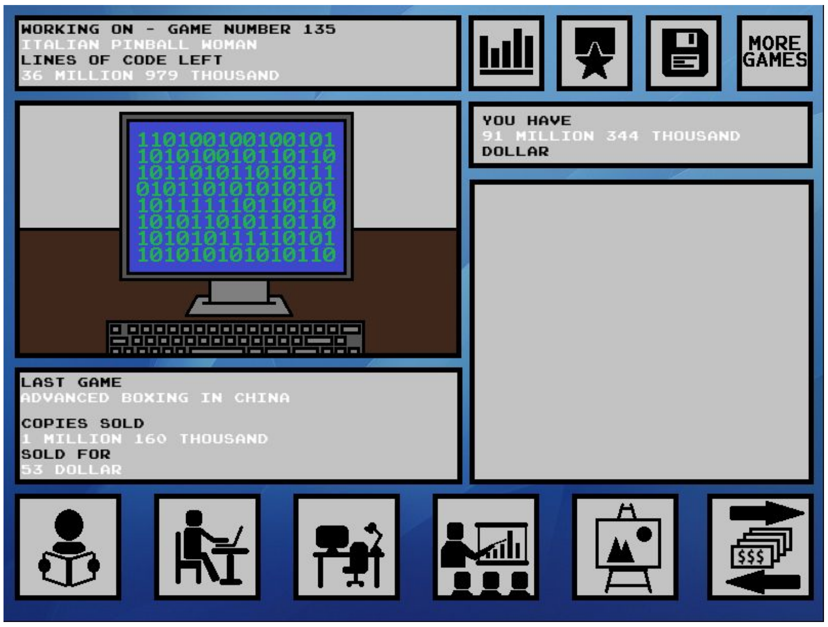
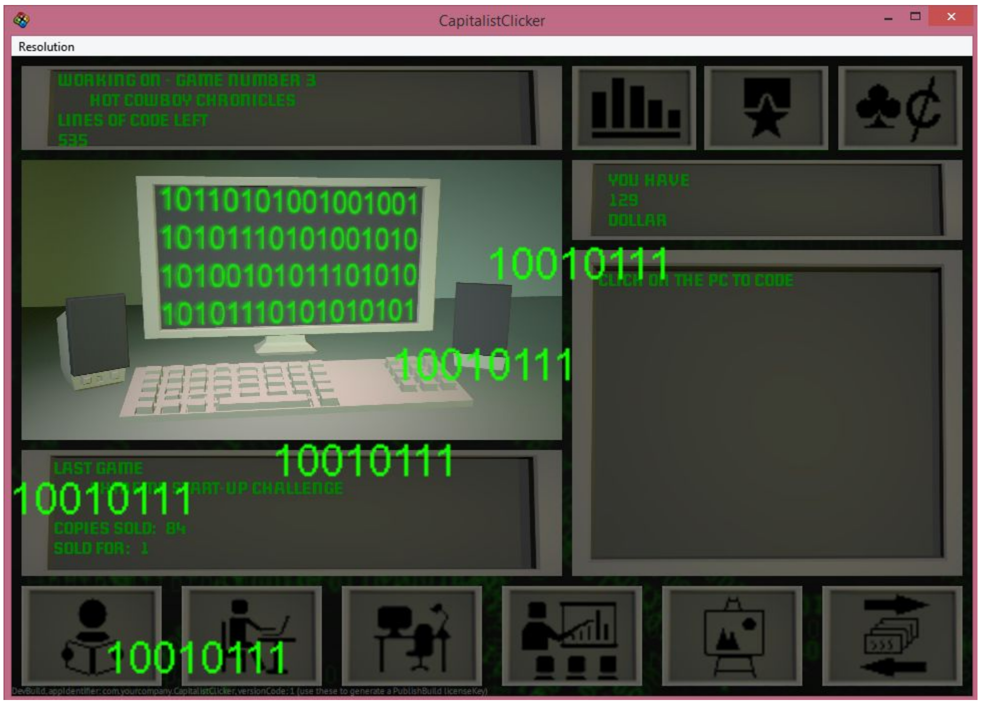
How To Make An Incremental/Idle Game with V-Play

The end result of this tutorial will be a clone/direct rip-off of Blobzone's "*Idle Game Dev*" on Newgrounds (-> http://www.newgrounds.com/portal/view/678296). It is an Idle Game resp. Incremental Game/ Clicker Game like "*AdVenture Capitalist*" or "*Cookie Clicker*".





Original Vs. Remake

Here we go...

First off, download the assets from this link:
https://github.com/ThomasKriz/IdleGameDevClone/blob/master/assetsForTutorial.zip
(Assets have either been created with Autodesk Maya, Photoshop or are ripped from the original game using the Windows Snipping Tool)

New Project

Create a **New - Empty V-Play 2 Project** with the name **IdleGameDevClone** and set the **interface orientation** to **Landscape**.
Now put the downloaded resources into the assets folder of your project.

In the **Main.qml**, create a **GameWindow** and a **Scene** and set the resolution(s).
Also, add a (background) **Image**.

```
import VPlay 2.0
import QtQuick 2.0

GameWindow {

  id: gameWindow
  activeScene: scene

  // this resolution is for iPhone 4 & iPhone 4S
  screenWidth: 960
  screenHeight: 640

     Scene {
     id: scene

     // the "logical size" - the scene content is auto-scaled to match the GameWindow size
     width: 480
     height: 320

        Image {
        anchors.fill: parent
        source: "../assets/bg.png"
     }
  }
}
```

Game Logic Variables

Underneath this game lies a very simple “economic simulation”. You are a game dev, everytime you finish a game, you receive money.
Each game takes a certain amount of “Lines of Code” to be finished -> These “Lines Of Code” get coded/diminish when the player clicks on the computer.
With the money, the player can upgrade his game dev company: He can increase his skill and hire programmers that automatically reduce the “Lines Of Code”, ...

In order to make all of this work, we have to introduce some variables to the **Scene**.

```
    property int money: 100 // the player starts with this money

  property int gameNo: 1 // the player starts developing his first game
  property int linesToCode: 100 // after 100 lines of code, the first game is finished

  property int linesOfCodePerSecond: 0 // leave it at zero, we’ll get to that soon
  property int linesOfCodePerClick: 0 // dito

  property string gameName: "Adult Soccer Simulator" // each game has its own name
  property string newGameName: " "     // we’ ll get to that later

    // the last game’s information is displayed and gets updated every time the current game is finished
  property int lastLinesToCode: 100
  property string lastGameName: " "
  property int copiesSold: 0
  property int soldFor: 0
```

Further variables will follow soon :) But first of all, lets display some of those we just added!

Display Money & LinesToCode/GameName

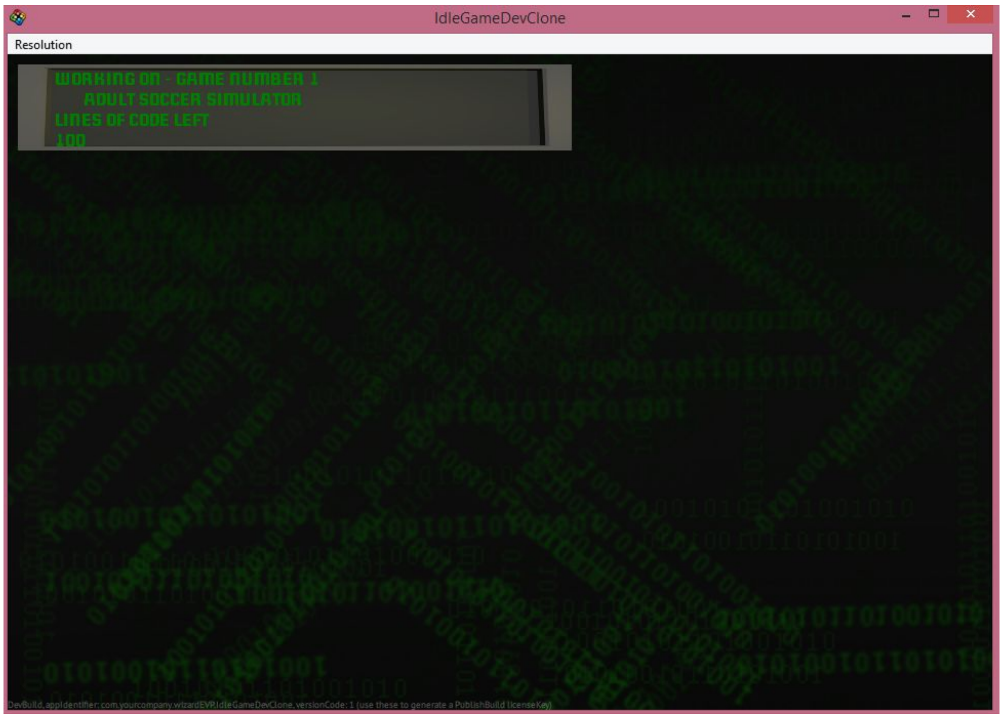
To make our text output nicer, quickly write the following into the **GameWindow.**

```
property alias myFont: myFont
  FontLoader {
    id: myFont
    source: "../assets/wheatonCapitals.ttf"
  }
```

This will allow us to use a custom font. Now, add this to the scene, just below the (background) Image.

```
Rectangle {
        id: topLeftStatus
        x: 5
        y: 5
        width: 270
        height: 42
        Image {
           anchors.fill: parent
           source: "../assets/boxWborder.png"
        }
        Text {
           y:2
           font.family: myFont.name
           color: "green"
           font.pixelSize: 8
           text: "        Working On - Game Number " + scene.gameNo + "
           " + scene.gameName +"
      Lines Of Code Left "+"
      " + scene.linesToCode
        }
     }
```

The **x** and **y** properties give the text a little padding, so that it doesnt overlap the grey border of the **boxWborder.png** background image.
Also, the text property is written in a bit of a weird way: The line breaks are recognised automatically, just as you put them (within the text property),.

Now, your app should look like this:


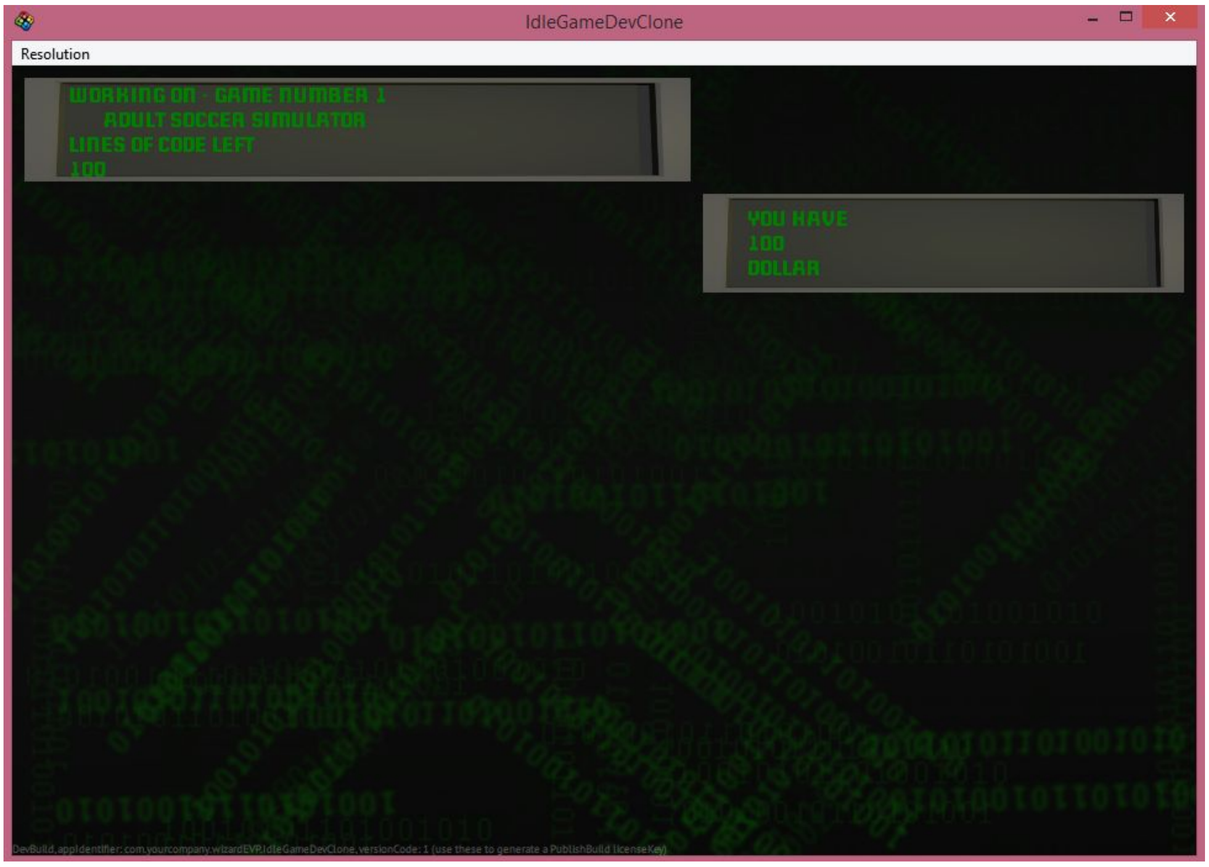
```
WORKING ON - GAME NUMBER 1
    ADULT SOCCER SIMULATOR
LINES OF CODE LEFT
100
```

DevBuild, appIdentifier: com.yourcompany.wizardEVP.IdleGameDevClone, versionCode: 1 (use these to generate a PublishBuild licenseKey)

Let’s make another “status bar” for the money. Add this -very similar- block of code below the above.

```
Rectangle {
        id: moneyStatus
        x: 280
        y: 52
        width: 195
        height: 40
        Image {
           anchors.fill: parent
           source: "../assets/boxWborder.png"
        }
        Text {
           font.family: myFont.name
           color: "green"
           font.pixelSize: 8
           y: 5
           text: "          You Have
      " + scene.money +"
      Dollar "
        }
     }
```

Nice!



WORKING ON - GAME NUMBER 1
ADULT SOCCER SIMULATOR
LINES OF CODE LEFT
100

YOU HAVE
100
DOLLAR

DevBuild, appIdentifier: com.yourcompany.wizardEVP.IdleGameDevClone, versionCode: 1 (use these to generate a PublishBuild licenseKey)

TextDisplayArea

We’re going to add yet another rectangle to the **Scene**, this time it is a “TextDisplayArea”, whenever we hover over an interactive element this “TextDisplayArea” should display information on the available interaction.

Add a new .qml-File by the name of **TextDisplayArea.qml** to the qml directory and fill it with the following code.

```
import QtQuick 2.0

Item {
  property string textToDisplay: " "

  Rectangle{
    x: 280
    y: 97
    width: 195
    height: 163
    Image {
      anchors.fill: parent
      source: "../assets/boxWborder.png"
    }
    Text {
      x:13
      y:10
      font.family: myFont.name
      color: "green"
      font.pixelSize: 8
      text: textToDisplay
    }
  }

  function clearText(){
    textToDisplay = " "
  }

}
```

Again, the **x** and **y** Text properties give a bit of padding..
Add the **TextDisplayArea** to the **Main.qml** like this:

```
TextDisplayArea {
      id: textDisplay
    }
```

Now we can set the Text to be displayed from everywhere; just say:

```
textDisplay.textToDisplay = "Click on the PC to Code"
```

To clear, say: `textDisplay.clearText()`

Put The “Clicker” into “Clicker Game”

Let’s make these Lines Of Code go down! (We’re also going to see the **TextDisplayArea** in action.) In order to do that we have to introduce a clickable area to the game. Add a new .qml-File by the name of **ClickerWindow.qml** to the qml directory.

Add it to the **Scene** below the **TextDisplayArea**.

```
ClickerWindow{}
```

Fill the **ClickerWindow.qml** like this:

```
import QtQuick 2.0

Item {
  Rectangle{
    id: clickerWindow
    x: 5
    y: 52
    width: 270
    height: 140

    Image{
      anchors.fill: parent
      source: "../assets/computer_bytecode.png"
    }
    MouseArea{
      anchors.fill: parent
      hoverEnabled: true
      onEntered:{
        textDisplay.textToDisplay = "Click on the PC to Code"
      }
      onClicked:{
        scene.linesOfCodePerClick = scene.codingSkillLvl
        scene.linesToCode = scene.linesToCode - scene.linesOfCodePerClick
      }
      onExited: {
        textDisplay.clearText()
      }
    }
  }
}
```

If you hover over the nice computer image (I did that in Autodesk Maya), you’ll see some text popping up in the **TextDisplayArea**, but if you click it, you’ll get an error. We have to introduce another variable in our **Scene**.

```
        property int codingSkillLvl: 1
        property double codingSkillPrice: 25 // just add it too, will be important later
```

If you click on the computer now, the “Lines Of Code Left” will decrease by 1 each click, as the **codingSkillLvl** property is 1. But, what if you click on the computer after the lines of code have reached 0. Well, they reach a negative value. This is not what we want, whenever the Lines Of Code reach 0, a new game should be developed. Therefore we introduce a **Timer** in our **Scene**.

```
Timer{
        id: checkForGameFinished
        interval: 5
        running: true
        repeat: true
        onTriggered: {
           // check if game is finished
           if (scene.linesToCode <= 0) {
              scene.linesToCode = scene.lastLinesToCode
              scene.lastLinesToCode = scene.linesToCode
              scene.linesToCode = scene.lastLinesToCode + Math.ceil(((scene.lastLinesToCode * (15)) / 100)) +
(scene.gameNo * 10 * (scene.linesToCode/4))
              //scene.gameFinished()

           }
        }
     }
```

This timer basically checks if a game has been finished at all times, as its interval is very small.

For now, if a game is finished, the game just updates the **linesToCode** by a formula that may be adjusted freely in terms of balancing but what we need is an increment of the Game Number, a new game name, etc.

Therefore, we have to introduce a couple of new variables.

More Game Logic Variables

Add these below the **codingSkillLvL** and **codingSkillPrice** property.

```
property int programmerSkillLvl: 0
property double programmerSkillPrice: 100

property int workEnvironmentLvl: 0
property double workEnvironmentPrice: 250

property int marketingLvl: 0
property double marketingPrice: 500

property int graphicsLvl: 1
property double graphicsPrice: 1000
```

These five similarily named -Lvl & -Price variables correspond to the 5 buttons at the bottom of the screen that we are going to implement in the next step. But first, lets introduce the **gameFinished()** function.

```
function gameFinished() {
        scene.lastGameName = gameName

        gameName = “A new game name will be generated here soon :)”

        scene.copiesSold = (42 * Math.ceil(marketingLvl * 1.5)) + (scene.gameNo * 42)
        scene.soldFor = graphicsLvl
        scene.money += scene.copiesSold * scene.soldFor

        scene.gameNo++
    }
```

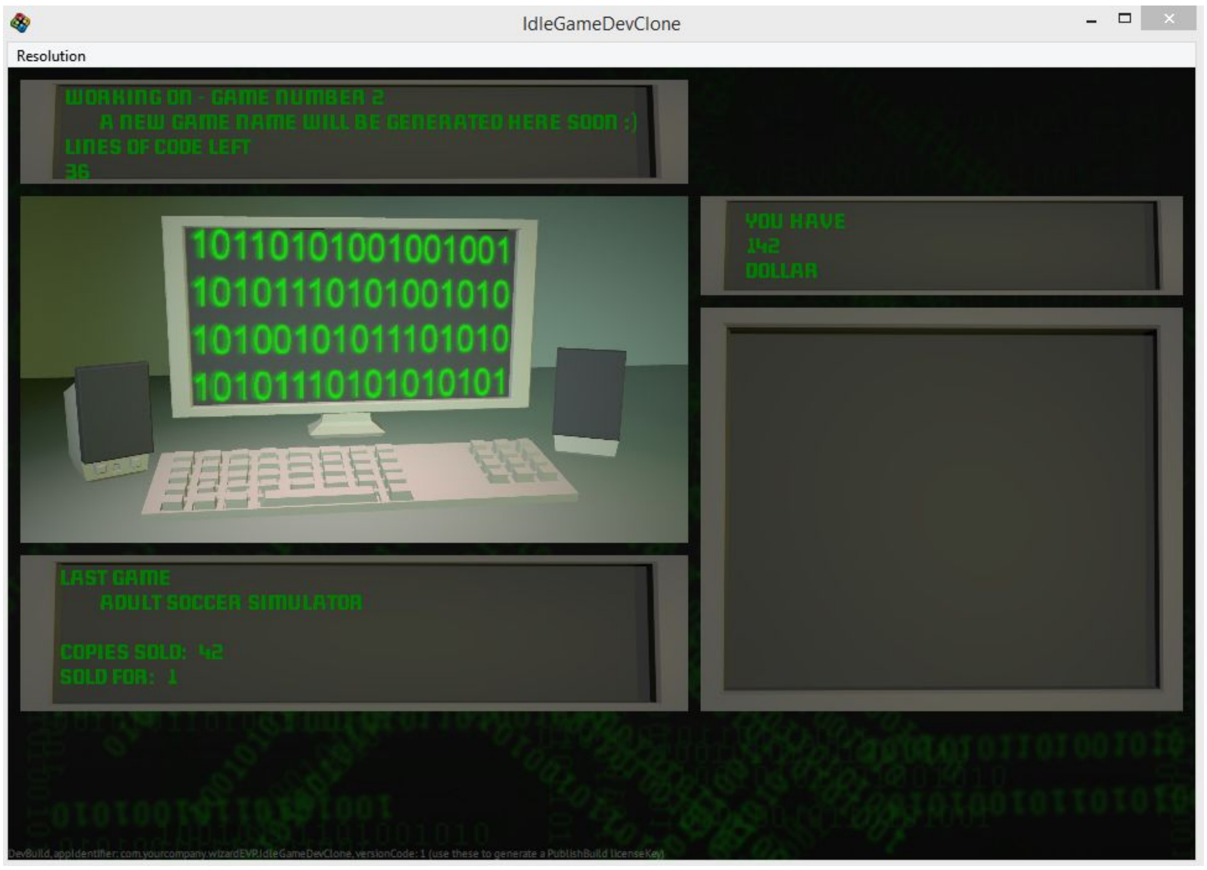
Alright, we already earn money when a game is finished. We’ll save the name generation for later… What about those **lastGameName**, **copiesSold**, **soldFor** variables? You know what, lets display them too!

Add this to the **Scene**, preferably below the **ClickerWindow{}**

```
Rectangle {
        id: lastGameStatus
        x: 5
        y: 197
        width: 270
        height: 63
        Image {
            anchors.fill: parent
            source: "../assets/boxWborder.png"
```

```
    }
    Text {
        y: 4
        font.family: myFont.name
        color: "green"
        font.pixelSize: 8
        text: "        Last Game" + "
        " + scene.lastGameName + "

Copies Sold:  " + scene.copiesSold +"
Sold For:  " + scene.soldFor
    }
}
```

Wow, looks like we’re going somewhere!



```
WORKING ON - GAME NUMBER 2
     A NEW GAME NAME WILL BE GENERATED HERE SOON :)
LINES OF CODE LEFT
36

YOU HAVE
142
DOLLAR

LAST GAME
     ADULT SOCCER SIMULATOR

COPIES SOLD:  42
SOLD FOR:  1

DevBuild, appIdentifier: com.yourcompany.wizardEVP.IdleGameDevClone, versionCode: 1 (use these to generate a PublishBuild licenseKey)
```

ButtonBarBelow

We have to invest our hard earned money into some upgrades, so lets set up a new .qml-File by the name of **ButtonBarBelow.qml**. This will contain 6 (hover) Buttons, 5 of them corresponding to the -Lvl & -Price variables from before.

Fill **ButtonBarBelow.qml** like this. (And add ButtonBarBelow{} to your Scene)

```
import QtQuick 2.0
import VPlay 2.0

Item{
   property int topY: 265

   //buttons will follow here
}
```

We **import VPlay 2.0** because we'll need the **SpriteSequenceVPlay**..
Anyways, as all 6 buttons nearly have the same code I will now start with the first button, the following ones will be a matter of copy and paste.

First of all, we are going to add the button that lets us upgrade our own clicking power.

```
Rectangle{
       id: upgradeClickingSkillButton
       x: 5
       y: topY
       width: 70
       height: 50

       SpriteSequenceVPlay{
           id:skillButtonAnimation
           defaultSource: "../assets/skillSequence.png"
           SpriteVPlay {
                  name: "idle"
                  frameWidth: 70
                  frameHeight: 50
                }
           SpriteVPlay {
                  name: "hover"
                  frameWidth: 70
                  frameHeight: 50
                  frameX: 70
                }
           SpriteVPlay {
                  name: "clicked"
                  frameWidth: 70
                  frameHeight: 50
                  frameX: 140
                }
```

```
    }

    MouseArea{
        anchors.fill: parent
        hoverEnabled: true
        onEntered:{
            skillButtonAnimation.jumpTo("hover")
            textDisplay.textToDisplay = "Upgrade Coding Skill.
" + scene.codingSkillPrice + "
Dollar

Current LVL " + scene.codingSkillLvl +"

Your coding skill determines how
many lines you can code per click"
        }
        onClicked:{
            if(scene.money - scene.codingSkillPrice >= 0){
                scene.codingSkillLvl += 1
                scene.money -= scene.codingSkillPrice
                scene.codingSkillPrice += Math.ceil(((scene.codingSkillPrice * 25 )/ 100))
                exited()
                entered() // call both exited and entered to refresh the text
                skillButtonAnimation.jumpTo("clicked")
            } else {
                skillButtonAnimation.jumpTo("hover")
            }
        }
        onExited: {
            skillButtonAnimation.jumpTo("idle")
            textDisplay.clearText()
        }
    }
  }
```

The **SpriteSequenceVPlay** basically holds three pictures, for different states of the button (clicked, hover, not clicked/not hover/idle). To enable hover, make sure to set the **MouseArea** to **hoverEnabled: true**

5 other buttons will follow now. Basically, the code for the **upgradeClickingSkillButton** has to be copypasted 5 times, then you have to adjust:

- **id** (Rectangle and SpriteSequenceVPlay)
- **defaultSource**
- the id before **.jumpTo(**“idle”**)**...
- The functionality within **onClicked**
- The text to be displayed when **hover**ed over

The code for each of the buttons follows:

```
Rectangle{
      id: hireProgrammerButton
      x: 5+70+10
      y: topY
      width: 70
      height: 50

      SpriteSequenceVPlay{
          id:programmerButtonAnimation
          defaultSource: "../assets/programmerSequence.png"
          SpriteVPlay {
                name: "idle"
                frameWidth: 70
                frameHeight: 50
              }
          SpriteVPlay {
                name: "hover"
                frameWidth: 70
                frameHeight: 50
                frameX: 70
              }
          SpriteVPlay {
                name: "clicked"
                frameWidth: 70
                frameHeight: 50
                frameX: 140
              }
      }

      MouseArea{
          anchors.fill: parent
          hoverEnabled: true
          onEntered:{
              programmerButtonAnimation.jumpTo("hover")
              textDisplay.textToDisplay = "Hire Programmer.
" + scene.programmerSkillPrice + "
Dollar

Current LVL " + scene.programmerSkillLvl +"

Programmers help you coding
your games"
          }
          onClicked:{
              if(scene.money - scene.programmerSkillPrice >= 0){
                  scene.programmerSkillLvl += 1
                  scene.money -= scene.programmerSkillPrice
                  scene.programmerSkillPrice += Math.ceil(((scene.programmerSkillPrice * 25 )/ 100))
                  exited()
                  entered() // call both exited and entered to refresh the text
                  programmerButtonAnimation.jumpTo("clicked")
              } else {
                  programmerButtonAnimation.jumpTo("hover")
              }
          }
          onExited: {
              programmerButtonAnimation.jumpTo("idle")
              textDisplay.clearText()
          }
      }
  }
```

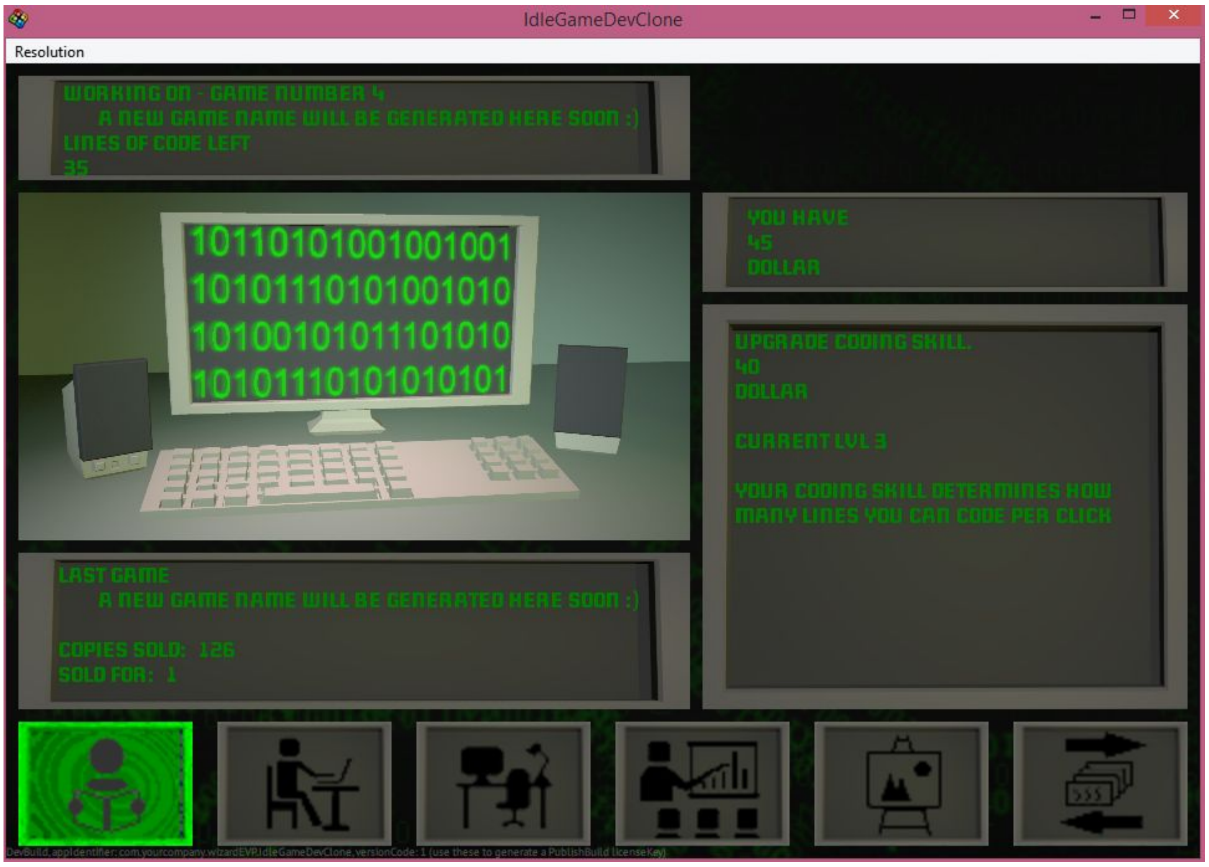
```
Rectangle{
      id: upgradeWorkEnvironmentButton
      x: 5+70+10+70+10
      y: topY
      width: 70
      height: 50

      SpriteSequenceVPlay{
          id:environmentButtonAnimation
          defaultSource: "../assets/environSequence.png"
          SpriteVPlay {
                name: "idle"
                frameWidth: 70
                frameHeight: 50
              }
          SpriteVPlay {
                name: "hover"
                frameWidth: 70
                frameHeight: 50
                frameX: 70
              }
          SpriteVPlay {
                name: "clicked"
                frameWidth: 70
                frameHeight: 50
                frameX: 140
              }
      }
      MouseArea{
          anchors.fill: parent
          hoverEnabled: true
          onEntered:{
              environmentButtonAnimation.jumpTo("hover")
              textDisplay.textToDisplay = "Upgrade Working Environment.
" + scene.workEnvironmentPrice + "
Dollar

Current LVL " + scene.workEnvironmentLvl +"

A good working environment helps
your programmers to work
faster and better."
          }
          onClicked:{
              if(scene.money - scene.workEnvironmentPrice >= 0){
                  scene.workEnvironmentLvl += 1
                  scene.money -= scene.workEnvironmentPrice
                  scene.workEnvironmentPrice += Math.ceil(((scene.workEnvironmentPrice * 25 )/ 100))
                  exited()
                  entered() // call both exited and entered to refresh the text
                  environmentButtonAnimation.jumpTo("clicked")
              } else {
                  environmentButtonAnimation.jumpTo("hover")
              }
          }
          onExited: {
              environmentButtonAnimation.jumpTo("idle")
              textDisplay.clearText()
          }
      }
  }
```

```
Rectangle{
      id: upgradeMarketingButton
      x: 5+70+10+70+10+70+10
      y: topY
      width: 70
      height: 50

      SpriteSequenceVPlay{
        id:marketingButtonAnimation
        defaultSource: "../assets/marketingSequence.png"
        SpriteVPlay {
              name: "idle"
              frameWidth: 70
              frameHeight: 50
            }
        SpriteVPlay {
              name: "hover"
              frameWidth: 70
              frameHeight: 50
              frameX: 70
            }
        SpriteVPlay {
              name: "clicked"
              frameWidth: 70
              frameHeight: 50
              frameX: 140
            }
      }

      MouseArea{
        anchors.fill: parent
        hoverEnabled: true
        onEntered:{
          marketingButtonAnimation.jumpTo("hover")
          textDisplay.textToDisplay = "Upgrade Marketing.
" + scene.marketingPrice + "
Dollar

Current LVL " + scene.marketingLvl +"

Marketing helps you to
sell more copies of your games"
        }
        onClicked:{
          if(scene.money - scene.marketingPrice >= 0){
            scene.marketingLvl += 1
            scene.money -= scene.marketingPrice
            scene.marketingPrice += Math.ceil(((scene.marketingPrice * 25 )/ 100))
            exited()
            entered() // call both exited and entered to refresh the text
            marketingButtonAnimation.jumpTo("clicked")
          } else {
            marketingButtonAnimation.jumpTo("hover")
          }
        }
        onExited: {
          marketingButtonAnimation.jumpTo("idle")
          textDisplay.clearText()
        }
      }
  }
```

```
Rectangle{
      id: upgradeGraphicsButton
      x: 5+70+10+70+10+70+10+70+10
      y: topY
      width: 70
      height: 50

      SpriteSequenceVPlay{
          id:graphicsButtonAnimation
          defaultSource: "../assets/graphSequence.png"
          SpriteVPlay {
                name: "idle"
                frameWidth: 70
                frameHeight: 50
              }
          SpriteVPlay {
                name: "hover"
                frameWidth: 70
                frameHeight: 50
                frameX: 70
              }
          SpriteVPlay {
                name: "clicked"
                frameWidth: 70
                frameHeight: 50
                frameX: 140
              }
      }

      MouseArea{
          anchors.fill: parent
          hoverEnabled: true
          onEntered:{
              graphicsButtonAnimation.jumpTo("hover")
              textDisplay.textToDisplay = "Upgrade Graphics.
" + scene.graphicsPrice + "
Dollar

Current LVL " + scene.graphicsLvl +"

Upgrade your graphics and sell
your games for more cash"
          }
          onClicked:{
              if(scene.money - scene.graphicsPrice >= 0){
                  scene.graphicsLvl += 1
                  scene.money -= scene.graphicsPrice
                  scene.graphicsPrice += Math.ceil(((scene.graphicsPrice * 25 )/ 100))
                  exited()
                  entered() // call both exited and entered to refresh the text
                  graphicsButtonAnimation.jumpTo("clicked")
              } else {
                  graphicsButtonAnimation.jumpTo("hover")
              }
          }
          onExited: {
              graphicsButtonAnimation.jumpTo("idle")
              textDisplay.clearText()
          }
      }
  }
```

```
Rectangle{
      id: sellYourCompanyButton
      x: 5+70+10+70+10+70+10+70+10+70+10
      y: topY
      width: 70
      height: 50

      SpriteSequenceVPlay{
         id:resetCompanyButtonAnimation
         defaultSource: "../assets/resetSequence.png"
         SpriteVPlay {
               name: "idle"
               frameWidth: 70
               frameHeight: 50
             }
         SpriteVPlay {
               name: "hover"
               frameWidth: 70
               frameHeight: 50
               frameX: 70
             }
         SpriteVPlay {
               name: "clicked"
               frameWidth: 70
               frameHeight: 50
               frameX: 140
             }
      }

      MouseArea{
         anchors.fill: parent
         hoverEnabled: true
         onEntered:{
            resetCompanyButtonAnimation.jumpTo("hover")
            textDisplay.textToDisplay = "Reset Your Company.

(Not implemented)"
         }
         onClicked:{
            //resetCompanyButtonAnimation.jumpTo("clicked")

         }
         onExited: {
            resetCompanyButtonAnimation.jumpTo("idle")
            textDisplay.clearText()
         }
      }
   }
```

There you have it! Six nice-looking buttons.



WORKING ON - GAME NUMBER 4
A NEW GAME NAME WILL BE GENERATED HERE SOON :)
LINES OF CODE LEFT
35

YOU HAVE
45
DOLLAR

UPGRADE CODING SKILL.
40
DOLLAR

CURRENT LVL 3

YOUR CODING SKILL DETERMINES HOW MANY LINES YOU CAN CODE PER CLICK

LAST GAME
A NEW GAME NAME WILL BE GENERATED HERE SOON :)

COPIES SOLD: 126
SOLD FOR: 1

```
DevBuild, appIdentifier: com.yourcompany.wizardEVP.IdleGameDevClone, versionCode: 1 (use these to generate a PublishBuild licenseKey)
```

For their functionality: Upgrading the Coding Skill works, so do Marketing & Graphics Upgrades. The last button, “Reset your company” has no functionality, it is just a reference to the original “*Idle Game Dev*”.

So, this leaves us with two not-working buttons: *Hire Programmer* and *Working Environment*. Let’s take those into consideration…

Programmer Timer & Balancing

Below the first **Timer**, add another **Timer** to the **Scene**:

```
Timer {
        id: programmerTimer
        interval: 1250
        running: true
        repeat: true
        onTriggered: {
           scene.linesOfCodePerSecond = scene.programmerSkillLvl*4 + scene.workEnvironmentLvl*4

           if (scene.linesToCode > 0){
              scene.linesToCode = scene.linesToCode - scene.linesOfCodePerSecond
           }
        }
    }
```

The **interval** will determine the pace of the programmers working on the game. Each time the **Timer** is triggered, the Lines Of Code get reduced by coding programmers. Hire a programmer and upgrade his working environment to see this timer in action.

Formulas like this
scene.linesOfCodePerSecond = *scene*.programmerSkillLvl*4 + *scene*.workEnvironmentLvl*4
have to be balanced well in order to make the game fun to play (and not too slow, too fast, ...).

The problem of balancing occurs in several places:

scene.copiesSold = (42 * *Math*.ceil(*marketingLvl* * 1.5)) + (*scene*.gameNo * 42)

scene.linesToCode = *scene*.lastLinesToCode + *Math*.ceil(((*scene*.lastLinesToCode * (15)) / 100)) + (*scene*.gameNo * 10 * (*scene*.linesToCode/4))

scene.graphicsPrice += *Math*.ceil(((*scene*.graphicsPrice * 25)/ 100))

…

graphicsLvl and **codingSkillLvl** are very straightforward:
The price of one copy sold is directly equivalent to the Graphics Lvl.
(GraphicsLvl = 1, Price = 1)
The same goes for the codingSkillLvl and the “power of one click”, but this balancing issue is addressed much later when we delve into the Achievements...

Generating Random Game Names

Here comes the fun part, and “*Idle Game Dev*”’s Unique Selling Proposition ;)

Create a new .qml-File: **RandomNameGenerator.qml**

```
import QtQuick 2.0

Item {
  property int arrayLength: 3
  property string tmpString: " "

  property variant words1: ["MEGA", “SUPER”, “CRAZY”]
  property variant words2: ["DRAGON", "PRIEST", "MAFIA"]
  property variant words3: ["QUEST", "CHRONICLES", " "]

  function createRandomGameName(){
      tmpString = " " + words1[Math.floor(utils.generateRandomValueBetween(0,arrayLength))] + " " +
words2[Math.floor(utils.generateRandomValueBetween(0,arrayLength))] + " " +
words3[Math.floor(utils.generateRandomValueBetween(0,arrayLength))]
      scene.newGameName = tmpString
      console.debug(tmpString)
  }
}
```

Fill the Arrays with as many funny words as you want! I used 45 words in each one of them. Make sure to adjust the **arrayLength** property accordingly (e.g. 45).
Also, make sure to use either **Math.floor** or **Math.ceil** on utils.generateRandomValueBetween() as this function would otherwise return a floating point number which wont be applicable as an array index value.

Also, add the **RandomNameGenerator** to the Scene:

```
RandomNameGenerator{
        id: randomNameGenerator
    }
```

Call its **createRandomGameName()** function over its id within **gameFinished()** and apply the newly generated game name.

```
//gameName = "A new game name will be generated here soon :)"

        randomNameGenerator.createRandomGameName()
        gameName = newGameName
```

Cool! You’ll never know what’s next: CRAZY PRIEST QUEST or MEGA MAFIA?

Special Effect: Spawn Some Bytecode

Time for a little Special Effect that will visually enhance our clicking endeavours! Whenever you click on the computer, a transparent bytecode image should spawn.

```
property int spawnBytecodeX: 0
property int spawnBytecodeY: 0

    EntityManager {
        id: entityManager
        entityContainer: scene
    }

    Component {
        id: bytecodeThingie
        EntityBase {
            id: bytecodeEntity
            entityType: "bytecode"

            property int bytecodeX: 0
            property int bytecodeY: 0

            Image {
                x: bytecodeX
                y: bytecodeY
                source: "../assets/bytecode.png"
            }
            MovementAnimation {
                target: parent
                property: "pos"
                minPropertyValue: Qt.point(-600, -600)
                maxPropertyValue: Qt.point(800, 800)
                velocity: Qt.point(utils.generateRandomValueBetween(-400,400),
utils.generateRandomValueBetween(-400,400))
                running: true
                onLimitReached: {
                    bytecodeEntity.removeEntity()
                }
            }
        }
    }

    function spawnBytecode() {
        entityManager.createEntityFromComponentWithProperties(bytecodeThingie, {
            "bytecodeX": scene.spawnBytecodeX,
            "bytecodeY": scene.spawnBytecodeY
        })
    }
```

createEntityFromComponentWithProperties is needed so that each Entity is created with its own properties.

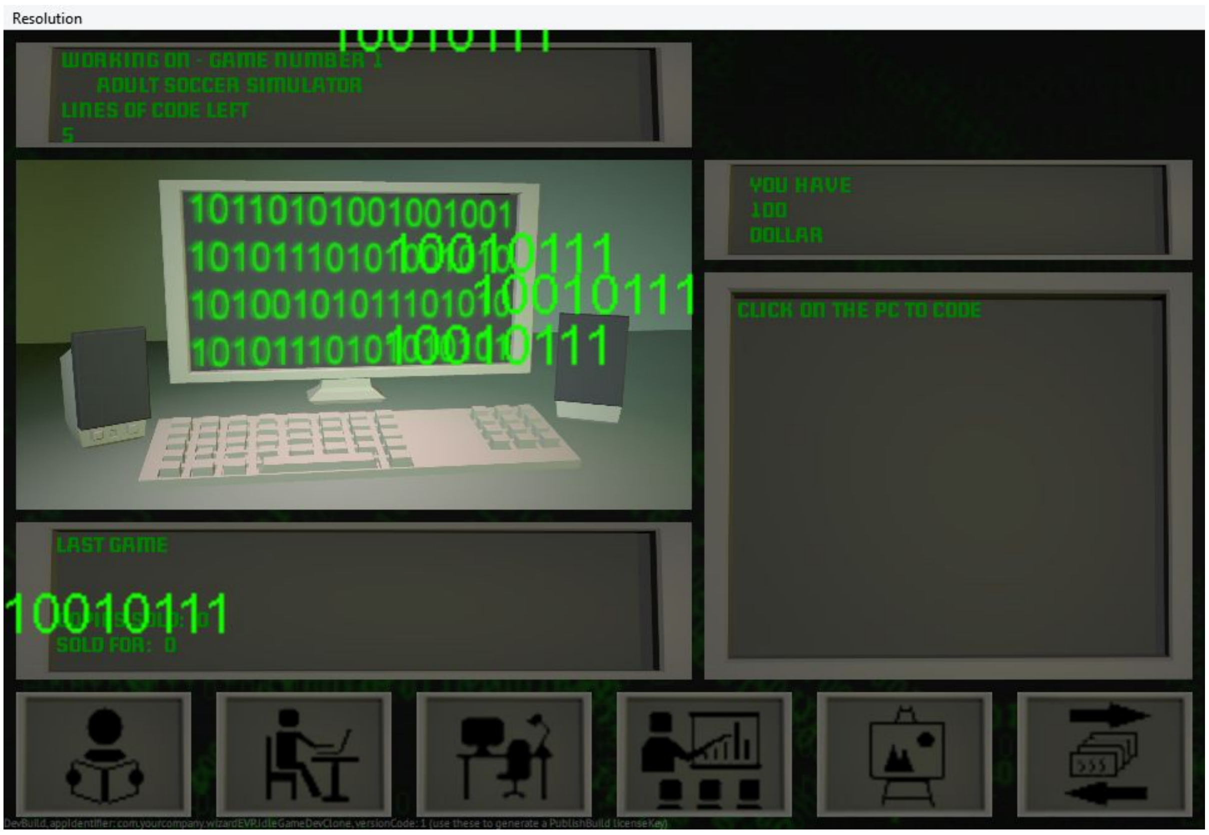
utils.generateRandomValueBetween(-400,400) can also return negative values which is nice as it enables the **bytecodeEntity** to fly in all 4 directions.

The "global" properties spawnBytecodeX and spawnBytecodeY are needed so that the **ClickerWindow** can tell the **bytecodeEntity** where to spawn.

Just add this to the end of the onClicked function within ClickerWindow.qml:

```
scene.spawnBytecodeX = mouseX
scene.spawnBytecodeY = mouseY + 40
scene.spawnBytecode()
```

And here we go!



Side note: Mouse cursor not visible in picture.
Another Side note: Entity pooling might be a better solution but it didn't work

Anyways, time for the finishing touches! It's all about statistics and achievements.

Statistics

Statistics help us keep track of the player's progress. This is important, as we will soon introduce achievements triggered by certain milestones.

At first, add some "global" variables below the wall of properties into the **Scene**.

```
// statistics
    property int totalLinesCodedProgrammers: 0
    property int totalLinesCodedAlone: 0
    property int totalGamesMade: 0
    property int totalGamesSold: 0
    property int totalMoneyEarned: 0
```

Let's create yet another .qml-File, this time it's called **StatusAndAchievementBar.qml**

Add it to your **Scene** (StatusAndAchievementBar{}) and fill the .qml-File like this:

```
import QtQuick 2.0
import VPlay 2.0

Item {

  Rectangle{
    id: statusButton
    x: 280
    y: 5
    width: 62
    height: 42

    SpriteSequenceVPlay{
      id:statusButtonAnimation
      defaultSource: "../assets/statsSequence.png"
      SpriteVPlay {
          name: "idle"
          frameWidth: 62
          frameHeight: 42
        }
      SpriteVPlay {
          name: "hover"
          frameWidth: 62
          frameHeight: 42
          frameX: 62
        }
    }
    MouseArea{
      anchors.fill: parent
      hoverEnabled: true
      onEntered:{
        statusButtonAnimation.jumpTo("hover")
```

```
            textDisplay.textToDisplay = "Total Lines Coded
" + (scene.totalLinesCodedAlone+scene.totalLinesCodedProgrammers) +"
Total Lines Coded Alone
" + scene.totalLinesCodedAlone +"
Total Games Made
" + scene.totalGamesMade +"
Total Games Sold
" + scene.totalGamesSold +"
Total Money Earned
" + scene.totalMoneyEarned +"
Lines Of Code Per Second
" + scene.linesOfCodePerSecond +"
Lines Of Code Per Click
" + scene.linesOfCodePerClick
        }
        onExited: {
            statusButtonAnimation.jumpTo("idle")
            textDisplay.clearText()
        }
      }
    }
}
```

Basically, this works just like one of the Buttons from the **ButtonBarBelow** but it is not clickable. All it does, is display text in the **TextDisplayArea** when hovered over.

Now, all of the Variables have to be updated in the corresponding places!

Within the **programmerTimer**:

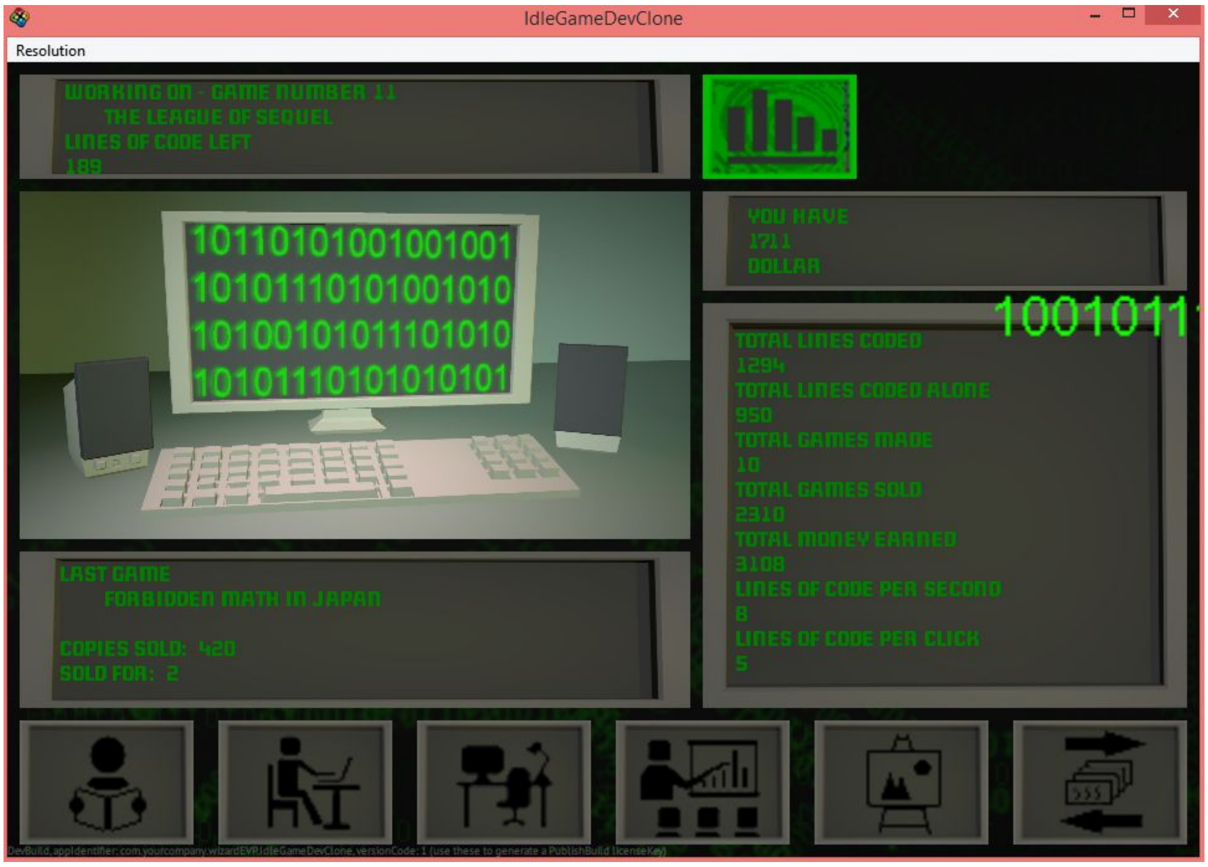
```
            scene.linesToCode = scene.linesToCode - scene.linesOfCodePerSecond
            scene.totalLinesCodedProgrammers += scene.linesOfCodePerSecond // this
```

Within the **ClickerWindow.qml**

```
            scene.linesToCode = scene.linesToCode - scene.linesOfCodePerClick
            scene.totalLinesCodedAlone += scene.linesOfCodePerClick // this
```

Within **gameFinished()**

```
            scene.copiesSold = (42 * Math.ceil(marketingLvl * 1.5)) + (scene.gameNo * 42)
        scene.totalGamesSold += scene.copiesSold // this
        scene.soldFor = graphicsLvl
        scene.money += scene.copiesSold * scene.soldFor
        scene.totalMoneyEarned += scene.copiesSold * scene.soldFor // this
        scene.gameNo++
        scene.totalGamesMade++ // this
```



WORKING ON - GAME NUMBER 11
THE LEAGUE OF SEQUEL
LINES OF CODE LEFT
189

YOU HAVE
1711
DOLLAR

TOTAL LINES CODED
1294
TOTAL LINES CODED ALONE
950
TOTAL GAMES MADE
10
TOTAL GAMES SOLD
2310
TOTAL MONEY EARNED
3108
LINES OF CODE PER SECOND
8
LINES OF CODE PER CLICK
5

LAST GAME
FORBIDDEN MATH IN JAPAN

COPIES SOLD: 420
SOLD FOR: 2

DevBuild, appIdentifier: com.yourcompany.wizardEVP.IdleGameDevClone, versionCode: 1 (use these to generate a PublishBuild licenseKey)

The statistics in action… Time for the achievements.

Achievements

The achievements are milestones that are triggered when the player reaches certain goals. They are displayed in a similar manner as the statistics. Much like with the statistics, we have to introduce “global” variables into the **Scene**.

```
// achievement variables
        property bool coded100ThousandLines: false          // + 25% coding skill
        property bool codedMillionLines: false              // + 50% coding skill
        property bool coded10MillionLines: false            // + 100% coding skill
        property bool coded100MillionLines: false           // + 200% coding skill

        property bool codedThousandLinesAlone: false        // + 100% clicking skill
        property bool coded10ThousandLinesAlone: false      // + 150% clicking skill
        property bool coded100ThousandLinesAlone: false     // + 200% clicking skill
        property bool codedMillionLinesAlone: false         // + 250% clicking skill

        property bool made25games: false                    // + 50% sales
        property bool made100games: false                   // + 100% sales
        property bool made250games: false                   // + 200% sales
        property bool made1000games: false                  // + 300% sales

        property bool sold100ThousandGames: false           // + 25% sales
        property bool soldMillionGames: false               // + 50% sales
        property bool sold10MillionGames: false             // + 100% sales
        property bool sold100MillionGames: false            // + 200% sales
```

We have to check for the completion of these goals within the ever-running interval: 5 **Timer**:

```
onTriggered: {
            // check for achievements
            if(!scene.coded100ThousandLines && scene.totalLinesCodedProgrammers >= 100000){
              scene.coded100ThousandLines = true
              achievementUnlockedNotifier.showAchievementUnlocked()
            } else if(!scene.codedMillionLines && scene.totalLinesCodedProgrammers >= 1000000){
              scene.codedMillionLines = true
              achievementUnlockedNotifier.showAchievementUnlocked()
            } else if

...
            if(!scene.made25games && scene.gameNo >= 25){
              scene.made25games = true
              achievementUnlockedNotifier.showAchievementUnlocked()
            } else if
...
            // check if game is finished
            if (scene.linesToCode <= 0) {
...
```

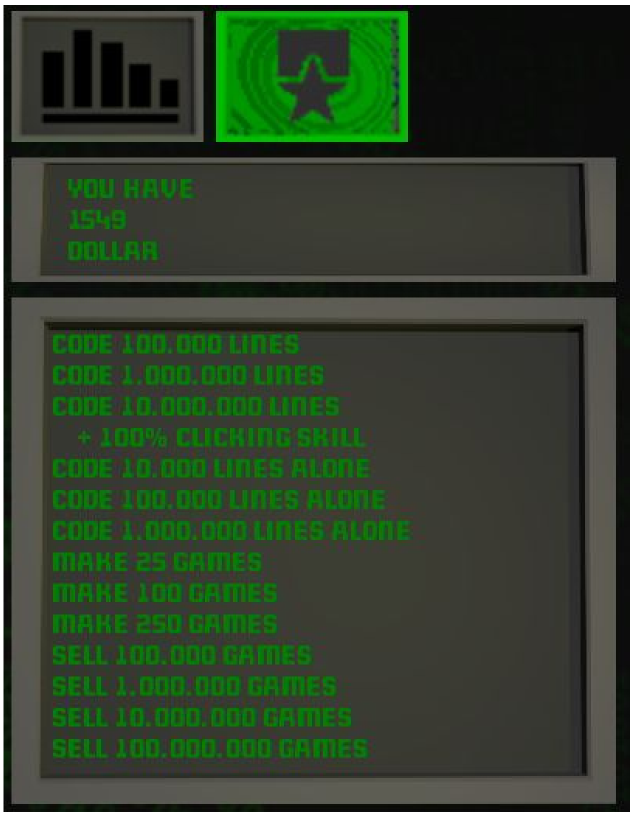
I left out stuff at the **…** sections but it should be easy to fill that in given the names of the variables. Also, the achievementUnlockedNotifier.showAchievementUnlocked() is not important yet, we'll save this for later but make sure to write that call everytime an achievement is unlocked.

Now, add another hover-button to the **StatusAndAchievementBar.qml**, much like the one for the statistics.

```
Rectangle{
    id: achievementButton
    x: 346
    y: 5
    width: 62
    height: 42
    SpriteSequenceVPlay{
      id:achievementButtonAnimation
      defaultSource: "../assets/achievementSequence.png"
      SpriteVPlay {
          name: "idle"
          frameWidth: 62
          frameHeight: 42
        }
      SpriteVPlay {
          name: "hover"
          frameWidth: 62
          frameHeight: 42
          frameX: 62
        }
    }
    MouseArea{
      property string tmpString: " "
      anchors.fill: parent
      hoverEnabled: true
      onEntered:{
        achievementButtonAnimation.jumpTo("hover")
        tmpString = ""
        if(scene.coded100ThousandLines){tmpString += "  + 25% coding skill"} else { tmpString += "Code 100.000 Lines"}
        tmpString += "
"
        … // the if-clauses go on here...
        if(scene.sold10MillionGames){tmpString += " + 100% sales"} else { tmpString += "Sell 10.000.000 Games"}
        tmpString += "
"
        if(scene.sold100MillionGames){tmpString += "    + 200% sales"} else { tmpString += "Sell 100.000.000 Games"}
        tmpString += "
"
        textDisplay.textToDisplay = tmpString
      }
      onExited: {
        achievementButtonAnimation.jumpTo("idle")
        textDisplay.clearText()
      }
    }
  }
```

Again, I left out if-clauses at … // the if-clauses go on here… as this is quite a massive wall of text ;) You get the idea though, if you have unlocked the achievement, instead of its condition (e.g. Sold 1 Million Games) it will display its reward (e.g. + 25% coding speed).

The achievements are can now be viewed and unlocked but their rewards do not apply yet.


YOU HAVE
1549
DOLLAR

CODE 100.000 LINES
CODE 1.000.000 LINES
CODE 10.000.000 LINES
+ 100% CLICKING SKILL
CODE 10.000 LINES ALONE
CODE 100.000 LINES ALONE
CODE 1.000.000 LINES ALONE
MAKE 25 GAMES
MAKE 100 GAMES
MAKE 250 GAMES
SELL 100.000 GAMES
SELL 1.000.000 GAMES
SELL 10.000.000 GAMES
SELL 100.000.000 GAMES

Therefore, apply the rewards in relevant places.

Within the **onTriggered** function in the **programmerTimer**:

```
scene.linesOfCodePerSecond = scene.programmerSkillLvl*4 + scene.workEnvironmentLvl*4 //balance here

            // apply achievements
            if(scene.coded100ThousandLines){scene.linesOfCodePerSecond +=
Math.ceil(scene.linesOfCodePerSecond/4)}
            if(scene.codedMillionLines){scene.linesOfCodePerSecond +=
Math.ceil(scene.linesOfCodePerSecond/2)}
            if(scene.coded10MillionLines){scene.linesOfCodePerSecond = scene.linesOfCodePerSecond*2}
            if(scene.coded100MillionLines){scene.linesOfCodePerSecond = scene.linesOfCodePerSecond*3}
```

Within the **onClicked** function in the **ClickerWindow.qml**:

```
scene.linesOfCodePerClick = scene.codingSkillLvl //+ (scene.gameNo*3*scene.codingSkillLvl) // balance here

            // apply achievements
            if(scene.codedThousandLinesAlone){scene.linesOfCodePerClick = scene.linesOfCodePerClick * 2}
            if(scene.coded10ThousandLinesAlone){scene.linesOfCodePerClick =
Math.ceil(scene.linesOfCodePerClick * 2.5)}
            if(scene.coded100ThousandLinesAlone){scene.linesOfCodePerClick = scene.linesOfCodePerClick * 3}
            if(scene.codedMillionLinesAlone){scene.linesOfCodePerClick = Math.ceil(scene.linesOfCodePerClick *
3.5)}
```

Within **gameFinished()**, where the copies are sold:

```
scene.copiesSold = (42 * Math.ceil(marketingLvl * 1.5)) + (scene.gameNo * 42)

        // apply achievements
        if(scene.made25games){scene.copiesSold += Math.ceil(scene.copiesSold/2)}
        if(scene.made100games){scene.copiesSold = scene.copiesSold*2}
        if(scene.made250games){scene.copiesSold = scene.copiesSold*3}
        if(scene.made1000games){scene.copiesSold = Math.ceil(scene.copiesSold*3.5)}

        if(scene.sold100ThousandGames){scene.copiesSold += Math.ceil(scene.copiesSold/4)}
        if(scene.soldMillionGames){scene.copiesSold += Math.ceil(scene.copiesSold/2)}
        if(scene.sold10MillionGames){scene.copiesSold = scene.copiesSold*2}
        if(scene.sold100MillionGames){scene.copiesSold = scene.copiesSold*3}
```

Now, the achievements do apply! For example, if you code 1.000 Lines by clicking (without programmers), you'll recognize a speed increase.

Finishing Touches

Nearly done, but you haven't forgotten the achievementUnlockedNotifier.showAchievementUnlocked(), have you? Create a new .qml-File and name it **AchievementUnlockedNotifier.qml** and fill it with the following code:

```
import QtQuick 2.0
import VPlay 2.0

Item {
  Rectangle{
     x: -54
     y: 20
     width: 50
     height: 50
     Image{
        anchors.fill:parent
        source: "../assets/boxWborder.png"
     }
     Text{
        anchors.centerIn: parent
        font.family: myFont.name
        color: "green"
        font.pixelSize: 6
        text: "Achievement
Unlocked!"
     }
     MovementAnimation{
        id: achievementNotifierAnimation
        target: parent
```

```
            property: "x"
            minPropertyValue: -55
            maxPropertyValue: 5

            property bool goingRight: false // true if going right, false if going left

            onLimitReached: {
                running = false
                if(goingRight){
                    shiftLeft()
                }
            }
            function shiftRight(){
                goingRight = true
                achievementNotifierAnimation.running = true
                achievementNotifierAnimation.velocity = 30
            }
            function shiftLeft(){
                goingRight = false
                achievementNotifierAnimation.running = true
                achievementNotifierAnimation.velocity = -30
            }
        }
    }
    function showAchievementUnlocked(){
        achievementNotifierAnimation.shiftRight()
    }
}
```

VPlay 2.0 is imported because of the **MovementAnimation**, the **x** property is negative because this little **Rectangle** is placed outside the screen (at least for the iPhone :)). Whenever an achievement is unlocked, it moves to the right, notifies the player of him unlocking an achievement and then flies to the left into obscurity.

Add this to your **Scene** and you're good to go as we already have added achievementUnlockedNotifier.showAchievementUnlocked() whenever an achievement is unlocked before.

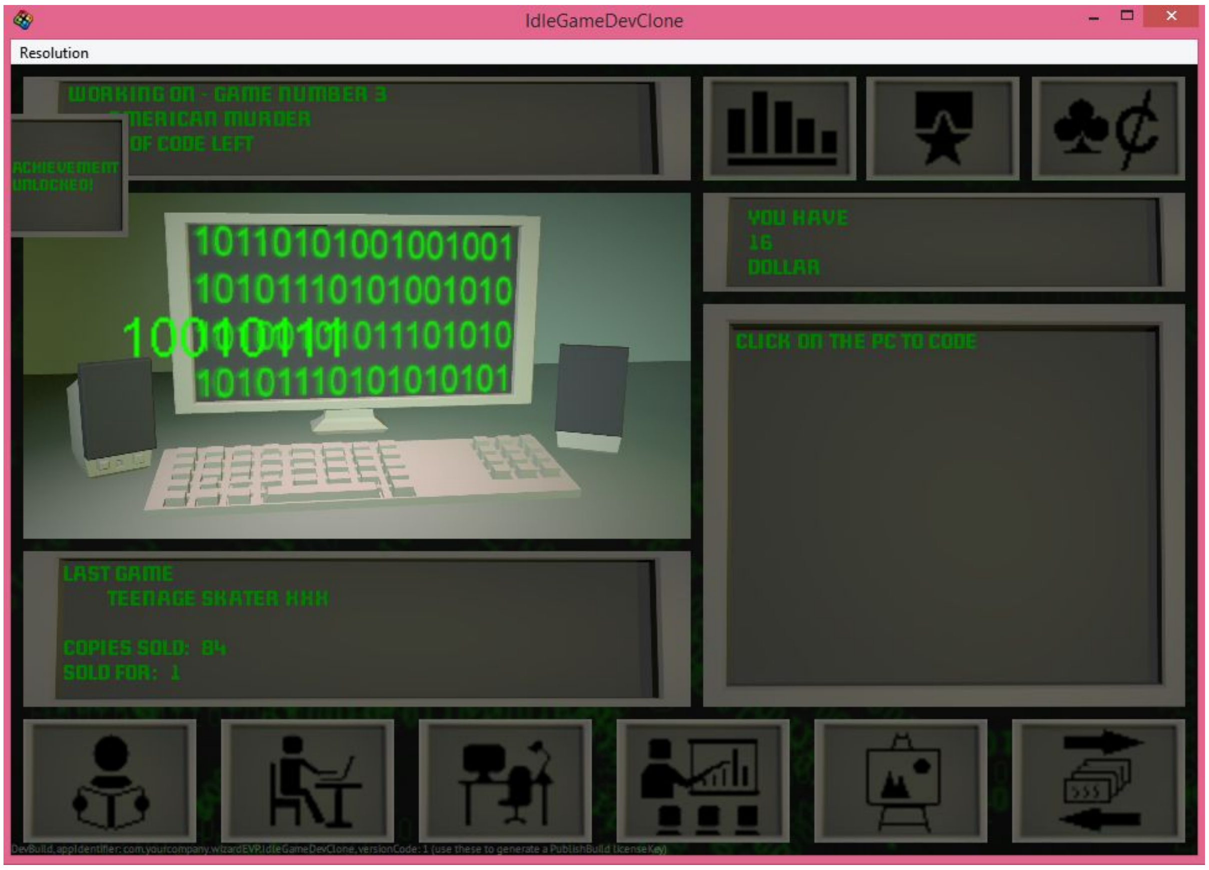
```
AchievementUnlockedNotifier{
                id: achievementUnlockedNotifier
            }
```

Nice, this makes unlocking an achievement even more fun.

There’s still one more thing to do but it’s a small task. Add an extra button without any functionality to the top right corner. Drop this code to the end of **StatusAndAchievementBar.qml**:

```
Rectangle{
    id: extraButton
    x: 280+62+4+62+5
    y: 5
    width: 62
    height: 42

    SpriteSequenceVPlay{
      id:extraButtonAnimation
      defaultSource: "../assets/extraSequence.png"
      SpriteVPlay {
          name: "idle"
          frameWidth: 62
          frameHeight: 42
        }
      SpriteVPlay {
          name: "hover"
          frameWidth: 62
          frameHeight: 42
          frameX: 62
        }
    }
    MouseArea{
      anchors.fill: parent
      hoverEnabled: true
      onEntered:{
        extraButtonAnimation.jumpTo("hover")
        textDisplay.textToDisplay = "Implement further features here :-)"
      }
      onExited: {
        extraButtonAnimation.jumpTo("idle")
        textDisplay.clearText()
      }
    }
  }
```

That’s it! Yay!
(Though you could still tweak the balancing for days...)



DevBuild, appIdentifier: com.yourcompany.wizardEVR.IdleGameDevClone, versionCode: 1 (use these to generate a PublishBuild licenseKey)

(The extra button to the top right, and the **AchievementUnlockedNotifier** on his way back to the left in the top left corner)

Download the finished game here:
https://github.com/ThomasKriz/IdleGameDevClone/blob/master/IdleGameDevClone.zip