Submitted by Karthick Mohanasundaram
Student ID: 15200702

REPORT – Breakout
Assignment 5

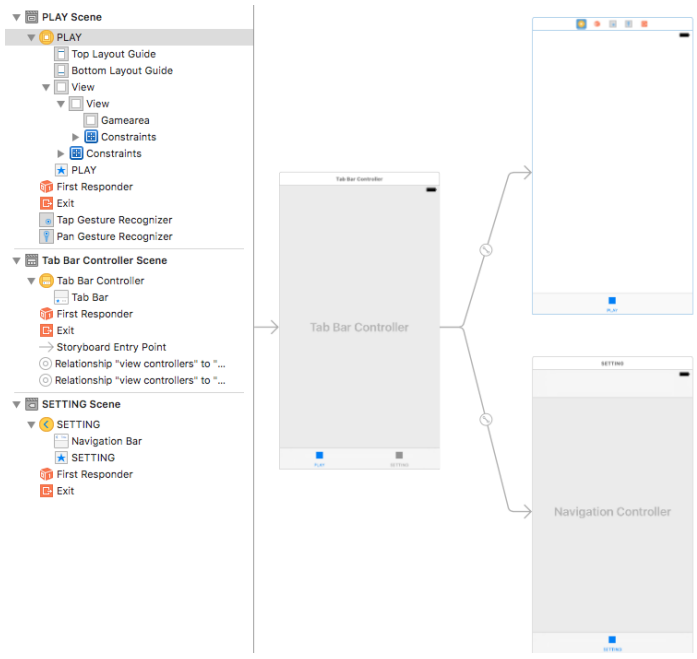
OBJECTIVE:

An iOS game to play Breakout and should allow for some level of customisation using a separate MVC.

Device: *iPhone 6s*

Set a Tab bar Controller

- Drag a tab bar controller into Main.storyboard. Add a View to play view controller and drag and drop pan gesture recognizer and tap gesture recognizer from interface builder to it.



- Connect view , action tap gesture and action pan gesture to BreakOutVC.swift

Set up the bricks

- Create a swift file “BreakOutBehaviour”; subclass of UIDynamicBehavior, UICollisionBehaviorDelegate.
- Create a UIView (name it as brickView) and set the necessary attributes to it inside nested for loop. The total iteration of nested for loop is total number of rows + columns. Calculate the origin of each bricks for each iteration. Write a function in BreakOutBehaviour called “addBrick(_ brick: UIView)” to set dynamic animator and collision behaviour to the array of bricks. Pass each brickView to addBrick function as parameter.

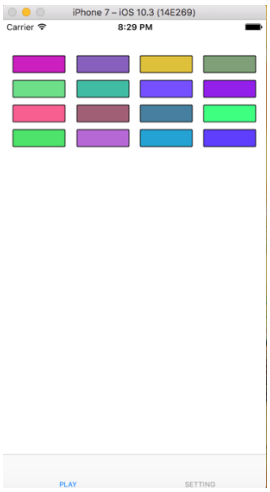
BreakOutVC.swift

```
//draws no. of bricks; total of Brick_Rows + Brick_Columns
for row in 0 ..< Brick_Rows {

    for column in 0 ..< Brick_Columns {

        //15.0 is the distance between each brick

        //50.0 is the distance between from the top of the window to the first
            line of bricks.

        //calculating the origin position of the bricks for each iteration
        let brickViewOrigin = CGPoint(x: CGFloat(15.0) + (brickWidth +
            CGFloat(15.0)) * CGFloat(column),y: CGFloat(50.0) + (brickHeight +
            CGFloat(10.0)) * CGFloat(row))

        //draws a brick at specified coordinates and size
        let brickView = UIView(frame: CGRect(origin: brickViewOrigin, size:
            (CGSize(width: brickWidth,height: brickHeight))))
        let Brick_BgColor = UIColor.RandomColor()
        brickView.layer.backgroundColor = Brick_BgColor.cgColor
        brickView.layer.borderColor = Brick_BorderColor.cgColor
        brickView.layer.borderWidth = CGFloat(1.0)
        brickView.layer.cornerRadius = CGFloat(2)
        brickView.type = game_things.brick


        GameBehaviour.game_protocol = self
        //calls addBrick func in BreakOutBehaviour file to apply dynamic
            animatore and collider behavior to each bricks
        GameBehaviour.addBrick(brickView)
    }

}
```

BreakOutBehaviour.swift

```
//set dynamic animator and collision behavior to the array of bricks
var brick_attachment_behaviour: [UIView: UIAttachmentBehavior] = [:]
func addBrick(_ brick: UIView)
{
    dynamicAnimator?.referenceView?.addSubview(brick)
    //enables bricks to engage collision with others
    collider.addItem(brick)

    //sets the bricks anchor point at center
    brick_attachment_behaviour[brick] = UIAttachmentBehavior(item: brick,
        attachedToAnchor: brick.center)

    //applies dynamic animatore to bricks
    addChildBehavior(brick_attachment_behaviour[brick]!)
    brick_behavior.addItem(brick)
}
```

***Output*:**



Create Paddle

- Create a UIView (name it as paddleView) and set the required attributes, origin and size to it.
- Write a function in BreakOutBehaviour called "func addpaddle(_ paddle: UIView)" to set dynamic animator and collision behaviour to it. Pass paddleView as a parameter.
- Connect the action pan gesture to BreakOutVC to move the paddle horizontally across the screen. Call the function "func movepaddle(_ view: UIView, translation: CGPoint)" to update the new anchor point of paddle with respect to pan gesture.

BreakOutVC.swift

```
let paddleOrigin = CGPoint(x: gamearea.bounds.midX - CGFloat(50.0),
                           y: gamearea.bounds.maxY - CGFloat(15.0))
//draws a paddle subview in parent view, places at specified coordinates
paddleView = UIView(frame: CGRect(origin: paddleOrigin, size: CGSize(width:
    100.0, height: 10.0)))
paddleView!.layer.backgroundColor = paddle_BgColor.cgColor
paddleView!.layer.borderColor = paddle_BorderColor.cgColor
paddleView!.layer.borderWidth = CGFloat(1.0)
paddleView!.type = game_things.paddle
GameBehaviour.addpaddle(paddleView!)
```

BreakOutBehaviour.swift

```
//set dynamic animator and collision behavior to paddle
var paddle_attachment_behavior: UIAttachmentBehavior? = nil
func addpaddle(_ paddle: UIView)
{
    dynamicAnimator?.referenceView?.addSubview(paddle)
    collider.addItem(paddle)

    //sets the paddle anchor point at center
    paddle_attachment_behavior = UIAttachmentBehavior(item: paddle,
        attachedToAnchor: paddle.center)

    addChildBehavior(paddle_attachment_behavior!)
    paddle_behavior.addItem(paddle)
}
```

pangesture - BreakOutVC.swift

```
//setting the pangesture to move paddle across the screen
@IBAction func moveGamePaddle(_ panGesture: UIPanGestureRecognizer) {
    switch panGesture.state {
    case .began: fallthrough
    case .changed: fallthrough
    case .ended:
        let translation = panGesture.translation(in: gamearea)
        GameBehaviour.movepaddle(paddleView!, translation: translation)
        panGesture.setTranslation(CGPoint.zero, in: gamearea)
    default:
        break
    }
}
```

BreakOutBehaviour.swift

```
//updates the new anchor point of paddle with respect to pan gesture
func movepaddle(_ view: UIView, translation: CGPoint) {
    view.center.x += translation.x
    paddle_attachment_behavior?.anchorPoint.x = view.center.x
    dynamicAnimator?.updateItem(usingCurrentState: view)
}
```

Output:

Create ball

- Create a UIView (name it as ballView) and set the required attributes, origin and size to it.
- Write a function in BreakOutBehaviour called “func addball(_ ball: UIView)” to set dynamic animator and collision behaviour to it. Pass ballView as a parameter.
- Connect the action tapgesture to BreakOutVC to get it to bounce. Call the function “func bounceBall(_ view: UIView, angle: CGFloat, magnitude: CGFloat)” to apply push behavior to ball to change travelling path accordingly when collides with paddle and bricks.

BreakOutVC.swift

```
//draws a ball subview in parent view, places at specified coordinates
let ballOrigin = CGPoint(x: gamearea.bounds.midX - 10,
                         y: gamearea.bounds.maxY - CGFloat(40.0))

ballView = UIView(frame: CGRect(origin: ballOrigin, size: Ball_Size))
let Ball_BgColor = UIColor.RandomColor()
ballView!.layer.backgroundColor = Ball_BgColor.cgColor
ballView!.layer.borderColor = Ball_BorderColor.cgColor
ballView!.layer.borderWidth = CGFloat(1.0)
ballView!.layer.cornerRadius = Ball_Size.height / 2.0
ballView!.type = game_things.ball
GameBehaviour.addball(ballView!)
```

BreakOutBehaviour.swift

```
//set dynamic animator and collision behavior to ball
func addball(_ ball: UIView){

    dynamicAnimator?.referenceView?.addSubview(ball)
    //enables ball to engage collision with others
    collider.addItem(ball)

    ball_behavior.addItem(ball)
}
```

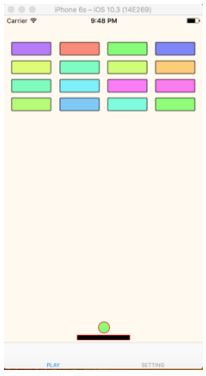
tap gesture - BreakOutVC.swift

```
//setting the tapgesture to bounce the ball when a touch is made on the view
@IBAction func pushBall(_ tapGesture: UITapGestureRecognizer) {
    switch tapGesture.state {
    case .ended:
        GameBehaviour.bounceBall(ballView!, angle: 1.80, magnitude:CGFloat(setting.
            speed))
    default:
        break

    }
}
```

BreakOutBehaviour.swift

```
//applies push behavior to ball to change travelling path accordingly when collides
    with paddle and bricks
func bounceBall(_ view: UIView, angle: CGFloat, magnitude: CGFloat) {

    let ball_bounce = UIPushBehavior(items: [view], mode: UIPushBehaviorMode.
        instantaneous)
    ball_bounce.angle = angle
    ball_bounce.magnitude = magnitude
    ball_bounce.action = {
        ball_bounce.dynamicAnimator?.removeBehavior(ball_bounce)
    }
    dynamicAnimator?.addBehavior(ball_bounce)
}
```

Output:



Bounce the ball off the wall

- Set the collision behaviour to the boundary of the view to bounce the ball when it made contact.

- Call the function “func drawBorders(_ view: UIView)” when the game begins. This function set the collision behaviour to top, left and right boundary of the view.
- The game ends when the ball hits the bottom boundary of the view.

BreakOutBehaviour.swift

```
//set collision behaviour to boundary; top, left and right
func drawBorders(_ view: UIView) {

    let topLeft = CGPoint(x: view.frame.minX, y: view.frame.maxY)
    let topRight = CGPoint(x: view.frame.maxX, y: view.frame.maxY)

    collider.addBoundary(withIdentifier: "Bottom" as NSCopying, from: topLeft, to:
        topRight)

}
```

Game behaviour implementation

- The “collisionBehaviour” function is called whenever there is a collision is started between any two dynamic items.

```
//called when a collision is started between any two dynamic items.
func collisionBehavior(_ behavior: UICollisionBehavior, beganContactFor item1:
    UIDynamicItem, with item2: UIDynamicItem, at p: CGPoint) {
    let (ballView, brickView) = collidedItems([item1, item2])

    if brickView != nil {

        //game ends when ball hits final brick
        if brick_attachment_behaviour.count == 1 {

            //plays winning sound
            play(sound: "cheer")

            //called to begin new game
            game_protocol?.win()

            //disabling the collision behaviour of ball
            collider.removeItem(ballView!)
            ballView?.removeFromSuperview()
        }
        deleteSubviews(view: brickView!, temp: true)
    }
}
```

- Call the function “func collidedItems(_ items: [UIDynamicItem]) -> (ballView: UIView?, brickView: UIView?)” to identify the collided items.

```
//called when collision occurs to identify the collided items
func collidedItems(_ items: [UIDynamicItem]) -> (ballView: UIView?, brickView:
    UIView?) {

    var ballView: UIView? = nil
    var brickView: UIView? = nil
    for item in items {
        if let view = item as? UIView {
            switch view.type {
            case .ball:
                ballView = view
            case .brick:
                brickView = view
            default:
                break
            }
        }
    }
    return (ballView, brickView)
}
```

- If the function “collidedItems( )” returns brickview as collided items, remove the collided bricks from the view. Meanwhile, check if “brick_attachment_behaviour: [UIView: UIAttachmentBehavior] = [:]” count is equal to 1, if so, call the function “win( )” in BreakOutVC.swift

```
//called when a collision is started between any two dynamic items.
func collisionBehavior(_ behavior: UICollisionBehavior, beganContactFor item1:
    UIDynamicItem, with item2: UIDynamicItem, at p: CGPoint) {
    let (ballView, brickView) = collidedItems([item1, item2])

    if brickView != nil {

        //game ends when ball hits final brick
        if brick_attachment_behaviour.count == 1 {

            //plays winning sound
            play(sound: "cheer")

            //called to begin new game
            game_protocol?.win()

            //disabling the collision behaviour of ball
            collider.removeItem(ballView!)
            ballView?.removeFromSuperview()
        }
        deleteSubviews(view: brickView!, temp: true)
    }
}
```

- Call the function “deleteSubviews(view: brickView!, temp: true)” to remove the collided bricks in BreakOutBehaviour.swift.

```
//deletes only hit bricks or all subviews from the view based on the value of
    animated
func deleteSubviews(view: UIView, temp: Bool = true) {

    //add transition befor bricks is removed when ball hits over it
    if temp == true {

        //sound
        play(sound: "bone")

        UIView.transition(with: view, duration: 0.5,
                          options: .transitionFlipFromTop,
                          animations: { },
                          completion: { (finished: Bool) -> Void in
                            self.deleteSubviews(view: view, temp: false)
        })
        return
    }


    //remove ball, bricks, and paddle before the new game begins
    switch view.type {
    case .ball:
        ball_behavior.removeItem(view)
    case .paddle:

            paddle_behavior.removeItem(view)

    case .brick:
        if brick_attachment_behaviour[view] != nil {
            brick_behavior.removeItem(view)
            removeChildBehavior(brick_attachment_behaviour[view]!)
            brick_attachment_behaviour[view] = nil
        } else { return }
    default:
        return
    }
    collider.removeItem(view)
    view.removeFromSuperview()
}
```

- The function “func collisionBehavior()” is called when there is a collision between ball and boundary of the view. Implement the losing condition by checking whether the ball hits the bottom boundary with the help of identifier “Bottom”. Call the func “end()” when the ball hits the bottom boundary of the view.

```
//Called when a collision is started between ball and boundary.
func collisionBehavior(_ behavior: UICollisionBehavior, beganContactFor item:
    UIDynamicItem, withBoundaryIdentifier identifier: NSCopying?, at p: CGPoint) {
    let (ballView, _) = collidedItems([item])


    if let borderID = identifier as? String {

        switch borderID {

        //test whether the ball hits the bottom boundary
        case "Bottom":
                //plays losing sound
                if brick_attachment_behaviour.count != 0
                {
                play(sound: "endbuzzer")
                }
                //game ends
                game_protocol?.end()
                collider.removeItem(ballView!)
                ballView?.removeFromSuperview()
        default:
            break
        }
    }
}
```

- Override the init() method in BreakOutBeahviour to add the dynamic items as a child to custom dynamic behaviour.

```
//adds the dynamic behavior items; ball, paddle and brick, as a child.
  override init() {
      super.init()
      addChildBehavior(collider)
      addChildBehavior(ball_behavior)
      addChildBehavior(paddle_behavior)
      addChildBehavior(brick_behavior)
  }
```

- Create a swift file “setting” to implement customization options; no. of bricks, ball size and ball speed.
- Build and run the app.

Run

1. Click the “Click to play.”
2. Tap on the screen to bounce the ball.
3. Drag the paddle to hit ball when comes toward the paddle.
4. Click setting to customizing the game.

Setting

1. Enter the no. of rows and click “ok.” The effect of this setting is only available when new game begins.
2. Click slow, medium and high button to increase the speed.
3. Click small, medium and big to set the size of ball, The effect of this setting is only available when new game begins.