Utvecklardokumentation - AndroidRally

1. Hur kompileringsprocessen går till

Projektet finns på https://github.com/lucaspi/AndroidRally och sedan finns ett nästan fullständigt Eclipse-projekt placerat i mappen AndroidRally. Här saknas vissa viktiga filer (t.ex. .classpath, .project osv.) som behövs för att kunna öppna projektet. Detta för att undvika konflikter i Git.

Vad man kan göra är följande:
Ladda ner zip-filen “project_files.zip” och extrahera filerna till roten av det “ofullständiga Eclipse-projektet” i AndroidRally-mappen. Detta kommer sedan att bli Eclipse-projektet.

Main-metoden för androidapplikationen finns i MainActivity.

(Vill man köra testerna får man lägga till mappen “test” som en “Source folder” i “Build path”, samt lägga till JUnit som ett bibliotek under fliken “Libraries” i “Build path”. Har man inte JUnit installerat kan man lägga till det som ett plug-in till Eclipse och sedan göra om proceduren ovan.)

2. Applikationens huvuddelar

Applikationen består av en server-del och en klient-del. Servern har flera spel körandes samtidigt, och klienten och modellen är helt skilda och kommunicerar endast genom Client, på klient-sidan, och GameController, på server-sidan.

2.1 Server

Innehåller en klass som heter GameController som hanterar och skapar GameModel, som är huvudklassen för modellen. Se UML-diagram under rubriken UML nedan för mer detaljerad vy.

2.2 Klient

På mobilen finns en klass som heter Client och som används för att lättare skicka och ta emot data från servern. Denna klass är en singleton då den endast bör skapas en gång (då man startar applikationen) och sedan håller i den enda anslutning till servern som applikationen behöver. (Se “Protokoll” för mer info om hur datan som skickas ser ut).

3. Designval

Applikationen är riktad till Android 4.2 (JELLY_BEAN_MR1) och högre, men stödjer minst Android 2.2.x (FROYO).

3.1 Modifierad MVC

I klassisk MVC känner controllern till både vyn och modellen. Vyn känner till modellen och modellen skickar händelser till vyn när den behöver uppdateras på samma sätt som att vyn skickar händelser till controllern då något händer.

I fallet med AndroidRally ser designen något annorlunda ut. Med tanke på att spelet är utvecklat på ett sådant sätt att det ska fungera att spela över internet med ett server/klient-tänk ser

uppdelningen något annorlunda ut. I “serverdelen” av programmet kommer spellogiken (dvs. modellen) att befinna sig samt en controller som hanterar denna. Någon vy behövs inte här utan denna befinner sig i androidapplikationen där även en controller finns som hanterar inmatning från användaren. Sättet kommunikationen mellan servern och klienten sker på är att klienten säger till när den vill utföra vissa handlingar och får sedan resultatet av servern, såvida det inte handlar om att servern skickar en push-notifikation om att alla spelare (vid ett nätverksspel) är klara.

3.2 Vyn

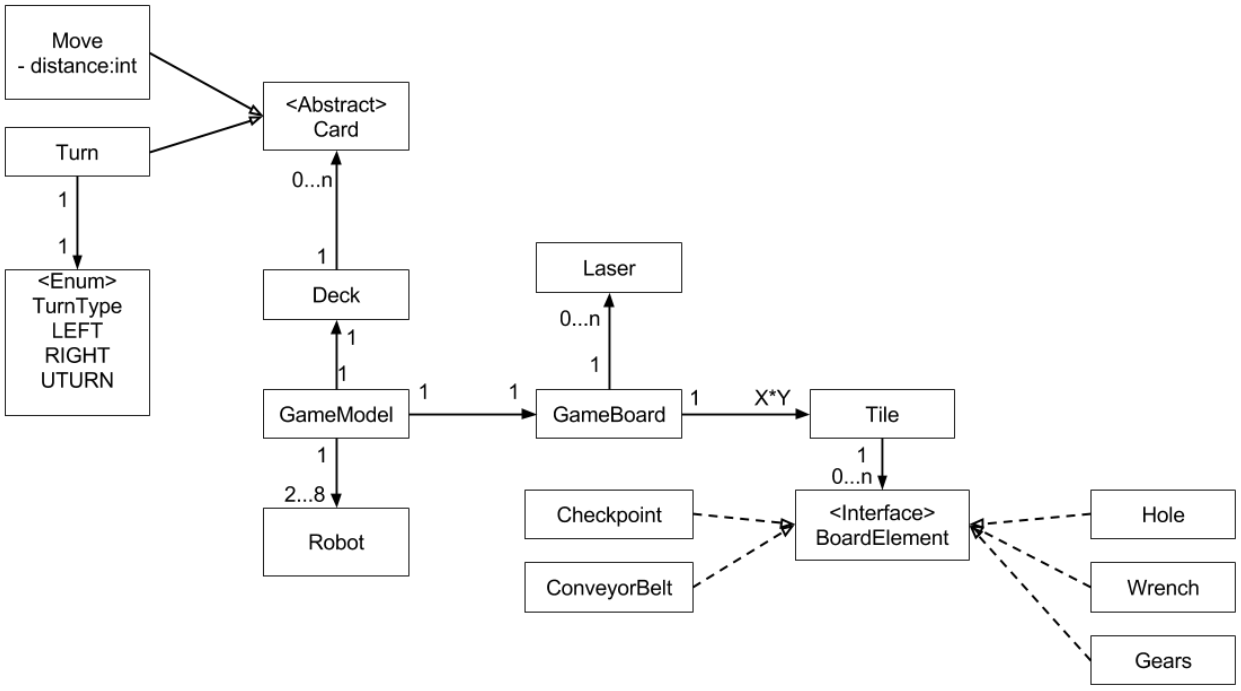
Då spelet ska kunna ligga i bakgrunden och uppdateras trots att ingen användare är aktiv i spelet så finns ingen riktig spellogik på mobilsidan. Detta gör att det enda som finns på mobilen är vyn och väldig simpel modelldata som är inbakad i vy-klasserna.

3.3 Tester

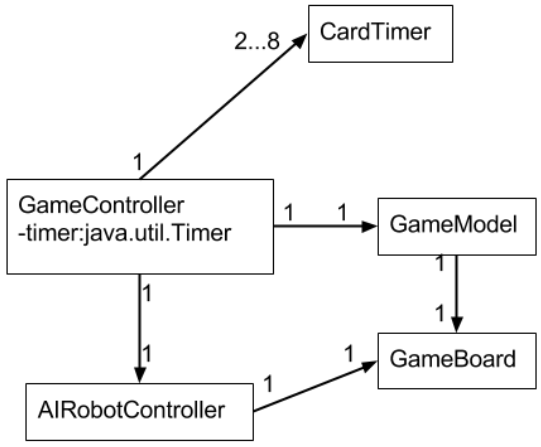
Endast några delar av spelet har automatiska test och då främst vissa metoder i modellen samt även huvudmetoden i AI-controllern. Resterande delar av modellen samt hela vy-paketet har testats manuellt.

4. UML

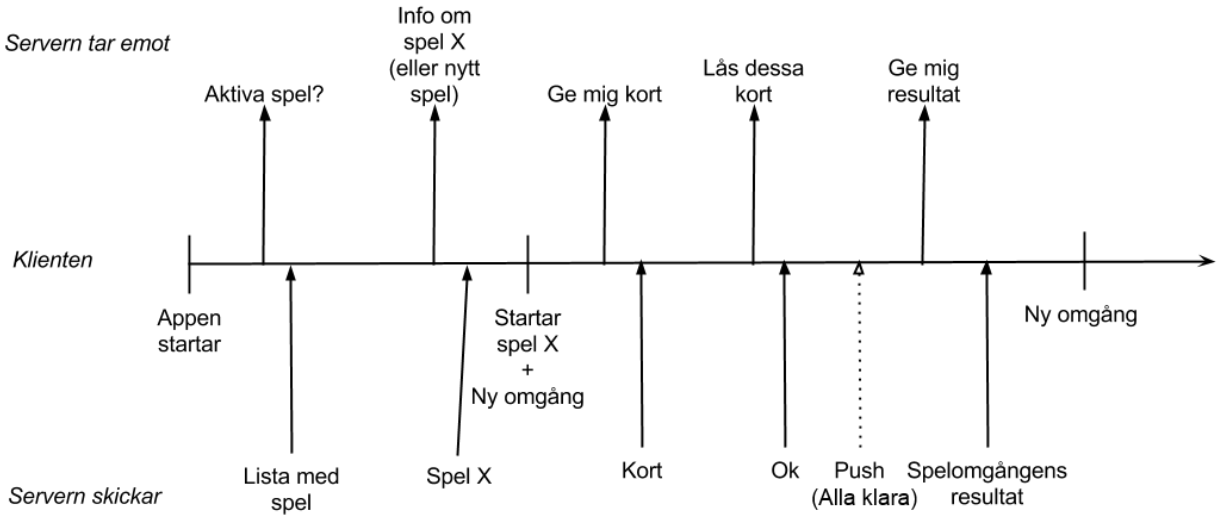
4.1 Modell



4.2 Serverdel av spelet



5. Flödesdiagram



6. Externa beroenden

Själva spelet (förutom menyer) använder sig av spelbiblioteket LibGDX. Det är detta ramverk som står för att rita ut applikationen ungefär 30 gånger per sekund. Användandet av LibGDX:s animationer har också förenklat skapandet av grafiken för spelet.

7. Protokoll

All data som skickas till servern måste ange ett par parametrar för att datan ska kunna hanteras rätt. För att ange vem spelaren är så måste klienten skicka sitt ID-nummer som är helt unikt och som klienten blir tilldelad av servern då den försöker ansluta med ID-numret -1. Klienten måste också ange vilket spel den försöker ansluta till, och därför har varje spel ett unikt ID. För att inte privata spels ID ska ge konflikt så är serverbaserade spels ID nummer positiva och privata negativa.

Klienten måste sedan ange vilken slags data som den skickar och lägger därför till en sträng som beskriver detta. Sist lägger klienten till den data den försöker skicka och väntar sedan på svar.

Detta betyder att all data skickas på formen:

[klientens ID]$[spelets ID]$[beskrivande sträng]$[data…]

(Notera: Då vi ej har en fungerande server så finns inte dessa beskrivande strängar deklarerade någonstans då de ej används.)

Nedan följer hur de all data ser ut när den skickas mellan klienten och servern.

7.1 Ladda ett spel

När en spelare laddar in ett nytt spel så måste denna få ett par värden för att vyn ska kunna ladda in spelet korrekt. Om spelaren laddar in ett sparat spel så laddas först data som vyn behöver direkt från telefonens minne. Därefter, både om det är ett nytt spel eller ett sparat, så får klienten några värden från servern för att kunna starta rätt. Dessa värden är:

- Hur många sekunder kort-timern är.
- Hur många timmar rund-timern är.
- Hur många sekunder det är kvar tills nästa runda tar slut.

Denna sträng kommer då vara på formatet:
[kort-timer i sekunder]:[rund-timer i timmar]:[sekunder tills nästa runda]

7.2 Spelplan

GameController.getMap() ger spelplanen i form av en sträng. “y” innebär ny kolumn och “x” innebär ny rad. En eller flera siffror anger vad som skall finnas på en ruta, ingen siffra innebär en tom ruta. Siffrorna anges efter x:et.

Ex:
En plan som är 3x3 med ett element A i ruta (2, 2), d.v.s. i mitten.
yxxxyxxAxyxxx

Alla riktningar använder sig av konstanter i GameBoard vilka är NORTH = 0, EAST = 1, SOUTH = 2, WEST = 3. Element på en ruta betecknas enligt följande:

- Hole = 1
- Checkpoint = [nummer för checkpoint]2
    - Ex: 12 ger checkpoint nummer 1.
- ConveyorBelt = [antal steg en robot flyttas på en omgång][riktning]3
    - Ex: 213 ger 2 steg, riktning 1.
- Gear medurs = 14, Gear moturs = 4
- Repair = 5
- Wall = [riktning]6
    - Ex: 16 ger vägg på östra sidan av en ruta.
- Laser = [riktning]7
    - Ex: 17 ger laser på östra sidan av en ruta och som då kommer att skjuta åt motsatt riktning; väst.
- Dock = [nummer för startplatsen]8
    - 18 anger att detta är startplatsen med en etta på sig, och som då kommer bli startplats för robot med ID-nummer 0, ty robotarnas ID-nummer är i intervallet [0,7] och nummer för startplatser i [1,8].

För flera element på samma ruta ska ":" anges mellan. Ex: 18:16 anger att det både finns en startplats på rutan och en vägg på den östra sidan. Nedan följer ett exempel på en plan på 12x16 rutor:

yxxxxxxx213xxxxxxx16xxyxx12xxxxx213xxxxx26xx78x16xyxxxxxxx213xxx32xxxxx58:16xyxxxx26x07xx213xxxxx26xxx38xyxxxx26xxx14x223x223x223x223x223xxx16xyxxxx26xxxxxxxxxxx18:16xyxxxx26xxxxxxxxxx28:16xyxxxx26xxx1x103x103x103x103x103xxxxyxxxx26x07xx113xxxxx26xxx48:16xyxxxxxxx113xxx22xxxxx68:16xyxx5xxxxx113xxxx16:17x26xx88:16xxyxxxxxxx113xxxxxxxxx

7.2.1 Editor

För att förenkla för utvecklare att skapa banor finns det en editor med bland source-filerna. Denna är skapad då den långa strängen i exemplet ovan inte är särskilt lätt att sätta sig in i. Dock är editorn i sitt nuvarande stadie endast tänkt för utvecklare och är därför inte allt för användarvänlig.

7.3 Kort

Kort skickas både från servern och klienten, men kan se lite olika ut beroende på hur den ska användas.

7.3.1 Klienten ber om kort

Modellen får in vilken spelrunda som efterfrågas och vilken robot som vill ha kort. Korten returneras sedan på formatet: [prioritet]:[prioritet]: … (för mellan 5-9 kort).
Men om ett kort är låst så anges "L[index för position];" framför.

Exempel: "410:420:480:660:L4;90", vilket bland annat betyder att kortet med prioritet 90 är låst på position 4.

7.3.2 Klienten skickar in valda kort

Då klienten skickar in sina valda kort skickar den, istället för prioritet på korten, index på kortet då den kom från servern. Om servern skickade "410:420:480:660:L4;90" så fick kortet 410 index 0, 420 index 1, och så vidare. Om klienten inte valt ett kort på en viss position returneras -1 i dess ställe. Då kan klienten returnera till exempel: “-1:2:6:3:4”, vilket ger ett slumpkort som första kort. Om klienten returnerar ogiltiga värden, till exempel “2:2:2:3:2”, så kommer detta tolkas som “2:-1:-1:3:-1”.

7.4 Klienten tar emot data om en runda

7.4.1 Actions

Alla händelser som ska visas under en runda i klienten sätts samman i en lång sträng. “;” och “#” används som skiljetecken mellan olika actions, “#” innebär att nästföljande action ska ske samtidigt som föregående action. “;” innebär att nästa action ska ske efter föregående action är klar.
Exempel: “[action1];[action2]#[action3];[action4]” där 2 och 3 händer samtidigt.

7.4.2 Definition av action

En movement action anger en robot och dess nya position, sedan är det klientens uppgift att vrida och flytta roboten på ett korrekt sätt. En action beskrivs på formatet “[robotID]:[robotens riktning][position x-värde][position y-värde]” Ex: “2:21005”, robot två har riktning två och står på positionen (10, 5). Observera att positionen alltid skall anges med två siffror. Om en robot går utanför spelplanen eller går ner i ett hål följs det av F#[robotID]:[antal liv kvar].
Exempel: “2:21005;F#2:2” vilket gör att roboten med ID-numret 2 kommer att röra sig till positionen (10, 5) och sedan falla och bara ha 2 liv kvar.

7.4.3 Kort

Rundan delas upp i fem delar, varje del innefattar ett kort per spelare och efterföljande händelser. Varje ny delrunda börjar med “R#[siffra]”, där siffran berättar vilket kort som spelas. första kortet beskrivs med noll och sista kortet med fyra.
Exempel: “R#0;[action1];[action2];R#1;[action 3]”

7.4.4 Faser

Varje delrunda delas in i ett antal olika faser. Först kommer all förflyttning från korten och eventuellt förflyttning för att en robot blivit knuffad. Sedan ska rullbanden röra på sig vilket beskrivs med “B1[avstånd]” Eftersom rullbanden som förflyttar robotar ett längre avstånd ska röra sig först kommer “B12” komma före “B11” i de fall det finns rullband med avståndet två. Efter “B1[avstånd]” följer de eventuella actions som rullbandet orsakar. Nästa fas är vridning från gears, den fasen beskrivs med “B4” följt av eventuella movement actions. “B5” betecknar att alla lasrar skjuts och sedan följer en uppdatering av alla träffade robotars status, vilken beskrivs senare. “B6” berättar om någon robot har nått en checkpoint på formen [robotID]: [checkpoint]. Sista fasen är “B7” vilken har hand om att återuppliva en död robot, det görs genom en vanlig movement action.

Exempel: “B12#[action1]#[action2];B11;B4” där action1 och 2 händer samtidigt som rullbandet med avstånd 2 är aktiv.

Då en robot blir skjuten skickas en uppdatering av robotens status under fas “B5” på formatet [robotID]:[om roboten tappade ett liv av skottet: 1, annars 0][antal liv kvar][skada].
Exempel: “2:036”, robot två blev skjuten när han hade 3 liv och 5 skada.

7.4.5 Game over
Då en robot förlorar skickas “L#[robotID]” och då en robot vinner skickas “W#[robotID]”. Efter en robot har vunnit är spelet slut och ingenting mer kommer att skickas.
Exempel: “L#1” anger att roboten med ID-nummer 1 har förlorat.
Exempel: “W#1” anger att roboten med ID-nummer 1 har har vunnit och att spelet är slut.

7.4.6 Exempel på en runda
R#0;3:00611;3:00610;0:10612;1:30607;2:30312;4:00208;B12;B11;B4;B5#1:032#3:033;B6;R#1;4:00207;4:00206;4:00205;3:00609;3:00608;1:30507;1:30407;0:10712;2:30212;B12#1:30408#1:30408;B11#1:30409#1:30409;B4;B5;B6;R#2;4:00204;4:00203;4:00202;0:10812;0:10912;3:00607;3:00606;1:30309;2:20212;B12;B11;B4;B5#1:033;B6;R#3;4:00201;1:30209;2:30212;3:10606;0:30912;B12;B11;B4;B5#2:032;B6;R#4;0:30812;1:20209;2:00212;4:30201;3:30606;B12;B11;B4;B5#1:034#2:034;B6;B7

8. Spara / ladda spel
Spara och ladda spel kan göras både på modellen och telefonens vy. Servern kan ej sparas ner korrekt då den ej är helt implementerad.

8.1 Vyn
På telefonen sparas data med hjälp av IOHandler, som kräver en SharedPreferences instans för att fungera korrekt. Datan om ett spel på användarens mobil sparas på formatet:
“[spelarens robotID]c[data för robot 0]a[data för robot 1]ab[sträng som representerar banan]”

Strängen för banan är representerad precis på samma sätt som när den skickas mellan klient och server. (Se *7.2 Spelplan* för mer information om denna sträng).

Datan för en robot sparas på formatet:
[x-position i fönstret]:[y-position i fönstret]:[rotation i grader moturs]:[liv]:[skada]
Exempel: “200.0:40.0:0.0:3:0”, som ger position (200, 40), rotation 0 grader, 3 liv kvar och 0 skada.

8.2 Modellen

Modellen kan också sparas ner då den ligger på telefonen som privat spel. Då den sparas ner skrivs den på formatet:
“[grundläggande data]b[info om robot 0]c[info om robot 1]cb[sträng för banan]”.

- Grundläggande data: “[korttimer i sekunder]:[rundtimer i timmar]:[antal spelare]”
- Info om roboten kan delas in i två delar: “[blandad info]a[robotens kort]”
    - Blandad info: “[x-position på kartan (två siffror)][y-position på kartan (två siffror)][rotation där 0 = söder, 1 = öst…][antal liv][skada]:[den sista checkpoint den hamnade på]:[x-position för respawn (två siffror)][y-position för respawn (två siffror)]”
    - Robotens kort: “[prioritet för kort 0]:[...]:[...]:[...]:[prioritet för kort 4]:”
      Dessa sparas för att kunna sätta rätt låste kort då spelet laddas.
- Strängen för banan: Ser likadan ut som när den skickas mellan klient och server.
  (Se *7.2 Spelplan* för mer information om denna sträng).