Objective-C

Урок 8

Жизненный цикл приложения и контроллера

Обзор жизненного цикла приложения и контроллера на iOS. Заключительный этап в создании игры «Пинг-понг» на Objective-C.

Жизненный цикл приложения

Жизненный цикл

Для написания грамотного кода мы должны знать, чего ожидать от системы. Рассмотрим основные статусы приложения:

1. Не запущено;
2. Неактивно;
3. Активно;
4. Находится в фоне;
5. Приостановлено.

При запуске и при каждом изменении статуса приложение посылает сообщения классу **AppDelegate**, в котором можно обработать эти изменения. Важно понимать, в какой момент и какой метод вызывается.

Изначально приложение находится в статусе «не запущено». При запуске пользователем оно переходит в **Foreground**, где изначально становится неактивным. На этом этапе выполняется код программы, но не обрабатываются события интерфейса. Затем приложение переходит в статус «активно»: выполняется код и обрабатываются все пользовательские события.

При запуске пользователем другого приложения текущее переходит в состояние неактивного, а после него – в **Background**. В этом состоянии код программы выполняется только ограниченное время (как правило, для синхронизации данных). События на данном этапе уже не обрабатываются. На этом этапе можно производить обновление контента, чтобы к моменту возвращения в активное состояние данные были актуальны.

По истечении предоставленного системой времени приложение переходит в статус «приостановлено». Далее система может уничтожить все его данные из памяти и полностью закончить его работу.

Методы жизненного цикла приложения

Рассмотрим подробнее методы **AppDelegate**, с помощью которых можно отслеживать изменения статуса приложения.

1. **Метод успешного запуска.**

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    return YES;
}
```

Он вызывается, когда система полностью подготовила все необходимые данные для работы приложения. Это значит, что по его завершении пользователь сможет приступить к работе с приложением.

2. **Метод перехода в неактивное состояние.**

```
- (void)applicationWillResignActive:(UIApplication *)application {

}
```

Как правило, в этом методе ставятся на паузу активные процессы и останавливается обновление пользовательского интерфейса.

3. **Метод перехода в активное состояние.**

```
- (void)applicationDidBecomeActive:(UIApplication *)application {

}
```

В этом методе необходимо обновить все процессы, которые были остановлены при переходе приложения в неактивное состояние.

4. **Метод перехода приложения в Background.**

```
- (void)applicationDidEnterBackground:(UIApplication *)application {

}
```

Этот метод сохраняет основные данные или состояние приложения, а также запускает процессы по обновлению его контента.

5. **Метод перехода приложения из Background в Foreground.**

```
- (void)applicationWillEnterForeground:(UIApplication *)application {

}
```

В этом методе можно остановить все процессы обновления контента.

6. **Метод, который отражает закрытие приложения пользователем.**

```
- (void)applicationWillTerminate:(UIApplication *)application {

}
```

Жизненный цикл контроллера

Жизненный цикл

Рассмотрим жизненный цикл отдельного контроллера. Первоначально он инициализируется, и в него осуществляется переход. Затем контроллер устанавливает свои свойства, и только после этого появляется на экране. Когда контроллер закрывается, он уничтожает все свойства из памяти и освобождает ее.

Методы жизненного цикла контроллера

Как и у **AppDelegate**, у каждого контроллера есть методы, которые отражают его текущее состояние:

1. **Метод, который вызывается сразу после загрузки.**

```
- (void)viewDidLoad {
    [super viewDidLoad];

}
```

При вызове этого метода все необходимые свойства контроллера уже созданы и готовы к использованию.

2. **Метод до отображения на экране.**

```
- (void)viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];

}
```

Этот метод вызывается перед появлением представления на экране. Он подходит для обновления состояний или переменных, которые влияют на отображение пользовательского интерфейса.

3. **Метод после отображения на экране.**

```
- (void)viewDidAppear:(BOOL)animated {
    [super viewDidAppear:animated];

}
```

Этот метод вызывается сразу после отображения представления на экране. Подходит для конфигурации размеров или схожих величин, так как все компоненты уже появились на экране.

4. **Метод до закрытия контроллера.**

```
- (void)viewWillDisappear:(BOOL)animated {
    [super viewWillDisappear:animated];

}
```

Этот метод вызывается перед закрытием контроллера. В нем можно обновить пользовательские настройки, которые были заданы в контроллере.

5. **Метод после закрытия контроллера.**

```
- (void)viewDidDisappear:(BOOL)animated {
    [super viewDidDisappear:animated];

}
```

Этот метод вызывается сразу после закрытия контроллера. В нем можно очистить данные, используемые в нем.

6. **Метод для обработки поворота экрана.**

```
- (void)viewWillTransitionToSize:(CGSize)size
withTransitionCoordinator:(id<UIViewControllerTransitionCoordinator>)coordinator
{
    [super viewWillTransitionToSize:size withTransitionCoordinator:coordinator];

}
```

Этот метод вызывается при повороте экрана. Его можно применять для анимации.

Завершаем игру «Пинг-понг»

Обработка нажатий

Чтобы передвигать ракетки по игровому полю, необходимо добавить соответствующие методы:

```
- (void)touchesBegan:(NSSet<UITouch *> *)touches withEvent:(UIEvent *)event {
    for (UITouch *touch in touches) {
        CGPoint point = [touch locationInView:self.view];
        if (_bottomTouch == nil && point.y > HALF_SCREEN_HEIGHT) {
            _bottomTouch = touch;
            _paddleBottom.center = CGPointMake(point.x, point.y);
        }
        else if (_topTouch == nil && point.y < HALF_SCREEN_HEIGHT) {
            _topTouch = touch;
            _paddleTop.center = CGPointMake(point.x, point.y);
        }
    }
}

- (void)touchesMoved:(NSSet<UITouch *> *)touches withEvent:(UIEvent *)event {
    for (UITouch *touch in touches) {
        CGPoint point = [touch locationInView:self.view];
        if (touch == _topTouch) {
            if (point.y > HALF_SCREEN_HEIGHT) {
                _paddleTop.center = CGPointMake(point.x, HALF_SCREEN_HEIGHT);
                return;
            }
            _paddleTop.center = point;
        }
        else if (touch == _bottomTouch) {
            if (point.y < HALF_SCREEN_HEIGHT) {
                _paddleBottom.center = CGPointMake(point.x, HALF_SCREEN_HEIGHT);
                return;
            }
            _paddleBottom.center = point;
        }
    }
}

- (void)touchesEnded:(NSSet<UITouch *> *)touches withEvent:(UIEvent *)event {
    for (UITouch *touch in touches) {
        if (touch == _topTouch) {
            _topTouch = nil;
        }
        else if (touch == _bottomTouch) {
            _bottomTouch = nil;
        }
    }
}

- (void)touchesCancelled:(NSSet<UITouch *> *)touches withEvent:(UIEvent *)event
{
    [self touchesEnded:touches withEvent:event];
```

Рассмотрим подробнее каждый метод.

```
- (void)touchesBegan:(NSSet<UITouch *> *)touches withEvent:(UIEvent *)event {
    for (UITouch *touch in touches) {
        CGPoint point = [touch locationInView:self.view];
        if (_bottomTouch == nil && point.y > HALF_SCREEN_HEIGHT) {
            _bottomTouch = touch;
            _paddleBottom.center = CGPointMake(point.x, point.y);
        }
        else if (_topTouch == nil && point.y < HALF_SCREEN_HEIGHT) {
            _topTouch = touch;
            _paddleTop.center = CGPointMake(point.x, point.y);
        }
    }
}
```

Метод **touchesBegan** вызывается при прикосновении пользователя к экрану. Так как iOS может обрабатывать несколько нажатий, то в данный метод передается целый массив объектов **UITouch**.

В теле метода совершается проверка нажатия: устанавливается, где именно оно было совершено. Если нажатие было в верхней половине экрана и там больше нет текущих нажатий – значит, его совершил игрок, который располагается вверху. Соответственно, для переменной **topTouch** присваивается значение этого прикосновения. После этого ракетка игрока перемещается на необходимую точку – для этого применяется центр ракетки. Аналогично и для игрока, который располагается внизу экрана. Так же переносится его ракетка и присваивается нижнее прикосновение.

```
- (void)touchesMoved:(NSSet<UITouch *> *)touches withEvent:(UIEvent *)event {
    for (UITouch *touch in touches) {
        CGPoint point = [touch locationInView:self.view];
        if (touch == _topTouch) {
            if (point.y > HALF_SCREEN_HEIGHT) {
                _paddleTop.center = CGPointMake(point.x, HALF_SCREEN_HEIGHT);
                return;
            }
            _paddleTop.center = point;
        }
        else if (touch == _bottomTouch) {
            if (point.y < HALF_SCREEN_HEIGHT) {
                _paddleBottom.center = CGPointMake(point.x, HALF_SCREEN_HEIGHT);
                return;
            }
            _paddleBottom.center = point;
        }
    }
}
```

Метод **touchesMoved** отслеживает перемещение пальца. Он вызывается, когда пользователь совершил прикосновение и сместил палец, не отрывая его от экрана.

В теле метода проверяется нажатие. Если оно уже существует, то необходимо обновить положение ракетки – как для верхнего игрока, так и для нижнего.

```
- (void)touchesEnded:(NSSet<UITouch *> *)touches withEvent:(UIEvent *)event {
    for (UITouch *touch in touches) {
        if (touch == _topTouch) {
            _topTouch = nil;
        }
        else if (touch == _bottomTouch) {
            _bottomTouch = nil;
        }
    }
}
```

Этот метод отвечает за завершение прикосновения. В этот момент выполняется тело метода и совершается проверка прикосновения. Если оно соответствует прикосновению верхнего или нижнего игрока, то оно обнуляется.

```
- (void)touchesCancelled:(NSSet<UITouch *> *)touches withEvent:(UIEvent *)event
{
    [self touchesEnded:touches withEvent:event];
}
```

Этот метод вызывается при прерывании прикосновении – если пользователь при перемещении пальца вышел за пределы экрана. Такое прикосновение надо обнулить. Для этого вызывается метод **touchesEnded**.

Основные игровые методы

Наше приложение уже действует и обрабатывает прикосновения – можно переходить к реализации игровой логики. Ниже представлены все игровые методы:

```
- (void)displayMessage:(NSString *)message {
    [self stop];
    UIAlertController *alertController = [UIAlertController
alertControllerWithTitle:@"Ping Pong" message:message
preferredStyle:(UIAlertControllerStyleAlert)];
    UIAlertAction *action = [UIAlertAction actionWithTitle:@"OK"
style:(UIAlertActionStyleDefault) handler:^(UIAlertAction * _Nonnull action) {
        if ([self gameOver]) {
            [self newGame];
            return;
        }
        [self reset];
        [self start];
    }];
    [alertController addAction:action];
    [self presentViewController:alertController animated:YES completion:nil];
}

- (void)newGame {
    [self reset];

    _scoreTop.text = @"0";
```

```
    _scoreBottom.text = @"0";

    [self displayMessage:@"Готовы к игре?"];
}

- (int)gameOver {
    if ([_scoreTop.text intValue] >= MAX_SCORE) return 1;
    if ([_scoreBottom.text intValue] >= MAX_SCORE) return 2;
    return 0;
}

- (void)start {
    _ball.center = CGPointMake(HALF_SCREEN_WIDTH, HALF_SCREEN_HEIGHT);
    if (!_timer) {
        _timer = [NSTimer scheduledTimerWithTimeInterval:1.0/60.0 target:self
selector:@selector(animate) userInfo:nil repeats:YES];
    }
    _ball.hidden = NO;
}

- (void)reset {
    if ((arc4random() % 2) == 0) {
        _dx = -1;
    } else {
        _dx = 1;
    }

    if (_dy != 0) {
        _dy = -_dy;
    } else if ((arc4random() % 2) == 0) {
        _dy = -1;
    } else  {
        _dy = 1;
    }

    _ball.center = CGPointMake(HALF_SCREEN_WIDTH, HALF_SCREEN_HEIGHT);

    _speed = 2;
}

- (void)stop {
    if (_timer) {
        [_timer invalidate];
        _timer = nil;
    }
    _ball.hidden = YES;
}

- (void)animate {
    _ball.center = CGPointMake(_ball.center.x + _dx*_speed, _ball.center.y +
_dy*_speed);
    [self checkCollision:CGRectMake(0, 0, 20, SCREEN_HEIGHT) X:fabs(_dx) Y:0];
    [self checkCollision:CGRectMake(SCREEN_WIDTH, 0, 20, SCREEN_HEIGHT)
```

```
X:-fabs(_dx) Y:0];
    if ([self checkCollision:_paddleTop.frame X:(_ball.center.x -
_paddleTop.center.x) / 32.0 Y:1]) {
        [self increaseSpeed];
    }
    if ([self checkCollision:_paddleBottom.frame X:(_ball.center.x -
_paddleBottom.center.x) / 32.0 Y:-1]) {
        [self increaseSpeed];
    }
    [self goal];
}

- (void)increaseSpeed {
    _speed += 0.5;
    if (_speed > 10) _speed = 10;
}

- (BOOL)checkCollision: (CGRect)rect X:(float)x Y:(float)y {
    if (CGRectIntersectsRect(_ball.frame, rect)) {
        if (x != 0) _dx = x;
        if (y != 0) _dy = y;
        return YES;
    }
    return NO;
}

- (BOOL)goal
{
    if (_ball.center.y < 0 || _ball.center.y >= SCREEN_HEIGHT) {
        int s1 = [_scoreTop.text intValue];
        int s2 = [_scoreBottom.text intValue];

        if (_ball.center.y < 0) ++s2; else ++s1;
        _scoreTop.text = [NSString stringWithFormat:@"%u", s1];
        _scoreBottom.text = [NSString stringWithFormat:@"%u", s2];

        int gameOver = [self gameOver];
        if (gameOver) {
            [self displayMessage:[NSString stringWithFormat:@"Игрок %i выиграл",
gameOver]];
        } else {
            [self reset];
        }

        return YES;
    }
    return NO;
}
```

Теперь рассмотрим подробнее каждый из них, определяя функционал.

```
- (void)displayMessage:(NSString *)message {
    [self stop];
    UIAlertController *alertController = [UIAlertController
alertControllerWithTitle:@"Ping Pong" message:message
preferredStyle:(UIAlertControllerStyleAlert)];
    UIAlertAction *action = [UIAlertAction actionWithTitle:@"OK"
style:(UIAlertActionStyleDefault) handler:^(UIAlertAction * _Nonnull action) {
        if ([self gameOver]) {
            [self newGame];
            return;
        }
        [self reset];
        [self start];
    }];
    [alertController addAction:action];
    [self presentViewController:alertController animated:YES completion:nil];
}
```

Этот метод необходим для вывода сообщений в виде диалоговых окон. На примере мы видим, как создается само диалоговое окно с переданным сообщением. Затем вводится действие, которое будет добавлено на это окно. При выборе действия совершается проверка – существует ли в данный момент проигравший игрок. Если нет – поле очищается и начинается игра. Далее это действие добавляется на окно, и оно показывается на экране.

```
- (void)newGame {
    [self reset];

    _scoreTop.text = @"0";
    _scoreBottom.text = @"0";

    [self displayMessage:@"Готовы к игре?"];
}
```

Этот метод настраивает новую игру. Очищается игровое и текстовое поле с очками игроков вызывается диалоговое окно.

```
- (int)gameOver {
    if ([_scoreTop.text intValue] >= MAX_SCORE) return 1;
    if ([_scoreBottom.text intValue] >= MAX_SCORE) return 2;
    return 0;
}
```

Этот метод выполняет проверку на проигравшего игрока. Если выиграл верхний игрок, то метод возвращает 1, если нижний – то 2. Если игроки не достигли максимального количества очков, то возвращается 0.

```
- (void)start {
    _ball.center = CGPointMake(HALF_SCREEN_WIDTH, HALF_SCREEN_HEIGHT);
    if (!_timer) {
        _timer = [NSTimer scheduledTimerWithTimeInterval:1.0/60.0 target:self
selector:@selector(animate) userInfo:nil repeats:YES];
    }
    _ball.hidden = NO;
}
```

Этот метод отвечает за настройку компонентов, необходимую для начала игры. Здесь мяч устанавливается на центр экрана, задается таймер, который будет вызывать метод **animate**.

```
- (void)reset {
    if ((arc4random() % 2) == 0) {
        _dx = -1;
    } else {
        _dx = 1;
    }

    if (_dy != 0) {
        _dy = -_dy;
    } else if ((arc4random() % 2) == 0) {
        _dy = -1;
    } else  {
        _dy = 1;
    }

    _ball.center = CGPointMake(HALF_SCREEN_WIDTH, HALF_SCREEN_HEIGHT);

    _speed = 2;
}
```

Этот метод необходим для сброса настроек.

```
- (void)stop {
    if (_timer) {
        [_timer invalidate];
        _timer = nil;
    }
    _ball.hidden = YES;
}
```

Данный метод останавливает игру: обнуляет таймер и скрывает мяч.

```
- (void)animate {
    _ball.center = CGPointMake(_ball.center.x + _dx*_speed, _ball.center.y +
_dy*_speed);
    [self checkCollision:CGRectMake(0, 0, 20, SCREEN_HEIGHT) X:fabs(_dx) Y:0];
    [self checkCollision:CGRectMake(SCREEN_WIDTH, 0, 20, SCREEN_HEIGHT)
X:-fabs(_dx) Y:0];
    if ([self checkCollision:_paddleTop.frame X:(_ball.center.x -
_paddleTop.center.x) / 32.0 Y:1]) {
        [self increaseSpeed];
    }
    if ([self checkCollision:_paddleBottom.frame X:(_ball.center.x -
_paddleBottom.center.x) / 32.0 Y:-1]) {
        [self increaseSpeed];
    }
    [self goal];
}
```

Данный метод вызывается таймером каждую 1/60 секунды и отвечает за обновление положения мяча с учетом его скорости. Здесь проверяется, не попадал ли мяч в стену и был ли он забит.

```
- (void)increaseSpeed {
    _speed += 0.5;
    if (_speed > 10) _speed = 10;
}
```

Данный метод увеличивает скорость. Максимальное значение скорости – 10.

```
- (BOOL)checkCollision: (CGRect)rect X:(float)x Y:(float)y {
    if (CGRectIntersectsRect(_ball.frame, rect)) {
        if (x != 0) _dx = x;
        if (y != 0) _dy = y;
        return YES;
    }
    return NO;
}
```

Этот метод проверяет, соприкасается ли мяч с краями экрана. Если да, то возвращается YES, иначе NO.

```
- (BOOL)goal
{
    if (_ball.center.y < 0 || _ball.center.y >= SCREEN_HEIGHT) {
        int s1 = [_scoreTop.text intValue];
        int s2 = [_scoreBottom.text intValue];

        if (_ball.center.y < 0) ++s2; else ++s1;
        _scoreTop.text = [NSString stringWithFormat:@"%u", s1];
        _scoreBottom.text = [NSString stringWithFormat:@"%u", s2];

        int gameOver = [self gameOver];
        if (gameOver) {
            [self displayMessage:[NSString stringWithFormat:@"Игрок %i выиграл",
gameOver]];
        } else {
            [self reset];
        }

        return YES;
    }
    return NO;
}
```

Этот метод проверяет, был ли забит мяч, и увеличивает значение очков у пользователя. После прибавления очков метод проверяет победителя. Если мяч был забит и победителя нет, то игра продолжается.

Использование методов жизненного цикла

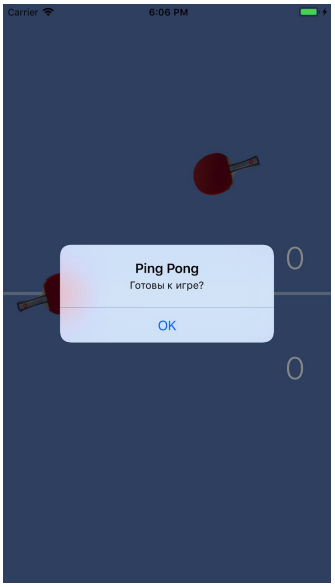
Для начала и конфигурирования игры необходимо применить 2 метода жизненного цикла.

```
- (void)viewDidAppear:(BOOL)animated {
    [super viewDidAppear:animated];

    [self becomeFirstResponder];
    [self newGame];
}

- (void)viewDidDisappear:(BOOL)animated {
    [super viewDidDisappear:animated];

    [self resignFirstResponder];
}
```

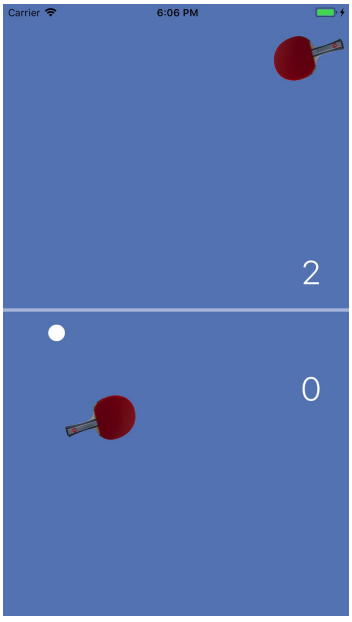
При отображении представления нажатия начинают отслеживаться, и игра запускается с помощью метода **newGame**. При закрытии контроллера (если приложение будет свернуто) отслеживание нажатий заканчивается.

Запуск готовой игры

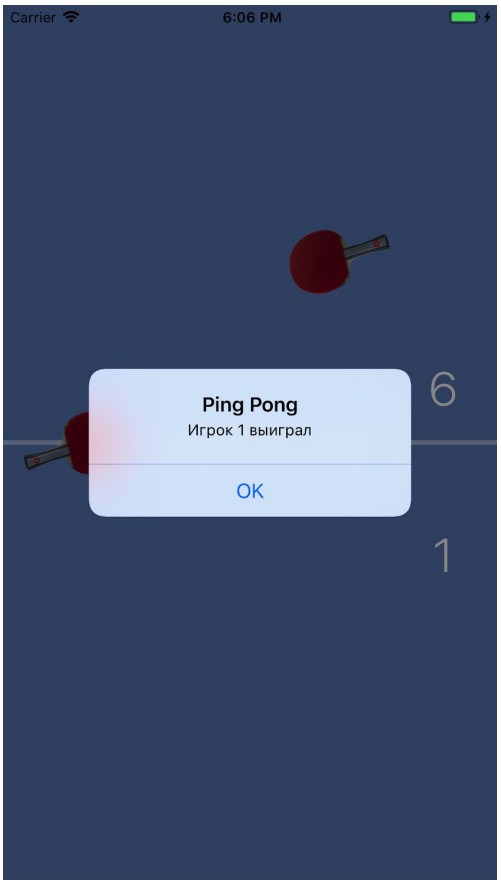
Теперь игра готова: можно приступать к ее использованию. Первоначальный экран выглядит так:



Игровой процесс:



По достижении заданного количества очков игра завершается:



Практическое задание

1. Применить интроспекцию к контроллеру игры, который вы написали на прошлой домашней работе.
2. Засвизлить любой метод
3. * Придумать где можно применить перенаправление методов
4. * Отрефакторить код игры пин понг.

Дополнительные материалы

1. http://proswift.ru/view-controller-lifecycle-zhiznennyj-cikl-view-controller/;
2. http://proswift.ru/ios-application-lifecycle-ili-zhiznennyj-cikl-ios-prilozheniya/.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. http://proswift.ru/view-controller-lifecycle-zhiznennyj-cikl-view-controller/;
2. http://proswift.ru/ios-application-lifecycle-ili-zhiznennyj-cikl-ios-prilozheniya/;
3. Стивен Кочан. «Программирование на Objective-C»;

4. Скотт Кнастер, Вакар Малик, Марк Далримпл. «Objective-C. Программирование для Mac OS X и iOS».