Задание V: Animation (Анимация)

Цель:

В этом Задании вы должны предоставить пользователям возможность играть в игру **Breakout**. Вашему приложению не обязательно иметь “счет” или другие элементы пользовательского интерфейса, которые кому-то хотелось бы иметь, но оно должно позволять пользователю настраивать игру.

Материалы

Вы должны заставить работать это приложение на реальном приборе, поэтому вам необходимо быть участником Developer Program (или бесплатной University Developer Program или нормальной Apple Developer Program за $99/год)

Обязательные пункты задания

1. Создайте простую **Breakout** игру, используя **Dynamic Animator**. Смотрите примерное изображение этой игры в разделе **Screen Shots (Экраны приложения)**, чтобы получить общую идею этой игры, но ваш UI не должен выглядеть в точности, как в **Screen Shots (Экраны приложения)**. Креативность приветствуется и будет вознаграждена.

2. Когда “кирпич” ударяется мячиком, то должна происходить какая-то анимация. Например, “кирпич” может переворачиваться или вспыхивать другим цветом, прежде чем исчезнет и т.д. Покажите нам, что вы знаете как анимировать изменения в **UIView**.

3. В дополнение к поддержке жеста **pan** для перемещения “ракетки”, вы должны поддерживать жест **tap**, который “толкает” ударный мячик в случайном направлении с определенной силой (то есть заметной, но не разрушающей игру силой!).

4. Когда все “кирпичи” уничтожены (или игра закончилась по другой причине), дайте сигнал (alert) пользователю об окончании игры и переустановите

“кирпичи” для следующей игры.

5. Ваша игра должна быть сконструирована так, чтобы поддерживать не менее 4-х различных параметров, которые управляют способом, каким ведется игра (например, число “кирпичей”, ударная сила мячика, число ударных мячиков, наличие гравитационного поля, “специальные кирпичи”, которые вызывают интересное поведение и т.д.).

6. Используйте **Tab Bar Controller**, чтобы добавить еще одну закладку к вашему UI, которая содержит статическую таблицу **Table View** с управляющими элементами, которые позволят пользователю установить этим 4+ различным параметрам игры значения. Ваша игра должна начать использовать их немедленно (как только вы кликните на кнопку возврата к главной закладке игры).

7. MVC для настройки этой игры должен использовать по крайней мере по одному из следующих 3-х iOS классов: **UISwitch**, **UISegmentedControl** и **UIStepper** (вы можете использовать **UISlider** вместо **UIStepper**, если считаете, что это больше подходит вашим настройкам).

8. Конфигурация вашей игры должна сохраняться постоянно между запусками приложения.

9. Ваше приложение должно работать как на iPhone, так и на iPad. Вам решать какое различие будет между двумя платформами (если вообще будет).

Подсказки (Hints)

1. Это задание намеренно не имеет четко определенных возможностей вашей игры, давая вам простор для демонстрации своей креативности. К хорошему разработчику iOS предъявляются требования иметь не только хорошие программисткие навыки, но и креативные способности.

2. Не переусложните реализацию вашей игры Breakout (у вас много другой работы в этом Задании).

3. Например, для упрощения ситуации ударный мячик, “кирпичи” и “ракетку” вы можете просто сделать закрашиваемыми прямоугольниками (вам даже не нужно создавать **subclass** **UIView** для всех этих элементов, просто используйте цвет фона этих **UIViews** - свойство **backgroundColor**).

4. Настоятельно рекомендуется управлять “столкновениями” с “кирпичами” с помощью границ (**boundaries**) в поведении **UICollisionBehavior**, а не заставлять такие нарисованные “кирпичи” принимать участие во взаимных столкновениях.

Вы должны синхронизировать границы (**boundaries**) “кирпичей” с их нарисованными **frames**, “кирпич” не должен перемещаться после того как он был размещен в пределах границ View вашего MVC, так что это сильно облегчает задачу. Но это только подсказка, а не обязательный пункт Задания.

5. Прыгающий мячик, будучи **UIView**, напротив, должен принимать участие во всех столкновениях (с “кирпичами”, с “ракеткой”, с границами игрового пространства). Потому что прыгающий мячик передвигается по всему игровому пространству, и его поведение управляется физическими принципами, то есть движком **Dynamic Animation**.

6. “Ракетка”, даже если она перемещается с помощью жеста **pan**, возможно, также должна быть границей (boundary) (границей, которая постоянно удаляется (removing) и добавляется (adding) к поведению **UICollisionBehavior**). В противном случае “ракетка” будет двигаться в ответ на удар мячика, а нам нужно, чтобы она двигалась только с помощью жеста **pan**.

7. Заметьте, что в обязательгых пунктах задания ничего не говорится о том, чтобы отображать счет игры (score). Смотри Дополнительные пункты Задания.

8. Возможно, для более понятного кода будет целесообразно создать **BreakoutBehavior** **subclass** класса **UIDynamicBehavior**, который использует метод **addChildBehavior** для добавления поведения “столкновение”, “мета” поведений динамических объектов и других “поведений” к комплекному поведению **BreakoutBehavior**.

9. Ваше поведение **BreakoutBehavior** могло бы управлять множеством вещей, связанных с поведенческой концепцией, таких как толчок (push), границы “кирпичей” и “ракетки”, а также поведением прыгающего мячика (мячиков).

10. Он мог бы управлять некоторыми **UIViews**, которые синхронизируются с грвницами (“ракетки” и “кирпичей”). И опять, это только подсказка, а не обязательный пункт Задания.

11. Возможно, вы не захотите, чтобы **referenceView** вашего динамического аниматора было бы топовым **view** вашего **MVC**. Потому что что произойдет, если вы захотите разместить другие элементы пользовательского интерфейса вашей игры (например, счет и т.д.)? Помыслите немного в архитектурном плане проектирования вашей игры, прежде чем погрузитесь в решение того, какой код должен быть в **BreakoutBehavior**, какой код должен быть в **Controller**, и какой код должен быть в **subclass** **UIView**. Существует бессчетное количество приемлемых решений, но у вас должен быть план.

12. Так как число “кирпичей” похоже должно быть конфигурируемым (не

фиксированным), вам определенно нужно будет уметь создавать и добавлять для “кирпича” (а также для мячика и “ракетки”) **UIViews** в коде, а не на storyboard (это одна из важных идей этого Задания, которая должна дать вам опыт такого создания).

13. Вы будете обнаруживать столкновения между прыгающим мячиком и границами “кирпича” с помощью делегата **collisionDelegate** класса **UICollisionBehavior**.

14. Когда вы добавляете границу (boundary) к **UICollisionBehavior**, вы должны передать еще и идентификатор, который вернется к вам, когда делегат уведомит вас о том, что произошло столкновение с этой границей. Этот идентификатор должен быть **NSCopying** (то есть объектом, котрый реализует **NSCopying** протокол). **NSString** и **NSNumber**, оба реализуют этот протокол (а вследствии взаимозаменяемости (**bridging**), это означает, что вы можете передавать **String** или **Int**, **Double** и т.д). Но, когда этот **NSCopying** вернется к вам назад в вашем **collisionDelegate**, то для того,чтобы его использовать, вам нужно будет выполнить “кастинг” от **NSCopying** к **String** или **NSNumber** с помощью **as** или **as?**.

15. “Толчок”, который должен генерировать ваш жест **tap**, представляет собой просто мгновенное (instantaneous) **UIPushBehavior** и больше ничего. Вы должны задать ему определенное значение для соответствующего размера прыгающего мячика.

16. Поведение “толчок” не причиняет никакого вреда (кроме того, что напрасно занимает память), если вы оставите его подсоединенным к аниматору после того, как “толчок” выполнен, так что удаление его через его собственное свойство **action** было бы желательно. Будьте внимательны, если возникнет необходимость в разрыве циклических ссылок памяти с использование **unowned** или **weak** списка захвата (**capture list**).

17. Вы можете управлять такими вещами, как упругость (elasticity), трение (friction), сопротовление (resistance) и вращение (rotation) анимируемых объектов. используя экземпляр класса **UIDynamicItemBehavior**. Просто создайте один экземпляр, добавьте к нему элементы, которые вы хотите наделить этими атрибутами, и добавьте поведение к аниматору. Вы можете устанавливать / переустанавливать эти атрибуты в любое время. Будьте внимательны и не создайте множество таких поведений, сражающихся за установку атрибутов того, что уже анимируется.

18. Достаточно хитроумным способом можно определить, когда прыгающий мячик покинул игру (либо не попав на “ракетку” и упав вниз, либо передвигаясь так быстро, что выпрыгнул полностью из всех границ - да, такое может произойти!).

Место, где это можно определить - это **action** вашего **UICollisionBehavior**. Просто проверяйте в нем, вышли ли динамические объекты (мячики), управляемые **UICollisionBehavior**, за пределы области **referenceView** аниматора, которому принадлежит **UICollisionBehavior**. Если да, то вы могли бы убрать этот динамический объект из "поведения" (а также убрать его **UIView** из его **superview** в зависимости от того, как вы организуете свой код).

19. Вам нужно решить, что делать с мячиком, когда игра впервые стартовала или когда мячик вернулся в игру после ухода за пределы **referenceView** (или даже что делать с мячиком при вращении прибора, то есть когда происходит **autorotation**). Самая простая вещь, которую вы можете делать - это поместить мячик **прямо впереди "ракетки"**. При таком способе расположения, если пользователь будет применять жесты **taps** для толчков, то преимущественными направлениями будут направления вверх к "кирпичам" или удар непосредственно о "ракетку" и затем быстрое движение вверх (а если он в середине движения, а вы сделали автовращение, он должен продолжать двигаться к "кирпичам" или границам или удариться изо всех сил о "ракетку" и отскочить).

20. Говоря об автовращении (**autorotation**), возможно, вы захотите позиционировать "кирпичи" вручную в методе **viewDidLayoutSubviews** (то есть, не пытайтесь использовать для этого **Autolayout**).

21. Иногда физика мячика и его столкновения будут заставлять мячик прыгать там, где он никогда не сможет вернуться на "ракетку"! Это нормально, именно для этого дана возможность "толчка" с помощью жеста **tap** как обязательный пункт Задания. Это выведет мячик из этого состояния.

22. Иногда некоторая физическая среда, которую вы могли бы установить, заставляет мячик бегать как сумасшедший (то есть ускрояя его до очень высоких скоростей или что-то подобное этому). Возможно, будет лучше ограничить возможности пользователя по таким установкам параметров игры, которые приводят к таким сумасшедшим ситуациям.

23. Будьте внимательны и не передвигайте границу вашей "ракетки" непосредственно на верхнюю часть прыгающего мячика. Иначе мячик может угодить в ловушку и попасть внутрь "ракетки".

24. Переключение на Установки (Settings) в середине игры может заставить мячик уйти из игровой зоны во время вашего отсутствия. Это нормально, но лучшая возможность предложена в **Дополнительных пунктах**, которые нужно посмотреть.

25. Вам предоставлена полная свобода для установления других ограничений на

использование вашего жеста **tap,** чтобы “толкнуть” мячик. Если вы думаете, что эта “фишка” слишком похожа на мошеничество, то можно запретить ее использовать на некоторое время после удара о “кирпич” или представить себе, что она “охлаждается” или что-то еще.

26. Возможно, вы захотите сделать **BezierPath** границу для вашей “ракетки” овалом (даже если “ракетка” все еще выглядит прямоугольником). Это заставит прыгающий мячик отскакивать от “ракетки” более интересно.

27. Забавно иметь в конфигурации игры “специальные кирпичи”, когда их ударяют, то это заставляет происходить другие вещи. И ничего не говорится о том, что “кирпич” должен исчезать при первом же ударе.

28. Никогда не помещайте **UIView** размером size (0, 0) в “поведение”. Оно этого не любит.

29. Никогда не добавляйте “поведение” к аниматору, который управляет поведение **UIView**, которое не принадлежит view иерархии **referenceView** аниматора.

30. Если вам придется самому перемещать **UIView**, который “удерживается” аниматором, не забудьте вызвать метод **updateItemUsingCurrentState** в аниматоре.

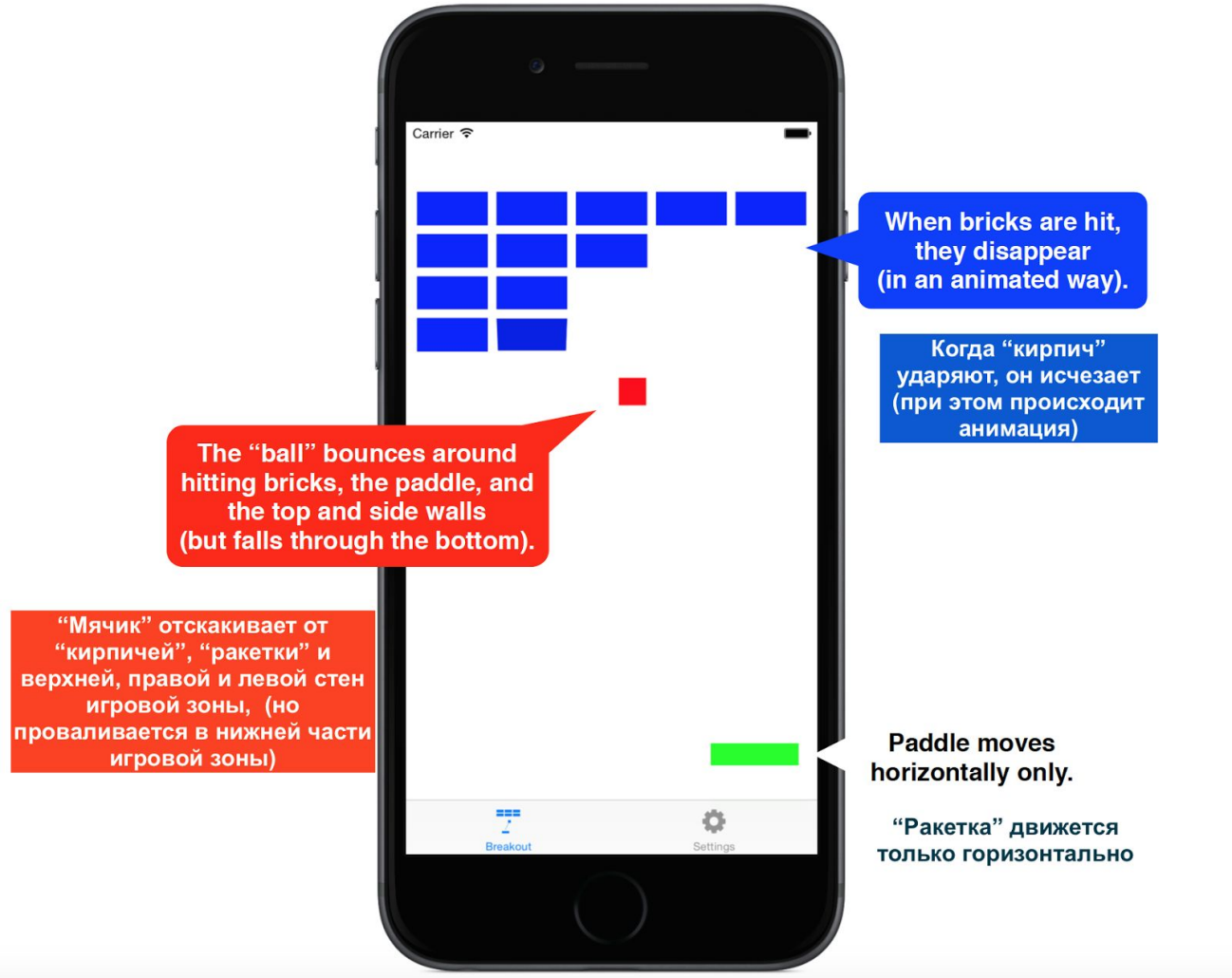
31. Все “поведения” знают динамический аниматор, который их анимирует (у них есть свойство для его получения).

32. И все динамические аниматоры знают **referenceView**, а которых они анимируют.

33. Если у вас есть привычка писать не обощенные (non-generic) решения проблемы (например, просто печатать, пока не получится то, что нужно) , вам следует отойти от этой првычки в этом Задании, потому что вы захотите сделать свою игру насколько возможно “конфигурационной” и это заставит вас проявить максимум изобретательности при создании MVC, отвечающего за настройки. Жесткая привязка некоторых значений элементов вашего класса может сделать гибкость настройки параметров игры трудно реализуемой.

Screen Shots (Экраны приложения)

Важно понимать, что приведенный в этом разделе примерный вид экрана - это не Обязательный пункт Задания. Фактически, Обязательным пунктом Задания является показать некоторую креативность, а не просто скопировать этот вид экрана. Мы редко включаем примерные виды экранов приложений, потому что это направляет вас к тому, чтобы сделать именно то, что приведено на этих экранах. Не эта цель того, что мы здесь представили. Это приведено для тех, кто только из текста Задания абсолютно не может понять о чем идет речь. Так что … пожалуйста, не делайте ваше решение абсолютно похожим на это …



Что нужно изучать

Это частичный список концепций, в которых это задание увеличивает ваш опыт работы или продемонстрирует ваши знания.

- UIDynamicAnimator
- UICollisionBehavior
- UIDynamicItemBehavior
- UIPushBehavior
- UIGravityBehavior

- UIView анимация (например, transitionWithView и/или animateWithDuration)
- Добавление UIViews в коде (а не на storyboard)
- Жесты (Gestures)
- NSUserDefaults
- Как избежать циклических ссылок в памяти в замыканиях
- UIAlertController
- Static UITableView
- UISwitch
- UIStepper
- UISlider
- UISegmentedControl

Оценка (Evaluation)

Во всех заданиях требуется написание качественного кода, на основе которого строится приложение без ошибок и предупреждений, следовательно вы должны тестировать полученное приложение до тех пор, пока оно не начнет функционировать правильно согласно поставленной задачи.

Приведем наиболее общие соображения, по которым задание может быть отклонено:

- Приложение не создается.
- Приложение не создается без предупреждений.
- Один или более пунктов в разделе **Обязательные задания (Required Tasks)** не выполнены.
- Не понята фундаментальная концепция задания .
- Код - небрежный или тяжелый для чтения (например, нет отступов и т.д.).
- Ваше решение тяжело (или невозможно) прочитать и понять из-за отсутствия комментариев, из-за плохого наименования методов и переменных, из-за непонятной структуры и т.д.
- UI в полном беспорядке. Элементы UI должны быть выравнены и размещены с зазором для того, чтобы хорошо выглядеть.
- **Public** и **private** API разграничены неправильно.

Часто студенты спрашивают: “Сколько комментариев кода нужно сделать?” Ответ - ваш код должен легко и полностью быть понятным любому, кто его читает. Вы можете предполагать, что читатель знает SDK, но вам не следует предполагать, что он уже знает решение проблемы.

Дополнительные задания (Extra Credit)

Мы пытались сделать Дополнительные задания таким образом, чтобы это расширяло ваши знания в той области, которую вы изучали на этой недели. Попытка выполнить хотя бы некоторые из них настоятельно рекомендуется для получения наибольшего эффекта от этого курса.

1. Используйте сложный **Dynamic Animation**. Например, вы можете найти креативный способ использовать **action** метод в “поведении” или использовать что-то наподобие линейной скорости (**linearVelocity**) в ваших вычислениях.

2. Как упоминалось выше, креативность вознаграждается, особенно интересные установки (**settings**) параметров игры.

3. Поддерживайте Счет в вашей игре. Давайте очки за то, что вы считаете значимым, при этом более искусная игра должна поошряться более высоким счетом.

4. Сделайте какой-нибудь художественный дизайн в пользовательском интерфейсе (либо нарисуйте что-то, либо используйте изображения).

5. Остановка игры во время перехода, например, к установкам иры - это продвинутая возможность, потому что вы должны “заморозить” мячик там, где он есть, но когда вы возвращаетесь, вам нужно запустить мячик с той же самой динейной скоростью, что у него была прежде. Попробуйте сделать это. Все это связано с управлением линейной скоростью мячика.

6. Интегрируйте акселерометр куда-нибудь в ваше приложение (может быть реальное гравитационное воздействие на полет прыгающего мячика?). Посмотрите документацию по **CoreMotion**. Мы будем изучать **CoreMotion** на этом курсе немного позже, но это может быть хорошим упражнением для практического изучения чего-нибудь без объяснений на лекции (хорошая тренировка для финального проекта, и даже может быть вы хотите использовать **CoreMotion** в вашем финальном проекте, и не можете дождаться пока об этом будет рассказано на лекции). Плюс это действительно крутая “фишка” для этого приложения.