Michael Gasparovic
Professor Silvestre Guenole
iOS Development

# Assignment 5

## Architecture

- The overall architecture is a tabbed application with two outlets
  - Single Page View
    - Game View
      - Ball Objects
        - UICollisionBehavior
        - UIPushBehavior
        - UIDynamicItemBehavior
      - Brick Views
      - Paddle Views
      - “New Game” Button
      - UIDynamicAnimator
  - UIViewController
    - Slider Cell
    - Stepper Cell
    - Segmented Cell

## Game View

- The main game view houses the entirety of the game within it
- As you can see above, it holds each of the major parts of the game as objects or renders them as subviews to the gameView outlet
- When the view appears the user is prompted to start a new game
- Once the new game outlet action is activated, the build game function is called
  - Build game calls
    - Create balls
      - Create balls loops through the number of balls (obtained from the settings object) putting it’s center at the center of the screen, rounding the square, setting it’s background color to blue, adding the view to the game view, and add it to the global array of ball objects
    - Create Paddle
      - The paddle view is made, center is set to the center of the screen at the bottom (but above the menu bar), sets the background color to black, and adds the paddle to all the ball collider objects
    - Create Bricks
      - The variables such as health and number of rows are obtained, then we loop through the columns and rows, in each creating a brick object in iterative locations and setting their health accordingly, as well as adding the brick collider object to all the balls
    - Add Behaviors

- Here we just declare the animator object and add each ball to the object's sub behavior list
- This function was left over from when I had all the collider and pusher objects within the game view

```
func buildGame() {
    self.createBalls(num: self.settings.numBalls)
    self.createPaddle()
    self.createBricks(columns: 5)

    self.addBehaviors()

    self.newGameButtonOutlet.isHidden = true
}
```

- When a collision is detected
  - The function detects an identifier (which means that it's a paddle or a brick hit)
    - Paddle hits are ignored
    - Brick hits are registered and the health of the brick is decremented and color changed or the brick is removed entirely
  - Otherwise the ball is hitting an edge
    - If that edge is the bottom edge, the ball is removed and if there are no balls left destroy game is called
      - Destroy game, removes all the behavior objects and removes all the subviews from the game view and the player is just presented with the new game button

```
private func destroyGame() {
    if self.animator != nil {
        self.animator.removeAllBehaviors()
    }
    for view in gameView.subviews {
        if ((view as? UIButton) != nil) {
            continue
        }
        view.removeFromSuperview()
    }
    self.newGameButtonOutlet.isHidden = false
}
```

**Ball Objects**

- Each ball object houses a series of behavior objects and a settings object
- Once the object is created
  - The view passed in is set to the view in the ball object
  - The push behavior is created

```
class Ball: UIDynamicBehavior {

    var ball: UIView!

    private var collider: UICollisionBehavior!
    private var push: UIPushBehavior!
    private var behavior: UIDynamicItemBehavior!

    private let settings = Settings()
```

    - The angle is randomly generated
    - The speed is gotten from settings
  - The collider object is created and the delegate is set to the delegate passed in
  - The dynamic item behavior (which dictates how the object would move and interact with other objects) is created
- The ball object also contains an add and remove collider function to add and remove views from the collider object contained within the Ball object
- There is also a random angle object that is used to set a random angle (this is used when a player taps on the screen)

```
private var balls: [Ball]!
private var bricks: [String: UIView]!
private var brickHealth: [String: Int]!
private var maxHealth: Int!
private var paddle: UIView!
```

**Paddle/Brick View**

- These objects were set to be as simple as possible, as such they are plainly views that are added to the collider object of each ball

**Settings Object**

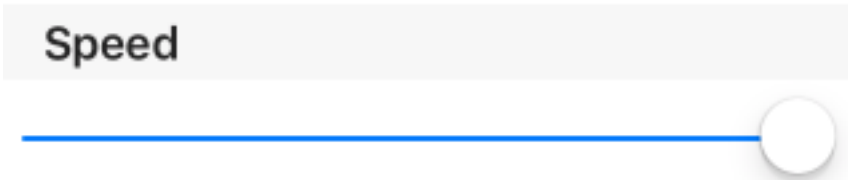
- The settings object is a thin wrapper around user defaults
  - It contains:
    - Speed variable
    - Number of Balls variable
    - Number of Brick Rows Variable
- Having this class allows for the code to look nicer
  - UserDefaults.standard.double(forKey: “speed”)
  - settings.speed

```
class Settings {

    private let user = UserDefaults.standard

    var speed: Double {
        get { return user.double(forKey: "Speed") }
        set { user.set(newValue, forKey: "Speed") }
    }
    var numBalls: Int {
        get { return user.integer(forKey: "Num_Balls") }
        set { user.set(newValue, forKey: "Num_Balls") }
    }
    var numBrickRows: Int {
        get { return user.integer(forKey: "Num_Brick_Rows") }
        set { user.set(newValue, forKey: "Num_Brick_Rows") }
    }
}
```

**Cell Types**

- There are three primary cell types, each with it's own simple class
  - Slider Cell
    - This cell controls the speed of the ball and it's range is from .02 to .08

```
class SliderCell: UITableViewCell {

    let settings = Settings()

    @IBOutlet weak var sliderControl: UISlider!

    @IBAction func changeSpeed(_ sender: UISlider) {
        self.settings.speed = Double(sender.value)
    }

    override func layoutSubviews() {
        self.sliderControl.value = Float(self.settings.speed)
    }
}
```



- Segmented Cell
  - This cell controls the number of balls on the screen, allowing the player to pick from 1…5

```
class StepperCell: UITableViewCell {

    let settings = Settings()

    @IBOutlet weak var numRows: UILabel!

    @IBOutlet weak var stepperControl: UIStepper!

    @IBAction func changeNumRows(_ sender: UIStepper) {
        self.settings.numBrickRows = Int(sender.value)
        self.numRows.text = "\(Int(sender.value))"
    }

    override func layoutSubviews() {
        var value = self.settings.numBrickRows
        if value > 9 { value = 9 }
        else if value < 0 { value = 0 }
        self.stepperControl.value = Double(value)
        self.numRows.text = "\(value)"
        self.stepperControl.maximumValue = 9.0
        self.stepperControl.minimumValue = 1.0
    }
}
```

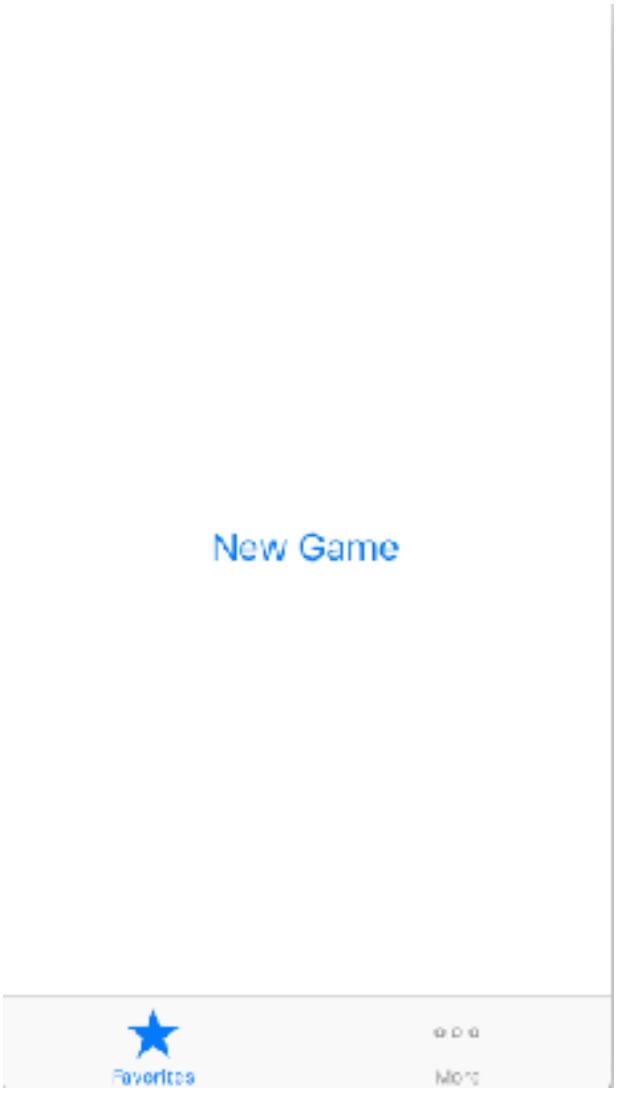

- Stepper Cell
  - This cell is a plus/minus stepper cell that controls the number of rows that will be displayed on the screen

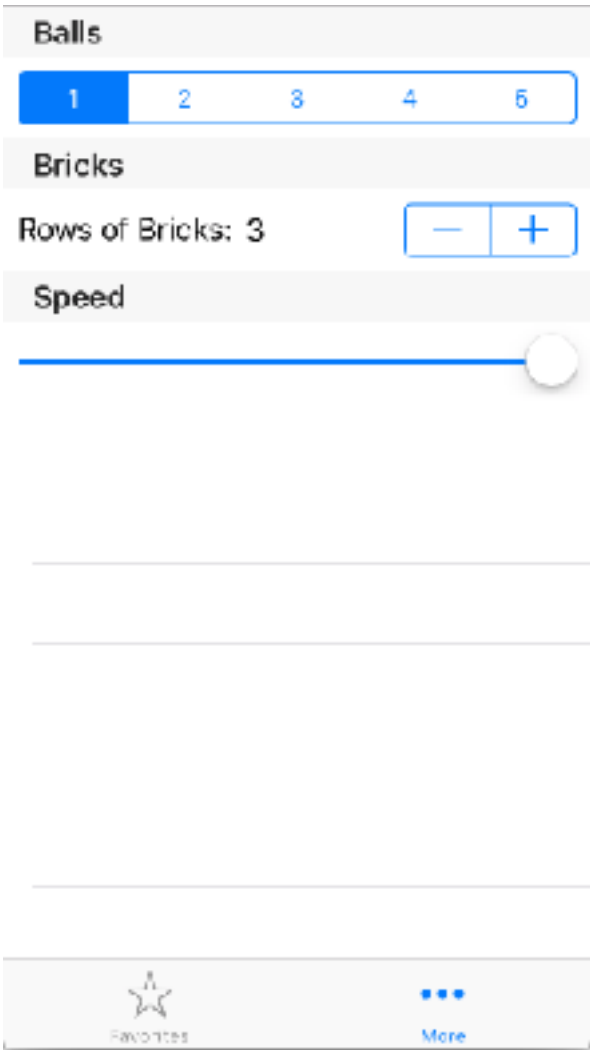
```
class SegmentedCell: UITableViewCell {

    let settings = Settings()

    @IBOutlet weak var segmentedControl: UISegmentedControl!

    @IBAction func changeNumberBalls(_ sender: UISegmentedControl) {
        self.settings.numBalls = sender.selectedSegmentIndex + 1
    }

    override func layoutSubviews() {
        self.segmentedControl.selectedSegmentIndex = self.settings.numBalls - 1
    }
```

## Other Notes

- I originally tried to fit all the behavior models on the game view, but found this wouldn't work (reason unknown) but moving all the behaviors to the ball object seemed to make it work
- This project made me realized why there are cell id's, as you may design a variety of cells and then programmatically call them as necessary
- Settings TVC is straight forward, which is why I left it out of the sub descriptions
- I was able to remove the status bar at the top by adding


"application.isStatusBarHidden = true" to the app delegate


New Game
Favorites
More


Balls
1
2
3
4
5
Bricks
Rows of Bricks: 3
Speed
Favorites
More