



TV-SPEL FÖR MOBILA ENHETER

MEDIETEKNISKT KANDIDATPROJEKT

NORRKÖPING, 2015-06-08

Av

ADAM TELLIA  
ALICE LUNDIN  
CHRISTOFFER ENGELBREKTSSON  
MARCUS LILJA  
MICHAEL NOVÉN  
OSKAR CARLBAUM  
SANDRA TRÅVÉN  
*Linköpings Universitet*

HANLEDARE OCH KUND

KARLJOHAN LUNDIN PALMERIUS

# Sammanfattning

Denna rapport behandlar arbetet med att utveckla ett system för TV-spel för mobila enheter. Digital teknik har utvecklats mycket under de senaste decennierna och en telefon är inte längre bara en telefon utan även en miniräknare, dator, kamera, etc. Så varför inte även en spelkontroll och en spelkonsol? En trådlös TV-spelslösning skulle vara smidigare än att vara beroende av en statisk låda med speciella tillhörande kontroller som dagens konsoller använder eftersom det skulle utnyttja den teknik användarna redan har. Istället för att de ska införskaffa den specifika kontrollen kan de använda sin smarta telefon.

Gruppen arbetade *agilt* enligt metoden *Scrum* med en bestämd organisation och arbetsmetodik som optimerade arbetet så att alla resurser utnyttjades till fullo. Samtliga gruppmedlemmar hade individuella ansvarsområden för att öka delaktigheten och sprida ut huvudansvaret. Ansvar delades ut vid projektets uppstart och undersöktes och förbättrades under projektets gång.

Systemet utvecklades i utvecklingsmiljön Android Studio och har stöd av ramverken libGDX och KryoNet. Dessa ramverk har hjälpt utvecklingsteamet mycket men gruppen har även behövt skapa egna lösningar för bland annat scenhantering, förflyttning av kamera och bestämma vem som leder en spelomgång.

Till systemet kan mobila enheter användas som handkontroller. Systemet kan kopplas till en TV och fungera som en spelkonsol. Tredjepartsutvecklare kan enkelt använda detta system för att utveckla nya spel. Systemet utvecklades i första hand till Android och en god kodstruktur prioriterades. Ett kompletterande spel i 2D utvecklades där flera spelare kan styra varsin bil på en bana som visas på en gemensam enhet. Mobila enheter används för styrning av bilen och kan roteras som en ratt.

# Innehåll

<b>Sammanfattning</b>	<b>ii</b>
<b>Figurer</b>	<b>vi</b>
<b>Tabeller</b>	<b>vii</b>
<b>Typografiska konventioner</b>	<b>viii</b>
<b>1 Inledning</b>	<b>1</b>
1.1 Bakgrund . . . . .	1
1.2 Syfte . . . . .	1
1.3 Frågeställningar . . . . .	2
1.4 Avgränsningar . . . . .	2
1.5 Spelidé . . . . .	2
<b>2 Relaterat arbete</b>	<b>3</b>
2.1 Arbetsmetod inom mjukvaruutveckling . . . . .	3
2.2 Liknande system . . . . .	3
2.3 Undersökning av ramverk . . . . .	4
2.3.1 LibGDX . . . . .	4
2.3.2 KryoNet . . . . .	4
2.4 TV-uppkoppling . . . . .	5
<b>3 Utvecklingsprocessen</b>	<b>6</b>
3.1 Scrum . . . . .	6
3.2 Arbetsrutiner . . . . .	7
3.3 Organisation . . . . .	8
3.4 Schemaläggning . . . . .	9
3.5 Utvecklingsverktyg och versionshantering . . . . .	9
3.5.1 Android Studio . . . . .	9
3.5.2 Git och GitHub . . . . .	9
3.6 Kravhantering . . . . .	9

3.6.1	Krav från kund . . . . .	10
3.6.2	Krav från övriga intressenter . . . . .	10
<b>4</b>	<b>Implementation</b>	<b>12</b>
4.1	Modellering och Unified Modeling Language (UML) . . . . .	12
4.2	Systemarkitektur . . . . .	12
4.3	Serverstruktur . . . . .	13
4.4	Klientstruktur . . . . .	13
4.5	Nätverkskommunikation . . . . .	14
4.6	Spelet . . . . .	15
4.6.1	Obligatorisk struktur för samtliga spel . . . . .	15
4.6.2	Kamera . . . . .	15
4.6.3	Styrning . . . . .	16
4.6.4	Övrig spellogik . . . . .	17
4.6.5	Ljud och musik . . . . .	17
4.7	Testning . . . . .	18
4.7.1	Testning av tidsfördräjning . . . . .	18
4.8	Kodgranskning . . . . .	18
<b>5</b>	<b>Resultat</b>	<b>19</b>
5.1	Systemet Varsom Games . . . . .	19
5.1.1	Servern . . . . .	19
5.1.2	Klienten . . . . .	20
5.2	K'razy Racy . . . . .	21
5.2.1	Struktur och innehåll . . . . .	21
5.2.2	Spelkontroll . . . . .	22
5.2.3	Tidsfördräjning . . . . .	23
5.3	Resulterande arbetsmetodik . . . . .	23
<b>6</b>	<b>Analys och diskussion</b>	<b>26</b>
6.1	Utvecklingsmetodik . . . . .	26
6.2	Arbetsfördelning . . . . .	27
6.3	Användartester . . . . .	27
6.4	Ramverk . . . . .	27
6.5	Analys av tidsfördräjning . . . . .	28
6.6	Vidareutveckling av projektet . . . . .	28
6.6.1	Nätverk . . . . .	28
6.6.2	Spelet . . . . .	28
6.6.3	Tredjepartsutveckling av spel . . . . .	29

6.7 Arbetet i ett vidare sammanhang . . . . .	29
<b>7 Slutsatser</b>	<b>31</b>
7.1 Slutsatser utefter frågeställningar . . . . .	31
7.2 Slutsatser utefter mål . . . . .	32
<b>Litteraturförteckning</b>	<b>33</b>
<b>A Arbetsbelastning och statistik</b>	<b>35</b>
A.1 Statistik över kodbidrag . . . . .	36
A.2 Alias på GitHub . . . . .	38
A.3 Personliga bidrag . . . . .	39
<b>B Användartest och svar</b>	<b>41</b>
<b>C Pseudokod för projektion på kamerans vägbana</b>	<b>47</b>

# Figurer

3.1	Ett exempel av hur verktyget <b>Slack</b> kan användas . . . . .	7
3.2	Ett exempel av hur verktyget <b>Trello</b> kan användas . . . . .	8
4.1	Schema över server-klient-struktur . . . . .	13
4.2	Ett <i>use case</i> -diagram för servern . . . . .	13
4.3	Ett flöde för klientens upplägg . . . . .	14
4.4	Kamerans placering i förhållande till dess bana och till bilarna . . . . .	16
4.5	Banans bakgrundsbild (a) samt dess <i>pixmap</i> (b) . . . . .	17
5.1	Systemets huvudmeny . . . . .	19
5.2	Klientens anslutningsscen . . . . .	20
5.3	Klientens navigationsscen . . . . .	21
5.4	Spelets huvudmeny . . . . .	21
5.5	Spelet . . . . .	22
5.6	Spelkontrollens grafiska gränssnitt . . . . .	23
5.7	Flödeskarta med uppgifter för en <i>story</i> under tredje <i>sprinten</i> . . . . .	24
6.1	Dubbla vägbanor för kameran . . . . .	29
A.1	Statistik över gjorda ändringar i de kodfiler som innehållas av serverapplikationen . . . . .	36
A.2	Statistik över gjorda ändringar i de kodfiler som innehållas av klientapplikationen . . . . .	37
B.1	De första fyra frågorna i användartestet . . . . .	42
B.2	Fråga 5 till 8 i användartestet . . . . .	43
B.3	Fråga 9 till 12 i användartestet . . . . .	44
B.4	Fråga 13 till 15 i användartestet . . . . .	45
B.5	Fråga 16 till 19 i användartestet . . . . .	46
C.1	Pseudokod för projektion och vägpunktshantering . . . . .	47

# Tabeller

1	Typografiska konventioner	ix
5.1	Sammanställning av uppmätt tidsfördröjning	23
A.1	Användarnamn på GitHub för tolkning av statistik	38

# Typografiska konventioner

Tabell 1 innehåller förklaringar på termer som förknippas med systemutveckling. Många har sitt ursprung i engelskan och har blivit översatta i officiella rapporter sedan tidigare. Samtliga ord i tabelllen markeras med kursiv stil och böjs enligt konventioner för svenska språket där de förekommer i texten. Andra ord i texten som är på engelska men som förklaras i texten kursiveras också. Namn på företag och produkter skrivs i fetstil.

Ordlista	
<b>Agil</b>	Försvenskad version av det engelska ordet <i>agile</i> , beskriver något smidigt eller lättörligt.
<b>Scrum</b>	Ett <i>agilt</i> arbetsätt och processramverk för systemutveckling.
<b>Produktägare</b>	En person i eller utanför utvecklingsgruppen som kommer med krav och önskemål om tillägg och förändringar gällande produkten. Denna person sätter även upp en <i>produktbacklogg</i> . Produktägaren är kundens röst.
<b>Utvecklingsteam</b>	En självorganiseraende grupp om tre till nio personer som utvecklar produkten i fråga.
<b>Scrum Master</b>	En person som ser till att arbetsmetoden efterföljs. Assisterar <i>produktägaren</i> och utvecklingsgruppen med deras <i>Scrumrelaterade</i> uppgifter om så behövs. En tjänande ledare.
<b>Produktbacklogg</b>	En lista med funktionaliteter som ska implementeras i produkten. Den ska sorteras efter prioritet och bör uppdateras regelbundet av <i>produktägaren</i> .
<b>Sprint</b>	En <i>sprint</i> är en tidsperiod som är två till fyra veckor. Under denna period ska en redovisningsbar prototyp tas fram.
<b>Sprintbacklogg</b>	En lista med <i>stories</i> som ska färdigställas under <i>sprintens</i> gång.
<b>Story</b>	En eller ett par meningar där krav och önskemål definieras och förklaras, ur en användares, kunds, utvecklares eller liknandes perspektiv.
<b>Shading</b>	Process för ljussättning och färgläggning av datorgrafik.
<b>Sensor fusion</b>	Funktionalitet för att kombinera sensorer för att få mer exakt mädata.

<b>Sprite</b>	Tvådimensionell bild med position, skala och rotation.
<b>Gameplay</b>	Användarens upplevelser och interaktion i spelet.
<b>Multiplayer</b>	För flera spelare.
<b>Debugga</b>	Lokalisera och återgärda fel i koden.
<b>TCP</b>	<i>Transmission Control Protocol</i> skickar paket med data från en sändare till en mottagare där återkoppling ges ifall paketen kommer fram.
<b>UDP</b>	<i>User Datagram Protocol</i> skickar paket med data från en sändare till en mottagare där återkoppling inte ges.
<b>Broadcast</b>	Datorkommunikation som innebär att ett paket som skickas mottas av alla på nätverket.
<b>Multicast</b>	En sändning av information till en grupp mottagare samtidigt, men inte till alla i ett nätverk.

Tabell 1: Typografiska konventioner

# Kapitel 1

## Inledning

### 1.1 Bakgrund

Digital teknik har utvecklats mycket under de senaste decennierna och en telefon är inte längre bara en telefon utan även en miniräknare, dator, kamera, etc. Digitalt spelande har blivit en del av mångas vardag och det finns ett stort utbud på spel för telefoner. Spelupplevelsen för ett mobilspel skiljer sig fortfarande från till exempel ett TV-spel där de spelande sitter i samma rum och kan prata med varandra medan spelaren på mobil spelar ensam. Det här projektet kombinerar intresset för mobilspel och samhörigheten från TV-spelen genom att skapa ett system för mobila enheter som fungerar som en konsol. Fördelarna med systemet över en vanlig spelkonsol är att användare inte behöver införskaffa specifik teknik för att kunna spela utan kan använda sina smarta mobila enheter som de redan äger.

### 1.2 Syfte

Projektet ska resultera i ett system som körs på en surfplatta, där användare kan bläddra och välja mellan olika spel genom sin egen mobiltelefon eller surfplatta. Vidare ska ett *multiplayer*-spel i 2D utvecklas till detta system där användarna kan spela med hjälp av sina mobila enheter.

Systemets kod ska vara tillräckligt välstrukturerad och läsbar för att ett annat *utvecklingsteam* ska kunna ta över arbetet eller skapa nya spel till systemet utan större problem.

Gruppen ska arbeta *agilt* med metoden *Scrum* och optimera arbetet så mycket som möjligt genom självutvärdering och effektivisering. Gruppen ska undvika slötid och se till att samtliga gruppmedlemmar bidrar maximalt under hela projektet.

Följande mål fanns vid projektets början:

- Ett färdigt system för spel på en surfplatta med flera mobiltelefoner som kontroller ska utvecklas där åtta användare ska kunna vara aktiva samtidigt.
- Systemets kod ska vara läsbar och välstrukturerad.
- Systemet och ett kompletterande spel ska kunna släppas på **Google Play** vid projektets slut.
- Serverenheten ska kunna kopplas upp mot en TV för att lättare visa spelet och underlätta för spelare att spela samtidigt.

## 1.3 Frågeställningar

De frågeställningar som ställdes vid projektets början var följande:

- Kommer en *TCP*-anslutning att vara tillräcklig med avseende på tidsfördröjning och överföringssbelastning?
- Kan det externa ramverket libGDX, som är utvecklat för att underlätta spelutveckling, bidra i utvecklingen av projektet, och om inte, behöver egna motsvarande klasser som hanterar kamera och *shading* skrivas från grunden?
- Androidutveckling sker på olika API-nivåer där en högre nivå representerar en nyare version. Högre nivåer kan köra program som är utvecklade på lägre nivå men inte tvärtom. En avvägning behöver göras mellan användbar teknik och kompatibilitet till majoriteten av mobila enheter. Hur ska denna göras?
- Är det tillräckligt att använda värden från accelerometern på kontrollen för att styra bilen, eller behövs någon mer avancerad metod, exempelvis *sensor fusion*?

## 1.4 Avgränsningar

Kunden hade inga krav gällande målplattform. Gruppen undersökte möjligheten att utveckla för de två mest använda operativsystemen för mobila enheter [1], vilket är **Android** och **iOS**. Då det krävs en betald licens för att utveckla till **iOS**, vilket inte gruppen hade tillgång till, valde gruppen **Android** som målplattform.

Målgruppen som valdes vid projektets start var personer i åldrarna 12 till 35 år. Kunden uttryckte en önskan om att yngre spelare skulle kunna använda systemet men då det inte låg i gruppens intresse och det inte var ett krav prioriterades det bort.

Som tidigare nämnt skulle ett spel utvecklas för att testa systemet. Spelets grafik var av låg prioritet för att undvika att ta fokus från arbetets huvuduppgift; att utforma systemet. Spelet avgränsades därav till att vara i 2D och det beslutades att minimalt med tid skulle läggas på att få spelet att se bra ut innan systemet fungerande.

## 1.5 Spelidé

Gruppen beslutade att skapa ett bilspel. Fordonen ses från ovan och styrs med hjälp av de rotations-sensorer som de mobila enheterna besitter, till exempel gyroskop, accelerometer eller kompass.

Till skillnad från många andra *multiplayer*-bilspel visas detta inte i delad skärm utan det är endast en bild på skärmen där samtliga spelare ser sitt fordon. Bilden är fokuserad kring den bil som för tillfället leder och om en bil hamnar utanför skärmen har den användaren förlorat omgången.

En vinnare kan koras på två sätt. Antingen kör ledaren ifrån alla andra bilar så att det endast är ledarens bil kvar i bild, eller så avslutas spelet då ledaren korsar mållinjen efter ett förbestämt antal varv.

# Kapitel 2

## Relaterat arbete

Samtliga gruppmedlemmar har deltagit i föreläsningar med Karljohan Lundin Palmerius från mitten av januari till mitten av februari 2015. Utöver detta har stora delar av *Software Engineering* av Pfleeger och Atlee [2] lästs och övrig information har hittats på internet.

### 2.1 Arbetsmetod inom mjukvaruutveckling

Det finns statistik över hur stor del av alla IT-projekt som genomförs som går över budget, tar för lång tid och aldrig möter sina mål [3]. Enligt Bloch, Blumberg och Laartz[4] uppskattas hela 17% av alla IT-projekt gå så illa att de hotar hela existensen av företaget. Den lösning som föreslås av Mittal [3] är att arbeta *agilt* i kombination med att göra projektet så litet som möjligt samt ha bra stöd av projektets ledning.

Enligt Pfleeger och Atlee[2] är det optimala sättet att arbeta med mjukvaruutveckling att arbeta *agilt*. Detta bör göras genom att bryta ner arbetet i mindre faser och arbeta med dessa iterativt och inkrementellt. Pfleeger och Atlee[2] lägger även stor tyngd vid kontakt med kunden och en kontinuerlig kravanalys för att se till att arbetet är på väg åt rätt håll. Genom att arbeta på det sättet har kunden och utvecklingsteamet under hela projekttiden haft god uppfattning av huruvida projektet kommer att bli klart i tid och kan enkelt påverka resultatet. Läs mer om hur gruppen använde sig av detta i kapitel 3.1 för arbetsrutiner och 3.6 för kravhantering.

### 2.2 Liknande system

Det finns ett fåtal redan existerande system vars funktionalitet kan liknas vid det här projektet. Projektgruppen har studerat dessa system, vilket har givit inspiration och idéer till förbättringsområden till detta projekt.

Systemet **Couch+**[5], som utvecklas av det Jönköpingsbaserade *start up*-företaget **Couch+ AB**, är ett spelsystem som fungerar i princip som en spelkonsol. Användarna kan välja ett spel ifrån systemets meny och använda **Android**-enheter som handkontroller. Systemet går att kontrollera med handkontrollerna via ett fjärrkontrollsliknande gränssnitt. Till **Couch+** går det dock ej att utveckla egna spel.

Till **Sony's PlayStation 4**-konsol kan flera **PlayStation Vita**-enheter användas som handkontroller [6]. **PlayStation Vita** har även en pekplatta som kan användas till att navigera i menyer. **Vita**-enheter liknar dock mer en traditionell handkontroll jämfört med en mobil enhet eftersom de har flera styrspakar och fysiska knappar.

## 2.3 Undersökning av ramverk

De ramverk som undersöktes under projektets gång är libGDX och KryoNet. Gruppen tog beslut om användandet av ramverken utifrån undersökningar kring tidigare arbeten då de använts samt genom att praktiskt testa ramverken. En redogörelse för andra projekt som använt libGDX och KryoNet följer.

### 2.3.1 LibGDX

LibGDX är ett ramverk som underlättar spelutveckling för mobila plattformar med hjälp av färdiga klasser och funktioner som bland annat hanterar kamera, *sprites*, *shading*, grafiska gränssnitt och animationer. LibGDX är plattformsoberoende och det finns möjlighet att konvertera dess kod till en **Android**-applikation, en HTML webbsida, ett dator program och även en **iOS**-applikation om utvecklarlicens innehålls. LibGDX är *open source*, vilket innebär att källkoden är öppen för allmänheten.

Ahmed och Aule[7] har gjort en utvärdering av libGDX och utveckling för **Android**. De har vid projektets start aldrig använt libGDX och testar att skapa ett projekt samt bygga spel med olika funktioner. Under tiden förs noggrann dokumentation om svårighetsgrad och problem som uppkommer. Prestandan testas på olika sätt, exempelvis genom att skapa stora mängder objekt och undersöka om en tidsfördräjning uppstår. Författarna till rapporten uppfattar att ramverket är relativt enkelt för en nybörjare att använda och det går snabbt att få användbara resultat. Det negativa anser Ahmed och Aule[7] är att utvecklaren aldrig får förståelse för **Android**-utveckling i sig, utan blir beroende av ett speciellt verktyg.

Tekniskt sett saknar Ahmed och Aule[7] ett större erbjudande av klasser för dialogrutor då endast enkla verktyg som textrutor och knappar existerar. De saknade också ett bättre system för att ordna grafiska komponenter på skärmen och spenderade mycket tid på finjusteringar av positioneringar hos objekt.

En fördel är även att spelet kan köras direkt på utvecklarens dator. Det snabbar upp utvecklingsprocessen då långsamma emulatorer<sup>1</sup> och anslutna mobila enheter kan undvikas. Ahmed och Aule[7] drar slutsatsen att libGDX kan vara av fördel att använda för ett mindre erfaret utvecklingsteam då prestandan är bra för stora mängder data, många kodexempel finns att hitta samt då en stor mängd hjälpklasser finns att använda sig av. Hjälpklasserna låter utvecklingsteamet slippa undan viss avancerad implementation.

### 2.3.2 KryoNet

KryoNet[9] är ett Java-bibliotek som använder ett förenklat API för nätverkskommunikation. Enligt dokumentationen för biblioteket är KryoNet anpassat för att vara effektivt för klient- och serveranvändanden och har optimeras för applikationer där paket ska skickas så snabbt som möjligt.

Det finns betydligt mindre dokumentation om tidigare utförda projekt med KryoNet än libGDX. Grundaren till **Esoteric Software**, Nathan Sweet, är upphovsman till libGDX och medupphovsman till KryoNet[10]. Enligt dokumentationen för KryoNet[9] är KryoNet ett bra komplement till libGDX om det ska utvecklas till spel med avancerade nätverksfunktionalitet. KryoNet gör enligt dokumentationen nätverkskommunikationen mer lättförståelig och utvecklaren slipper bygga upp funktioner från grunden.

För att få grundförståelse för KryoNet och libGDX har gruppen tagit del av ett flertal *tutorials*, undervisningsguider, av utvecklaren som kallar sig Dermetfan[11] främst genom **Youtube**[12]. Enligt

<sup>1</sup>Emulator: En programvara i **Android Studio** som låter utvecklaren testa spelet på en simulerad mobil enhet på datorskärmen

Dermetfan är KryoNet och libGDX bra komplement och passande för nybörjare inom spelutveckling.

## 2.4 TV-uppkoppling

För att få systemet att fungera på TV krävs det att det finns möjlighet att spegla surfplattans bild på en TV. Alternativ för att kunna åstadkomma detta är följande:

- HDMI-kabel
- Chromecast <sup>2</sup>
- Android TV <sup>3</sup>

Majoriteten av mobila enheter har inte ett HDMI-uttag. De enheter som har HDMI-uttag har automatisk spegling av skärmen. För den majoritet av mobila enheter som inte har tillgång till HDMI-uttag finns möjligheten att spegla skärmen med hjälp av **Chromecast**, men detta fungerar inte bra för spel då det blir en födröjning från enheten som strömmar datan. Gruppen hade inte tillfälle att testa det tredje alternativet, **Android TV**, och kunde därmed ej utforska det alternativet.

---

<sup>2</sup>Chromecast: En enhet som strömmar mediadata till TV

<sup>3</sup>Android TV: En TV som använder sig av **Android** som operativsystem

# Kapitel 3

## Utvecklingsprocessen

### 3.1 Scrum

Gruppen har arbetat enligt riktlinjer för den *agila* arbetsmetoden *Scrum* (läs mer om *Scrum* i *The Scrum Guide* [13]). I *Scrum* delas projektiden upp i *sprintar*, där en *sprint* är två till fyra veckor lång. Till varje *sprint* ska ett antal *stories* utformas. *Sprintens* mål är att möjliggöra *stories* i tid till *sprintens* slut. En *story* kan exempelvis vara ”Som användare vill jag kunna stänga av systemet med min mobiltelefon.” *Storyn* delas sedan upp i *tasks* som beskriver det praktiska arbetet utvecklingsteamet måste göra för att fullfölja *storyn*. Ett exempel på en typisk *task* kan vara ”Implementera grafiskt gränssnitt för en meny hos klientapplikationen.” En *task* bör vara tillräckligt liten för att kunna avklaras under en dag.

Vid projektets uppstart planerades hur arbetet skulle läggas upp och hur organisationen skulle gå till. En projektplan utformades även. Detta gjordes under *sprint* noll, en förberedelsefas inför den första *sprinten*.

I *Scrum* ingår, enligt Schwaber och Sutherland[13], att varje dag börja med ett stående möte. Under detta möte ska gruppmedlemmarna en efter en beskriva kort vad som gjordes under föregående arbetstillfälle samt vad som ska göras under dagen. Beskrivningen görs efter vilka *tasks* som slutförts och ska åtas.

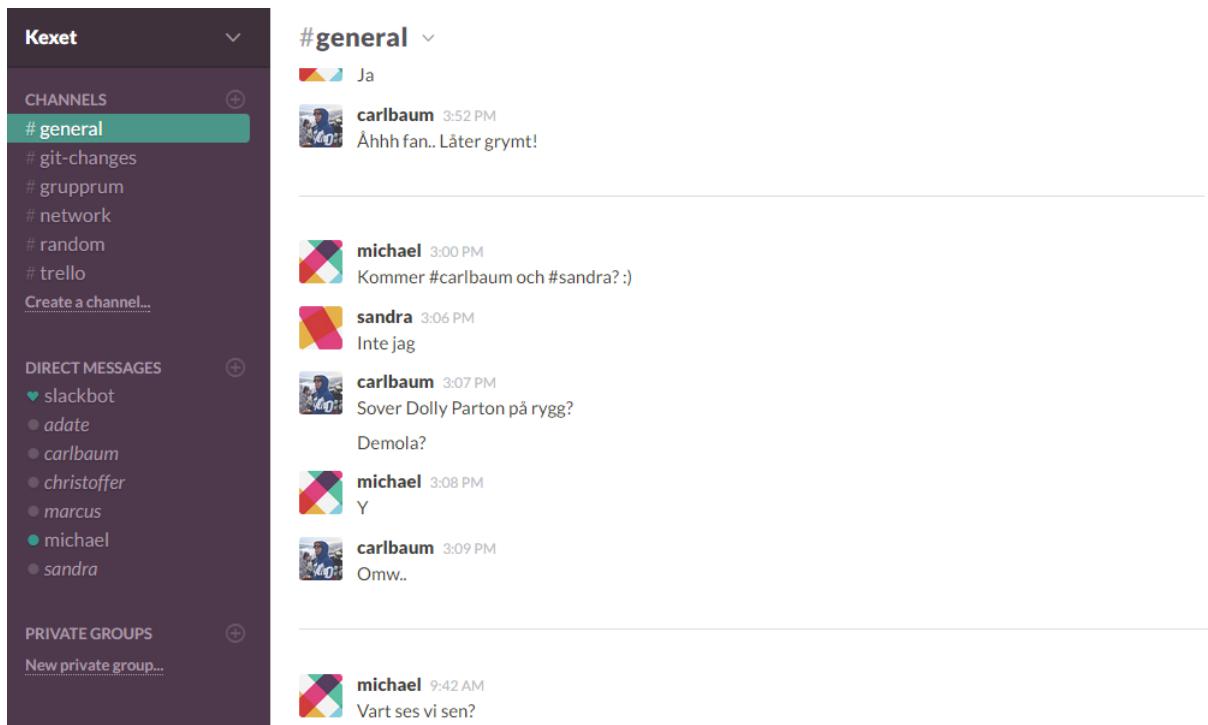
Vid start av en ny *sprint* hålls ett stort uppstartsmöte med hela utvecklingsgruppen där *sprintens* längd, mål, *stories*, *tasks*, ansvarsområden och schemaläggning bestäms. Detta möte bör enligt riktlinjer för *Scrum* ta ca fyra timmar [14]. Ett avstämningsmöte med kunden hålls i slutet eller precis efter uppstart av varje *sprint* för att hålla en nära kontakt med kunden samt säkerställa att kundens krav tillgodoses. Vad som diskuteras under dessa möten ska enligt Schwaber och Sutherland[13] dokumenteras. För att kunna användas under framtida *sprintplaneringar* och på så sätt undvika återupprepning av problem.

Ett möte avslutas enligt Schwaber och Sutherland[13] med ett *retrospective*, vilket är ett tillbakablicksmöte där den avklarade *sprinten* diskuteras. De frågor som ska besvaras under detta möte är följande:

- Vad har gått bra under *sprinten*?
- Vad har gått mindre bra under *sprinten*?
- Vad kan förbättras under nästa *sprint*?

## 3.2 Arbetsrutiner

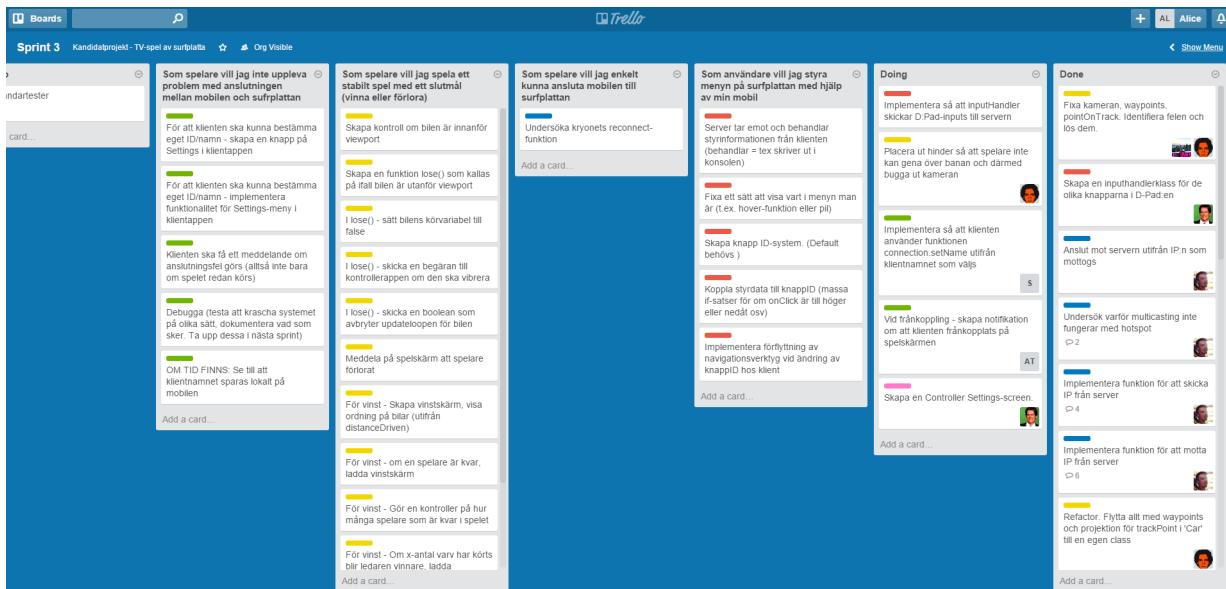
Kommunikation inom gruppen skedde över ett flertal olika medier. Verktyget **Slack** användes som en samlingsgrupp som även varit anslutet till de övriga programmen som användts. I **Slack** delades gruppkonversationer in efter teman vilket gjorde det möjligt för de utvecklare som samarbetade med en specifik uppgift att föra en privat diskussion utan att andra i gruppen skulle bli störda, se figur 3.1. Under fliken ”#grupprum” stod vilket rum som är bokat för dagen, medan det under ”#general” diskuterades mer allmänna frågor där det inte var ett stort problem om ett meddelande inte lästes av alla.



Figur 3.1: Ett exempel av hur verktyget **Slack** kan användas

Dokument som mötesprotokoll och agendor samt foton på *whiteboards* och UML-diagram lades upp i en mapp på **Google Drive** där samtliga gruppmedlemmar kunde redigera alla dokument. Läs mer om hur gruppen använt UML i avsnitt 4.1. **Google Drive** tillåter flera personer att redigera samma dokument samtidigt och visar även vilken ändring som är gjord senast och vem som gjorde ändringen.

Verktyget **Trello** användes för att strukturera arbetet i små uppgifter och för att hålla koll på vem som gör vad. De *stories* som bestämdes under uppstartsmötet för *sprinten* radades upp som kolumner. Under dessa kolumner placerades flyttbara kort med respektive *tasks*, se figur 3.2. Den utvecklare som åtog sig en uppgift skrev sitt namn på motsvarande kort och flyttar det till kolumnen ”*Doing*”. Då en uppgift var färdig flyttas den till kolumnen ”*Done*”, där den blev granskad av en annan gruppmedlem innan den blev godkänd som färdig. Då ett kort förflyttas i **Trello** meddelas detta även på **Slack**.



Figur 3.2: Ett exempel av hur verktyget **Trello** kan användas

Koden kommenterades kontinuerligt på engelska under projektets gång. Kommentarerna var tydliga och förklarade vad som gjorts i till exempel en funktion. Gruppen kom överens om att verktyget Javadoc skulle användas för att generera Javadoc-kommentarer i koden till HTML-format. Detta leder i regel till en bättre överblick som en utomstående läsare lättare kan följa, till skillnad från att behöva läsa igenom källkod.

### 3.3 Organisation

Huvudansvaret delades upp mellan gruppmedlemmarna för att jämma ut arbetsfördelningen samt skapa känsla av delaktighet hos alla gruppmedlemmar. Ansvarig för respektive område var inte nödvändigtvis den person som skulle utföra arbetet, utan ansvarade för att arbetet skulle genomföras. De roller som delades ut var följande:

- *Scrum Master* håller i möten och ser till att en agenda följs under större möten<sup>1</sup>.
- Sekreterare och dokumentationsansvarig för anteckningar under större möten och sammanställer dessa till mötesprotokoll. Det är också upp till denne att se till att alla dokument finns i gruppens gemensamma **Google Drive**-mapp.
- Modelleringsansvarig ansvarar för att systemets struktur hålls tydlig och att alla gruppmedlemmar är väl införstådda på systemets modeller.
- Kodansvarig ansvarar för att koden är läsbar och välkommenterad, att refaktorering sker och att kodstrukturen följer den struktur som bestämts under modelleringen.
- Testansvarig ansvarar för att testning av funktioner och system sker samt ordnar kodgranskningstillfällen i form av *walkthroughs*, läs mer i kapitel 4.8.
- Kommunikationsansvarig har huvudkontakten med kunden och uppdragsgivaren och har även huvudansvar för bokning av sal samt ansvarar för att all information når ut till alla gruppmedlemmar.

<sup>1</sup>Med större möten avses *sprintplaneringsmöten* och *retrospective*

- Kravanalytiker har huvudansvar för att användarundersökningar, kravhantering och kravanalys utförs och att arbetet prioriteras enligt kraven.

Samtliga ansvarsområden togs om hand utöver resten av arbetet i projektet och det var alltså ingen gruppmedlem som enbart ägnade sig åt sitt ansvarsområde. Under varje *sprint* delades gruppen in i *sprintspecifika* ansvarsområden representerade av en *story*. Exempelvis delades gruppen upp i par som vardera tog hand om en *story*. Vid arbete med *Scrum* kan det finnas en *produktägare*[13]. Detta valdes bort i projektet efter en diskussion mellan gruppmedlemmarna, läs mer om detta i avsnitt 6.1.

## 3.4 Schemaläggning

Vid varje *sprintplaneringsmöte* bestämdes ett schema för *sprinten*. Då samtliga gruppmedlemmar läser samma kurser var planering av schema inte komplicerat men hänsyn fick tas till övriga kurser samt tid till eventuella omtentor och andra engagemang. Projektet krävde inte arbete utanför arbetstid (kl 8-17, måndag-fredag), men inga hinder fanns för att arbeta vidare på egen hand utanför schemalagda tider. Om en gruppmedlem inte kunde närvara på ett pass togs denna tid ikapp på egen hand utanför utsatt arbetstid.

Ett gemensamt schema skapades i **Google Kalender** där även den bokade salen kunde registreras.

## 3.5 Utvecklingsverktyg och versionshantering

### 3.5.1 Android Studio

Gruppen valde att använda sig av **Android Studio** som är en gratis nedladdningsbar IDE byggd på **IntelliJ IDEA Community Edition** och använder sig av **Java**.

### 3.5.2 Git och GitHub

**Git** användes till versionshantering. Samtliga utvecklare hade ett eget konto på den webbaserade **Git-samlingsplatsen GitHub** där ett *repository*, en datastruktur som **Git** sparar information i, skapades för projektet **Varsom Games**. Genom kommandotolken **Git Bash** på **Windows** eller Terminal på **OSX**- och **Linux**-distributioner kunde utvecklarna lägga upp nya ändringar i projektet, hämta hem vad någon annan gjort, jobba på separata delar av projektet samt slå samman dessa delar till en och samma version. En fungerande version skulle alltid finnas uppe som alla kunde hämta vid behov. Versionshanteringen möjliggjorde också att kunna gå tillbaka och se tidigare versioner av samma kodfil, samt återställa till en tidigare version om olösliga problem uppstod.

## 3.6 Kravhantering

Kravhanteringen och kravanalysen sköttes under hela projektet men med fokus vid vissa tillfällen. Det skedde enligt listan nedan för att säkerställa att kraven förstods och sattes i perspektiv till varandra:

1. Kundens och intressenters krav samlades in. Hur detta skedde återfinns i underrubrikerna nedan.
2. Gruppen analyserade och förstod de givna kraven.

3. Kraven dokumenterades och det antagna systemets beteende modellerades och kan läsa mer om i kapitel 4.1.
4. Modeller och dokumentation bekräftades av kunden och andra intressenter så att de stämde överens med kraven.
5. Om det stämde var kravanalysen färdig. Om det inte stämde började arbetet om på steg ett, där metoden för att samla in kraven ändrades.

Listan bygger på teori av Pfleeger & Atlee[2]. En beskrivning av de olika typer av krav som behandlades och hur det utfördes följer nedan.

### 3.6.1 Krav från kund

Vid projektets start gavs en kortfattad beskrivning av produkten kunden efterfrågade. Under ett möte med kunden framgick att arbetet var väldigt mycket upp till utvecklarna själva men att systemet och dess funktionalitet skulle prioriteras över spel och grafik. En lista på de krav kunden specificerat sammantälldes, se nedan:

- Spelet ska vara spelbart för minst två användare samtidigt
- Som spelare ska jag kunna identifiera vilket fordon jag styr
- Tidsfördröjningen ska inte störa spelandet och får maximalt vara 200 millisekunder
- Som spelare ska jag kunna backa mitt fordon om jag fastnar
- Spelet ska kräva användarens fulla uppmärksamhet - om jag lägger ifrån mig kontrollern under spelets gång kommer jag förlora

Under projektets gång hade kommunikationsansvarige kontakten med kunden och såg till att möten blev inbokade inför varje ny *sprint*. Kundens krav sammantälldes efter mötet i en produktbeskrivning och den person med ansvar för kravanalysen såg till att kraven sattes i fokus och eventuella frågetecken rätades ut under kundmötena. Listan i kapitel 3.6 följdes ibland direkt under kundmötet och ibland under *sprinten* med återkoppling till kunden via mail eller extra inbokat möte.

### 3.6.2 Krav från övriga intressenter

Övriga intressenter under projektet innefattade främst de slutgiltiga användarna samt uppdragsgivaren.

För att undersöka intresse och krav hos den slutgiltiga användaren skapades ett formulär som skickades ut till vänner, släkt och bekanta till gruppens medlemmar. Se formuläret och en sammanställning av resultatet i bilaga B. Då den grafiska delen av systemet ännu inte var utvecklad innehöll formuläret mycket beskrivningar och hypotetiska frågor. Syftet med användartestet var att undersöka målgruppens intressen för att kunna göra en prioritering av funktionalitet att implementera.

Ett andra användartest skapades där användaren skulle få uppgifter som till exempel att byta namn eller starta en spelomgång. Medan användaren utförde uppgiften skulle en gruppmedlem anteckna vad användaren gjorde. Detta skulle ge gruppen en uppfattning om hur användarvänligt systemet och spelet var och hur det skulle kunna förbättras. Eftersom gruppen inte hade implementerat någon grafik när testet utvecklades valdes det att vänta med att köra testet tills det fanns ett gränssnitt som gruppen

ville få återkoppling till. När gränsnittet sedan var implementerat fanns inte tillräckligt med tid kvar på projektet för att hinna både genomföra testet och uppdatera gränsnittet efteråt så testet utfördes inte.

Utvecklarna och uppdragsgivaren hade avstämningssmöten med jämna mellanrum som bokades av uppdragsgivaren själv. Under dessa möten fick uppdragsgivaren en klar bild av hur arbetet i gruppen gick, hur långt projektet kommit i förhållande till milstolpar och projektplan och om projektet beräknades bli färdigt i tid. Krav från uppdragsgivaren handlade om tidsplanen, resurser eller arbetsmetoder och diskuterades både under avstämningssmöten och sedan under *sprintplaneringsmöten* för att genomföra kravanalysen. En lista på de krav uppdragsgivaren hade vid början av projektet följer:

- Gruppen ska arbeta *agilt* med metoden *Scrum*
- Alla gruppmedlemmar ska bidra under projektet, ingen gruppmedlem ska hamna utanför arbetet
- Arbetet ska göras i förhållande till en tidsplan med milstolpar för att försäkra att det blir färdigt i tid

# Kapitel 4

## Implementation

### 4.1 Modellering och Unified Modeling Language (UML)

Eftersom mjukvarusystem av större skala, såsom detta, ofta kan bli invecklade och komplicerade är det viktigt att modellera systemets struktur så att alla gruppmedlemmar förstår den. Det finns flera olika sätt att modellera upp systemarkitekturen och ett sätt som gruppen testade i det här projektet är UML.

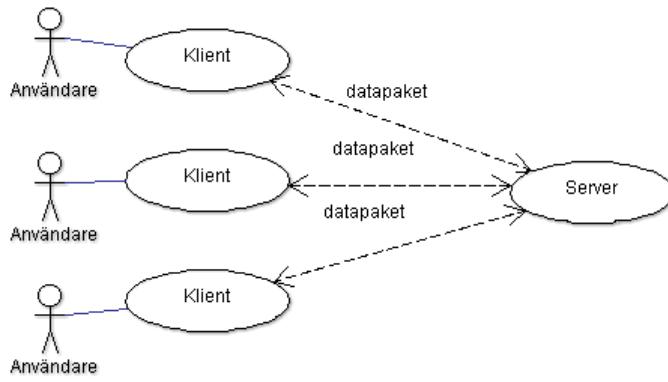
UML står för *Unified Modeling Language* och är ett objektorienterat system för att grafiskt modellera systemarkitektur. Flera typer av UML-diagram skapades under detta projekt såsom *Use-Case*-diagram, aktivitetsdiagram och klassdiagram.

Modellering var vid projektets start en ny metod för de flesta av gruppens medlemmar. Det visade sig användbart under skapandet av systemarkitekturen men också tidskrävande. Eftersom de flesta modeller som skapades i början av projektet kom att ändras under arbetets gång beslutade gruppen att inte göra allt för ingående UML-modeller utan använda *just-in-time-modelling*(JIT). Detta innebär att modeller skapas precis innan de behövs vilket minskar tiden som läggs på modellering och mindre modellering behöver göras om när nya beslut fattas. Alltså användes modelleringen främst för att hjälpa förståelsen över hur systemets moduler hänger ihop och för att kunna planera vidareutvecklingen av dessa istället för att användas som dokumentation. Gruppen ritade därför mestadels enklare modeller på papper och whiteboard istället för att lägga ned tid på att göra ingående UML-diagram.

Om modellering ej skulle använts initialt och koden skulle skapas spontant utan vidare planering finns stor risk att hela kodstrukturen skulle blivit ostrukturerad och svår att vidareutveckla.

### 4.2 Systemarkitektur

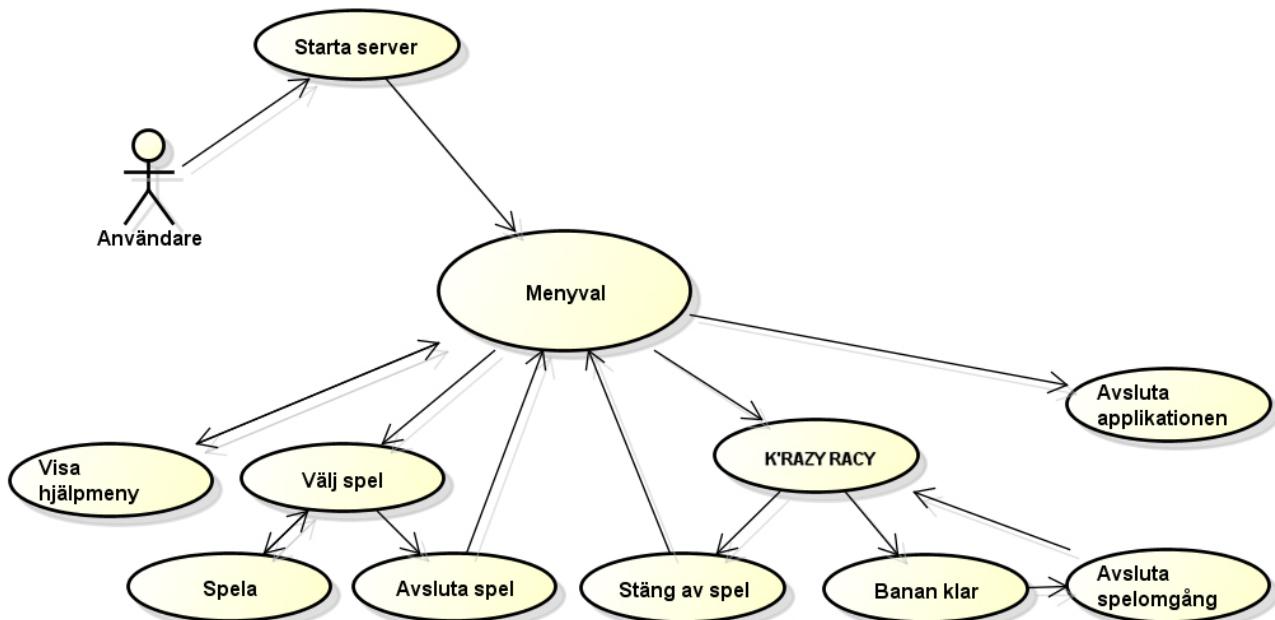
Spelarnas enheter skickar information till enheten som visar spelet, för enkelhetens skull kallas den för surfplatta i detta stycke men skulle även kunna vara någon annan mobil enhet. Exempel på information som skickas är hur enheterna är lutade och hur användarna trycker på pekskärmarna. Surfplattan tar sedan denna information och beräknar vad som ska visas på skärmen enligt spellogiken för pågående spel. Om något händer som en spelares enhet behöver veta skickas denna information tillbaka. Detta gör att spelarnas enheter aldrig behöver kommunicera med varandra vilket resulterar i att en server-klient-struktur är lämpad, se figur 4.1. Surfplattan är servern och spelarnas enheter är kunder.



Figur 4.1: Schema över server-klient-struktur

## 4.3 Serverstruktur

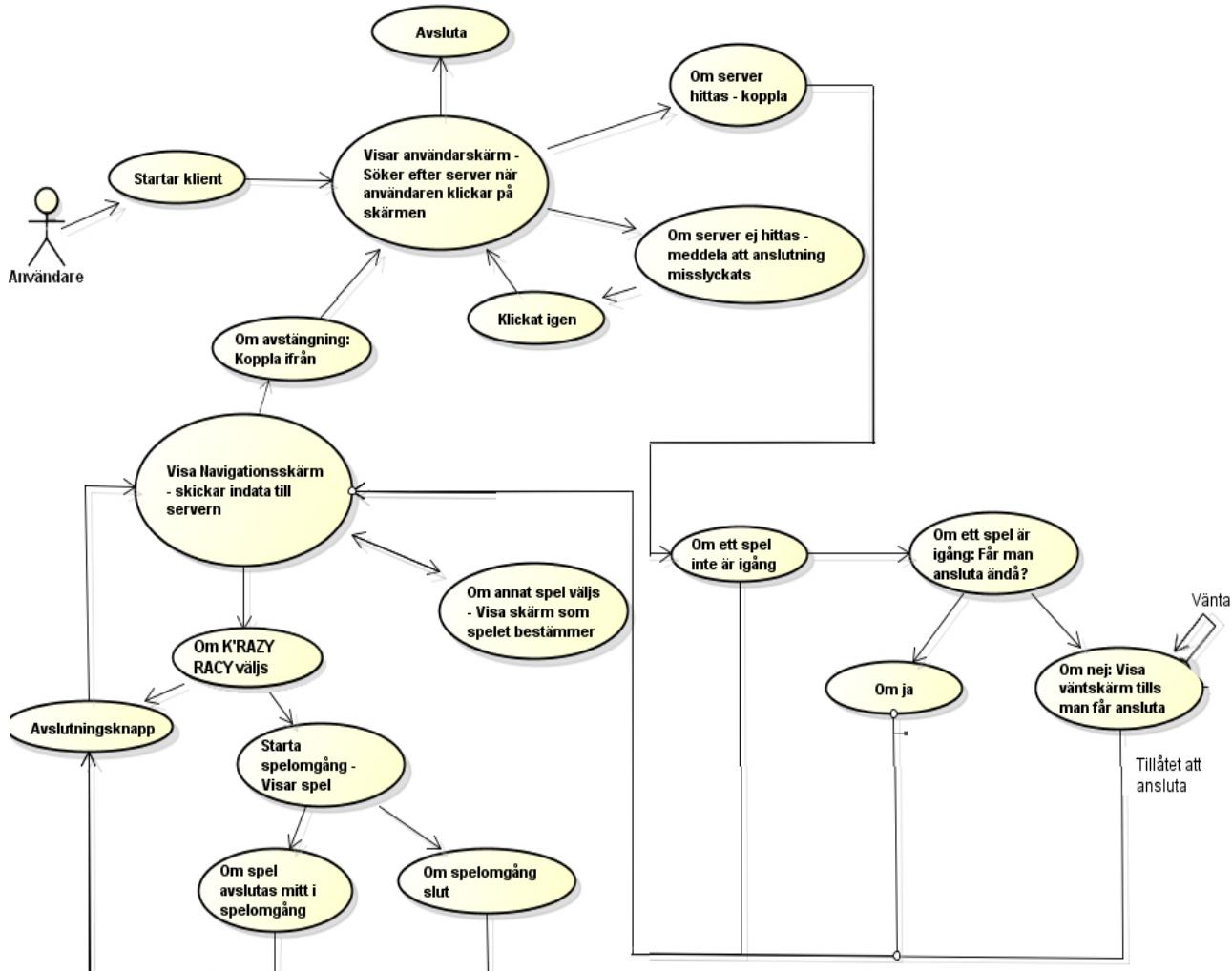
Serverns struktur är uppbyggd så att en användare alltid ska kunna ansluta om en server är igång. För att servern ska kunna utföra spelberäkningar samtidigt som den är öppen för anslutningar från fler klienter behövs två separata trådar; en tråd som utför alla renderingsberäkningar och en som tar emot anslutningsförfrågningar och skickar svar. Användaren kan bara påverka renderingstråden där denne kan navigera i menyer och spela spel. När en användare väljer att stänga av applikationen är det enda gången användaren påverkar anslutningstråden, annars arbetar anslutningstråden självständigt. Figur 4.2 visar ett *use case*-diagram för servern där menyalternativ visas.

Figur 4.2: Ett *use case*-diagram för servern

## 4.4 Klientstruktur

Strukturen för klienten fokuserar på navigation på servern. Klienten visar en scen som är anpassad efter vad som sker på servern, se figur 4.3. När klienten startas söker den efter en server att ansluta sig till. När en sådan hittats frågar klienten servern om ett spel körs och om klienten får medverka

i rådande spelomgång. Vissa spel tillåter inte att klienter som ansluter efter att spelet startats kan påverka spelet. Om ett sådant spel körs visas en scen som uppmanar användaren att vänta tills rådande spelomgång är slut. När det är tillåtet för de nya anslutningarna att påverka spelet skickar servern information om vilken scen som klienterna ska visa. Om inget spel är aktivt visas systemets egen navigationsscen [5.3](#).



Figur 4.3: Ett flöde för klientens upplägg

## 4.5 Nätverkskommunikation

Kommunikationen sker enligt ett server-klient-baserat förhållande, som nämnt i avsnitt 4.2. De är länkade med en *TCP*-koppling men det är också möjligt att skicka paket med *UDP*. Som stöd för nätverkskommunikationen användes ramverket KryoNet, tidigare nämnt i kapitlet om relaterat arbete under avsnitt 2.3.2.

*TCP* är användbart när det är viktigt att det skickade paketet når destinationen. Protokollet garanterar återkoppling och att paketen behandlas i den ordning som de skickades i. Det medför dock att kommunikationen kräver mer data och att det kan skapas en fördräjning i läsandet av paket. Om *TCP*-standarden används så skickas inte ett paket förrän paketet uppnått en viss datamängd. Istället valde gruppen att använda ramverket KryoNet och där är det redan inställt från början att packetet skickas så fort funktionen kallas.

*UDP* är ett lättviktigt protokoll som är kopplingslöst, det vill säga saknar återkoppling. Detta gör att paket som använder *UDP* inte går att kontrollera ifall de kommit fram. *UDP* använder mindre data än *TCP* och är därför lämpat för att skicka data som ”tål att tappas”. Vad det är för data förklaras längre ner i detta avsnitt. Dessutom kan problemet med tappad data oftast bortses ifrån eftersom förlusten är låg i stabila nätverk.

För att etablera kontakt mellan servern och klienterna så startar servern en *broadcast* som skickar ut ett meddelande innehållande sin egen IP-adress. En *broadcast* skickar ut paket med *UDP* till alla adresser som finns i LAN och WLAN. När klienterna fångat upp meddelandet så används det för att koppla upp sig till servern.

Under tiden servern och klienter är uppkopplade används både *TCP* och *UDP* för att skicka paket mellan enheterna. Valet av protokoll beror på hur viktigt det är att paketet kommer fram. Paket som måste nå slutdestinationen är paket som bara skickas en gång, till exempel paket med information om avstängning och paus. De är operationer som skulle skapa förvirring och missförstånd om de inte skedde när en användare tror att det ska ske. För att garantera att dessa paket kommer fram skickas de med *TCP*. Paket som skickas kontinuerligt är inte lika viktiga att de når sin destination eftersom näckommende paket skickas kort inpå och innehåller minimala skillnader. Ett sådant paket märks oftast inte av om det inte kommer fram och ”tål att tappas”. Paket med information om klienters lutning eller om gasknappen är nedtryckt är exempel på paket som skickas kontinuerligt. Dessa paket skickas med *UDP*.

## 4.6 Spelet

### 4.6.1 Obligatorisk struktur för samtliga spel

Eftersom det är tänkt att flera olika spel ska kunna spelas genom systemet är det viktigt att samtliga spel följer en specifik uppbyggnad och struktur. Av denna anledning skapades den abstrakta klassen *VarsomGame*. Instanser av denna klass kan startas från systemets huvudmeny.

Varje spels huvudklass ska vara en påbyggnad av *VarsomGame* genom Java-kommandot *extends*. Denna huvudklass måste använda kommandot *@Override* för att använda *VarsomGame*-metoden *handleDataFromClients()*, vilken servern dirigerar inkommande speldata till. Det är sedan upp till utvecklarna för spelet att skriva om funktionen så att datan hanteras och skickas vidare till korrekt funktion inom spelet.

### 4.6.2 Kamera

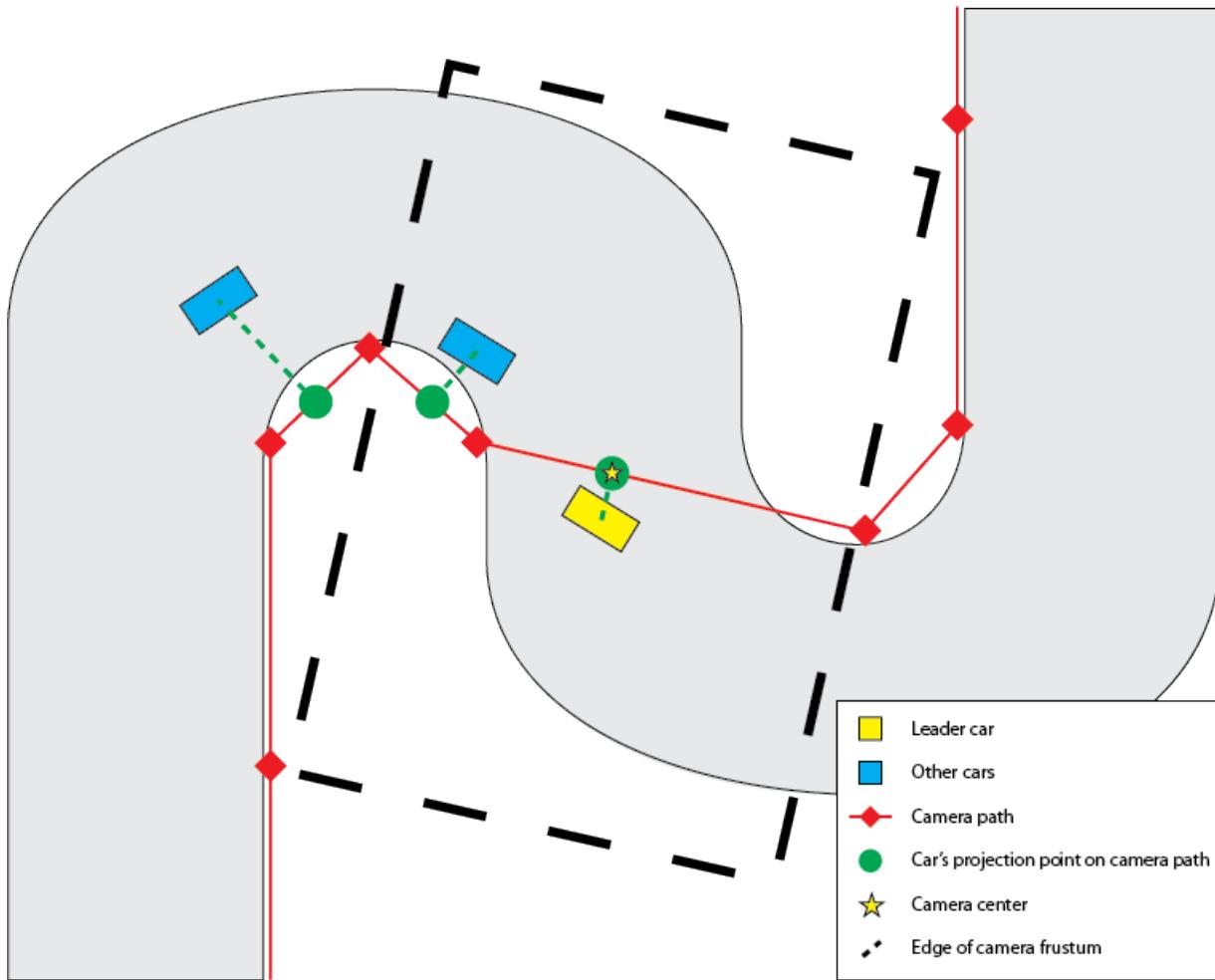
För att underlätta rendering av spelobjekt valde utvecklarna att använda en ortografisk kamera från libGDX. Kameran använder parallellprojektion som gör att alla objekt ser lika stora ut oavsett avstånd från kameran, vilket är precis vad ett 2D-spel behöver. Eftersom kameran ska följa ledarbilen men samtidigt stanna kvar inom banan beslutades det att kameran ska följa en förbestämd vägbana. Bilarnas positioner ska projiceras på denna vägbana och kameran följer ledarbilens projektionspunkt. Detta förhindrar att ledaren kan köra av banan och dra med sig kameran så att andra spelare hamnar utanför bild.

För att veta vilken linje av kamerans vägbana som projektionen ska ske på lagras den linje som projektionspunkten befinner sig på för tillfället. Med hjälp av denna linje sätts kamerans rotation. När projektionspunkten nått linjens ändpunkt väljs nästa linje som projektionslinje. Om punkterna placeras i mitten eller längs ytterkurvan finns det risk för att projektionspunkten inte når linjens ändpunkt. Därför placerades kamerans vägpunkter alltid på innerkurvor. Se figur 4.4.

Projektionsformeln som används kan ses i ekvation 4.1, där  $p$  är den resulterade projektionspunkten,  $\vec{v}$  är den aktuella projektionslinjen,  $a$  är startpunkten till  $\vec{v}$  och  $\vec{u}$  är vektorn från  $a$  till bilens centrum.[15]

$$p = a + \frac{\vec{u} \cdot \vec{v}}{|\vec{v}|^2} \vec{v} \quad (4.1)$$

Projektionspunkten används även för att bestämma i vilken ordning bilarna ligger. Varje gång punkten uppdateras beräknas längden av dess förflyttning och summeras sedan med den totala summan av dess tidigare förflyttningar. Den bil vars totala summa är störst är således ledarbilen. Pseudokoden som skrevs innan den faktiska koden återfinns i figur C.1 i bilaga C.



Figur 4.4: Kamerans placering i förhållande till dess bana och till bilarna

### 4.6.3 Styrning

Användaren ska kunna styra sin bil som om kontrollen vore en ratt. För att åstadkomma detta används enhetens accelerometer. Mer specifikt används accelerometerens x- och y-komponenter. För att räkna ut enhetens rotation används därefter funktionen *atan2* [16] från Javas Math-bibliotek. En vanlig arcus-tangens-operation resulterar i en vinkel i radianer inom omfånget  $[-\pi/2, \pi/2]$  till skillnad från *atan2* vilken ger omfånget  $[-\pi, \pi]$ . Denna funktion tar in både accelerometers x- och y-värde och ger tillbaka en vinkel i radianer.

Under *gameplay* skickar de klienter vars bil fortfarande är med i spelet kontinuerligt ett paket med en textsträng som innehåller relevant information till servern. Daten dirigeras vidare till spelets *handleDataFromClients*-funktion, se avsnitt 4.6.1. Denna sträng delas där upp i tre delar; en boolean som

representerar om gasknappen är nedtryckt eller inte, en *boolean* för om användaren trycker på backknappen eller ej, samt en *float* för klientenhetens rotation från nollläget i *landscape-mode*, dvs. det värdet som *atan2*-funktionen genererat. Denna information skickas då vidare till korrekt bil med hjälp av dennes anslutningsid.

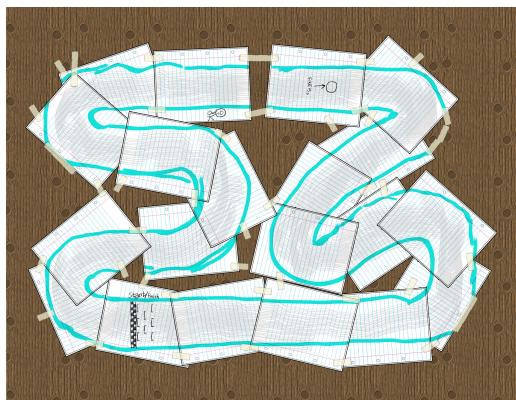
Bilarna är instanser av klassen *Car*, vilken kan konvertera de inkommande variablerna till acceleration och vridning på bilens däck. Vridningen på däcken styr, likt i verkligheten, bilens riktning. Den inkommande vinkelns justeras något med hjälp av en känslighetsfaktor för att ge bättre spelkänsla.

#### 4.6.4 Övrig spellogik

*Gameplay* hanteras i en klass som kallas *GameScreen*, vilket är en påbyggnad av libGDXs *Screen*-klass. Det är i denna klass som uppdatering och rendering sker. Kamera, grafiskt användargränssnitt och fysik sköts även här.

För att underlätta arbetet med fysiksimulation och kollisioner mellan bilar och hinder användes box2D, vilket är ett API just för detta. Implementation av dess inbyggda funktionalitet kräver att de objekt som ska påverkas av fysik placeras i en instans av box2D:s *World*-klass, som hanterar fysiken. Detta *World*-objekt tillhör och uppdateras också i *GameScreen*. I nuvarande utförande är det endast hinder och bilar som är skapade med hjälp av box2D och påverkas av fysik.

I *GameScreen* skapas också den bana som användaren valt från huvudmenyn. Varje bana är en påbyggnad av den abstrakta klassen *Track*, vilken skapats specifikt för detta spel. I varje påbyggnad av *Track* finns banans bakgrundsbild, hinder, kamerans vägbana samt varje bils startpunkt. Det finns även en tillhörande *pixmap* till bakgrundsbilden. Ur denna kan specifika pixelvärdet hämtas, vilket används för att påverka bilarna på olika sätt. För närvarande består denna *pixmap* endast av vitt och svart, vilket representerar om bilen befinner sig på banan eller utanför dess gränser. Om bilen befinner sig på svart är den utanför banan och kan då inte köra lika fort. Se förhållandet mellan bakgrundsbilden och dess *pixmap* i figur 4.5a och 4.5b.



(a) Track



(b)Pixmap

Figur 4.5: Banans bakgrundsbild (a) samt dess *pixmap* (b)

#### 4.6.5 Ljud och musik

För att förbättra spelkänslan och göra spelet mer estetiskt tilltalande lades ljudeffekter och musik till. LibGDX har stöd för ett flertal olika ljudformat, bland annat *.wav* och *.mp3*.

En statisk ljudhanteringsklass baserad på *Singleton*-designmönstret[17] skrevs, vars syfte är att kunna läsa in, spela upp och stoppa ljudfiler. Denna ljudhanteringsklass och dess publika metoder kan nås

från alla andra klasser i spelet. Ljudhanteraren konstruerades även så att det är möjligt att spela spelet i ljudlöst läge.

Ljuddesignen i spelet valdes till att vara ”galen och komisk” för att komplettera spelets tecknade grafik. De ljudeffekter och musikklinna som används i projektet följer *Creative Commons*-standard[18] och är fölaktligen gratis att använda i icke-kommersiella sammanhang som detta.

## 4.7 Testning

För att skapa ett stabilt system med hög standard var en viktig del att genomföra tester under projektets gång. I första *sprinten* implementerades automatiska tester i form av enhetstester. Dessa enhetstester skrevs med hjälp av ramverket JUnit som hjälper till med färdiga metoder för att testa applikationer som är skrivna i Java. Syftet med enhetstesterna var att kontrollera att metoder i klasser fungerat som förväntat. Dessa tester tog lång tid att skriva då ingen av gruppmedlemmarna hade tidigare erfarenhet av automatiska tester. Efter första *sprinten* skedde fortsättningsvis all testning i form av loggar i konsolen samt öppen diskussion mellan gruppmedlemmar. Användartester har även utförs på individer utanför gruppen vilket beskrivs i sektion 3.6.2.

### 4.7.1 Testning av tidsfördröjning

Då kunden ställt krav på spelets tidsfördröjning skulle denna testas. Testen utfördes genom att filma både kontroller och surfplatta under en spelomgång. Fordonetets hjul som indikerar dess svängning gjordes väl synliga och kontroller vreds från horisontellt läge till att luta ca 45 grader. Hjulen reagerade efter en viss tidsfördröjning på svängningen och roterade med 45 grader. Detta filmades med hög upplösning och bildhastighet på 120 bilder per sekund. Det gjordes med mobiltelefonen **OnePlus One**. Filmen lades sedan in i filmredigeringsverktyget **Adobe Premiere Pro CC 2014** som möjliggör att scanna bild per bild genom filmen för att mäta den tid det tog mellan svängning hos kontrollen och rotation hos hjulen på skärmen, det vill säga tidsfördröjningen.

## 4.8 Kodgranskning

Gruppen använde olika metoder för att granska koden. Den metod som fungerade bäst för gruppen var att efter varje *sprintavslut* ha en genomgång av den kod som skrevs under föregående *sprint*. En dator kopplas in till en TV eller en projektör och varje gruppmedlem får förklara vad de har gjort under *sprinten* och vad deras producerade kod gör. Under genomgången kommenteras koden med förklarande text och tillhörande kommentarer om vad som behöver refaktoreras eller ändras på. Avslutningen av genomgången är att koden refaktoreras för att bli mer lättläst och att den samtidigt följer Googles Java Style Guide.[19] Under diskussionen om refaktorering kom gruppen överens om vad variabler, funktioner och klasser ska döpas till för att koden ska bli enkel att följa. Det är viktigt att namnen är beskrivande och tydliga för att förenkla vidareutveckling.

Parprogrammering[20] användes vid behov under varje *sprint*. När någon stötte på ett problem, vare sig det handlade om förståelsen av existerande kod eller oklarheter kring hur en *task* ska lösas, kunde en annan gruppmedlem hjälpa till. Detta gjordes för att det är enkelt att ensam stirra sig blind på den egna koden. En annan person kan då hjälpa till med att ge ett annat perspektiv på problemet. Parprogrammering användes dock bara då det var absolut nödvändigt, eftersom parprogrammering är en mycket resurskrävande arbetsmetod.

# Kapitel 5

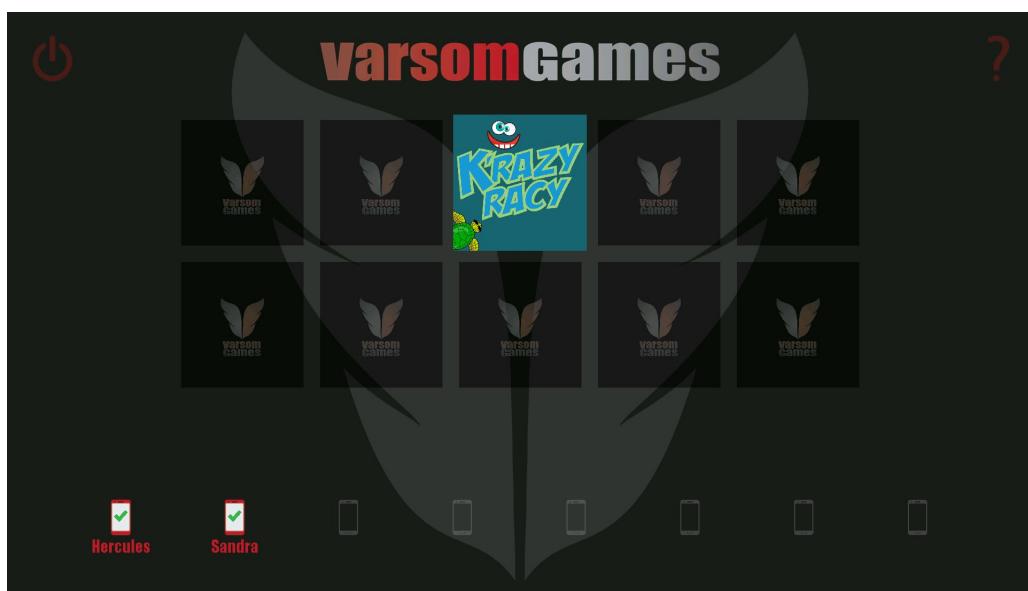
## Resultat

### 5.1 Systemet Varsom Games

Systemet består av två applikationer; en serverapplikation och en klientapplikation. Serverapplikationen körs fördelaktligen på en surfplatta eller dator för att få en större skärm att spela på. Klientappen körs på varje spelares egen mobil eller surfplatta.

#### 5.1.1 Servern

När servern startas visas huvudmenyn för systemet, se figur 5.1. Alla spel som användaren kan spela representeras av halvgenomskinliga bilder i menyn. För att starta ett spel måste användaren markera spelet och välja det. Det spel som är markerat blir lite större än de andra och slutar vara genomskinligt. För att byta vilket spel som är markerat kan användaren både använda klientens navigationskontroll på sin telefon efter att den anslutit sig till servern eller svepa fingret över serverenhetens skärm.



Figur 5.1: Systemets huvudmeny

För att välja ett spel kan användaren klicka på serverns skärm på tillhörande visningsbild när den är markerad eller använda sin telefon. Åtta spelare kan vara anslutna samtidigt på servern vilket indikeras av de åtta släckta telefonikonerna under visningsbilderna. När en användare ansluts till servern lyser

en av de släckta telefonerna upp i rött och vitt och spelarens namn visas under den. I figur 5.1 är två spelare anslutna.

Högst upp i vänstra hörnet finns en ikon för att stänga av servern. Klickar användaren på knappen kommer en ruta upp med en fråga om användaren är säker på att den vill stänga av systemet. Denna fråga kan besvaras genom att klicka på serverns skärm eller med navigationskontrollen på klienten. Om användaren inte vill klicka på serverns skärm kan användaren välja knappen via klienten.

I det andra hörnet, högst upp till höger, finns en hjälp-ikon i form av ett frågetecken. Om denna knapp väljs visas en sidomeny på höger sida av skärmen med förslag på anledningar och åtgärder till olika anslutningsproblem användarna skulle kunna råka ut för. Om serverapplikationen varit igång ett tag utan att ett spel startats kan det vara för att någon användare har problem med att ansluta. Då lyser hjälp-ikonen upp för att dra till sig uppmärksamhet så användaren kan få den hjälp den behöver.

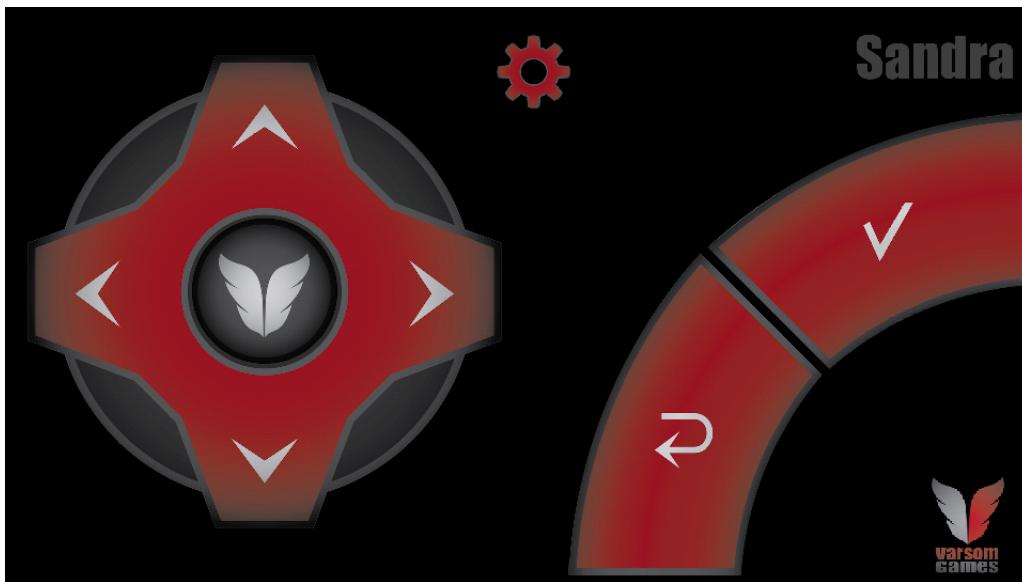
### 5.1.2 Klienten

Klientens olika tillstånd visas för användaren i form av olika scener. När applikationen startas visas en anslutningsscén, se figur 5.2, där användaren behöver klicka på skärmen för att ansluta till en server på samma nätverk. I nuläget ansluts klientapplikationen till den första servern som hittas.



Figur 5.2: Klientens anslutningsscén

När användaren är ansluten visas navigationsscenen, se figur 5.3. Navigationsscenen har pilar åt höger, vänster, upp och ned, också kallat *d-pad*, som användaren kan använda för att flytta markören på servern i de menyer som visas på servern. Det finns en inställningsknapp som möjliggör att användaren kan ställa in personliga inställningar som t.ex. namn och vibration i telefonen. Den ser ut som ett kuggjhul och är placerad högst upp och i mitten av scenen. Medan en användare ändrar inställningar kan alla andra anslutna användare fortfarande navigera i menyerna. En väljknapp och en bakåtknapp har placerats ut i form av en fjärdedels cirkel på höger halva av scenen, detta för att följa hur tummen naturligt rör sig över skärmen. Med väljknappen aktiveras det val som är markerat och med bakåtknappen tas användaren tillbaka till föregående meny på servern. Användarens namn visas högst upp till höger.



Figur 5.3: Klientens navigationsscen

## 5.2 K'razy Racy

### 5.2.1 Struktur och innehåll

När användaren startar spelet kallas systemet på spelets huvudklass, som följer uppbyggnaden för spel som nämns i avsnitt 4.6.1.

När spelet startas dirigerar huvudklassen programmet vidare till en så kallad *splash screen*, vilket är en skärm som visas medan bilder och filer laddas in. Detta medför att spelet och menyer visas snabbare längre fram.

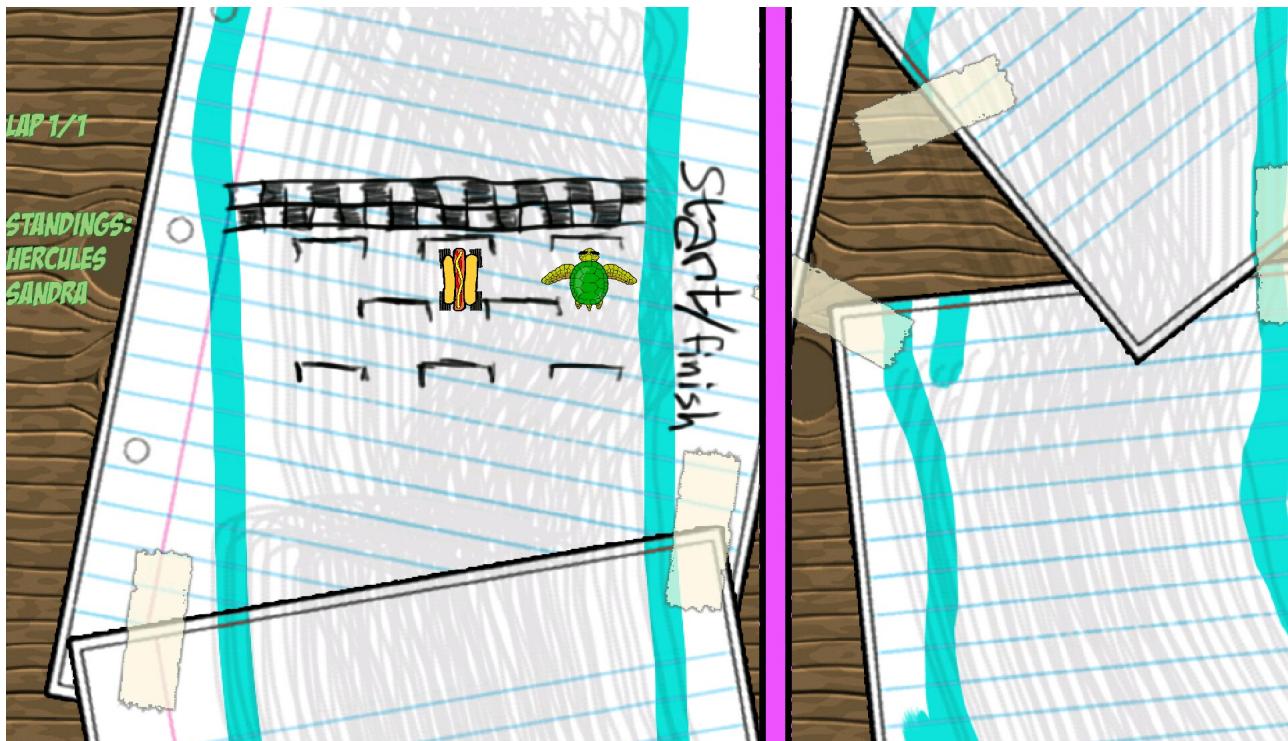
Efter att alla filer har laddats visas spelets huvudmeny, se figur 5.4. Här har användaren möjlighet att välja att starta en spelomgång eller att avsluta spelet. Dessutom visas en lista över samtliga anslutna spelare.



Figur 5.4: Spelets huvudmeny

När en bana har valts startas själva spelet, se figur 5.5. Det skickas då ett paket till samtliga anslutna enheter vilket säger åt dem att visa spelkontrollscenen som visas i figur 5.6. I systemets och spelets huvudmeny visas systemets navigationsscén. Det skickas även ett paket som meddelar klienterna att de ska skicka data tillbaka till servern och spelet. Det är denna data som behandlas i metoden *handleDataFromClients()* och kommer att styra bilarna. Mer om detta i avsnitt 4.6.3, om styrning.

För att förtydliga vilken bil som tillhör vilken användare zoomas kameran in på varje bil i början av varje runda. Det namn som användaren skrivit in på sin kontroll visas då vid den bil användaren ska styra.



Figur 5.5: Spelet

När spelet är över, det vill säga då en vinnare har utsetts, visas en resultatskärm. I denna illustreras hur de olika spelarna presterade. Användaren får även möjlighet att starta en ny spelomgång eller gå tillbaka till menyn. För en överblick av de olika scenerna i spelstrukturen, se figur 4.3.

### 5.2.2 Spelkontroll

Användaren styr som tidigare nämnt sin bil med hjälp av sin egen telefon eller surfplatta. Som nämndes ovan gasar eller backar användaren genom att trycka på respektive knapp, se spelkontrollens grafiska gränssnitt i figur 5.6. Själva styrningen sköts med hjälp av enhetens rotation likt en riktig bilratt. Pausknappen i mitten pausar spelet och visar en pausmeny på servern där användarna kan välja att lämna spelet eller fortsätta spela.



Figur 5.6: Spelkontrollens grafiska gränssnitt

### 5.2.3 Tidsfördröjning

Tidsfördröjningen för spelet upplevdes i slutet av *sprint* 4 som för stor enligt kunden. Den uppmättes som beskrivet i avsnitt 4.7.1. Koden analyserades sedan och det upptäcktes att det vid ett par tillfällen skrevs ut kommentarer i utvecklarförstret för felsökningssyften. Då dessa togs bort minskade tidsfördröjningen vilket gjorde den accepterbar för kundens krav. Tidsfördröjningen mättes även vid lokalt nätverk genom separat router, *wi-fi*, och ett eget nätverk, *hotspot*. Se en sammanställning av tidsfördröjningen i tabell 5.1.

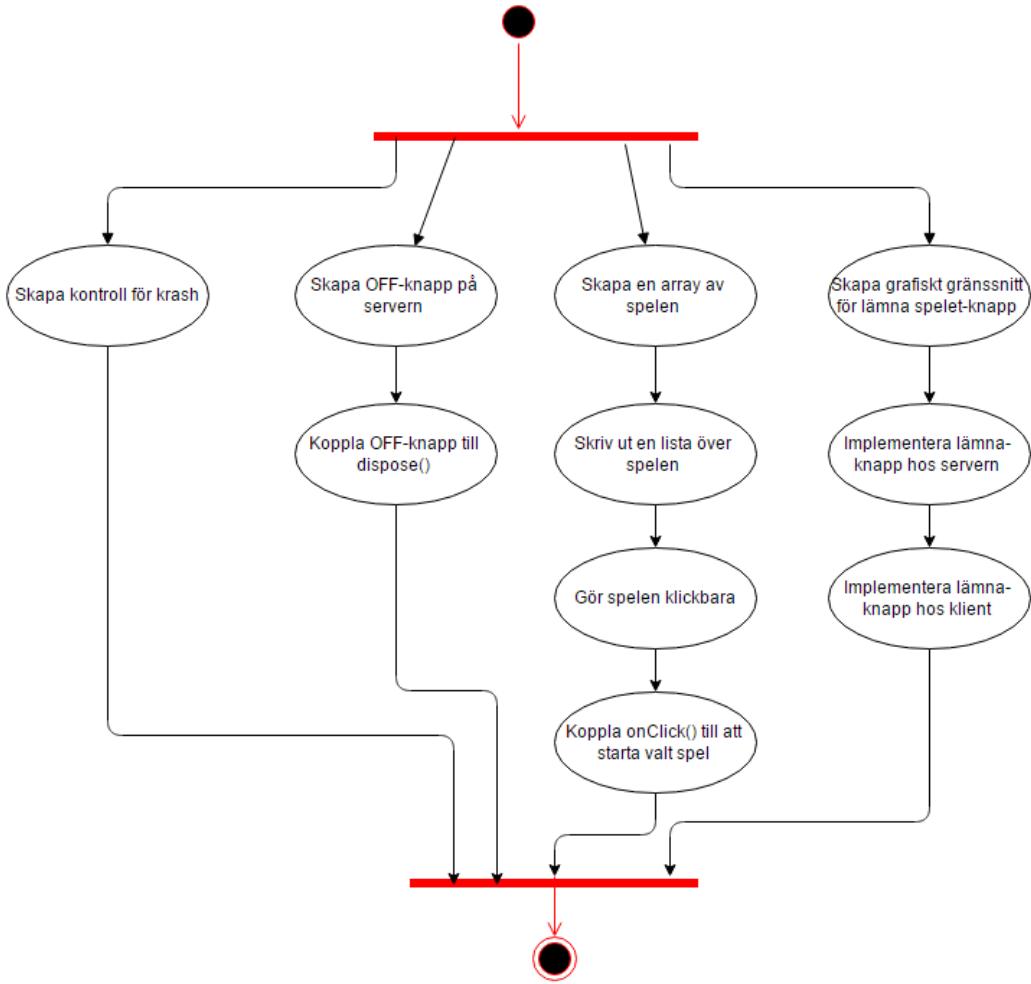
Version	Tidsfördröjning [ms]
Original, med kommentarer	350
Utan kommentarer, på <i>hotspot</i>	80
Utan kommentarer, på <i>wifi</i>	160

Tabell 5.1: Sammanställning av uppmätt tidsfördröjning

## 5.3 Resulterande arbetsmetodik

Att arbeta *agilt* med *Scrum* innebar att stora delar av arbetsplaneringen var flexibel, vilket resulterade i att gruppens medlemmar kunde testa sig fram i arbetsmetodik. *Scrum* har varit en stabil grundstruktur för arbetet, där mycket information och tips på vägen finns att hitta på nätet. I de första *sprintarna* var planeringsmötena kortare och *stories* och *tasks* var mer generella. Slutsatsen som drogs var dock att de uppgifter som var bäst specificerade var lättast att ta sig an och ju fler specificerade och små uppgifter som fanns desto mindre risk fanns för resursslöseri, i form av att någon gruppmedlem inte hade något att göra eller gjorde samma sak som någon annan. De längsta planeringsmötena resulterade i de mest lyckade *sprintarna*.

Genom att implementera *Scrum* kunde dessutom flödeskartor på kritiska vägar under *sprintarna* ritas ut, se ett exempel i figur 5.7 och tidsanvändningen blev mer effektiv.



Figur 5.7: Flödeskarta med uppgifter för en *story* under tredje *sprinten*

Arbetet startades med en planeringsfas, *sprint noll*. Den upplevdes dock som för lång då en del av det tekniska arbetet startades redan under den. En diskussion hur en optimal *sprint noll* skulle utförts finns i diskussionen, kapitel 6.1.

Gruppen anser att de dagliga stående mötena har varit viktiga för gemenskapen och känslan av att all kod är gemensam. De har även gett möjlighet att uppdatera samtliga gruppmedlemmar om någonting utförts utanför arbetstid. Gemenskapen har varit god och gruppen anser att en viktig aspekt har varit att arbeta nära varandra. Att sitta och arbeta tillsammans i samma rum möjliggjorde optimalt kompetensutbyte genom diskussioner direkt då problem uppstod.

Arbetsfördelningen som uppstod tack vare utdelat huvudansvar av olika områden var till stor hjälp då en specifik uppgift skulle utföras, exempelvis bokning av kundmöte eller anordnande av *walkthroughs*. Genom att dela ut specifika ansvarsområden undveks att en specifik gruppmedlem ofrivilligt gavs för mycket eller för lite ansvar. Vid specifika problem visste gruppen vem de skulle vända sig till, vilket effekтивiserade arbetet och motverkade kaos. Dock var vissa ansvarsområden för vagt beskrivna till en början vilket skapade förvirring. Dessa blev därför omformulerade efter hand.

De tekniska rollerna, exempelvis nätverksansvarig eller spelbaneansvarig, har flyttats runt för att motverka att viss kod tillhör en viss person. Detta har gjort att samtliga gruppmedlemmar har en god förståelse för större delen av koden och relativt snabbt skapade förståelse för nyskrivna funktioner.

En diskussion kring för- och nackdelar med ämnesuppdelning finns i kapitel 6.1. På grund av olika intresse förekom dock specialiseringar, då exempelvis en gruppmedlem intresserade sig extra mycket för grafiken och därav hade mer kunskap inom det.

# Kapitel 6

## Analys och diskussion

I detta kapitel kommer det utförda arbetet att diskuteras och analyseras. Både arbetet i gruppen och implementationen har haft stor inverkan på det färdiga resultatet och behandlas i separata delar. En diskussion har förts under projektets gång kring en vidareutveckling av systemet, samt vad som skulle kunna göras efter projektets slut.

### 6.1 Utvecklingsmetodik

Ingen av gruppens medlemmar hade arbetat *agilt* med metoden *Scrum* före projektets start, vilket gjorde att det var mycket information att ta in vid projektets början. Gruppen hade problem med att komma igång med arbetet och med att avgöra vad en eventuell *sprint* noll skulle innehålla. En debatt kring huruvida *sprint* noll verkligen tillhör metoden *Scrum* eller ej förs generellt, läs exempelvis debattartiklar av Prakash [21] och Moskalenko [22]. Gruppen valde till slut att starta med *sprint* noll utan *story* och *tasks* för att skaffa sig kunskap om hur *Scrum* skulle användas på bästa sätt innan arbetet satte igång på riktigt. I efterhand skulle detta arbetet lika gärna ha kunnat gjorts under första *sprinten*. Ett mer erfaret arbetslag hade exempelvis kunnat starta arbetet med en *sprint* enbart för efterforskningar med uppgifter som ”läs anvisad litteratur” och ”håll intervjuer med tio eventuella testanvändare av systemet”.

Sammanfattat anses *Scrum* haft en positiv inverkan på projektet, då det gett gruppen konkreta metoder att arbeta med. Ett exempel på en situation då *Scrum* varit till stor hjälp var då gruppmedlemmar var sjuka eller befann sig på annan ort över dagen. På grund av gruppens *sprintbacklogg* kunde gruppmedlemmarna enkelt ha kontroll över att de inte åtog sig samma uppgifter.

När gruppen diskuterade vilken person som skulle vara produktägare föredrog medlemmarna att produktägarens uppgifter skulle hanteras gemensamt. Det ledde till att alla medlemmar deltog i beslut om vilken *story* som skulle prioriteras under kommande *sprint* vid varje *sprintplanering*.

Det diskuterades till en början huruvida gruppen skulle delas in i ämnesområden. Det skulle medföra att gruppmedlemmarna kunde specialisera sig inom ett visst område och se till att projektet höll god kvalitet inom det. Under *sprint* noll skapades ämnesområden som användes på detta sätt för att samla in så mycket information som möjligt inför resten av projektet. För att kunna arbeta i ett projekt strikt uppdelat i ämnesområden skulle det krävas någon övergripande ansvarig som såg till att samtliga delar sammankopplades, något som gruppen insåg under samma *sprint*. Då gruppen inte utsett en person med huvudansvar splittrades kompetenserna snabbt och kommunikationen mellan ämnesområdena blev fattig.

Det diskuterades därför inför första *sprinten* om detta uppdrag skulle tas vidare eller ändras helt. En klar fördel med ämnesområden är att gruppen tillsammans kan nå en högre grad av kompetens inom

respektive område. En högre grad kompetens skulle kunna spara tid för gruppen, då varje enskild gruppmedlem inte behöver lägga tid på att lära sig övriga ämnesområden. Nackdelarna ligger både i integrationen av systemet samt i gruppmedlemmarnas egna intresse av kompetens. För att det färdiga resultatet ska vara integrerat mellan grafik, nätverk och spelstruktur måste utvecklarna av samtliga delar kommunicera. Annars riskeras att en ämnesgrupp oavsiktligt sätter käppar i hjulet för en annan. Vid integrationsproblem bör kompromisser ske vilket kan vara svårt att hantera för en oerfaren grupp utan övergripande kunskap inom alla ämnen. I gruppmedlemmarnas intresse låg att efter projektets avslut kunna redogöra för så stor del av systemutvecklingen som möjligt. Majoriteten av gruppen prioriterade alltså en lägre men bredare kompetens över en högre kompetens inom ett specifikt ämne. Alltså valdes ämnesgrupper bort.

## 6.2 Arbetsfördelning

De ansvarsområden som delades ut i början av projektet har uppfattats som nödvändiga och ungefär lika belastande. I och med att till exempel ingående modellering och ett användartest prioriterats bort har vissa ansvarsområdagens belastning minskat. Varje medlem har skött sina uppgifter självständigt vilket har underlättat för de andra gruppmedlemmarna.

Gruppen känner att alla har bidragit med den mängd arbete som förväntats och att efter att tasks hade delats upp bättre än i första sprinten har alla alltid arbetat med något och det har alltid funnits något att göra. Vissa medlemmar har jobbat utanför planerad tid och på så vis bidragit med mer än andra. Mer information om hur olika medlemmar arbetat finns i bilaga A.

## 6.3 Användartester

Få användarter har gjorts i det här projektet. Det första testet gjordes tidigt i utvecklingen så att gruppen fick en uppfattning om vilka av gruppens idéer som passade målgruppen. Efter det ville gruppen fokusera på kodutvecklingen och bara testa efter att alla idéer var förverkligade eftersom det kändes onödigt att lägga tid på tester som skulle ge kommentarer om sådant som ändå skulle implementeras. Detta var ett orutinerat val då testning ska ske tidigt och ofta. Det gör att utvecklarna hela tiden vet vad intressenter vill ha och tycker om olika idéer. Dessutom, om det är skisser som testas istället för färdiga program kan val göras att ändra eller kasta bort idéer om något inte visar sig vara uppskattat utan att mycket tid på läggs på att utveckla dem först. En slutsats gruppen har dragit från detta är att gruppen borde ha skissat på grafik och testat den redan i början av projektet även om funktionalitet inte fanns. Det hade gjort att gruppen kunde justera grafik utan att lägga tid på kodskrivande och även få förslag på funktionalitet.

## 6.4 Ramverk

I början av projektet diskuterades för- och nackdelar med att skriva all kod specifikt för Android och att använda sig av ett eller flera ramverk som möjligtvis skulle kunna underlätta arbetet. Den största nackdelen med att använda ett ramverk ansågs vara att systemet blir mer begränsat och att det kan bli svårt att bygga vidare. Det var problematiskt då ett av målen var att koden skulle kunna tas över av andra utvecklare. Fördelen med ett ramverk var att kunna få tillgång till färdiga klasser som kan hjälpa till med exempelvis nätverk och grafik i spelet. Då gruppen hade en begränsad tidsperiod på sig att slutföra projektet ansågs det vara nödvändigt att ta hjälp av externa ramverk för att kunna uppnå slutmålen.

Efter att ha arbetat med ramverken har gruppen funnit att de har underlättat arbetet väsentligt. På grund av de klasser som ramverken har bidragit med har fokus kunnat läggas på de klasser som är unika för systemet och spelet. Saker gruppen inte behövt lägga tid på tack vare ramverken var att läsa på om och skriva kod för nätverkskommunikation på en djupare nivå och utveckla klasser för scener, fysik, knappar och bilder vilket har sparat mycket tid. Om gruppen skulle göra om projektet, även utan tidsbegränsning, skulle ramverken användas eftersom gruppen haft stor hjälp av dem.

## 6.5 Analys av tidsfördröjning

Genom att ta bort kommentarer i koden förbättrades tidsfördröjningen drastiskt. Det är förståeligt då det tar tid för programmet att skriva ut text för varje bild. Spelet körs i 60 *frames per second* och alltså skedde 60 utskrifter varje sekund. Att det var skillnad på tidsfördröjningen på olika nätverk är också till stor del förståeligt, då det lokala nätverket användes av flera andra datorer under tiden och passerade genom en extern router. *Hotspot*:ens uppkoppling var enbart tillägnad det aktiva spelet vilket undvek eventuella hinder i anslutningen.

Felavläsning i mätningen kan ha skett på grund av behandlingen av sändningen av paket med *UDP*. Då vissa paket som skickas via *UDP* kan försvinna utan återkoppling finns risk att en del av tidsfördröjningen beror på paket som inte kommit fram. Dock skulle det snarare ge upphov till en ”ryckig” rörelse eftersom nästkommande paket inte väntar in de borttappade. Huruvida tidsfördröjningen som uppmäts beror på protokollöverföringen eller andra aspekter är okänt för gruppen och vid vidareutveckling av projektet skulle detta kunna undersökas närmare.

## 6.6 Vidareutveckling av projektet

### 6.6.1 Nätverk

För tillfället är det inte möjligt att se alla aktiva servrar som finns i nätverket. Klienten kommer koppla upp sig till den första servern som den får ett *broadcast*-paket ifrån. En möjlig utveckling av systemet skulle vara att klienten samlar alla meddelanden och listar samtliga servrar. Därefter väljer användaren vilken server som ska anslutas till.

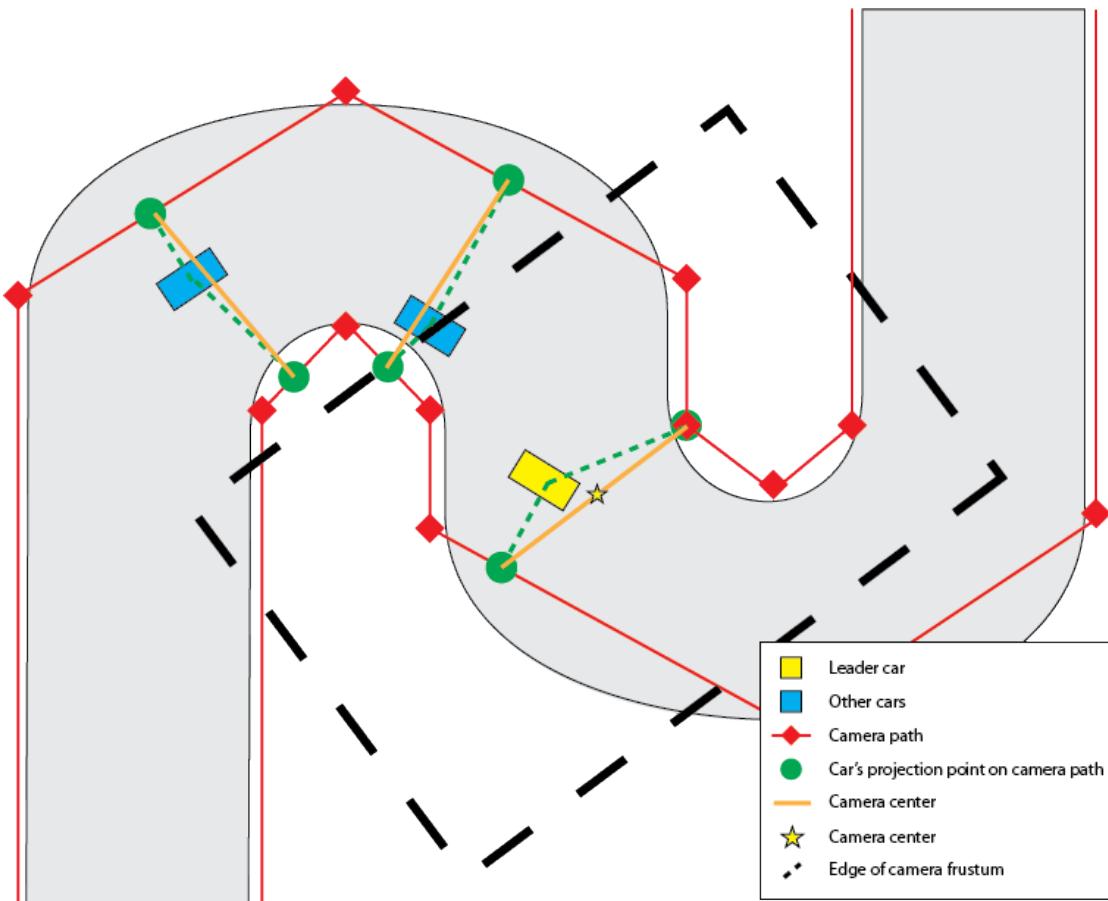
En annan teknik som bör bytas ut i ett senare skede är *broadcast* och istället använda *Multicast*. Alla enheter som vill skicka och ta emot paket måste ingå i en *Multicast*-grupp. När enheten gått med så är det möjligt att skicka och få paket till skillnad från *broadcast*. *Multicast* effektiviseras även genom att den bara skickar meddelandet till de specifika enheterna som ingår i gruppen.

Om spelet skulle nå en större skala aktiva användare kan det vara en idé att ha en dedikerad server som hanterar alla servrar som finns uppe. Då är det möjligt för klienten att direkt kunna hitta den servern den är ute efter.

### 6.6.2 Spelet

För tillfället saknas en del funktionalitet i spelet som gruppen inte hann med att implementera under projektets gång. Fler spelinställningar såsom val av bil och färg, antal spelrundor och hur många varv en spelomgång ska bestå av var tänkt att införas. Gruppen ville även skapa möjligheten att plocka upp diverse vapen längs banan för att kunna skjuta dessa på motståndarna eller olika typer av temporära upgraderingar. Det skulle ge speldynamiken ytterligare en dimension vilken kan göra det mer underhållande.

Spelkamerans förflyttningar och rotationer är aningen ojämna och skulle kunna förbättras. T.ex. skulle en yttre vägbana kunna införas. Bilarna skulle projicera sin position även på denna. Det skulle då gå att få en linje mellan de två projektionspunkterna. Kamerans position skulle då ligga i mitten av denna linje och ha samma rotation som den. Se figur 6.1.



Figur 6.1: Dubbla vägbanor för kameran

### 6.6.3 Tredjepartsutveckling av spel

Ett mål har varit att andra utvecklare enkelt ska kunna skapa sina egna spel till Varsom-systemet. Grundtanken var att användarna via Varsom-menyn kunde ladda in spelfiler som finns på enheten. För tillfället måste de nya spelen dock läggas in via systemets källkod.

I dagsläget är det klienten som har sparat informationen om de olika scenerna som den ska visa. Det betyder att om ett nytt spel till systemet har utvecklats måste både serverapplikationen och klientapplikationen uppdateras. Ett önskemål är att servern har information om hur scenerna klienten ska visa ser ut och skickar det till klienten. Om utvecklare inte vill skapa en egen scen ska **Varsom**-systemets navigationsscén användas.

## 6.7 Arbetet i ett vidare sammanhang

Tekniken blir mer och mer mobil, så ett system som är helt trådlöst och inte bundet till ett uttag eller en stationär låda är helt rätt i tiden. En möjlig framtid är att denna typ av mobila system utkonkurrerar systemen med en ansluten konsol, exempelvis **Xbox** och **PlayStation**.

Förutom spel finns fler framtida möjligheter för ett system som Varsom. Ett av de trendigaste ämnena inom teknik idag är *Internet of things*, det vill säga ickebemannade saker som styrs genom internet. Läs exempelvis om hur finansföretaget **Goldman Sachs** använder *Internet of Things* [23]. Genom att applicera systemets uppbyggnad på detta sätt skulle exempelvis ett system för att styra elektronik inom hemmet kunna utvecklas. En användare kan ha en mobil enhet kopplad till en diskmaskin, tvättmaskin, torktumlare eller kaffekokare och använda sin egen mobiltelefon uppkopplad mot sitt hemnätverk för att styra dessa maskiner.

# Kapitel 7

## Slutsatser

### 7.1 Slutsatser utefter frågeställningar

- **Frågeställning 1:** *Kommer en TCP-anslutning att vara tillräcklig med avseende på tidsfördröjning och överföringsbelastning?*

En anslutning via *TCP* är tillräcklig för att hantera skickande och mottagande separat. Protokollet kan dock medföra en tidsfördröjning eftersom återkoppling ingår i *TCP*. Den medförde att onödiga resurser lades på irrelevant återkoppling. För att minimera detta problemet har ett ytterligare protokoll, *UDP*, använts som komplement. I *UDP* finns ingen återkoppling. Det gör att meddelanden kan skickas kontinuerligt utan att skadas av mindre paketförluster och kan skickas med kortare mellanrum.

Med avseende på tid är gruppen medveten om att en viss tidsfördröjning sker, men är osäkra på var i systemet det problemet uppkommer. Det kan alltså hända att *TCP*-anslutningen inte är tillräcklig för att överföra paket i realtid.

- **Frågeställning 2:** *Kan det externa ramverket libGDX underlätta utvecklingen av projektet, och om inte, behöver egna motsvarande klasser som hanterar kamera och shading skrivas från grunden?*

Det externa ramverket libGDX var av stor användning vid utvecklandet av systemet, främst med avseende på kameran. Det beslutades därför att libGDX skulle användas.

- **Frågeställning 3:** *Androidutveckling sker på olika API-nivåer, där en högre nivå representerar en nyare version. Högre nivåer kan köra program som är utvecklade på lägre nivå, men inte tvärtom. En avvägning behöver göras mellan förhållandet användbar teknik och kompatibilitet till majoriteten av mobila enheter. Hur ska denna göras?*

Då libGDX är oberoende av **Android Studios** inställningar för API har ingen hänsyn tagits till kompatibilitet hos olika versioner av **Android**. Dock är libGDX ännu ej kompatibelt till **iOS** för enheter utan utvecklarlicens, vilket ej tagits hänsyn till.

- **Frågeställning 4:** *Är det tillräckligt att använda värden från accelerometern på kontrollen för att styra bilen, eller behövs någon mer avancerad metod, exempelvis sensor fusion?*

Accelerometern ger tillräcklig indikation för att styra. Dock är det möjligt att den uppmätta tidsfördröjningen beror på att det är inte tillräckligt att använda enbart en sensor utan att en kombination av sensordata behövs. *Sensor fusion* kan vara en möjlig lösning på problemet men det har inte hunnit undersökas.

## 7.2 Slutsatser utefter mål

- **Mål 1:** *Ett färdigt system för spel på en surfplatta med flera mobiltelefoner som kontroller ska utvecklas där åtta användare ska kunna vara aktiva samtidigt.*

Ett färdigt system för spel på surfplatta med mobiltelefon som kontroll där åtta spelare kan vara aktiva samtidigt har utvecklats. Fler användare kan vara aktiva i systemet utanför spelet, men spelet är i nuläget begränsat till åtta spelare.

- **Mål 2:** *Systemets kod ska vara läsbar och välstrukturerad.*

Systemets kod är välstrukturerad, men saknar på flera ställen ordentlig dokumentation i form av Javadoc.

- **Mål 3:** *Systemet och ett kompletterande spel ska kunna släppas på Google Play vid projektets slut.*

Systemet och spelet ligger sedan 2015-06-03 uppe som två separata applikationer på **Google Play** och kan laddas ner gratis.

- **Mål 4:** *Serverenheten ska kunna kopplas upp mot en TV för att lättare visa spelet och underlätta för flera spelare att spela samtidigt.*

För koppling från surfplatta till TV krävs att surfplattan är utrustad med en HDMI-ingång. Det finns inga andra tillräckligt stabila överföringsmetoder på marknaden idag. **Android TV** kommer vara ett bra alternativ att använda. Dock säljs det inte i större skala i dagens läge.

# Litteraturförteckning

- [1] *Mobile/Tablet Operating System Market Share*, hämtad: 2015-05-13  
<http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1>
- [2] Shari Lawrence Pfleeger och Joanne M. Atlee, *Software Engineering, Fourth Edition, International Edition*, Pearson 2010
- [3] Saurabh Mittal, *Agile and its value - Where is it going wrong?*, Scrum Alliance, 2013-09-11, hämtad: 2015-05-10  
[www.scrumalliance.org/community/articles/2013/september/agile-its-value-part-1.aspx](http://www.scrumalliance.org/community/articles/2013/september/agile-its-value-part-1.aspx)
- [4] Michael Bloch, Sven Blumberg och Jürgen Laartz, *Delivering large-scale IT projects on time, on budget, and on value*, McKinsey&Company, 2012-10, hämtad: 2015-05-10  
[www.mckinsey.com/insights/business\\_technology/delivering\\_large-scale\\_it\\_projects\\_on\\_time\\_on\\_budget\\_and\\_on\\_value](http://www.mckinsey.com/insights/business_technology/delivering_large-scale_it_projects_on_time_on_budget_and_on_value)
- [5] Okänd författare, *Couch+*, Couch+, okänt publiceringsdatum, hämtad: 2015-06-02  
[www.couchplus.com](http://www.couchplus.com)
- [6] Okänd författare, *Fjärrspel*, Sony, okänt publiceringsdatum, hämtad: 2015-06-02  
<http://www.playstation.com/sv-se/explore/ps4/features/remote-play>
- [7] Rakiv Ahmed och Jonas Aule, *An evaluation of the framework Libgdx when developing a game prototype for Android devices*, KTH - Royal Institute of Technology, 2011-06, hämtad: 2015-06-01  
<http://www.csc.kth.se/utbildning/kth/kurser/DD143X/dkand11/Group7Henrik/Rakiv.Ahmed.Jonas.Aule.report.pdf>
- [8] Kjetil Mehl, *Peer-to-peer Game State Replication*, Norwegian University of Science and Technology, 2013-04, hämtad: 2015-06-01  
[http://brage.bibsys.no/xmlui/bitstream/handle/11250/253211/644190\\_FULLTEXT01.pdf?sequence=1&isAllowed=y](http://brage.bibsys.no/xmlui/bitstream/handle/11250/253211/644190_FULLTEXT01.pdf?sequence=1&isAllowed=y)
- [9] Nathan Sweet, *Kryonet*, EsotericSoftware, 2015-05-10, hämtad: 2015-06-02  
[www.github.com/EsotericSoftware/kryonet](http://www.github.com/EsotericSoftware/kryonet)
- [10] Okänd författare, *Esoteric Software*, Kickstarter, 2014-09, hämtad: 2015-06-04  
<https://www.kickstarter.com/profile/esotericsoftware/created>
- [11] Dermetfan Homepage, hämtad: 2015-06-02  
[www.dermetfan.net](http://www.dermetfan.net)

- [12] Dermetfan Youtube-channel, hämtad: 2015-06-04  
[www.youtube.com/user/dermetfan](http://www.youtube.com/user/dermetfan)
- [13] Ken Schwaber och Jeff Sutherland, *The Scrum Guide, The definitive guide to Scrum: The rules of the game*, Juli 2013, hämtad: 2015-02-15  
[www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf](http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf)
- [14] Okänd författare, *Sprint planning meeting*, Scrum Methology, 2010-08-20, hämtad: 2015-02-15  
[www.scrummethodology.com/scrum-meetings](http://www.scrummethodology.com/scrum-meetings)
- [15] George Baravdish, *Linjär algebra TNA002*, 2012
- [16] Okänd författare, *Dokumentation för klassen Math i Java*, Oracle, okänt publiceringsdatum, hämtad: 2015-05-13  
[www.docs.oracle.com/javase/7/docs/api/java/lang/Math.html#asin\(double\)](http://www.docs.oracle.com/javase/7/docs/api/java/lang/Math.html#asin(double))
- [17] Okänd författare, *Singleton*, Singleton Pattern, onkänt publikationsdatum, hämtad: 2015-06-04  
<http://www.oodesign.com/singleton-pattern.html>
- [18] Okänd författare, *Creative Commons*, About Creative Commons, okänt publikationsdatum, hämtad: 2015-06-04  
[www.creativecommons.org/about](http://www.creativecommons.org/about)
- [19] Okänd författare, *Google Java Style*, Google, 2014-03-21, hämtad: 2015-06-03  
<https://google-styleguide.googlecode.com/svn/trunk/javaguide.html>
- [20] Cerisegruppen, *Parprogrammering*, 2014-07, hämtad: 2015-05-13  
[www.csc.kth.se/tcs/projects/cerise/parprogrammering/](http://www.csc.kth.se/tcs/projects/cerise/parprogrammering/)
- [21] Anurag Prakash, *What is Sprint Zero?*, Scrum Alliance, 2013-09-04, hämtad: 2015-05-13  
[www.scrumalliance.org/community/articles/2013/september/what-is-sprint-zero](http://www.scrumalliance.org/community/articles/2013/september/what-is-sprint-zero)
- [22] Vyacheslav Moskalenko, *The Power of Sprint Zero*, Luxoft, 2014-03-28, hämtad: 2015-05-13
- [23] Okänd författare, *The Internet of Things - The Next Mega-Trend*, Goldman Sachs, 2014-09, hämtad: 2015-05-13  
[www.goldmansachs.com/our-thinking/outlook/internet-of-things/](http://www.goldmansachs.com/our-thinking/outlook/internet-of-things/)

## Bilaga A

### Arbetsbelastning och statistik

Figur A.1 och A.2 nedan visar statistiken över förändringar gjorda i kodfilerna som laddats upp på versionshanteringsverktyget **GitHub** både för gruppen sammanlagt och uppdelat per person. Den övre, gröna grafen representerar hela gruppens sammanlagda arbete och de undre orange graferna representerar personprestationer. Statistiken ger dock inte en exakt bild över hur mycket varje person bidragit med i kodväg, då parprogrammering enbart visas som bidrag av en person samt de tillägg som gjorts men valts att inte användas i slutgiltiga versionen inte syns. En förklaring över vilken graf som tillhör vilken gruppmedlem utefter användarnamn finns i tabell A.1.

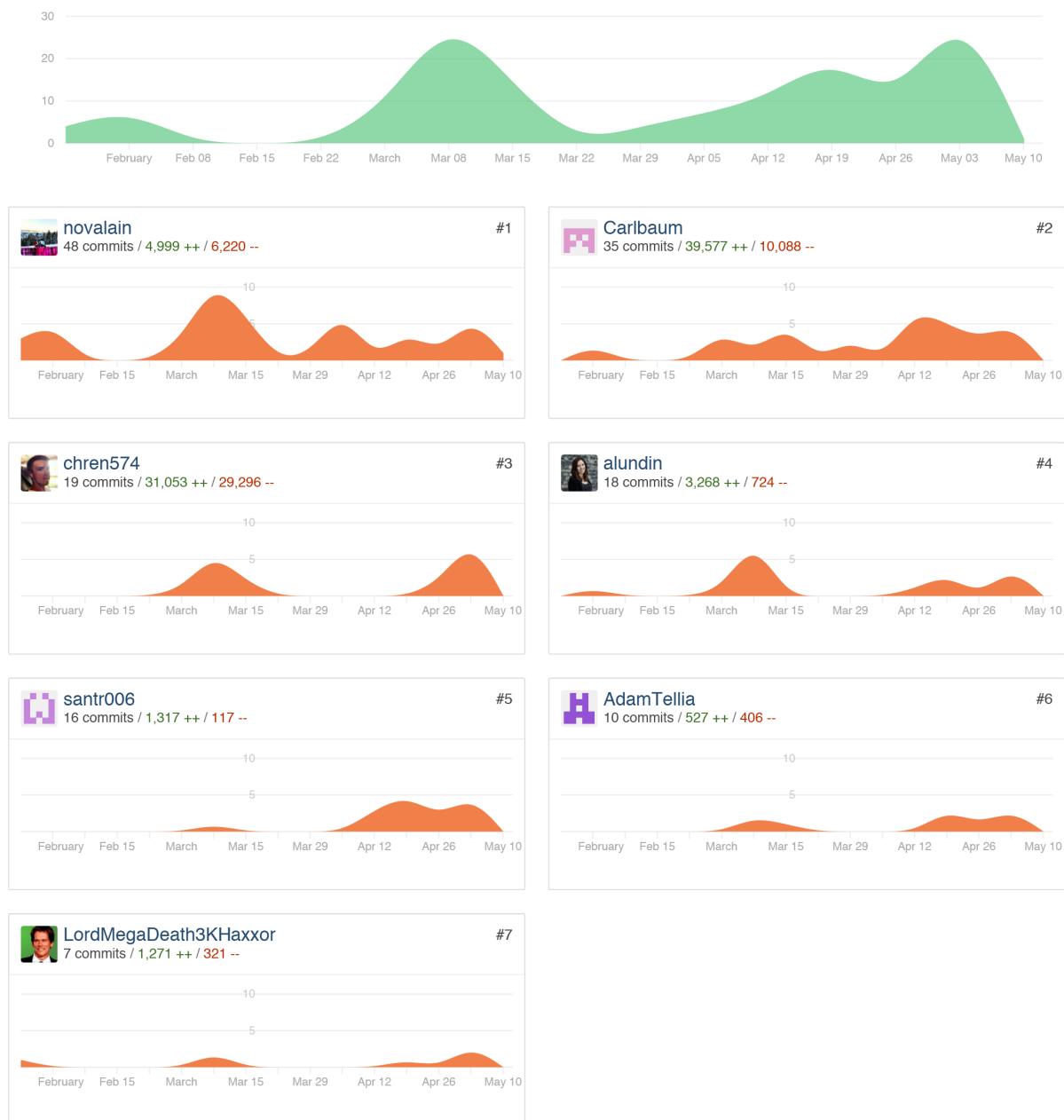
En närmare beskrivning av vad var och en av gruppmedlemmarna bidragit med finns i bilaga A.3. Gruppen anser att arbetsbelasningen varit jämnt fördelad mellan gruppmedlemmarna och upplever inte att någon hamnat utanför projektets arbete.

## A.1 Statistik över kodbidrag

Jan 25, 2015 – May 12, 2015

Contributions to Varsom-System, excluding merge commits

Contributions: **Commits** ▾



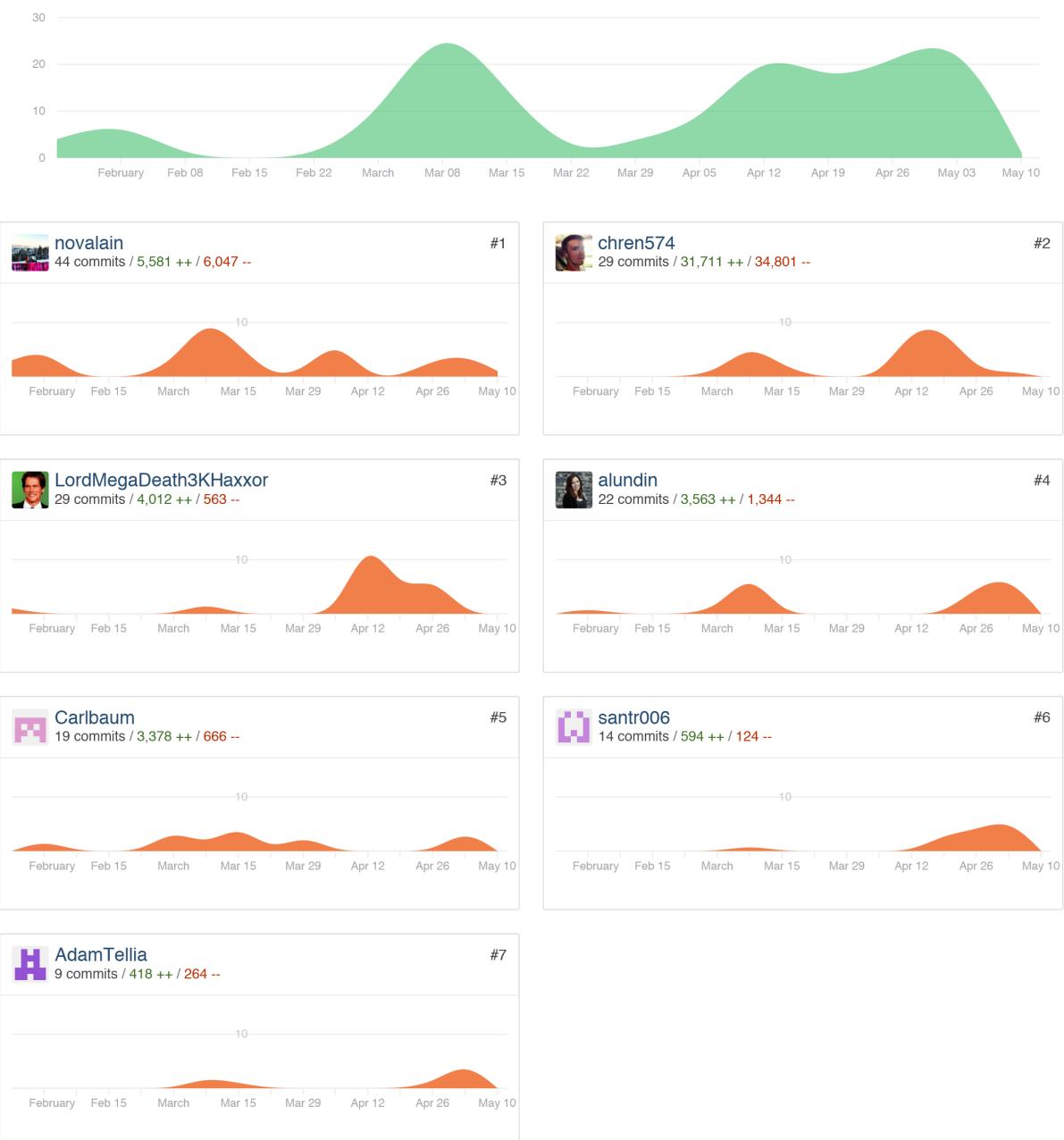
Figur A.1: Statistik över gjorda ändringar i de kodfiler som innehållas av serverapplikationen

## A.1. STATISTIK ÖVER KODBIDRAG BILAGA A. ARBETSBELASTNING OCH STATISTIK

Jan 25, 2015 – May 12, 2015

Contributions to ControllerApp, excluding merge commits

Contributions: **Commits** ▾



Figur A.2: Statistik över gjorda ändringar i de kodfiler som innehållas av klientapplikationen

## A.2 Alias på GitHub

Användarnamn	Person
AdamTellia	Adam Tellia
alundin	Alice Lundin
Carlbaum	Oskar Carlbaum
chren574	Christoffer Engelbrektsson
LordMegaDeath3KHaxxor	Marcus Lilja
novalain	Michael Novén
santr006	Sandra Tråvén

Tabell A.1: Användarnamn på GitHub för tolkning av statistik

## A.3 Personliga bidrag

### Adam Tellia

Adam var dokumentationsansvarig och sekreterare. I början av projektet arbetade han med nätverkskommunikationen och efter att den delen blev avklarad har hans fokus legat på att fixa diverse funktionaliteter som behövts i systemet.

### Alice Lundin

Alice hade ansvarsområde *Scrum Master*. Vid projektets start var Alice en del av den grupp som samlade information om nätverkskommunikation. Hon jobbade senare med spelet och att få objekt att följa en specifik bana, med funktioner för att starta eller lämna spelet och systemet och under sista *sprinten* med att navigera i menyer med hjälp av klientapplikationen.

### Christoffer Engelbrektsson

Christoffer var kommunikationsansvarig. I *sprint* noll så gjordes efterforskning på olika tekniker för att strömma data till TV och vilken typ av nätverksteknik applikationera skulle använda för att kommunicera med varandra. De resterande *sprintarna* lades den mesta tiden på att implementera nätverksfunktionalliteten i applikationerna.

### Marcus Lilja

Marcus har under projektets gång varit modelleringsansvarig. Vid projektets start arbetade Marcus med att avgöra vilka API:er som skulle behövas för att rendera grafik och skapa GUI. Under större delen av projektet jobbade han med med skapandet och strukturen för GUI:system till server- och klientappen samt spelet med hjälp utav libGDX. Arbete har även lagts på att göra spelet responsivt till olika skärmstorlekar och att lägga in ljud. Marcus var även med och utvecklade navigationskontrollen till klientappen.

### Michael Novén

Michael var testansvarig och hade ansvaret över Git och såg till att struktur fanns där, samt hjälpte med diverse problem relaterade till Git och mergekonflikter. Utöver detta har han arbetat med spellogik och det som behövts för tillfället.

### Oskar Carlbäum

Oskar var under projektets gång kodansvarig. Hans fokus har annars legat mest på spellogik men har också skapat den mesta grafiken.

### Sandra Tråvén

Sandra har varit kravanalytiker. Under projektets början var Sandra en del av den grupp som samlade information om nätverkskommunikation. Hon hjälpte till med att skapa strukturen för koppling mellan

systemet och spelens och har arbetat mycket med scenerna och kommunikationen som behövts för att klienten ska byta scen beroende på servern.

## Bilaga B

### Användartest och svar

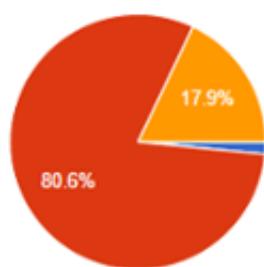
Testet inleddes med en förklarande text:

Hej! Vi är en projektgrupp på Linköpings universitet och håller på att utveckla ett spelsystem vid namn Varsom Games. Med Varsom kan du spela TV-spel med dina vänner var som helst! Det gör du med din mobil som kontroll och surfplatta som skärm. Kopplar du upp plattan till en TV kan alla se vad som händer! Ni behöver inte längre en konsol med tillhörande kontroller för att spela, alla har med sig sin egen kontroll! Vi planerar även att utveckla ett bilspel att spela på systemet.

Kort om spelet: Du och dina vänner kontrollerar era bilar genom att vrida telefonen som en ratt för att svänga samt trycka på skärmen för att gasa. Spelet är i 2D och alla bilar kommer synas på samma skärm. Kameran filmar er uppifrån och följer den spelare som kör först. Om någon hamnar så långt bakom den som ligger först att bilen inte längre syns på skärmen åker den spelaren ut. Den spelare som har sin bil kvar sist på skärmen vinner.

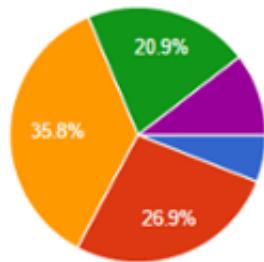
Testets frågor samt svarsammanställning följer i figur **B.1**, **B.2**, **B.3**, **B.4** och **B.5**.

Din ålder



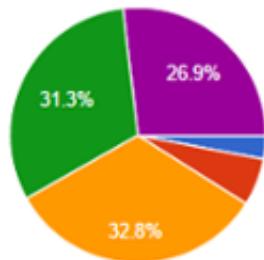
0-15	1	1.5 %
16-25	54	80.6 %
26-40	12	17.9 %
41-supergammal	0	0 %

Hur stor användning skulle du ha för det här systemet?



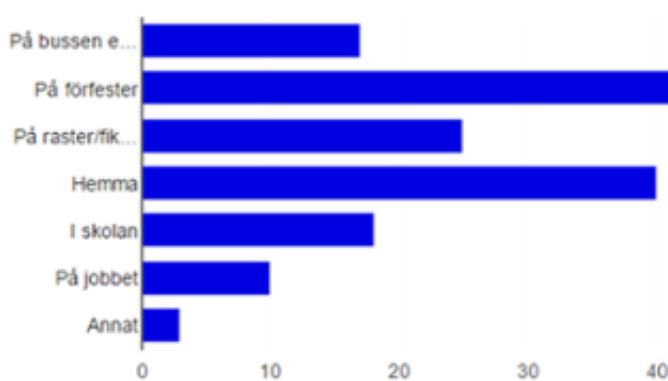
1 - Ingen alls	4	6 %
2	18	26.9 %
3	24	35.8 %
4	14	20.9 %
5 - Precis vad jag vill ha	7	10.4 %

Hur kul låter det här spelet?



1 - Jättetråkigt	2	3 %
2	4	6 %
3	22	32.8 %
4	21	31.3 %
5 - Jättekul	18	26.9 %

I vilka situationer skulle du spela spelet?

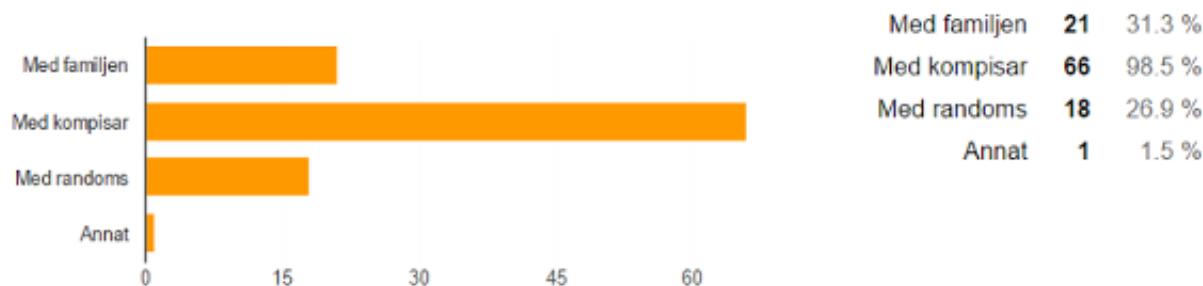


På bussen eller tåget	17	25.4 %
På förfester	43	64.2 %
På raster/fikapausen	25	37.3 %
Hemma	40	59.7 %
I skolan	18	26.9 %
På jobbet	10	14.9 %
Annat	3	4.5 %

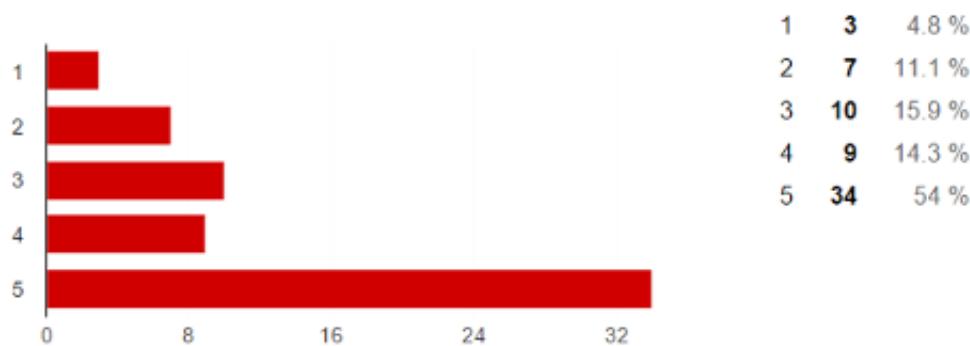
Figur B.1: De första fyra frågorna i användartestet

**Vilka andra situationer?**

Generellt sett pauser
-
Hemma hos adam
i tp404
Hemmaväll. Behöver inte förfest för att man ska umgås med andra i varandras hem.
På ToanII
Senast jag spelade spel var det Quake 3 som gällde, så jag är lite efter här (därav användningen också...)

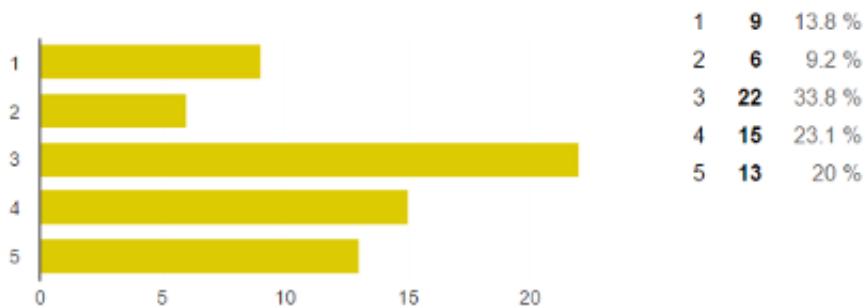
**Med vem skulle du spela spelet?****Med vilka andra?**

-
Alla som har ett intresse för spel
Min sköldpadda
Med fiender.
Vore balst om det var över internet, så att jag kan ha ett spel för mina vänner om jag vill det, och ett spel för randoms
Kanske adam, men jag skulle äga honom

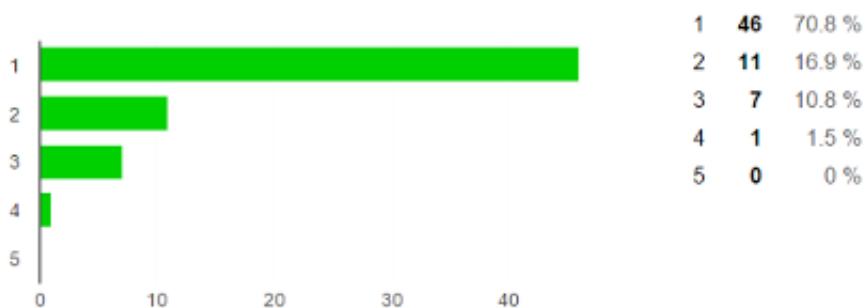
**Vinna [Värdera hur roligt/viktigt det är för dig att dessa funktioner finns med i ett spel.]**

Figur B.2: Fråga 5 till 8 i användartestet

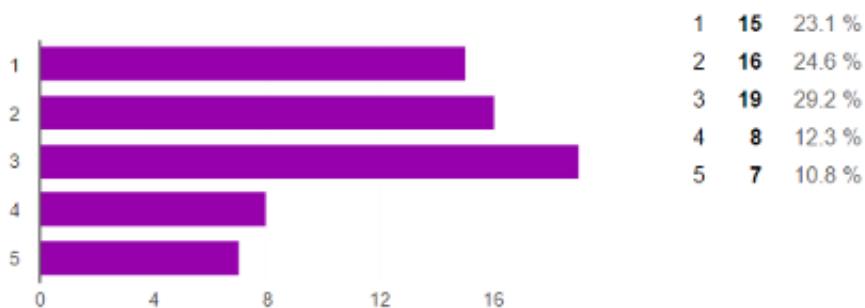
**Se ditt high score [Värdera hur roligt/viktigt det är för dig att dessa funktioner finns med i ett spel.]**



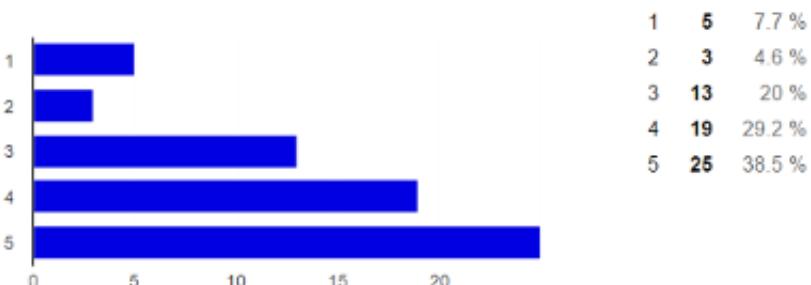
**Skriva ut ett körkort [Värdera hur roligt/viktigt det är för dig att dessa funktioner finns med i ett spel.]**



**Samla achievements [Värdera hur roligt/viktigt det är för dig att dessa funktioner finns med i ett spel.]**

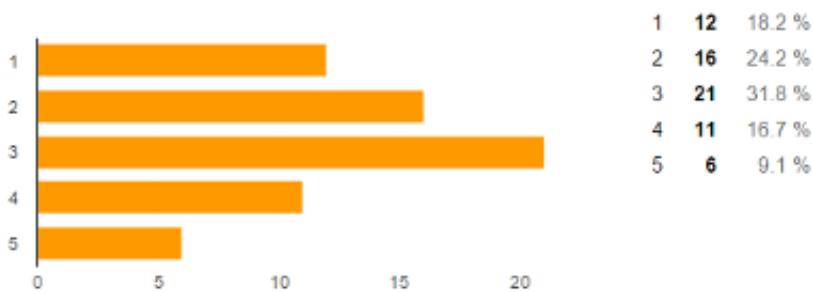


**Sabotera för andra spelare [Värdera hur roligt/viktigt det är för dig att dessa funktioner finns med i ett spel.]**

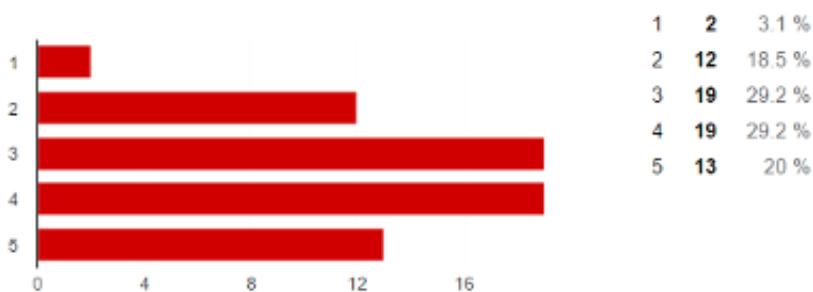


Figur B.3: Fråga 9 till 12 i användartestet

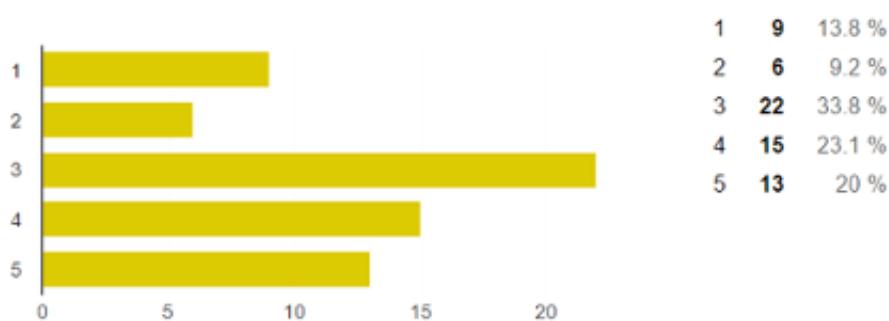
Hjälpa andra spelare [Värdera hur roligt/viktigt det är för dig att dessa funktioner finns med i ett spel.]



Använda strategi [Värdera hur roligt/viktigt det är för dig att dessa funktioner finns med i ett spel.]

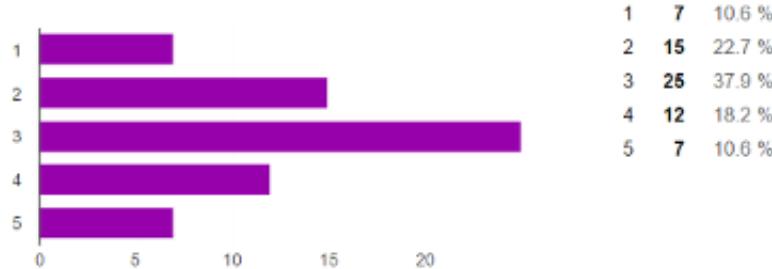


Se ditt high score [Värdera hur roligt/viktigt det är för dig att dessa funktioner finns med i ett spel.]

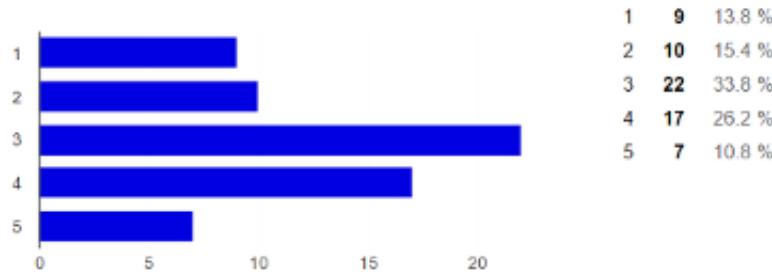


Figur B.4: Fråga 13 till 15 i användartestet

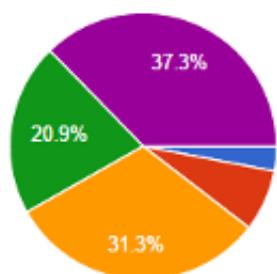
Vara aktiv även efter att du förlorat [Värdera hur roligt/viktigt det är för dig att dessa funktioner finns med i ett spel.]



Skräddarsy ditt eget fordon [Värdera hur roligt/viktigt det är för dig att dessa funktioner finns med i ett spel.]

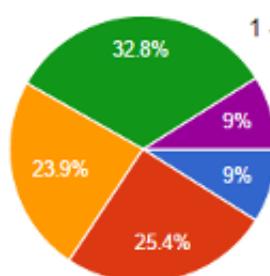


Vad skulle du tycka om man kunde spara sina resultat?



1 - Jätteonödigt	<b>2</b>	3 %
2	<b>5</b>	7.5 %
3	<b>21</b>	31.3 %
4	<b>14</b>	20.9 %
5 - Jättekul	<b>25</b>	37.3 %

Hur stor roll spelar bra grafik för dig?



1 - Ingen roll, jag spelar spel även om grafiken är dålig	<b>6</b>	9 %
2	<b>17</b>	25.4 %
3	<b>16</b>	23.9 %
4	<b>22</b>	32.8 %
5 - Stor roll, annars vill jag inte spela	<b>6</b>	9 %

Figur B.5: Fråga 16 till 19 i användartestet

## Bilaga C

# Pseudokod för projektion på kamerans vägbana

```
/**  
* * PSEUDO CODE - Algorithm..  
* 0.      in Car class, update car (should be done before this function is called)  
* 1.      calculate quadrant of point  
* 2.      project carPos on current line (get the global position of the projected point, pointOnTrack)  
* 3.      check if waypoint reached  
* 3.1     if reached{  
* 3.1.1    calc delta, Delta = distance from pointOnTrack's prev pos, to end of currentLine  
* 3.1.2    add delta to traveledDistance.  
* 3.1.3    update currentLineStartPos,  
* 3.1.4    update pointOnTrack to end of currentLine  
* 3.1.5    update the waypointIndex, make sure it doesn't go out of bounds  
* 3.1.6    update currentLineEndPos,  
* 3.1.7    update currentLine's angle  
* 3.1.8    update currentLineVec  
* 3.1.9    }  
* 3.2     else if NOT reached{  
* 3.2.1    if pointOnTrack is further from currentLine's end point than it was in the previous  
*           update,(car is going the wrong way) {  
* 3.2.1.1    if pointOnTrack has preceded(?) the currentLine  
*             set pointOnTrack to currentLine's start position  
*             calc delta.. delta = distance from previous pointOnTrack to currentLine's start position.  
* 3.2.1.2    }  
* 3.2.1.3    else if pointOnTrack is still on the line {  
*             calc delta.. delta = distance from previous pointOnTrack to current pointOnTrack  
* 3.2.1.4    }  
* 3.2.1.5    subtract delta from traveledDistance  
* 3.2.2    else if pointOnTrack got closer to currentLine's end point or is still  
*             calc delta.. delta = difference of (the distance from previous pOT to currentLineEnd & the  
*             distance from current pOT to currentLineEnd)  
* 3.2.2.1    add delta to traveledDistance  
* 3.2.2.2    }  
* 3.2.2.3    }  
* 4        set prevProjPoint = projectionPoint;  
*  
* @param carPos  
* @return  
*/
```

Figur C.1: Pseudokod för projektion och vägpunktshantering