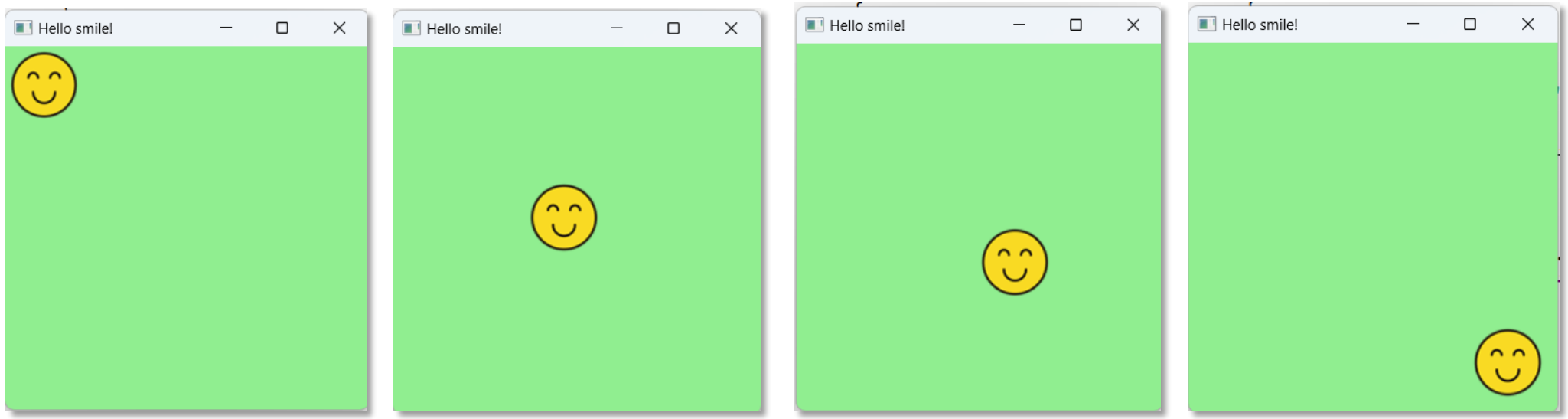
Анимация в QML

*Анимация нужна не только для создания визуальных эффектов - она может помочь обратить внимание пользователя на ту или иную часть действий, выполняемых вашей программой.

Анимация при изменении свойств

Для анимации свойств существует элемент ***PropertyAnimation.*** С его помощью можно изменять несколько свойств одновременно, например **размер и прозрачность**.

Одновременное изменение двух свойств: ***x*** и ***y***. В результате чего изображение проделает путь из верхнего левого угла в нижний правый угол.



Анимация при изменении свойств

```
import QtQuick 2.15
    Rectangle {
        color: "lightgreen"
        width: 300;  height: 300
        Image {
            id: img
            x: 0;  y: 0
            source: "qrc:/smile.png"
        }
        PropertyAnimation {
            target: img
            properties: "x, y"
            from: 0;  to: 300 - img.height
            duration: 1500                  // время выполнения анимации в мс
            running: true                   // запуск анимации
            loops: Animation.Infinite  // число повторений анимации
            easing.type: Easing.OutExpo // идентификатор типа смягчающей линии
        }
}
```

Базовый элемент анимации

Значение	График динамики
OutExpo	

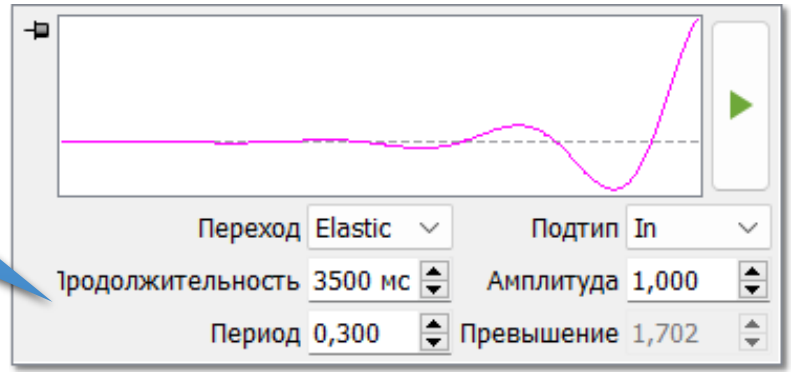
Эффект изменяющейся скорости

Для создания эффекта изменяющейся скорости анимации применяются смягчающие линии.

Анимация может ускоряться в самом начале, достигать максимальной скорости в середине, а затем начать замедляться. В примере используется стандартный идентификатор типа смягчающей линии **easing.type**

Значение	График динамики
InQuad	
OutInQuad	
InCirc	

Окно интерактивной настройки анимации



Элемент ***PropertyAnimation*** является базовым для более специфичных типов элементов анимации:

- ***NumberAnimation*** - для изменения числовых свойств;
- ***ColorAnimation*** - для изменения свойств цвета;
- ***RotationAnimation*** - для поворота элементов.

Анимация числовых значений

Элемент анимации числовых значений ***NumberAnimation*** предоставляет, в сравнении с элементом ***PropertyAnimation***, более эффективную реализацию для анимирования свойств типа ***real*** и ***int***.

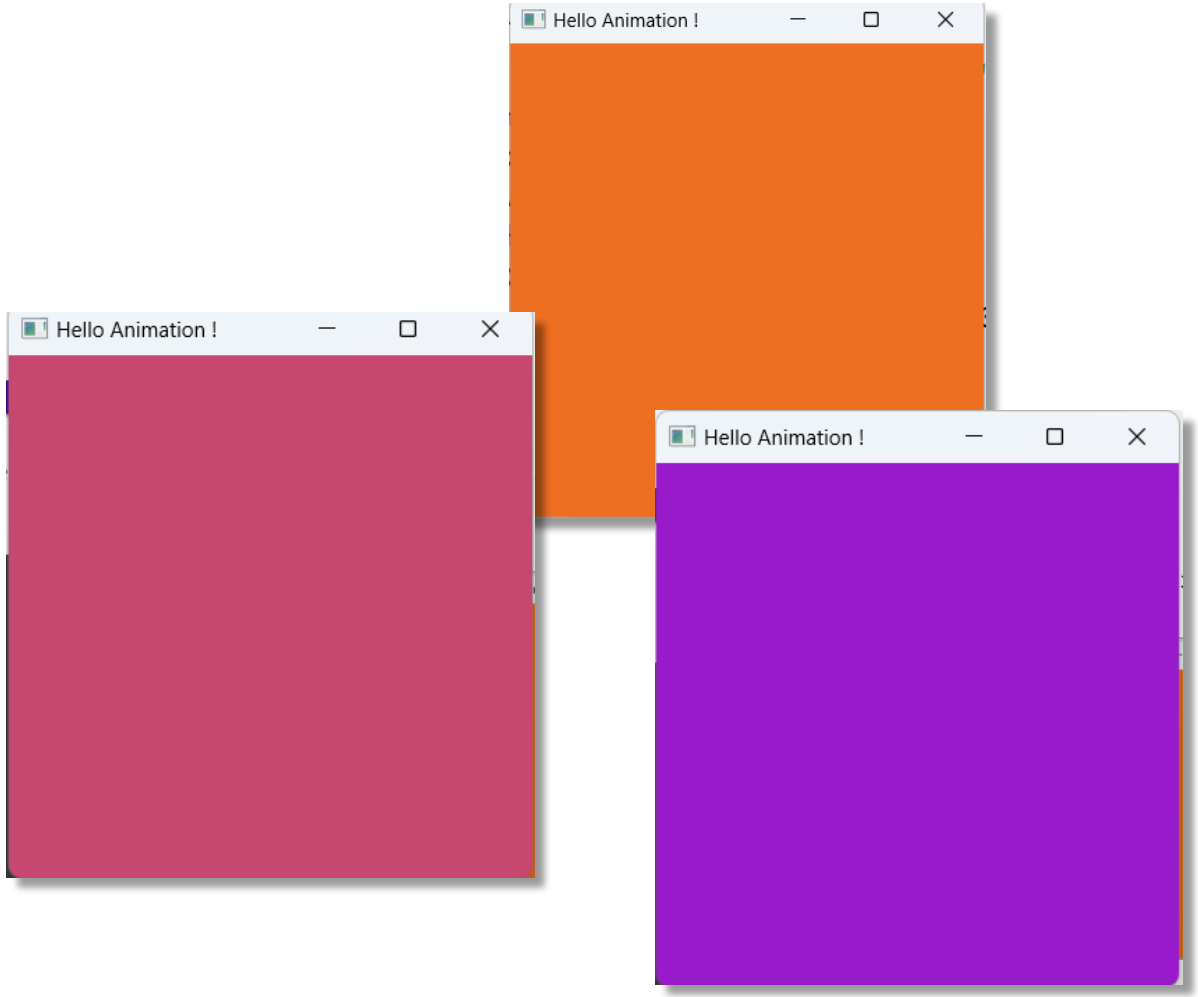
```
import QtQuick 2.15
    Rectangle {
        width: 300 ;  height: 100
        color: "lightgreen"
        Rectangle {
            x: 0;   y: 0
            height: 100
            color: "red"
            NumberAnimation on width {
                from: 300
                to: 0
                duration: 2000
                easing.type: Easing.InOutCubic
            }
    }
}
```

Пример демонстрирует использование этой анимации для изменения значения свойства ширины элемента (***width***)

Анимация с изменением цвета

Элемент ***ColorAnimation*** управляет изменением цвета элементов. Для изменения значения цвета предоставляет свойства **from** и **to**.

```
import QtQuick 2.15
Rectangle {
    width: 200
     height: 200
     ColorAnimation on color {
        from: Qt.rgba(1, 0.5, 0, 1)
        to: Qt.rgba(0.5, 0, 1, 1)
        duration: 1500
        running: true
        loops: Animation.Infinite
    }
}
```



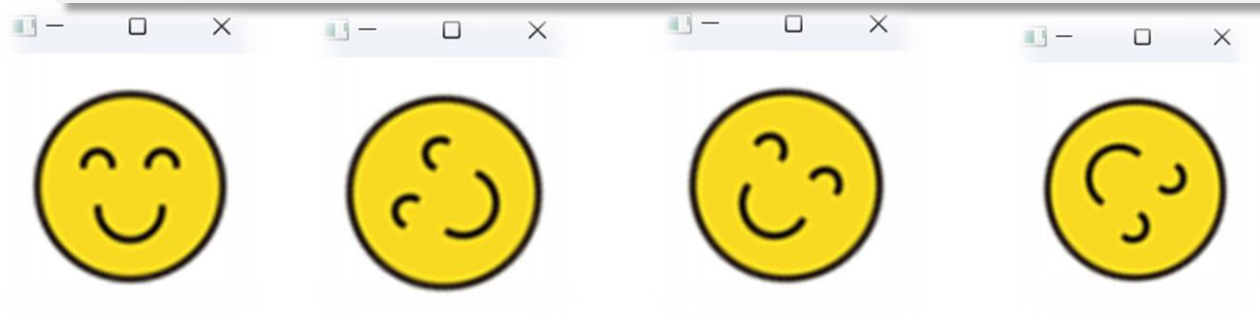
Анимация с поворотом

```
import QtQuick 2.15
Rectangle {
  width: 150; height: 150
   Image {
      source: " qrc:/smile.png"
      anchors.centerIn: parent
      smooth: true
     RotationAnimation on rotation {
         from: 0
         to: 360
         duration: 2000
         loops: Animation.Infinite
         easing.type: Easing.InOutBack
     }
  }
}
```

Элемент ***RotationAnimation*** описывает поворот элемента. Он предоставляет свойство **direction**, с помощью которого можно задавать направление поворота. Это свойство может принимать следующие значения:

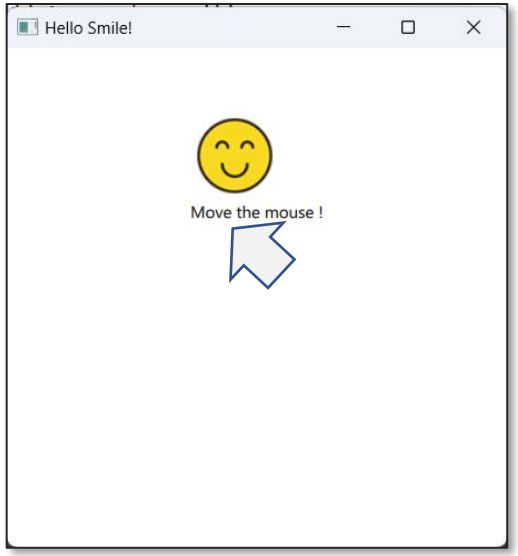
- *RotationAnimation.Clockwise* - поворот по часовой стрелке;
- *RotationAnimation.Counterclockwise* - поворот против часовой стрелки;
- *RotationAnimation.Shortest* - поворот в сторону наименьшего угла поворота от значения, заданного в свойстве **from**, и до значения - **to**.

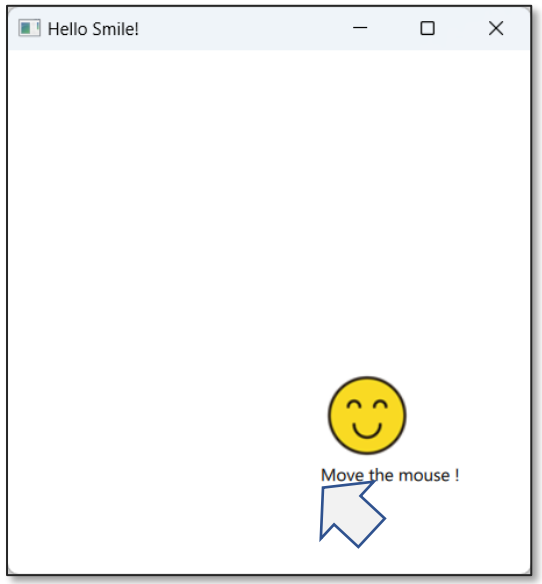
По умолчанию поворот осуществляется в *направлении часовой стрелки.*

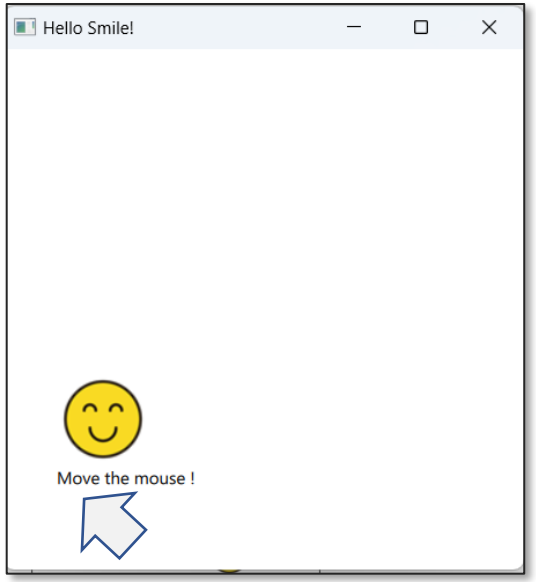


Анимация поведения

Иногда необходимо выполнить анимацию в момент, когда происходит изменение какого-либо из свойств элемента. Например, чтобы растровое изображение следовало за указателем мыши из одной позиции в другую, красиво анимировало при смене позиций. Это значит, что каждый раз, когда свойство изменяется, должна запускаться анимация. Именно для этого существует элемент анимации поведения ***Behavior.***



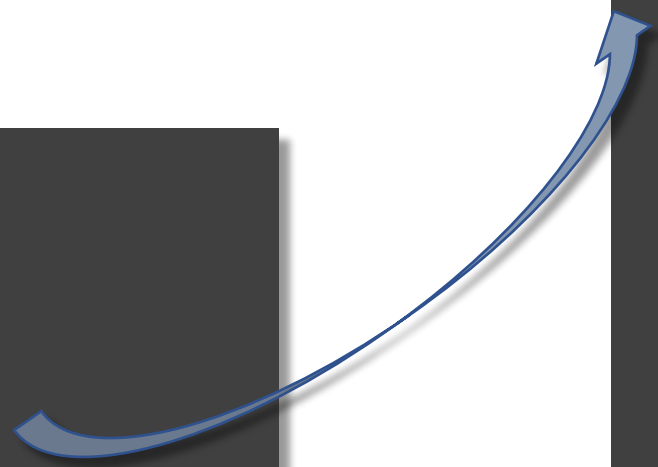




Анимация поведения

```
import QtQuick 2.15
Rectangle {
    id: rect
    width: 360; height: 360
   . . .

   MouseArea {
       anchors.fill: rect
       hoverEnabled: true
       onMouseXChanged: img.x = mouseX
       onMouseYChanged: img.y = mouseY
}
```



```
Image {
    id: img
    source: " qrc:/smile.png"
    x: 10;  y: 10
    smooth: true
    Text {
        anchors.verticalCenter: img.verticalCenter
        anchors.top: img.bottom
        text: "Move the mouse ! "
        Behavior on x {
            NumberAnimation {
                duration: 1000
                easing.type: Easing.OutBounce
            }
        }
        Behavior on y {
            NumberAnimation {
                duration: 1000
                easing.type: Easing.OutBounce
            }
        }
    }
}
```

Параллельные и последовательные анимации

Иногда с одним объектом нужно выполнить несколько различных анимаций. В этом случае анимации могут быть объединены в одну общую анимацию. Это делается при помощи специальных элементов групп. Группы анимаций могут выполняться **параллельно** или **последовательно**:

- **Последовательные анимации** задаются при помощи элемента ***SequentialAnimation***. В этом элементе каждый потомок анимации выполняется *в порядке очереди*.

- Элемент ***ParallelAnimation*** задает **параллельную группу**, в которой потомки анимации запускаются и исполняются одновременно.

Параллельные и последовательные анимации могут быть вложены друг в друга.
Если элемент группы опущен, то анимации будут выполняться *параллельно*, например:

```
Rectangle {
  . . .
   PropertyAnimation on x {to: 50; duration: 1000)
   PropertyAnimation on y {to: 50; duration: 1000}
  . . .
}
```

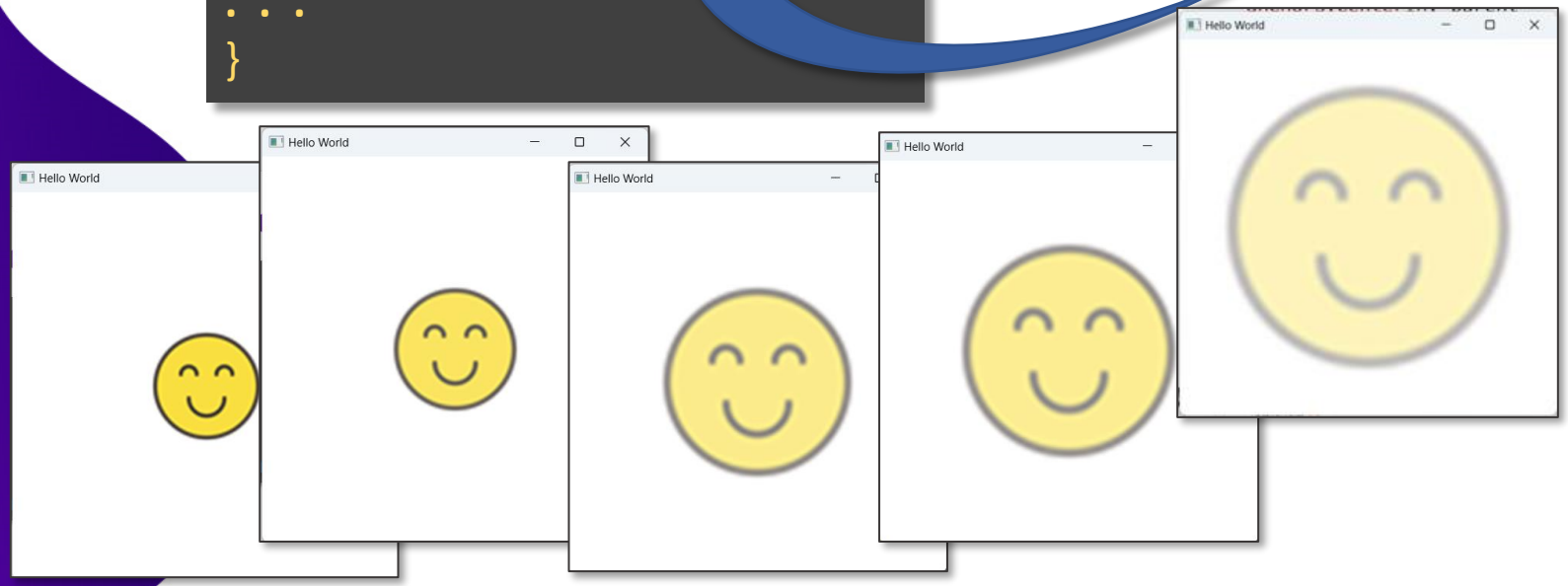
Пример параллельных анимаций

```
import QtQuick 2.15
Rectangle {
    width: 400;  height: 400
    Image {
        id: img
        source: " qrc:/smile.png"
        smooth: true
        anchors.centerIn: parent
    }
. . .
}
```

Изменяем сразу два свойства одновременно:

- увеличение изображения (свойство scale) с фактором от 0.1 до 3
- изменение прозрачности (свойство opacity) (от 1 до 0).

```
ParallelAnimation {
    NumberAnimation {
        target: img
        properties: "scale"
        from: 0.1;  to: 3.0;
        duration: 2000
        easing .type: Easing.InOutCubic
    }
    NumberAnimation {
        target: img
        properties: "opacity"
        from: 1.0 ;    to: 0;
        duration: 2000
    }
    running: true  // запуск анимации
    loops: Animation.Infinite
}
```



Пример последовательных анимаций

Представим, что у нас есть некий объект, который висит наверху, и если мы нажатием мыши его отпустим, то он упадет. После падения он перевернется вокруг своей оси, полежит некоторое время на полу, а затем самостоятельно поднимется вверх.

```
import QtQuick 2.15
   Rectangle {
      width: 130
      height: 450
      Image {
         id: img
         source: " qrc: / smile.png"
         smooth: true
         . . .
      }
  }
```

```
Text {
      anchors.horizontalCenter: img.horizontalCenter
      anchors . top: img.bottom
      text: "Click me ! “
}
MouseArea {
      anchors.fill: img
      onClicked: anim. running = true
}
SequentialAnimation {
      id: anim
      NumberAnimation {  // 1
            target: img
            from: 20;  to: 300
            properties: “y"
            easing.type: Easing.OutBounce
            duration: 1000
      }
       . . .
}
```

```
RotationAnimation { // 2
      target: img
      from: 0 ;   to: 360
      properties: "rotation"
      direction: RotationAnimaion.Clockwise
      duration: 1000
}
PauseAnimation {    // 3
      duration: 500
}
NumberAnimation {   // 4
      target: img
      from: 300 ;   to: 20
      properties: "y"
      easing.type: Easing.OutBack
      duration: 1000
}
```

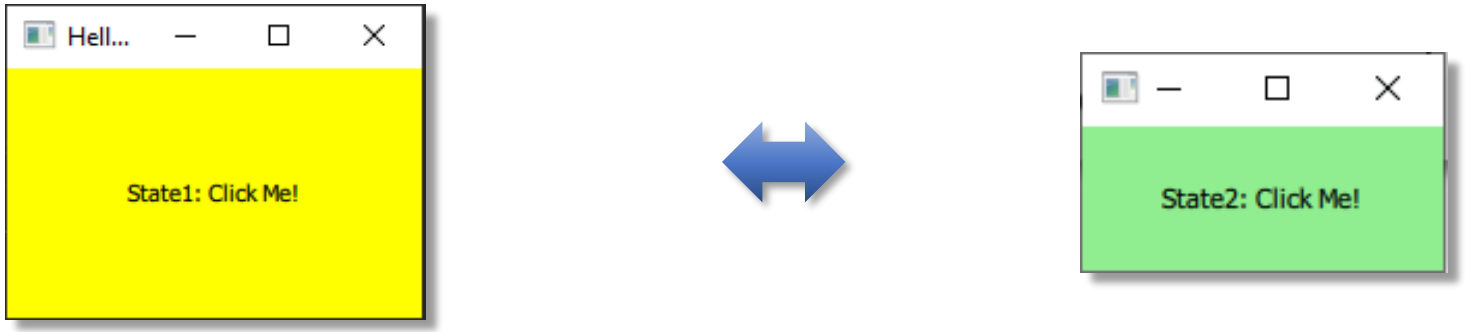
Состояния и переходы

Состояния похожи на шаги в истории, и вы можете их образно сравнить с отдельными кадрами на кинопленке. Наблюдая за кадрами кинофильма, можно сказать, например, что только минуту назад супермен был в состоянии полета в пункте «А», а теперь он уже находится в состоянии приземления в пункт «В».

Для того чтобы движения объектов были плавными, нужно внедрить между какими-то двумя состояниями какой-нибудь красивый **переход**.

Состояния

Состояния в языке *QML* представлены при помощи элемента ***State***. Каждое отдельно взятое состояние - это конфигурация, так как с его помощью можно свойствам элементов присвоить целые наборы значений. Из состояний можно сформировать целые списки, а так же с их помощью можно запускать на выполнение функции ***JS***, управлять изменением фиксации и изменять элементы предков.



*Смена состояний в примере будет осуществляться нажатием мыши на область элемента.

Состояния

```
import QtQuick 2.15
Rectangle {
    id: rect
    width: 360;   height: 360
    state: "State2"
    Text {
       id: txt
       anchors.centerIn: parent
    }
    . . .
   MouseArea {
     anchors.fill: parent
     onClicked:
          parent.state = (parent.state == "State1") ? "State2 " : "State1"
   }
}
```

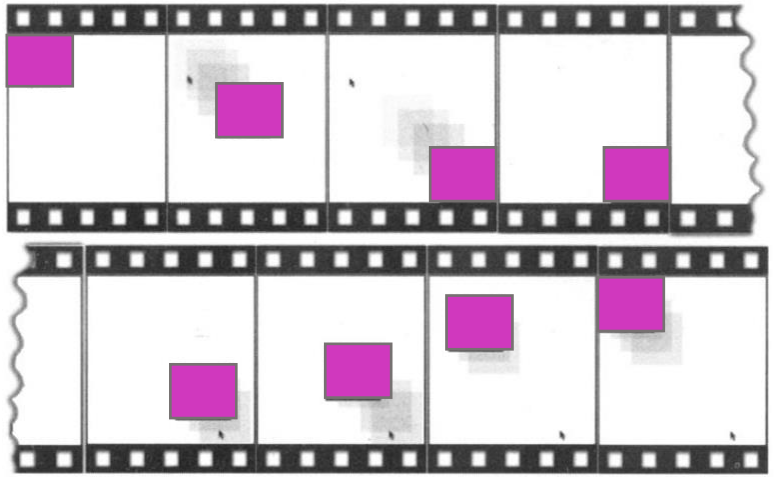
```
states: [
   State {
      name: "State1"
      PropertyChanges {
         target: rect
         color: "lightgreen"
         width: 150 ; height: 60
      }
     PropertyChanges {
         target: txt
         text: "State2: Click Me!"
     }
   },
. . .
]
```

```
State {
      name: "State2"
     PropertyChanges {
         target: rect
         color: "yellow"
         width: 200
         height: 120
    }
    PropertyChanges {
         target: txt
         text: "State1: Click Me!“
    }
}
```

Переходы

Переходы применяются к двум и более **состояниям** и описывают, как между состояниями должна проходить анимация, то есть определяют, как элементы изменяются со сменой одного состояния другим.

Анимационный движок выполнит всю необходимую работу и решит, как наилучшим образом распределить кадры по времени. Он определит и то, какие кадры должны быть созданы и показаны, а какие можно не создавать и не показывать, а это очень важно, так как влияет на производительность. Синтаксис для определения переходов практически идентичен синтаксису определения состояний.



*В примере создается анимация из двух состояний, соединенных переходами.

Переходы

```
import QtQuick 2.15
    Item {
        width: 300 ; height: 300
        Rectangle {
            id: rect
            width: 100 ; height: 100
            color: "magenta"
            state: "State1"
            Text {
                anchors.centerIn: parent
                text: "Click me ! "
            }
            MouseArea {
                anchors.fill: rect
                onClicked:
                    rect.state = (rect.state == "State1") ? "State2“ : "State1“
        }
        states: [
            . . .
        ]
        truncitions: [
            . . .
        ]
    }
}
```

```
states: [
    State {
      name: "State1"
      PropertyChanges {target: rect; x: 0; y: 0}
    },
    State {
        name: "State2"
        PropertyChanges {target: rect; x: 200; y: 200}
    }
]
```

```
transitions: [
    Transition {
        from: "State1"; to: "State2"
        PropertyAnimation {
            target: rect;  properties: “x, y";
            easing.type: Easing.InCirc
            duration: 1000
        }
    },
    Transition {
        from: "State2"; to: "State1"
        PropertyAnimation {
            target: rect; properties: “x, y";
            easing.type: Easing.InBounce
            duration: 1000
        }
    }
]
```

Переходы

```
transitions:
Transition : {
    from: " * " ; to: " * "
    PropertyAnimation {
        target: rect;
        properties: "x, y";
        easing.type: Easing.InCirc
        duration: 1000
    }
}
```

* Если бы мы использовали одинаковые смягчающие линии, то код обоих элементов ***PropertyAnimation*** был бы идентичен, и тогда мы могли бы сократить его и использовать шаблонный переход.

Модуль частиц

Модуль частиц ***QtQuick.Particles*** позволяет создавать потрясающие визуальные эффекты. Эти эффекты производятся отображением большого количества частиц и могут очень пригодиться в реализации видеоигр для симулирования взрывов, летающих метеоритов, разлетающихся космических кораблей и многого другого.

Четыре следующих компонента составляют стержень модуля частиц:

- элемент ***ParticleSystem*** является центральным - он запускает системный таймер для управления временной линией;
- элемент ***Emitter*** (эмиттер) - излучает частицы;
- элемент ***ParticlePainter*** - служит для отображения частиц. Сам элемент может быть элементом растрового изображения ***ImageParticle***, а также и элементом шейдера, представленным элементом ***CustomParticle*** .
- элемент ***Affector*** (эффектор)- изменяет поведение частиц после их создания эмиттером.

Тип	Описание
Age	Изменение срока жизни частиц
Friction	Сила трения
Turbulence	Сила турбулентных потоков
Wander	Изменение траектории частиц случайным образом

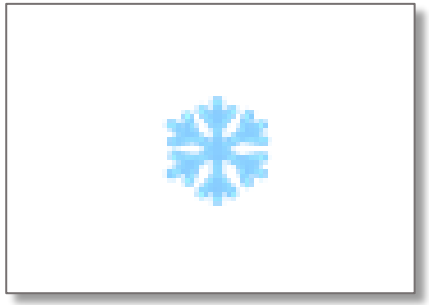
*Некоторые эффекторы модуля *QtQuick.Partices 2.0*

Модуль частиц

Для растровых изображений снежинок, которые являются частицами, мы используем элемент ***ImageParticle.***

Создаем эмиттер - он нам необходим не только для излучения частиц. А так же

- управление сроком жизни частиц (св-во **lifeSpan**).
- управление размерами частиц (св-ва **size**, **endSize** и **sizeVariation**).
- управление частотой излучения частиц в секунду (св-во **emitRate**).
- управление скоростью (св-во **velocity**). На самом деле это свойство может управлять не только скоростью - в нашем случае с помощью элемента ***AngleDirection*** оно так же управляет и направлением движения.



*Поскольку снежинки должны падать сверху вниз, нам нужно указать угол падения, - для этого мы присваиваем свойству **velocity** элемент ***AngleDirection.*** Угол излучения частиц задается свойством **angle** и варьируется в диапазоне от 0 до 360 градусов, при этом 0 градусов означает, что движение будет производиться вправо. Но нам нужно чтобы снежинки перемещались сверху вниз, поэтому указываем угол 90 град.

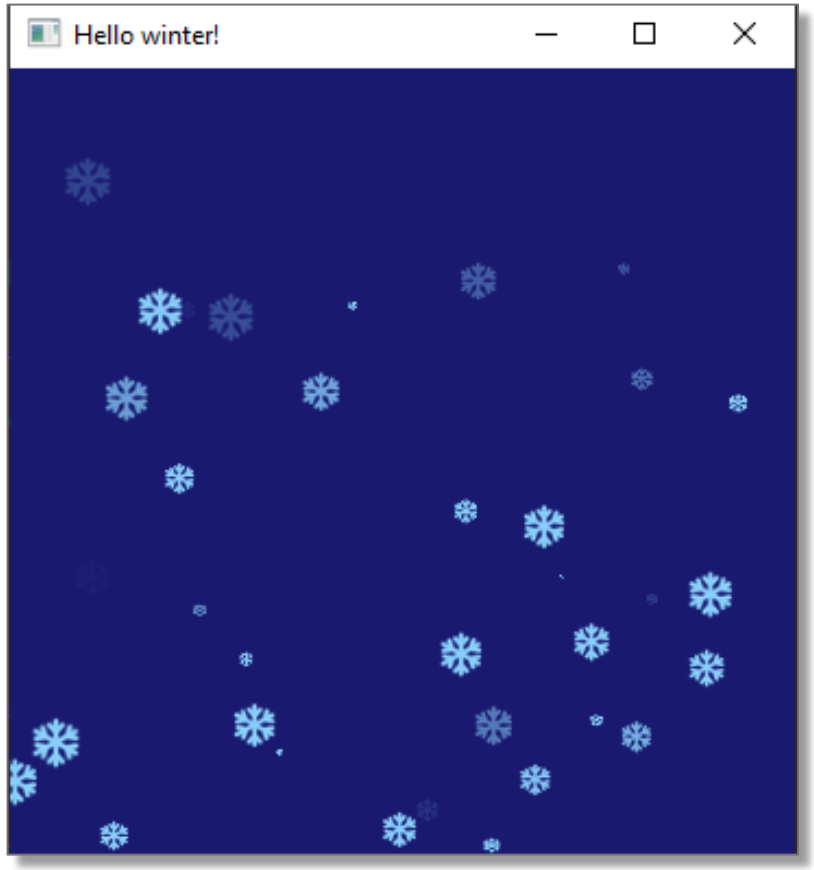
“Gravity”

Добавим к симуляции немного ветра - чтобы снежинки, пройдя какую-то часть пути, резко ускорились в каком-либо направлении, например вправо, - можно сделать нашу анимацию еще более реалистичной и впечатляющей. Теперь задействуем четвертый из основных элементов модуля частиц - *эффектор*.

*__Для демонстрации эффекта ветра__ возьмем элемент ***Gravity***. Его назначение, как и следует из его имени, это применение силы «гравитацию», с помощью которой он способен притягивать объекты в ту или иную сторону.

```
Gravity {
    y: parent.height / 2
    width: parent.width
    height: parent.height
    angle: 0  // притяжение вправо
    acceleration: 250
}
```

Пример.
Симуляция снегопада

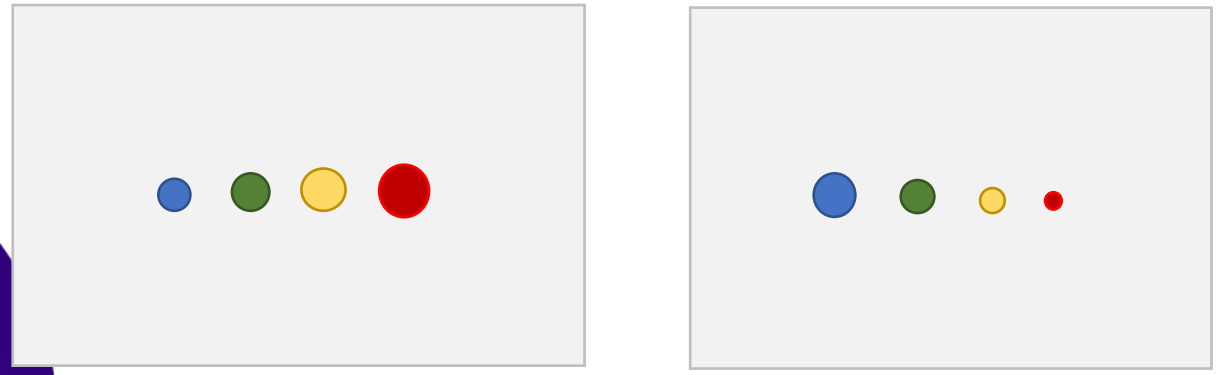



```
import QtQuick.Particles 2.0
    Rectangle {
        width: 360;    height: 360
        color: "MidnightBlue"
        ParticleSystem {
            anchors.fill: parent
            ImageParticle {
                source: "qrc: / snowflake. png“
            }
            Emitter {
                width: parent.width
                height: parent.height
                anchors.bottom: parent.buttom
                lifeSpan: 10000
                sizeVariation: 16
                emitRate: 20
                velocity:
                    AngleDirection {
                        angle: 90 ;// базовый угол
                        angleVariation: 10 // max изм. угла
                        magnitude: 100    // скорость частицы
                    }
            }
        // + Gravity
    }
```

Домашка # 11

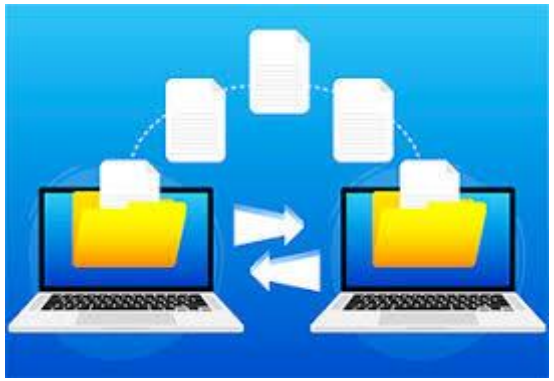
Использовать анимацию для привлечения внимания пользователя или наоборот чтобы скрыть к-л процесс в своих приложениях

1. Трансформация кнопки



2. Анимация загрузки

3. Перезапись информации



Может быть интересным

1. https://infogra.ru/ui/12-printsipov-primeneniya-animatsii-v-polzovatelskih-interfejsah
2. https://www.uprock.ru/articles/ui-animacii-vse-chto-vam-nuzhno-znat-primery
3. https://adn.agency/blog/article/printsipy_animatsii_v_dizayne_interfeysov_tezisy_animation_handbook
4. https://cloudmakers.ru/kontseptualnaya-animatsiya-sozdanie-dizajna-polzovatelskogo-interfejsa/
5. https://lottiefiles.ru/ui-animation-examples/
6. https://habr.com/ru/articles/306348/