

# Coloristance Developer Manual

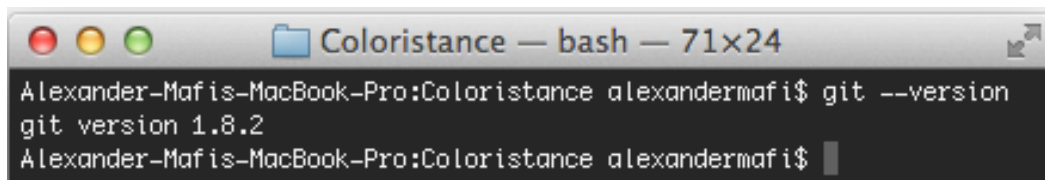
---

## Getting started

### Använda git-kommandon i terminalen

För att kunna ladda hem projektet måste terminalen (command line) kunna tolka git-kommandon. Om detta inte är inställt krävs följande:

1. Gå till <http://git-scm.com/downloads>
2. Hämta installationsfilen för ditt operativsystem
3. Installera den nedladdade filen
4. Öppna terminal på Mac eller övrig command line
5. Skriv följande: `git --version`



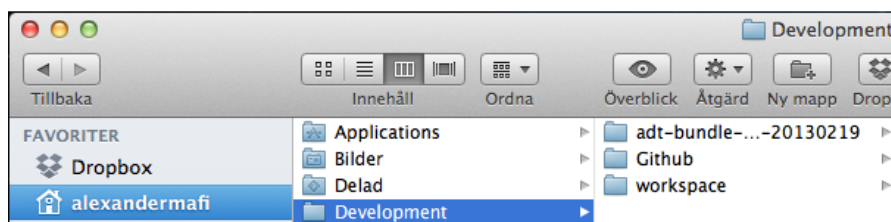
```
Alexander-Mafis-MacBook-Pro:Coloristance alexandermafi$ git --version
git version 1.8.2
Alexander-Mafis-MacBook-Pro:Coloristance alexandermafi$
```

6. Du är klar!

OBS använd inte Githubs egna program istället för terminalen (command line). Detta med anledning av att det inte fungerar. Den hittar bland annat inte alla förändringar. Fyra av fem problem som dyker upp beror på försök att göra något i GUI:t (Graphical User Interface) som inte fungerar. GUI:t är endast utformad efter de vanligaste funktionerna och kan ge upphov till problem om övriga kommandon används.

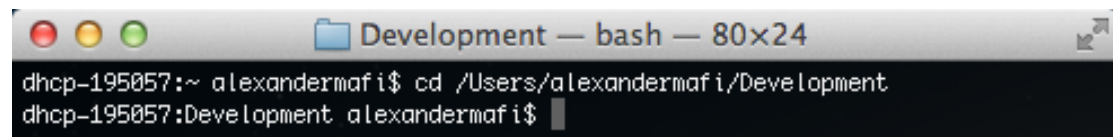
### Ladda ner projekt från github

Skapa en mapp där du vill lägga projektet, exempelvis Development.



För att terminalen ska förstå vart du vill lägga det nedladdade projektet måste du navigera till den skapade mappen Development i terminalen (command line). I detta fall ligger mappen Development i alexandermafi som i sin tur ligger i users. Därför skrivs i detta fall: `cd /Users/alexandermafi/development`. I Mac räcker det egentligen med att skriva `cd` och sedan dra in mappen för att få hela sökvägen.

Kommandot `cd` följt att mappnamn används för att förflytta sig mellan mappar på datorn. "cd" står för "change directory".



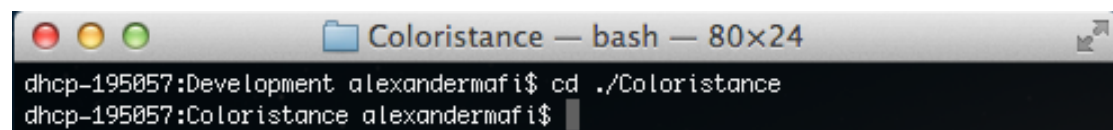
```
dhcp-195057:~ alexandermafi$ cd /Users/alexandermafi/Development
dhcp-195057:Development alexandermafi$
```

Detta gör så att terminalen vet vilken mapp den ska lägga projektet i.

Ladda ner projektet genom att skriva följande i terminalen:

- `git clone https://github.com/AndroidSquad/Coloristance.git`

Genom detta skapas en mapp som heter Coloristance helt automatiskt. Navigera till den mappen genom att skriva: `cd ./Coloristance`



```
dhcp-195057:Development alexandermafi$ cd ./Coloristance
dhcp-195057:Coloristance alexandermafi$
```

När det står Coloristance till vänster om ditt användarnamn (alexandermafi i detta fall) så befinner sig terminalen i rätt mapp och du kan nu använda git commandon för att påverka mappens innehåll. Om du använder Windows Powershell på en Windows-dator så står istället hela din katalog, exempelvis: `C:\Users\Simon\Documents\GitHub\Coloristance`

### Dependencies

- Java 6 SE development environment
- Android SDK
- Virtual and regular Android devices
- junit-4.11.jar

### Android SDK targets

- Minimum SDK: 8
- Target SDK: 17

### Importera projektet till Eclipse

För att kunna jobba med projektet i Eclipse måste appen och dess filer hämtas från github på länken under rubriken "Getting started".

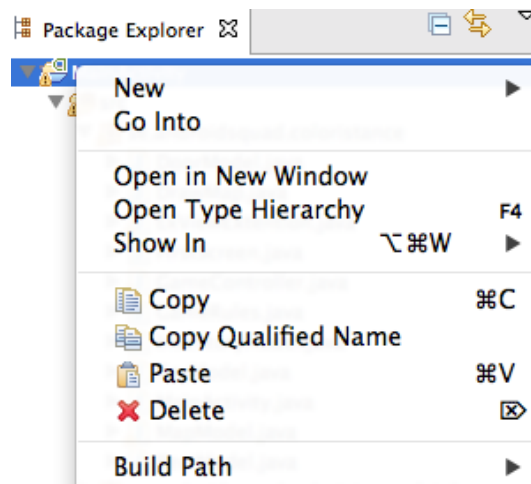
Utför följande steg för att importera projektet i Eclipse med ADT-plugin:

***File → Import → veckla ut mappen "android" och tryck på "Existing Android Code Into Workspace → Välj "Root Directory" → kryssa i projektet som dyker upp i rutan → tryck finish.***

Kodningen sker i Eclipse men all kontakt med repository på servern ska ske genom Terminalen (Command Line).

Trots detta kommer Eclipse visa fel *se.androidsquad.coloristance.tests* och detta beror på att du måste importera ett externt arkiv som heter *junit-4.11.jar* som laddas ned på <https://github.com/junit-team/junit/downloads>. Importen görs genom följande:

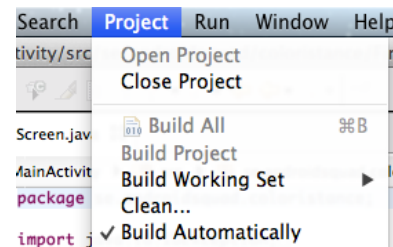
**Högerklicka på projektmappen → Build Path → Add External Archives → Välj junit-4.11.jar från den mapp du laddat ned den → Öppna.**



## Building and Installing

För att kompilera projektet används kommandot "Clean" i Eclipse. Det som händer är att clean-kommandot tar bort de gamla .class-filerna och bygger upp nya .class-filer. Detta utförs genom följande steg:

**Project → Clean...**



## Installing Android device as emulator

Applikationen kan installeras på en Androidenhet eller en virtuell emulator som skapats i Eclipse. Vi rekommenderar en fysisk Androidenhet då dessa oftast är betydligt snabbare.

För att skapa en virtuell emulator väljs:

**Window → AVD Manager**

För den virtuella enheten måste det specificeras vilken version av Android som ska köras, samt påtänkta fysiska specifikationer för enheten, såsom kamera etc. För vidare förklaring, se avsnittet som berör just detta på Android Developer: <http://developer.android.com/tools/devices/index.html>

För att använda en fysisk androidenhet som emulator krävs mer arbete, men resulterar i en bättre miljö att testa applikationen. För att installera en androidenhet som emulator måste de steg som beskrivs i följande länk på Android Developer genomföras:

<http://developer.android.com/tools/device.html>

## Release procedure

Sammanlagt har två releases släppts av applikationen. Den första releasen har fungerande men begränsad funktionalitet. Detta beskrivs i Release Notes 0.1 alpha.pdf, vilket är ett dokument som förklarar den första releasen. I detta dokument beskrivs även kända buggar och vilken funktionalitet som planeras tillkomma i den senare releasen.

Den andra och slutgiltiga releasen har mer omfattande dokumentation där det redogörs för vilka krav som ställs (requirements), test av funktionalitet och användarmanualer. Den andra releasen har en mer fulländad funktionalitet, och har dessutom genomgått mer rigorös testning av funktionaliteten.

## Building a release package

För varje release skapas en .apk fil. Denna .apk fil behövs för att kunna starta applikationen utan IDE, exempelvis Eclipse. För den första releasen är denna fil osignerad, men för den andra releasen är den signerad, vilket innebär att den går att ladda upp till Google Play.

För att skapa en osignerad .apk fil utförs följande steg:

***Högerklicka på projektmappen → Android Tools → Export Unsigned Application Package***

I vårt fall gick det inte att pusha .apk-filen initialt, då denna var med i .gitignore. För att lösa detta måste .apk extensions tas bort ur .gitignore.

Den slutgiltiga releasen måste vara en signerad .apk fil. För att skapa denna signerade fil utförs istället följande steg:

***Högerklicka på projektmappen → Android Tools → Export Signed Application Package***

## Organizing the distribution directory

Den skapade .apk filen bör ligga i en egen mapp för att kunna nås smidigare, och för att tydligt visa de olika release som kan finnas i ett projekt.

1. Därför skapas först en mapp vid namn "dist" i projektets rotkatalog.
2. Skapa sedan en mapp för varje version av releasen.
  - a. v.01alpha
  - b. v.01beta
3. Flytta sedan applikationens .apk fil till den senaste releasens mapp.

4. Döp om .apk applikationen till den senaste releasen. Exempelvis:  
Coloristance-01alpha.apk

### Release requirements

Varje mapp som kännetecknar en release innehåller följande:

- En .apk fil som är applikationens paket och det som krävs för att köra projektet.
- Ett dokument vid namn "Release Notes", korrelerande med den senaste releasen, som innehåller relevant dokumentation för just den releasen. Innehåll ska vara bland annat:
  - Ny funktionalitet
  - Kända buggar
  - Kommande funktionalitet

## Tests

### JUnit

JUnit används för att genomföra automatiska tester av applikationen. Dessa tester finns som separata klasser inom applikationen Coloristance. Testklasserna är:

- *DoorModelTest.java*
- *MapModelTest.java*
- *RectModelTest.java*

För att genomföra dessa tester väljer man att köra applikationen som ett JUnit test. Detta görs genom att välja:

***Run As → JUnit test***

### Continuous testing

Applikationens funktionalitet har hela tiden testats kontinuerligt, varje gång kod har skrivits i projektet. Denna testning har skett genom att applikationens funktionalitet har iakttagits på antingen en virtuell eller fysisk Androidenhet.

Utöver detta har Java-kommandot `Log.v(String, String)` använts för att skriva ut meddelanden i LogCat i Eclipse. Genom att skriva ut meddelanden vid ställen där det misstänktes finnas fel så kunde en testning av applikationen skötas manuellt i Eclipse.

## Arkitektur

Projektets javaklasser har delats in olika paket beroende på deras funktion i projektet. Projektets utformning har utgått från ett designmönster enligt MVC-modellen. En så tydlig uppdelning av funktionaliteten som i en MVC-modell var svårt att uppnå, och därför har inte denna modell implementerats helt och hållet.

Istället agerar en Activity i applikationen delvis både View och Controller. Modelklasserna är dock mer tydligt uppdelade.

De paket som använts för att dela in javaklasserna är:

- `se.androidsquad.coloristance.controllers`
  - Controllers är de klasser som ansvarar för att sammankoppla de övriga klasserna så att de tillhandahålls med nödvändig data. Activities kommer mestadels agera som controllers men även en del som views på grund av Androids uppbyggnad.
- `se.androidsquad.coloristance.database`
  - Som en separat databas innehåller detta paket nödvändig information rörande spelets olika banor: färgen på rummen, färgen på nycklarna, samt rummens och nycklarnas relation till varandra på kartan. Database innehåller specifikationer för den bana som används i spelet. En tvådimensionell lista med femsiffriga strängar används för att beskriva de olika rummen: deras färger och position. En liknande tvådimensionell lista beskriver de nycklar som finns i de olika rummen.
- `se.androidsquad.coloristance.models`
  - Innehåller spellogik som inte är specifikt för Android och känner inte till controllers och views.
- `se.androidsquad.coloristance.tests`
  - Innehåller de javaklasser som används för testning av applikationen, i JUnit.
- `se.androidsquad.coloristance.views`
  - Innehåller den grafiska logiken och representationen för Androidenheter.