Assignment V: Animation

Objective

In this assignment, you will let your users play the game Breakout. Your application will not necessarily have all the scoring and other UI one might want, but it will allow some customization of the game.

Materials

- You must get this assignment working on an actual device, therefore you will need to be a member of a Developer Program (either the free University Developer Program or the $99/year normal Apple Developer Program).

Required Tasks

1. Create a straightforward Breakout game using Dynamic Animator. See the image in the Screen Shots section below to get a general idea of the game, but your UI should **not** look exactly like the Screen Shot. Creativity is encouraged and will be rewarded.
2. When a brick is hit, some animation of the brick must occur. For example, the brick might flip over before fading out or it might flash another color before disappearing, etc. Show us that you know how to animate changes to a `UIView`.
3. In addition to supporting a pan gesture to move the game's paddle, you must support a tap gesture which pushes the bouncing ball in a random direction an appropriate (i.e. noticeable, but not game-destroying!) amount.
4. When all the bricks have been eliminated (or the game is otherwise over), put up an alert and then reset the bricks for the next game.
5. Your game should be designed to support at least 4 different variables that control the way your game is played (e.g. number of bricks, ball bounciness, number of bouncing balls, a gravitational pull, special bricks that cause interesting behavior, etc.).
6. Use a tab bar controller to add a second tab to your UI which contains a static table view with controls that let the user set these 4+ different variables to meaningful game-play values. Your game should start using them immediately (i.e. as soon as you click back on the main game play tab).
7. This game-play settings MVC must use at least **one of each** of the following 3 iOS classes: `UISwitch`, `UISegmentedControl` and `UIStepper` (you may substitute `UISlider` for the `UIStepper` if that's more appropriate to your setting).
8. Your game-play configurations must persist between application launchings.
9. Your application should work on both iPhone and iPad. It is up to you to decide what differences exist between the two platforms (if any).

Hints

1. This assignment is intentionally vague about the feature set of your game to make room for you to show us some creativity. Being a good iOS developer requires a lot of creativity in addition to the programming skills.
2. Don't overcomplicate the implementation of your Breakout game itself (you have plenty of other work to do in this assignment).
3. For example, to simplify things, the bouncing ball, the bricks and the paddle can all simply be filled-in rectangles (so you won't even need to subclass `UIView` for any of those things, just use `UIView`'s `backgroundColor` property).
4. It is highly recommended that you manage the collision behavior with the bricks using **boundaries** in a `UICollisionBehavior` rather than having those brick-drawing views actually participate in the collisions themselves. You'll have to keep the brick boundaries in sync with the `frame`s of the bricks, but the bricks don't move once they are laid out for a given `bounds` of your MVC's View, so that should be very easy. This is only a hint, not a required task.
5. The bouncing ball, on the other hand, almost certainly does want to be a `UIView` that is participating in the collisions (with the brick, paddle and wall boundaries). That's because the bouncing ball is moving all over the place and you want the physics engine to be able to control its behavior.
6. The paddle, even though it moves in response to a pan gesture, probably also wants to be a boundary (one that you are constantly removing and adding to the `UICollisionBehavior`). Otherwise, when the ball hits the paddle, the paddle might want to move in response and it should only move in response to the pan gesture.
7. Note that the required tasks say nothing about keeping score. See Extra Credit.
8. You might find it good for the clarity of your code to create a `BreakoutBehavior` subclass of `UIDynamicBehavior` which uses `addChildBehavior` to add the collision behavior, dynamic item behavior, et. al., to itself.
9. Your `BreakoutBehavior` could manage lots of behavior-oriented stuff like the push, the brick and paddle boundaries, and the behavior of the bouncing ball(s).
10. It could even manage some of the `UIView`s that are syncing up with the boundaries (the paddle and bricks). Again, this is a hint, not a required task.
11. You probably won't want the reference view of your dynamic animator to be your MVC's top level `view` property. Because what happens when you want to put other UI around your game (e.g. score, etc.)? Do a little bit of architecting before you dive in to decide what code is going to be in your `BreakoutBehavior`, what code is going to be in your Controller, and what code might be in a `UIView` subclass. There are lots of acceptable solutions, but try to have a plan.

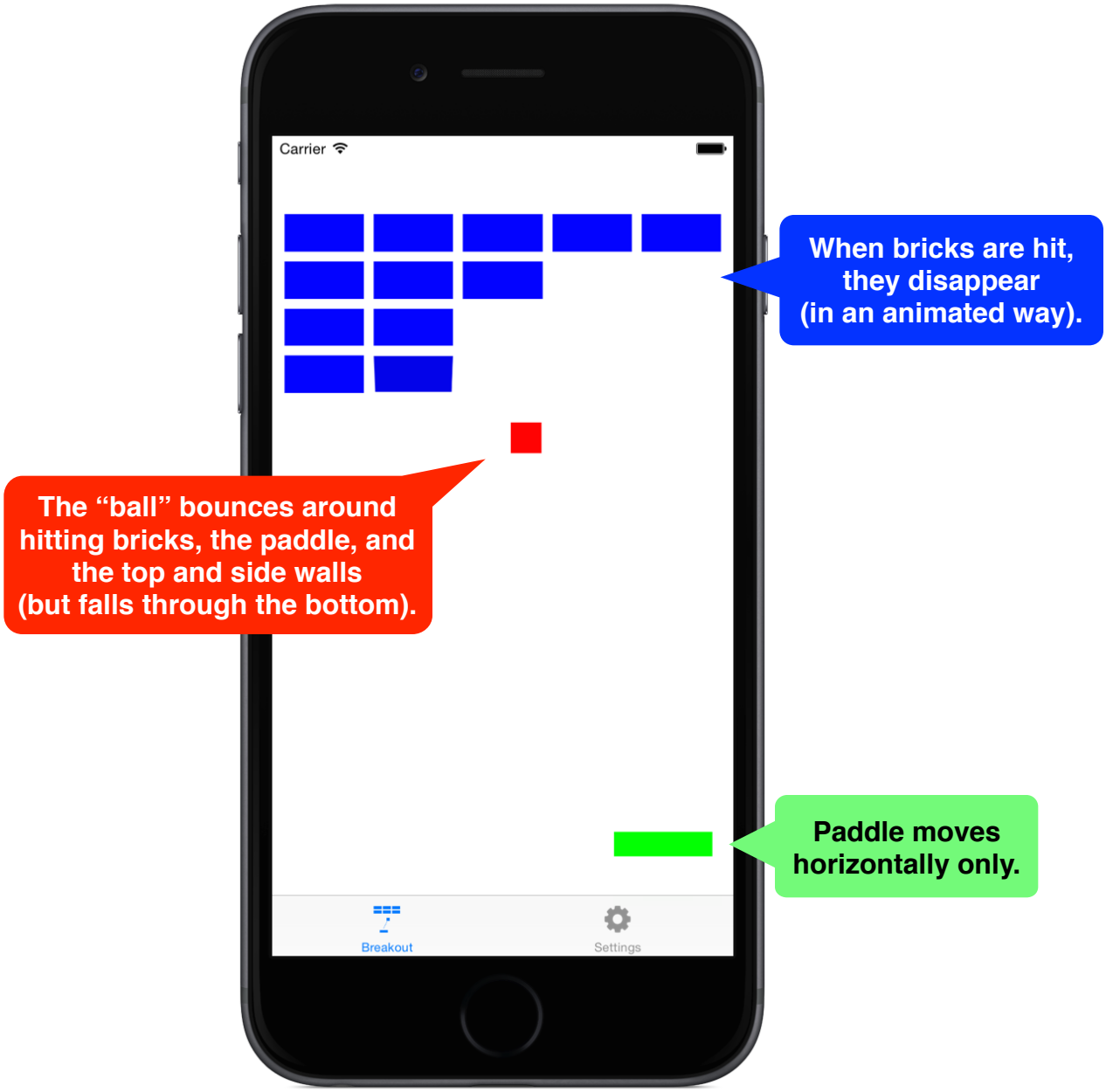
12. Since the number of bricks is likely going to be configurable (i.e. not fixed), you will almost certainly be creating and adding the brick (and ball and paddle for that matter) `UIViews` in code rather than through your storyboard (that's one of the things this assignment is intended to give you experience doing).
13. You will find out about collisions between the bouncing ball and brick boundaries with the `UICollisionBehavior`'s `collisionDelegate`.
14. When you add a boundary to a `UICollisionBehavior`, you get to pass along an identifier that will be fed back to you when the delegate gets notified of a boundary collision. That identifier is an `NSCopying` (i.e. an object that implements the `NSCopying` protocol). `NSString` and `NSNumber` both implement that protocol (and, because of bridging, that means you can pass a `String` or an `Int`, `Double`, etc.). But when that `NSCopying` is passed back to you in your `collisionDelegate`, you'll have to cast it from `NSCopying` to a `String` or `NSNumber` with `as` or `as?` in order to use it.
15. The push that your tap gesture must generate is just an instantaneous `UIPushBehavior`, nothing more. You'll have to set it up with an appropriate magnitude for the size of your bouncing ball.
16. The push behavior won't do any harm (other than wasting memory) if you leave it attached to the animator after it's done pushing, but removing it via its own `action` property would be better. Be careful to break any possible memory cycle using an `unowned` or `weak` capture list.
17. You can control things like the elasticity, friction, resistance and rotation of animated items using a `UIDynamicItemBehavior` instance. Simply create one, add any items to it that you want to have those attributes, then add the behavior to the animator. You can set/reset the attributes at any time. Be careful not to have multiple of these kinds of behaviors fighting to set the attributes of something that's being animated, though.
18. It's a little bit tricky to find out when the bouncing ball has left the game (either by getting past the paddle and falling off the bottom or by moving so fast that it jumps completely out of the boundaries—yes, that can happen!). The place to detect this is probably in the `action` of your `UICollisionBehavior`. Just check in there to see if any of the `items` (the balls) being managed by the `UICollisionBehavior` are outside the `bounds` of the `UICollisionBehavior`'s `dynamicAnimator`'s `referenceView`. If so, perhaps you could remove the item from the behavior (and also perhaps remove the `UIView` itself from its `superview` depending on how you are organizing your code).
19. You'll have to decide what to do with the ball when a game first starts or when a ball gets put back in play after going outside the `bounds` of the `referenceView` (or even what to do with the ball when autorotation happens). A simple thing to do is to place it **right in front of the paddle**. That way, when the user taps to push, most directions it would head will either go up toward the bricks or bounce right off the paddle and then shoot up there (and, if it is in the middle of moving and you

autorotate, it'll either keep moving toward the bricks or a boundary or smash right into the paddle immediately and bounce off).

20. Speaking of autorotation, you'll probably want to position your bricks manually in `viewDidLayoutSubviews` (i.e. don't try to use autolayout for that).
21. Sometimes the physics of the ball and its collisions will leave the ball bouncing around somewhere where it's not ever going to get back to the paddle! That's okay, that's why the tap-to-push feature is a required task. It will get your user out of those conditions.
22. Some physics that you might set up might cause the ball to go completely crazy (i.e. accelerate to very high speeds or something). It's probably best to try to limit the user's ability to adjust game settings to mostly avoid these crazy situations.
23. Be careful not to move your paddle boundary right on top of a bouncing ball or the ball might get trapped **inside** your paddle.
24. Switching over to Settings in the middle of playing breakout will probably cause the ball to go out of play while you're away. That's fine, but see the Extra Credit for a better option.
25. You can feel free to put some other limitations on the use of your tap-to-push if you think this feature is too much like cheating (like maybe that feature is disabled for some amount of time after a brick is hit or it has a "cooldown" or something).
26. You might want to make the bezier path boundary for your paddle be an oval (even if the paddle itself still looks like a rectangle). It makes the bouncing ball come off the paddle more interestingly.
27. A fun configurable is having "special bricks" that, when hit, cause other things to happen. And nothing says that a brick has to disappear on the very first hit.
28. Don't ever put a `UIView` that is size `(0, 0)` into a behavior. It will not like that.
29. And don't ever add a behavior to an animator that is controlling the behavior of a `UIView` that is not in that animator's `referenceView`'s view hierarchy somewhere.
30. If you ever have to move a `UIView` that the animator "has a hold of," don't forget to call `updateItemUsingCurrentState` in the animator.
31. All behaviors know the dynamic animator that is animating them (they have a property to get it).
32. And all dynamic animators know the `referenceView` in which they are animating.
33. If you are in the habit of writing non-generic solutions to problems (i.e. you just keep typing until it does what you want!), you'll want to break that habit for this assignment because you want your solution to be as "configurable" as possible so that you can have more flexibility to create your game-play settings MVC. Hard-wiring the way things work internally in your classes will make that much harder.

Screen Shot

It is impossible to overemphasize how important it is to understand that this screen shot is **not** a Required Task. In fact, it's a Required Task to show some creativity and **not** just copy this. We rarely include screen shots because they bias you toward doing things in a certain way. That is not the intent of this screen shot. It is just here for those of you who feel like you absolutely cannot figure out what is being asked of you from a text description alone. So … please don't submit a solution that looks exactly like this …



Things to Learn

Here is a partial list of concepts this assignment is intended to let you gain practice with or otherwise demonstrate your knowledge of.

1. `UIDynamicAnimator`
2. `UICollisionBehavior`
3. `UIDynamicItemBehavior`
4. `UIPushBehavior`
5. `UIGravityBehavior`
6. `UIView` animation (i.e. `transitionWithView` and/or `animateWithDuration`)
7. Adding `UIViews` in code (rather than storyboard)
8. Gestures
9. `NSUserDefaults`
10. Closure memory cycle avoidance
11. `UIAlertController`
12. Static `UITableView`
13. `UISwitch`
14. `UIStepper`
15. `UISlider`
16. `UISegmentedControl`

Evaluation

In all of the assignments this quarter, writing quality code that builds without warnings or errors, and then testing the resulting application and iterating until it functions properly is the goal.

Here are the most common reasons assignments are marked down:

- Project does not build.
- Project does not build without warnings.
- One or more items in the Required Tasks section was not satisfied.
- A fundamental concept was not understood.
- Code is visually sloppy and hard to read (e.g. indentation is not consistent, etc.).
- Your solution is difficult (or impossible) for someone reading the code to understand due to lack of comments, poor variable/method names, poor solution structure, long methods, etc.
- UI is a mess. Things should be lined up and appropriately spaced to "look nice."
- Public and private API is not properly delineated.

Often students ask "how much commenting of my code do I need to do?" The answer is that your code must be easily and completely understandable by anyone reading it. You can assume that the reader knows the SDK, but should not assume that they already know the (or a) solution to the problem.

Extra Credit

We try to make Extra Credit be opportunities to expand on what you've learned this week. Attempting at least some of these each week is highly recommended to get the most out of this course.

1. Use sophisticated Dynamic Animation. For example, you might find a creative way to use the `action` method in a behavior or use something like linear velocity in your calculations.
2. As mentioned above, creativity will be rewarded, especially interesting game-play settings.
3. Keep score in your game. Give points for whatever you think makes sense, but hopefully playing "more skillfully" results in a higher score.
4. Do some cool artistic design in your user-interface (either by drawing or using images).
5. Pausing your game when you navigate away from it (to go to settings) is a bit of a challenge (because you basically have to freeze the ball where it is, but when you come back, you have to get the ball going with the same linear velocity it had). Give it a try. It's all about controlling the linear velocity of the ball.
6. Integrate the accelerometer into your application somehow (maybe real-life gravity affects the flight of the bouncing ball?). Check out the documentation for the `CoreMotion` framework. We will be covering `CoreMotion` later in the quarter, but this could still be a good exercise for practice learning something without benefit of a lecture explanation (good practice for your final project and maybe you even want to use `CoreMotion` in your final project and can't wait for it to get covered in lecture). Plus it's just kind of a cool feature in this app.