BRICK BREAKER

BY,

SUHAS SURESH

NETHRAVATHI SATHNUR NAGARAJ

JEEVAN CHANDRASHEKAR

TO,

HAEYONG CHUNG

GENERAL DESCRIPTION

The game is called Brick Breaker. It is an arcade game. Aim of the games is the player has to break all the bricks in all the levels using the paddle without making the ball fall of the screen to win the game. The game contains 3 levels:

1. Easy
2. Medium
3. Hard

In the easy level, the player will play with 3 lives, the player has to break all the bricks(Rectangle boxes) on the screen with the ball using paddle. The ball speed is a bit slow. For each brick break, the player is awarded 1 point. The player has to break all the bricks to pass this level. When the player breaks all the bricks, player is advanced to the next level.

In the medium level, the player will play with 3 lives, the player has to break all the bricks(Hearts) on the screen with the ball using paddle. The ball speed is a bit fast compared to easy level. Each brick is stronger. The player has to hit the brick more than once to break the brick. For entire 1 brick break, the player is awarded 2 points. The player has to break all the bricks to pass this level. When the player breaks all the bricks, player is advanced to the next level.

In the high level, the player will play with 3 lives, the player has to break all the bricks(Smiles) on the screen with the ball using paddle. The ball speed is a bit fast. Each brick is stronger. The player has to hit the brick more than once to break the brick. For each brick break, the player is awarded 3 points. The player has to break all the bricks to win this game. Once the player has finished breaking all the bricks. The Game is finished and congratulations message is displayed along with the score.

CHARACTER DESIGN

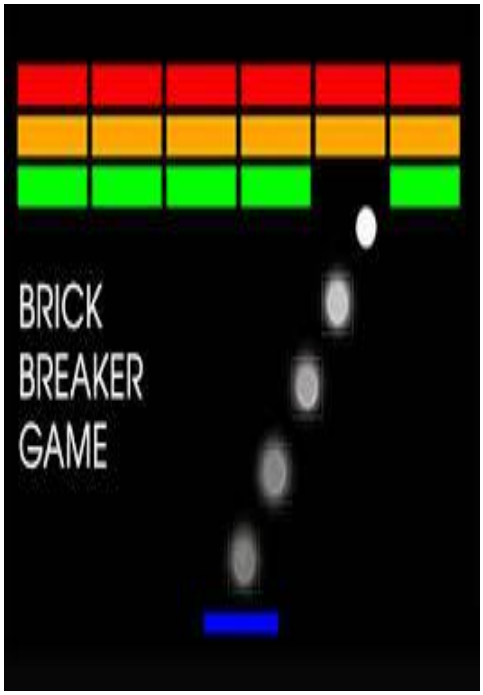
- Object oriented programming concepts is used to create different kinds of bricks in different levels. In level easy rectangle boxes are used. In level medium hearts are used and in hard level, smiles are used to break the bricks.
- Ball and the paddle is created with the help of physics to break the brick.
- Collision concepts is implemented to detect collision between:
    - Ball and bricks
    - Ball and paddle.
- Composer Scenes is used for transition from one level to another.

TARGET AUDIENCE

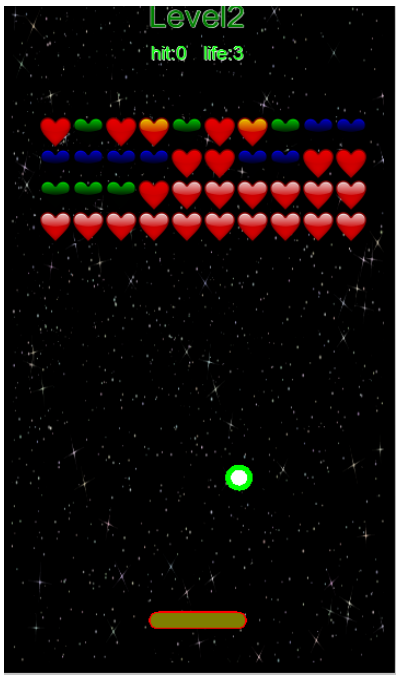
The intended audience for this game are kids of age groups 8 to 12. This game helps kids to enjoy playing the game with ease.

CONCEPT DRAWING

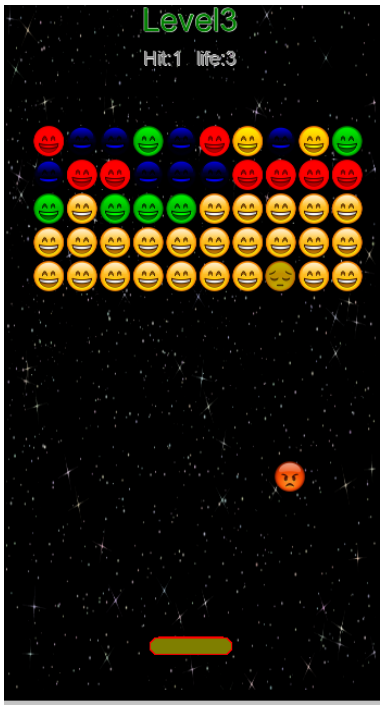
Level 1:



Level 2:



Level 3:



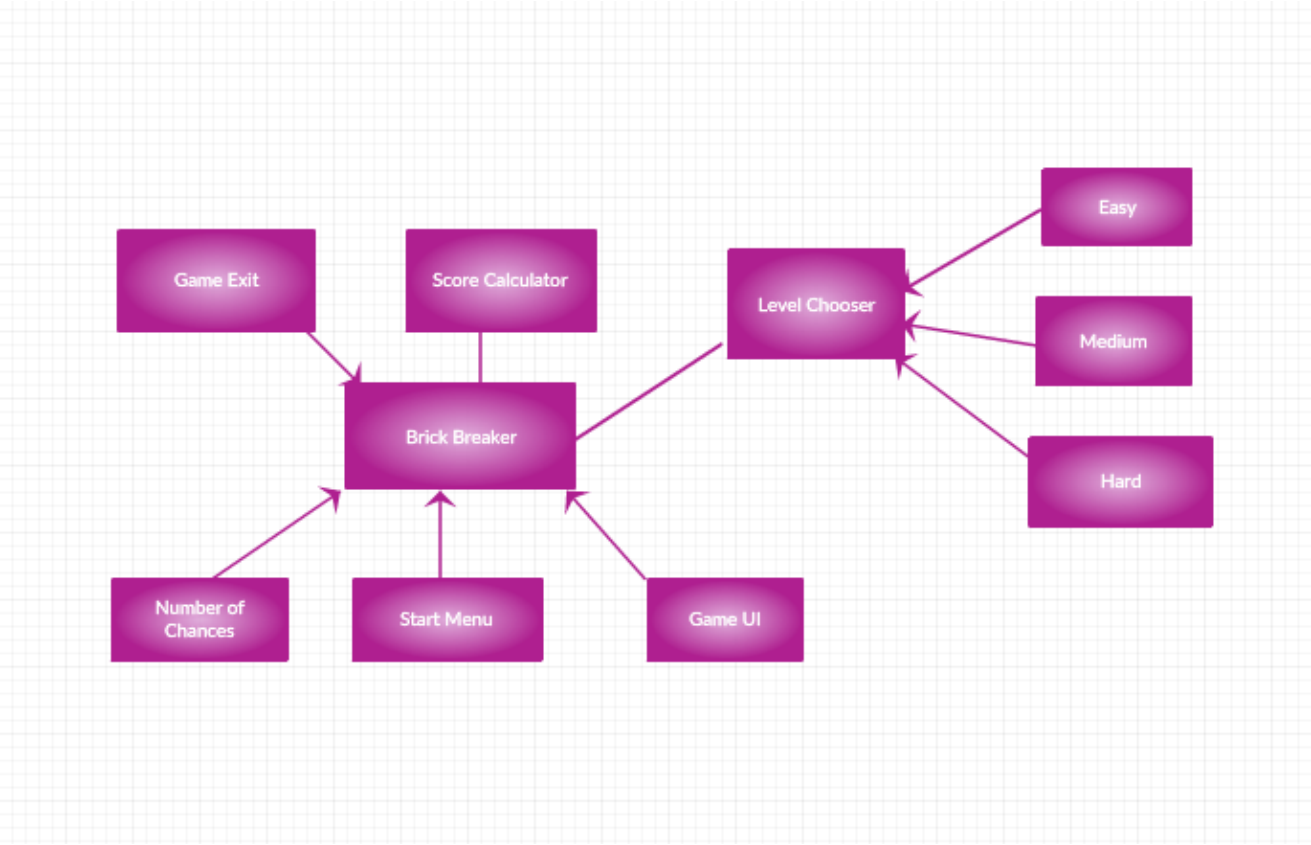
The above mentioned are some of the designs taken from the existing games. Our game concept is similar to these designs but will have more animations and different levels.

SCORING SYSTEM AND ACHIEVEMENTS

- Each time the balls goes off the playing area, the life of the player is decremented by 1
- When the life reaches zero, the game is over.
- There is a hit point assigned to bricks in each level, the hitpoints of the brick increases as the user progresses to the next level.
- Hitpoint of the brick is decremented by 1 each time the ball strikes the brick.
- When the hitpoint of the brick reaches zero, the brick is removed from the play area.

SOFTWARE DESIGN PLAN:

UML Diagram:



DESCRIPTION

In the above diagram the user can choose which level he/she wants to play and select the level. The Start screen will display the start button where the user can click and start from the first level. The score calculator will keep a track of the hitpoints. The Number of Chances is the number of times the user can play the game once started. Once the game is completed or at the beginning of the screen the user can just click on the exit menu and get out of the game.

IMPLEMENTATION NOTES

The game is composed of 3 levels:

- Easy
- Medium
- Hard

Initially, a start button is displayed for the user to start the game on click of the same it takes the user to the next scene where the user is asked to select the difficulty level of the game he/she wants to play.

Implementation of the home screen:

Start button is implemented using a widget as shown below.

```
local startBtn = widget.newButton(
    {
      x = 270,
      y = 675,
      id = "button1",
      label = "Start Game ",
      fontSize = 50,
      font = "Monotype Corsiva",
      labelColor = {default = {1,1,1}, over = {0,0,0,.5}},
      width = 250,
      height = 70,
      defaultFile = "startbutton.png",
      onEvent = handleButtonEvent
    })
```

Difficulty Level is designed using segmented control constituting 3 segments as shown below

```
local function onSegmentPress( event )
local target = event.target
print( "Segment Label is:",
target.segmentLabel )
print( "Segment Number is:",
target.segmentNumber )
if(target.segmentLabel=="Easy") then
  composer.removeScene( "level1", false )
  composer.gotoScene("level1", { effect="fade", time=400 });
elseif (target.segmentLabel=="Medium") then
  composer.removeScene( "level2", false )
  composer.gotoScene("level2", { effect="fade", time=400 });
else
```

```
  composer.removeScene( "level3", false )
  composer.gotoScene("level3", { effect="fade", time=400 });
end
end

local segmentedControl = widget.newSegmentedControl(
   {
      left = display.contentWidth/2-225,
      top = display.contentHeight/2 -60,
      sheet = segmentSheet,
      leftSegmentFrame = 1,
      middleSegmentFrame = 2,
      rightSegmentFrame = 3,
      leftSegmentSelectedFrame = 4,
      middleSegmentSelectedFrame = 5,
      rightSegmentSelectedFrame = 6,
      segmentFrameWidth = 40,
      segmentFrameHeight = 68,
      dividerFrame = 7,
      dividerFrameWidth = 4,
      dividerFrameHeight = 68,
      segmentWidth = 150,
      segments = { "Easy", "Medium", "Hard"},
      fontSize = 40,
      defaultSegment = 1,
      labelColor = { default={0,0,0}, over={1,0.3,0.8,1}},
      labelSize =30,
      labelFont = "Monotype Corsiva",
      emboss = true,
```

```
        onPress = onSegmentPress
    }
)
     sceneGroup:insert( segmentedControl )
   --end
end
end
```

Bricks across all levels are designed using object oriented programming concepts. A new file brick.lua consists of all the attributes and functions of the class brick. Sample code to show how bricks are created using function spawn

```
function Brick:spawn(row,column)
 self.shape=display.newCircle(self.xPos, self.yPos,15);
 self.shape.pp = self;  -- parent object
 self.shape.tag = self.tag; -- “Brick”
 self.shape:setFillColor (1,1,0);
 physics.addBody(self.shape, "kinematic");
end
```

overriding the above function in different levels to create different shapes for bricks.

Level 1(easy): Brick shape rectangle

```
function box:spawn(row,coloumn)
          box[row][coloumn] = display.newSprite(sheet, seqData);
            if (row==0) then
              box[row][coloumn]:setSequence("grey");
                                    box_placement={x=display.contentWidth-(72*6)/2-
300,y=display.contentHeight/2-400}
                                    box[row][coloumn].x=box_placement.x+(coloumn * 72) ;
                                     box[row][coloumn].y=box_placement.y +(row*69);
      elseif(row==1) then
                                 box[row][coloumn]:setSequence("blue");
```

```
                                box_placement={x=display.contentWidth-(72*6)/2-
300,y=display.contentHeight/2-400}
                                box[row][coloumn].x=box_placement.x+(coloumn * 72) ;
                                 box[row][coloumn].y=box_placement.y +(row*69);
      elseif(row==2) then
                                box[row][coloumn]:setSequence("yellow");
                                box_placement={x=display.contentWidth-(72*6)/2-
300,y=display.contentHeight/2-400}
                                box[row][coloumn].x=box_placement.x+(coloumn * 72) ;
                                    box[row][coloumn].y=box_placement.y +(row*69);
                            else
                                box[row][coloumn]:setSequence("red");
                                box_placement={x=display.contentWidth-(72*6)/2-
300,y=display.contentHeight/2-400}
                                box[row][coloumn].x=box_placement.x+(coloumn * 72) ;
                                    box[row][coloumn].y=box_placement.y +(row*69);
                            end
                                 physics.addBody(box[row][coloumn], "static");
                            box[row][coloumn]:setStrokeColor( 0, 0, 0 );
                            box[row][coloumn].strokeWidth = 3;
                            box[row][coloumn].tag="box";
                             box[row][coloumn].pp = self;
                        sceneGroup:insert(box[row][coloumn])
      end
```

Level 2(medium): Brick shape heart

```
   function box:spawn(row,coloumn)
    box[row][coloumn] = display.newSprite(sheet, seqData);
     box[row][coloumn]:setSequence("heart");
     box_placement={x=display.contentWidth-(60*6)/2-500,y=display.contentHeight/2-450}
     box[row][coloumn].x=box_placement.x+(coloumn * 60) ;
```

```
box[row][coloumn].y=box_placement.y +(row*60);
--box[row][coloumn].shape=self.shape;
physics.addBody (box[row][coloumn], "static");
box[row][coloumn]:setStrokeColor( 0, 0, 0 );
box[row][coloumn].strokeWidth = 3;
box[row][coloumn].tag="box";
box[row][coloumn].HP=2;
box[row][coloumn].pp = self;
box[row][coloumn].row=row;
box[row][coloumn].coloumn=coloumn;
end
```

Level 3(hard): Brick shape smile

```
function box:spawn(row,coloumn)
box[row][coloumn] = display.newSprite(sheet, seqData);
box[row][coloumn]:setSequence("smile");
box_placement={x=display.contentWidth-(61*6)/2-515,y=display.contentHeight/2-450}
box[row][coloumn].x=box_placement.x+(coloumn * 61) ;
box[row][coloumn].y=box_placement.y +(row*62);
physics.addBody (box[row][coloumn], "static");
box[row][coloumn]:setStrokeColor( 0, 0, 0 );
box[row][coloumn].strokeWidth = 3;
box[row][coloumn].tag="box";
self.shape=box[row][coloumn];
box[row][coloumn].HP = 3;
box[row][coloumn].pp = self;
 end
```

Ball and paddle: ball and paddle are designed using concepts of physics. Ball is associated with the collision event to detect collision with the bricks it is as shown below.

```
ball1 = display.newSprite( sheet, seqData );
    ball1:setSequence('angry');
    ball1.x=display.contentCenterX
    ball1.y=display.contentCenterY-50
physics.addBody (ball1, "dynamic", {bounce=1, radius=20} )
        ball1:applyForce(1.5,12,ball1.x,ball1.y)
        ball1:addEventListener( "collision", BoxCollision)
        ball1:addEventListener("collision", ballCollision)
local function ballCollision (event)
                                    if (event.phase == "began") then
                                            if(life>0) then
                                            if (event.other == bottom) then
                                            life = life - 1;
        audio.play( soundTable["loselife"] );
        show1.text = "life:" .. life;
                                                        event.target:removeSelf();
        if(levelcomplete==0) then
        t1=timer.performWithDelay( 2000, function()
                            display.contentCenterY-50, 20)
                            ball1.strokeWidth = 10
                            ball1:setFillColor(1, 1, 1)
                            ball1:setStrokeColor(0,1,0)]]
    ball1 = display.newSprite( sheet, seqData );
    ball1:setSequence('angry');
    ball1.x=display.contentCenterX
    ball1.y=display.contentCenterY-50
         physics.addBody (ball1, "dynamic", {bounce=1, radius=20} )
        ball1:applyForce(1.5,12,ball1.x,ball1.y)
        ball1:addEventListener( "collision", BoxCollision)
        ball1:addEventListener("collision", ballCollision)
```

```
  end,1 );
end
end
                                                    if (life == 0) then
                                                    audio.stop( backgroundMusicChannel );
                                                    ball1:removeSelf()
                                                     show1:removeSelf()
                                                        top:removeSelf()
bottom:removeSelf()
left:removeSelf()
right:removeSelf()
        timer.cancel( t1 )
        if(paddle and paddle.removeSelf)then
                                                            physics.removeBody(paddle)
                                                            paddle:removeSelf()
                                                    end
        for row = 1, 4 do
        for coloumn = 1,10 do
                if(box[row][coloumn] and box[row][coloumn].removeSelf)then
                        physics.removeBody(box[row][coloumn]);
                        box[row][coloumn]:removeSelf();
                        box[row][coloumn]=nil
                end
        end
        end
        --ball1:removeEventListener("collision", ballCollision);
        show:removeSelf()
        show1:removeSelf()
        name1:removeSelf()
        Runtime:removeEventListener("touch", move);
```

```
   timer.performWithDelay( 2000, function()
    finalscore=hit*300;
plagain = widget.newButton( {
   --x=display.contentCenterX, y=50
          x = display.contentCenterX,
          y = display.contentCenterY,
         id = "plagain",
          label = "       Score:  " .. finalscore.."\n      play again",
          fontSize = 60,
           font = "Monotype Corsiva",
          labelColor = { default={ 0, 0, 0 }, over={ 0, 0, 0 } },
              width = 450,
            height = 200,
            defaultFile = "startbutton.png",
         --  sheet = buttonSheet,
          --defaultFrame = 1,
          --overFrame = 2,
          --onEvent = playgain,
      } )
    ball1:addEventListener("collision", ballCollision);
paddle = display.newRoundedRect (display.contentCenterX, display.contentHeight-100, 150, 30,15);
            paddle.strokeWidth = 3
                  paddle:setFillColor(0.745098, 0.745098, 0.745098)
                  --paddle:setStrokeColor(1,0,0)
                  physics.addBody( paddle, "kinematic",{bounce=1});
                  local function move ( event )
                        if event.phase == "began" then
if(count<40) then
                              paddle.markX = paddle.x
end
```

```
                elseif event.phase == "moved" then
    if(count<40) then
                    local x = (event.x-event.xStart) + paddle.markX;

                    if (x <= 20 + paddle.width/2) then
                        paddle.x = 20+paddle.width/2;
                    elseif (x >= display.contentWidth-20-paddle.width/2) then
                        paddle.x = display.contentWidth-20-paddle.width/2;
                    else
                        paddle.x = x;
                    end
                     end
                end
            end

            Runtime:addEventListener("touch", move);
            sceneGroup:insert(paddle)
```

We have used image sheets to improve the appearance of various objects such as bricks and ball.

```
local options =
{
  frames = {
    { x = 5, y = 221, width = 61, height = 62}, --smile  face frame 1
    {x = 150, y = 149, width = 61, height = 62}, --sad face frame 2
    {x = 149, y = 221, width = 61, height = 62}, --crying face frame 3
    {x = 220, y = 149, width = 62, height = 65}, --ball horror frame 4
    {x = 7, y = 4, width = 59, height = 62}, --ball angry frame 5

  }
};
local sheet = graphics.newImageSheet( "image.png", options );
```

```
local seqData = {
  {name = "smile", frames={1}},
  {name = "sad", frames={2}},
   {name = "crying", frames={3}},
    {name = "horror", frames={4}},
   {name = "angry", frames={5}},
}


ball1 = display.newSprite( sheet, seqData );
     ball1:setSequence('angry');
```

there are various transitions from one scene to another such as

- Start scene to level selection

```
local options =
{
 effect = "slideLeft",
 time = 600
}

composer.gotoScene("scene1", options);
```

- Level selection scene to selected level

```
if(event.target.id == '1') then
        composer.removeScene( "level1", false )
        composer.gotoScene("level1", { effect="fade", time=400 });
    end
    if(event.target.id == '2') then
        composer.removeScene( "level2", false )
        composer.gotoScene( "level2" , { effect="fade", time=500 } )
    end
    composer.removeScene( "scene1", false )

  end
```

- Transition between different levels

```
        composer.removeScene( "level1", false )
        composer.gotoScene( "level2",{effect="slideLeft",time=500});
```

We have used custom events to get back to the level selection screen if the user wants to play again after losing a particular level.

```
plagain = widget.newButton( {
         --x=display.contentCenterX, y=50
```

```
                x = display.contentCenterX,
                y = display.contentCenterY,
               id = "plagain",
                label = "        Score:  " .. finalscore.."\n      play again",
                fontSize = 60,
                 font = "Monotype Corsiva",
                labelColor = { default={ 0, 0, 0 }, over={ 0, 0, 0 } },
                     width = 450,
                  height = 200,
                  defaultFile = "startbutton.png",
              --  sheet = buttonSheet,
                --defaultFrame = 1,
                --overFrame = 2,
                --onEvent = playgain,
            } )
            local function playAgainHandler(myEvent)
            --if(event.phase=='ended')
            composer.removeScene( "level1", false )
            composer.gotoScene("scene1", { effect="crossFade", time=500 });
            if(myEvent=="ended") then
            plagain:dispatchEvent( myEvent )
            end
            end

            plagain:addEventListener( "tap", playAgainHandler )
            local myEvent={ name="tap", target=plagain}

         sceneGroup:insert(plagain)
         end,1)
end
```

We have used a sound table to store all the different sounds that's been used in the game. It is as shown below.

```
local soundTable = {

        shootSound = audio.loadSound( "shoot.wav" ),
        hitSound = audio.loadSound( "hit.wav" ),
        explodeSound = audio.loadSound( "explode.wav" ),
        laserSound = audio.loadSound( "Laser_Shoot2.wav" ),
        powerup5 = audio.loadSound( "Powerup5.wav" ),
        loselife = audio.loadSound( "result.wav" ),
        bgscore= audio.loadStream( "Arcade-Fantasy.wav" ),
        heartbreak=audio.loadStream( "heartbreak.wav" ),
        sadSound=audio.loadStream( "Sad Sound1.wav" ),
        crySound=audio.loadStream( "Baby Crying1.wav" ),
        level2bg=audio.loadStream( "8-Bit-Mayhem.mp3" ),
```

```
        level3bg=audio.loadStream( "Monkey-Drama.mp3" )
    }
    return soundTable;
```

TARGET PLATFORM

- Handheld and Mobile devices(Android)
- Input technology:
    - Touch screen gestures (pan)
- Simulator: Corona Sdk(Samsung Galaxy S3)

PROJECT MILESTONE

I. The design of the actual game will be roughly done by the 16th of April.
II. The start screen and first level will be done by the 20th of April.
III. The second level will be done by the 23th of April.
IV. The third level will be finished by the 26th of April.
V. The report work will be finished as and when each level is completed.
VI. The integration and testing will be done as and when each level is completed.
VII. We will be meeting thrice a week to plan out accordingly for the above mentioned schedule.

TEAM COLLABORATION

- Suhas Suresh : Object oriented programming to design bricks, implementing physics concepts such as collision force and creating objects such as ball and paddle.
- Jeevan : Design of start screen, and implementing composer for transition from one level to another.
- Nethravathi: Implementing sprites for different objects such as bricks ball and paddle across 3 different levels.

REFERENCES

- https://en.wikipedia.org/wiki/Arkanoid
- http://www.gamasutra.com/view/feature/1630/breaking_down_breakout_system_and_.php?print=1
- https://www.playonloop.com/music-loops-category/videogame/
- http://soundimage.org/looping-music/