** Movement of the paddle and its interaction with the ball
* **Ashish Charan Tandi:** Movement of ball within game and interaction with bricks
* **Arushi Gambhir, Archit Naraniwal, Avdesh Dixit, Ashish Yerekar:** Designing of the Main Menu and all the in game menus (e.g. Pause Menu)

WORK AS FINISHED BY THE MEMBERS:

* **Arkopal Dutt:** Generation of bricks in the various levels, movement of paddle (final version) and the compilation of all the functions and menus into one integrated product
* **Arpit Agarwal:** creation of the windows/ game menus involved in the game
* **Ashish Charan Tandi:** Movement of ball within game, reflection of ball on bricks, and the walls
* **Archit Naraniwal, Ashish Yerekar:** Designing of the Main Menu and all the in game menus (they were not accepted by the coordinator)
* **Arushi Gambhir:** SRS Document (modified by the coordinator), User Manual, Instructions Window, Final Stage Report
* **Avdesh Dixit:** N/A

WORKING OF THE GAME:

* Game starts upon clicking the executable equivalent file of the game.
* Window appears showing logo of the game.
* Main Menu appears with the following options
  + PLAY
  + OPTIONS
  + QUIT
* On clicking the OPTIONS,another window appears with the following categories
* INSTRUCTIONS
* LEVELS
* CREDITS
* In game,the user can shoot the ball using the enter key. Using the movement keys( ‘a’ for left and ‘d’ for right ) the user will be able to control the paddle to reflect the ball onto the bricks and thus complete the object of finishing the level to proceed to other levels.
* For breaking each brick, points are awarded to the user.The score is updated simultaneously with each breaking brick and is displayed on the same window as the game progresses.
* One can quit the game at any point of time by pressing ‘control c’.

VARIOUS MODULES:

* GENERATION OF BRICKS AT VARIOUS LEVELS:  
  The following function uses a switch statement for different styles of brick placement to distinguish between the various levels offered to the user. It takes an integer value from the user marking the level he wants to play. With each level, the difficulty of the game increases because of the positioning of the bricks and due to the incrementation of the ball speed and the paddle speed. Also, the scoring differs at each level. Nevertheless, with extensive trials, we have made sure that the game is always playable.

void Build(int level)

{

int status(0);

switch (level)

{

case 1:

Number\_Bricks = 18;

Score\_Step\_Up = 10;

for(int i=0;i < 4;i++)

{

Position\_Brick[status][0]= 3.0 + i\*xdiff;

Position\_Brick[status][1]= 1.5 + i\*ydiff;

Presence\_Brick[status] = 1;

RectangleShape brick(Play\_Window, 3.0+i\*xdiff , 1.5+i\*ydiff , Blue , bricklength, brickwidth);

brick.Draw();

status++;

}

for(int j=0 ;j < 4;j++)

{

Position\_Brick[status][0]= 3.0 + (3+j)\*xdiff;

Position\_Brick[status][1]= 1.5 + (3-j)\*ydiff;

Presence\_Brick[status] = 1;

RectangleShape brick(Play\_Window, 3.0 + (3 + j)\*xdiff , 1.5 + (3 - j)\*ydiff , Blue , bricklength, brickwidth);

brick.Draw();

status++;

}

for(int k=1;k < 4;k++)

{

Position\_Brick[status][0]= 3.0 + k\*xdiff;

Position\_Brick[status][1]= 1.5 + (k-1)\*ydiff;

Presence\_Brick[status] = 1;

RectangleShape brick(Play\_Window, 3.0+k\*xdiff , 1.5 + (k-1)\*ydiff , Green , bricklength, brickwidth);

brick.Draw();

status++;

}

for(int l=1 ;l < 4;l++)

{

Position\_Brick[status][0]= 3.0 + (2+l)\*xdiff;

Position\_Brick[status][1]= 1.5 + (3-l)\*ydiff;

Presence\_Brick[status] = 1;

RectangleShape brick(Play\_Window, 3.0 + (2 + l)\*xdiff , 1.5 + (3 - l)\*ydiff , Green , bricklength, brickwidth);

brick.Draw();

status++;

}

for(int m=2;m < 4;m++)

{

Position\_Brick[status][0]= 3.0 + m\*xdiff;

Position\_Brick[status][1]= 1.5 + (m-2)\*ydiff;

Presence\_Brick[status] = 1;

RectangleShape brick(Play\_Window, 3.0+m\*xdiff , 1.5 + (m-2)\*ydiff , Yellow , bricklength, brickwidth);

brick.Draw();

status++;

}

for(int n=2 ;n < 4;n++)

{

Position\_Brick[status][0]= 3.0 + (1+n)\*xdiff;

Position\_Brick[status][1]= 1.5 + (3-n)\*ydiff;

Presence\_Brick[status] = 1;

RectangleShape brick(Play\_Window, 3.0 + (1 + n)\*xdiff , 1.5 + (3 - n)\*ydiff , Yellow , bricklength, brickwidth);

brick.Draw();

status++;

}

for (int s = 18; s < 48; s++) Presence\_Brick[s] = 0;

//Maybe change the speeds of the ball along the axes

break;

case 2:

Number\_Bricks = 48;

Score\_Step\_Up = 15;

for(int i=0;i < 6;i++)

{

for(int j=0;j < 8; j++)

{

RectangleShape brick(Play\_Window, 2.2+j\*xdiff , 1.5+i\*ydiff , Blue , bricklength, brickwidth);

brick.Draw();

Position\_Brick[status][0]= 2.2 + j\*xdiff;

Position\_Brick[status][1]= 1.5 + i\*ydiff;

Presence\_Brick[status]=1;

status++;

}

}

break;

case 3:

Number\_Bricks = 28 ;

Score\_Step\_Up = 20;

for(int i = 0; i < 8; i++)

{

for(int j = 0; j < 7; j++)

{

Position\_Brick[status][0]= 4.0 + j\*xdiff;

Position\_Brick[status][1]= 1.5 + i\*ydiff;

if ((i + j) % 2 == 0)

{

RectangleShape brick(Play\_Window, 4.0+j\*xdiff , 1.5+i\*ydiff , Yellow , bricklength, brickwidth);

brick.Draw();

Presence\_Brick[status]=1;

status++;

}

else Presence\_Brick [status] = 0;

}

}

for(int t = 42; t < 48; t++)

{

Presence\_Brick[t] = 1;

}

Xspeed = 1.2;

Yspeed = 0.8;

break;

case 4:

Number\_Bricks = 18;

Score\_Step\_Up = 30;

for(int i = 0; i < 8; i++)

{

for(int j = 0; j < 7; j++)

{

Position\_Brick[status][0]= 4.0 + j\*xdiff;

Position\_Brick[status][1]= 1.5 + i\*ydiff;

if ((i + j) % 5 == 0)

{

RectangleShape brick(Play\_Window, 4.0+j\*xdiff , 1.5+i\*ydiff , Yellow , bricklength, brickwidth);

brick.Draw();

Presence\_Brick[status]=1;

status++;

}

else if ((i + j) % 7 == 0)

{

RectangleShape brick(Play\_Window, 4.0+j\*xdiff , 1.5+i\*ydiff , Blue , bricklength, brickwidth);

brick.Draw();

Presence\_Brick[status]=1;

status++;

}

else Presence\_Brick [status] = 0;

}

}

for(int t = 42; t < 48; t++)

{

Presence\_Brick[t] = 1;

}

Xspeed = 1.2;

Yspeed = 1.0;

max\_paddle\_speed = 3.0;

break;

case 5:

Number\_Bricks = 26;

Score\_Step\_Up = 50;

for(int i = 0; i < 8; i++)

{

for(int j = 0; j < 7; j++)

{

Position\_Brick[status][0]= 4.0 + j\*xdiff;

Position\_Brick[status][1]= 1.5 + i\*ydiff;

if ((i + j) % 3 == 0)

{

RectangleShape brick(Play\_Window, 4.0+j\*xdiff , 1.5+i\*ydiff , Yellow , bricklength, brickwidth);

brick.Draw();

Presence\_Brick[status]=1;

status++;

}

else if ((i + j) % 7 == 0)

{

RectangleShape brick(Play\_Window, 4.0+j\*xdiff , 1.5+i\*ydiff , Blue , bricklength, brickwidth);

brick.Draw();

Presence\_Brick[status]=1;

status++;

}

else Presence\_Brick [status] = 0;

}

}

for(int t = 42; t < 48; t++)

{

Presence\_Brick[t] = 1;

}

Xspeed = 1.2;

Yspeed = 0.8;

max\_paddle\_speed = 3.0;

break;

}

}

* COLLISION OF THE BALL:
* void yreflect(), void xreflect(): These functions employ laws of reflection and reflect the ball on y-axis(x-axis) or a line parallel to y-axis(x-axis).

void yreflect()

{

Ydir=Ydir\*(-1);

}

void xreflect()

{

Xdir=Xdir\*(-1);

}

* void ifbrick(): This function serves the basic agenda of the game. Upon the collision of the ball with any of the bricks, the image of the brick is erased,score is updated and the ball is reflected in the direction(x-axis or y-axis) according to the position of the collision.

void ifbrick()

{

for(int j=0;j < 48;j++)

{

if(Presence\_Brick[j] == 1)

{

if((X\_ball>=Position\_Brick[j][0] - bricklength/2 - dia/2) &&

(X\_ball<=Position\_Brick[j][0] + bricklength/2 + dia/2) &&

(Y\_ball>=Position\_Brick[j][1] - brickwidth/2 - dia/2) &&

(Y\_ball<=Position\_Brick[j][1] + brickwidth/2 + dia/2))

{

if((X\_ball>=Position\_Brick[j][0] - bricklength/2) && (X\_ball <= Position\_Brick[j][0] + bricklength/2))

{

xreflect();

Presence\_Brick[j]=0;

Number\_Bricks--;

LevelScore = LevelScore + Score\_Step\_Up;

Play\_Window.RenderText(Position(2.0,15.2),LevelScore, Black,White);

}

else if((Y\_ball>=Position\_Brick[j][1]- brickwidth/2) && (Y\_ball<=Position\_Brick[j][1] + brickwidth/2))

{

yreflect();

Presence\_Brick[j]=0;

Number\_Bricks--;

LevelScore = LevelScore + Score\_Step\_Up;

}

else

{

Xdir=(-1)\*(Position\_Brick[j][0]-X\_ball)/(mod(Position\_Brick[j][0]-X\_ball));

Ydir=(-1)\*(Position\_Brick[j][1]-Y\_ball)/(mod(Position\_Brick[j][1]-Y\_ball));

Presence\_Brick[j]=0;

Number\_Bricks--;

LevelScore = LevelScore + Score\_Step\_Up;

}

RectangleShape destroybrick(Play\_Window,Position\_Brick[j][0],Position\_Brick[j][1],Black,bricklength,brickwidth);

destroybrick.Draw();

}

}

}

}

* void ifboundary(): As the game is, ball is reflected either from the walls or the bricks or the user-controlled paddle.This function takes into account all these various collisions of the ball and then reflects it in a direction that accords with ‘Laws of reflection’.

void ifboundary()

{

if(X\_ball<=X\_Play\_Area\_Left + dia/2 || X\_ball >= X\_Play\_Area\_Right - dia/2)

{

yreflect();

}

if(Y\_ball <= Y\_Play\_Area\_Top + dia/2)

{

xreflect();

}

if((X\_ball>=xpaddle - length\_paddle/2 - dia/2) &&

(X\_ball<=xpaddle + length\_paddle/2 + dia/2) &&

(Y\_ball>=ypaddle - width\_paddle/2 - dia/2) &&

(Y\_ball<=ypaddle + width\_paddle/2 + dia/2))

{

if((X\_ball>=xpaddle - length\_paddle/2) && (X\_ball <= xpaddle + length\_paddle/2))

{

xreflect();

}

else if((Y\_ball>=ypaddle- width\_paddle/2) && (Y\_ball<=ypaddle + width\_paddle/2))

{

yreflect();

}

else

{

Xdir=(-1)\*(xpaddle-X\_ball)/(mod(xpaddle-X\_ball));

Ydir=(-1)\*(ypaddle-Y\_ball)/(mod(ypaddle-Y\_ball));

}

}

}

* MOVEMENT OF THE PADDLE:

void movepaddle(): Movement of the paddle is controlled via this function.It moves the paddle in the direction and with speed as specified/controlled by the user.

void movepaddle()

{

Paddle\_Patch.SetPosition(Position(xpaddle, ypaddle));

Paddle\_Patch.Draw();

xpaddle = xpaddle + paddle\_dir\*paddle\_step\*paddle\_speed;

Paddle.SetPosition(Position( xpaddle, ypaddle ));

Paddle.Draw();

}

SCOPE OF IMPROVEMENT:

* Introduction of more levels and different types of bricks
* A more user-friendly interface

CONCLUDING REMARKS:

We felt that EzWindows was not a very convenient package to serve all our purposes. We learnt a lot about team work in the process. But the most important lesson was learning to meet deadlines. Hopefully in the future we can come up with something more creative and more useful.