# 1.Описание

Игра на android. Рабочее название - The Touch Game. Представляет простую игру, цель которой быстро нажимать на появляющиеся в случайном месте экрана нужные кнопки, набирая при этом очки. Включает несколько режимов:

* **Simple mode:**На экране находится одна кнопка в центре. На верхней панели находятся таймер и очки. При старте (первое нажатие на кнопку) таймер запускается (время-5 сек).  
  При нажатии кнопка исчезает, в рандомном месте появляется другая кнопка и начисляется очко, а таймер увеличивается (время увеличения будет определено тестом).  
  По истечении таймера игра окончена – очки фиксируются в оффлайн таблицу рекордов.
* **Standard mode**:

На экране появляется красный и зеленый отпечатки пальцев, располагаясь по центру, друг под другом. На таймере надпись “Press green button”. При старте (первое нажатие на любую кнопку) запускаем таймер (время – 10 сек).  
При нажатии на любую кнопку все кнопки на экране исчезают. Появляется от 1 до 5 кнопок, из них от 1 до всех зеленые, расположение случайно, но кнопки не перекрывают друг друга.

Требуется вновь нажимать на любую из зеленых кнопок.  
 При нажатии на зеленую наращивается очко и накидывается время, при нажатии на красную время таймера скидывается на некое время.  
Затем через определенный период (не плохо сделать его случайным) происходит 1 из 3 либо некоторые кнопки меняют цвет, либо исчезают, либо то же, что и по нажатии на кнопку без беспокойства таймера.

* **Некие другие**

Система должна включать также

* Экран главного меню, содержащий кнопки: standard mode, choose mode, rules,scores(не обязательно),about. Наверху название игры.
* rules:содержит правила, также неплохо бы добавить вывод правил режима при первом запуске режима
* about-описание разработчика
* Таблица рекордов для каждого режима.

Возможные добавления после релиза

* Доп. Режимы
* Локализация, смена звуков.

2.План разработки.

1. Реализация simple mode.

На данном этапе требуется полностью реализовать рабочий simple mode без рекордов в отдельном activity, который впоследствии будет служить основой всех остальных режимов. Необходимо реализовать дизайн и предусмотреть(хотя бы частично) возможность запуска на различных экранах.

1.Создается activity, создается game\_layout, содержащий пустое поле, но наверху присутствуют TimerTextView и ScoreTextView. В коде идет их обнаружение, создание переменных для очков и программное создание кнопки по центру экрана

2.Создается обработчик кнопки. По нажатию начисляются очки, кнопка исчезает и появляется в рандомном месте с анимацией scale.

3.Время реализовать таймер. Создается таймер, при нажатии на кнопку накидываем ему некое время.

4.Остался дизайн. Кидается скин кнопки под разные экраны, меняется тема, ставится некий звук. Вопрос ставить ли музыку?

5.Можно добавить меню, пока там, например, только выход, но при открытии меню стопорим таймер - будет как пауза.

**Тестирование –** требуется найти оптимальное значение констант положения и времени

После этого делается резервная копия проекта т.к., по сути, этот режим каркас для остальных режимов и они будут реализовываться наслоением на этот (хотя не обязательно).

1. **Реализация Standard mode.**

На данном этапе требуется реализовать полностью готовый и играбельный standard mode. Создание вести в отдельном activity, используя simple mode лишь для вставки отдельных кусков кода.

1.Создается новый Activity и layout копируем из simple mode. Создаем класс для наших кнопок, в нем должны быть свойства visible, isGreen, coords, а также метод вставки кнопки на экран.   
2.Создаем массив из COUNT\_OF\_BUTTONS кнопок. Обнаружение элементов экрана берем из кода прошлого режима. В onCreate создаем две первые кнопки. Ставим их по центру, как в условиях.  
3.Реализуем обработчик, пока что в результате нажатия делаем проверку на цвет и начисляем очки. Также очищаем весь экран.  
4.Самая сложная часть. Требуется разработать алгоритм задания случайных координат. Вынесем это в отдельную функцию. Выбираем 2 рандомных числа. Первое число видимых кнопок, второе – зеленых. Создаем массив, все зеленые, но пока не видимые. Идет проход по массиву с вызовом новой ф-ии SetRandomCoords (TheButton[] buttons, int num), которая задаем num-у эл-ту массива случ x и y, которые не перекрывают все кнопки из массива, которые видимы (пока пускай функция ставит кнопки по порядку).  
5.Если это работает, добавим таймер. С этим проблем быть не должно.  
6.Осталось добавить случайное поведение в определенные моменты времени.

1. **Реализация главного меню.**

На данном этапе требуется реализовать полностью рабочее главное меню с окончательным дизайном и связью с готовыми режимами, также нужно добавить меню во все режимы с кнопкой выхода в главное меню, а по завершении кнопки resart и toMainMenu. Следует убрать возможность поворота экрана.

1.Создается новое MainMenuActivity, делается главным. Верстаем кнопки.  
 2.Создаем еще 1 layout choose\_mode, там пока 1 кнопка simple\_mode.  
 3.Создаем обработчики кнопок, выводящие на нужный режим. Учитываем нажатие back;  
 4.Запрещаем поворот экрана.  
 5.Делаем нормальную реализацию метода onFinish() обоих режимов, хорошо бы сделать метод одинаково.  
 6.Добавляем игровое меню.  
 7.Добавляем кнопки вкл и выкл вибро и звук. Сначала звук. Сохраняем в префе. В игру грузим в зависимости от того есть или нет либо норм звуки, либо мгновенную пустоту, чтобы код не пичкать ifами.

1. **Доработка системы, адаптация к изменениям**1.Перехват back и вызов на нее меню. Кусок кода отвечающий за меню будем копировать в другие режимы, делаем его особняком, добавим паузу при вызове меню. (ОЙ! Напичкал костылей, ну и х\*\* с ним. Все работает и трогать не будем, будем вставлять в другие режимы).  
   2. Добавим звуки зеленых и красных, а также вибро на красных.  
   3. Надо сделать полностью общий финиш для всех режимов. Создаем класс Finisher. На конструктор передаем название режима и набранные очки и контекст  
   4.Дальше у него есть один публичный метод finishgame(). Переносим туда наработки из режимов и оставляем там только создание объекта и вызов его метода.  
   5. Добавим вибро и звук конца игры. Тестируем.
2. **Реализация таблицы рекордов**

Будем реализовывать класс для доступа к рекордам. Методы класса публичные-  
LoadRecords(string mode,string[] recordName, int[] recordscore)  
-saveresult(string mode,string Name, int score) возвращает позицию результата-  
replacerecord(name,score)

1.Пишем класс и его атрибуты – это преф, его имя  
 2.Пишем метод loadRecords. Для начала проверим изменяется ли в яве массив в функции. Делаем цикл. Забираем инфу из преф и суем в массив defaults – прочерк , 0.   
3.Пишем приватный метод сохранения всего массива в преф. В принципи легко.  
4.Пишем приватный метод вставки в массив  
5.Используя 2 пред метода пишем метод saveresult  
6. replacerecord(name,score)-ну это просто  
После реализации класса нужно сделать таблицу рекордов версткой и реализовать функцию заполнения таблицы с помощью созданного класса. лучше создать класс работы с таблицей т к еще его можно будет вызывать из класса финиш. Поехали!!!!  
7.верстаем лаяут с таблицей на 5 значений. Да, мы не сможем изменять кол-во рекордов константой, ну и х\*\* с ним.  
8.Пишем класс TableFiller. На конструктор актвити и режим.  
9.Метод fillTable. Вызываем лаяут, пишем режим. Забираем очки, вставляем в табличку инфу.  
10.Конкретно тестируем класс records, вызывая его для simple mode из экрана recordes  
11.Добавим на этот экран смену режимов.  
Если все это работает пора начинать обновление метода FinishGame()  
12. Пока просто будем добавлять рекорд со статичным именем. и при 1 месте выводить что побит рекорд. Тестим, забья на имя и меняя его в коде.  
13. Теперь надо сделать возможность выбора имени. Верстаем текствью и кнопку. Там будет отображаться текущее имя.  
14.Верстаем новый лаяут с Едиттекст куда надо имя написать и сохранить кнопочка (ну текствью еще конечно) , отмена кнопка еще.  
15.Обрабатываем кнопку сейв. Она сохраняет имя в преф, возвращает нас на прошлый экран , перезаписывает рекорд . Кнопка отмена аналогично без трогания префов.  
16.Добавим возможность из финишёра показывать рекорды только этого режима. Благо класс уже есть.

На этом этапе основная работа над проектом считается завершенной. Теперь надо реализовать экраны с правилами и абоут, что деоается только версткой и не представляет проблем помимо перевода на английский. Сейчас нужно разработать детальные рекомендации по добавлению режимов.

3. Инструкции по разработке доп. режимов.   
1. Новый режим должен быть тщательно продуман и описан, а также рекомендуется составить как можно более детальный план разработки.   
2. Конструирование нового режима осуществлять в отдельном проекте, в активити под названием “NameModeActivity”, верстается лаяут name\_mode. Верстается лишь верхняя часть, рекомендуется забрать из simple и доделать что надо.  
3.Код пишется независимо от готовых режимов, однако рекомендуется брать куски кода из них и поддерживать созданную в них структуру.  
4.Перед вставкой в основной проект режим должен быть качественно сделан и протестирован, однако звук, конец и меню не тронуты.   
5.Итак, если все хорошо, привожу алгоритм вставки режима в проект.   
6.Вставляем класс и лаяут, регаем активити, ориен – портрет, стринги копируем.  
7.Для начала добавим возможность запуска из меню выбор режима. Добавляем в лаяут кнопку, обрабатываем ее в активити и добавляем возможность запуска.  
8.Добавим звук и вибро с учетом префа конечно.  
9.Используем финишера, НАЗВАНИЕ РЕЖИМА ПРЕРЕДАЕМ СТРОЧНЫМИ БУКВАМИ, в финишере добавляем case.  
10.Добавим рекорды, в константы главного меню заносим инфу о новом режиме.  
11.Самый гадкий пункт. Надо добавить меню, копируем тот гадкий блок кода, изменяем нужные функции, не забываем об isFinish и т.д. Доводим до работоспособности.  
12.Если режим разрабатывался после правил, добавим их – правила в стринг, текствью на стр. правил, в активити – константа диалога, кейс в выбор диалога, новый обработчик диалога и вывоз диалоги при условии в онкреате .  
13.По завершении локализируем режим, что довольно быстро.

4. Доп. режимы.

1.Color mode.  
Описание – на экране появляется некоторое количество разноцветных отпечатков пальцев. Наверху текст “Press color” где color выбирается ранодомом из – (red, green ,blue ,yellow, white), причем надпись соответствующим цветом написана. Среди пальцев есть хотя бы 1 нужного цвета, остальные случайных цветов. Таймер запускается при нажатии на кнопку. Итак, наша цель нажать все кнопки нужного цвета и не трогать другие. При нажатии на нужную кнопку, получаем очко, она исчезает, если больше нужных кнопок нажатии на не на ту кнопку все также генерится вновь и все, наказание в том, что мы не получим 3 очка. Таймер здесь будет фиксирован (30 сек или посмотрим).  
  
План разработки:   
1. Создаем проект NewMode. Там 1 Activity и 1 лаяут. Названия соответствуют документации.  
2. Верстаем лаяут, взяв simple mode, однако теперь надо еще добавить строку “Press color”. Сразу не забываем о строковых ресурсах.  
3.Переходим к кодированию. Обнаруживаем все ресурсы.   
4.Оглядываясь на стандарт моде создаем класс для работы с кнопками, но теперь он содержит номер цвета от 0 до 4.  
5.Создаем массив кнопок.  
6.Пишем метод заполнения экрана на манер стандарт моде.  
7.Пишем обработчика, выполняя все требования.  
8.Добавляем таймер, в финиш пока ничего.  
9.Комлексно тестируем.  
10.Выполняем вставку в проект по документации.

**2. Calculator mode.**

­Описание – на экране появляется некоторое количество кнопок, на каждой из них некое число. Наверху на таймере надпись считай, а ниже простой примерчик (о формировании позже). На одной из кнопок верное решение. Итак, опишем процесс формирования примерчика. Сначала выбирается 1 из четырех операций. Затем выбирается 2 числа способом, зависящим от операции. Если плюс, то первое число из диапазона (предположительно от 0 до 99) и разница от числа до диапазона. Считаем результат. Если минус то первое число, второе от 0 до числа, и результат. Если умножение первое число от диапазон делить на 3, второе от диапазон делить на это число, считаем результат. С делением аналогично, но результат и число меняем местами. Итак, пример готов и выведен в текствью, среди чисел одно верное. Перерассчитывается нажатии на неверное сбрасывается кол-во секунд равное кол-ву неверных ответов. При нажатии на верное добавляется кол-во секунд, равное (TIME\_FOR\_DECISION – timeleft), где timeleft – время пройденное с момента последнего нажатия на кнопку. При нажатии на любую кнопку пример и экран перезаписываются.

План разработки:  
1. Берем проект теста, меняем названия.  
2.Меняем в лаяуте пресс колор на примерчик.  
3.Беремся за код, чистим его и пишем заново с оглядкой на колормоде, он ближе всего.  
4.Пишем класс для работы с кнопками, вместо цвета значение, в остальном как в колормоде.  
5. Создаем массив кнопок.  
6.Нужен метод случайного вывода, пока пример зададим статично, остальное скопируем из колормоде с адаптацией.  
7.Реализуем примерчик.  
8.Пишем обработчик.  
9.Добавляем таймер.  
10.Комлексно тестируем.  
11.Выполняем вставку в проект по документации.

**3.Hardcore mode**На экране одна кнопка. Лаяут как в симпле. По нажатии на кнопку первый раз очко не начислется (это же хардкор). Игра запускается. Кнопка прыгает по экрану каждые TIME\_FOR\_JUMP миллисекунд т.ч. не удастся нажимать на все (лучший вариант игры – держать палец в одном месте и ждать, когда кнопка появится там, но проблема в том, что ). По нажатию вне кнопки конец игры. По нажатию на кнопку получаем очко и накидываем немного времени, но незначительно.

План разработки:  
1. Берем проект теста, меняем названия, лаяут с симпла.  
2.Чистим код, начинаем писать обнаружение.  
3.В игре 1 кнопка, класс не нужен, но нужен метод установки на экран.  
4.Пишем метод установки кнопки в случайное место экрана.  
5.Обработчик делаем, но уже не клик, а тач.  
6.Теперь пишем таймер, пока все как в симпле.  
7.А вот и фишка режима. Добавляем прыжок кнопки.  
8.Комлексно тестируем.  
9.Выполняем вставку в проект по документации.

**4.MEMORY MODE.**

На экране зеленая кнопка. Лаяут из стандарта. Жмем на кнопку, игра началась. На экране некоторое количество кнопок красных и зеленых (количество – START\_COUNT+score/SCORE\_FOR\_PLUS\_BUTTON). На время TIME\_FOR\_MEMORY они имеют свой цвет и не реагируют на нажатия, затем все они стают белыми. Цель – нажать все бывшие зеленые. Очки начисляются как в колор моде(его и берем за основу), но нажатие всех поощряется мощнее. За идущие подряд безошибочные серии начислем еще больше очков. Число таких серий выводим на экран, начиная с 3. Таймера нет. Поражение при первом неверном нажатии.

План разработки:  
1. Берем проект теста, меняем названия, лаяут с симпла, добавляем серию и меняем стартовую надпись.  
2.Чистим код, начинаем писать обнаружение.  
3.Берем из колор класс.  
4.Пишем создание массива.  
5.Пишем функции заполнения.   
6.Копируем обработчик из колора, изменяем начисление очков.  
7.Пишем таймер.  
8.Добавляем исчезание цветов и игнор обработчика небелых.  
9.Тестируем.  
10.Пихаем.

1. **Добавление правил и информации о разработчике.**Думаю, эта часть не должна доставить проблем. В стрингах добавляем правила для каждого режима, создаем экран с ними и реализуем единовременный показ правил при заходе в режим. Экран инфо просто сверстать.  
   1. Добавим огромную тестовую строку и научимся создавать длинный экран.  
   2.Пишем по англ. правила для каждого режима.  
   3.Добавляем диалог с правилами в начало каждого режима, учтя префу.  
   4.Делаем фотку.  
   5.Пишем инфу.  
   6.Учимся делать ссылки.