

Spirometry in young children 2-7

Project

Spiro



Måling af vejrtrækningskapacitet hos mindre børn, ved brug af biofeedback stimulering.

Respiratory measurements of young children with asthma, cystic fibrosis and other lung diseases, using visual feedback stimulation.

Michael Kell Jensen s051154

Vejledere: Finn Gustafsson, Steen Silberg

DTU IT-Diploma Final exam. IMM-B.Eng-2013-3

Abstract

Dansk - Danish

Formålet med projektet er at indfange måledata om lungefunktionen hos astma- og Lunge patienter, målrettet til børn i alderen 2-7 år.

At forbedre kvaliteten af undersøgelses udførslen når børnene er på hospitalet for at blive undersøgt og sikre at man opnår en brugbar undersøgelse for den månedlige undersøgelse.

At forbedre målingerne gennem biofeedback i form af et spil der udføres under undersøgelsen og derved opnå en bedre patient historik om sygdommens forløb og diagnosticering.

Abstract

English - Engelsk

The goal of this project is to retain measurement data of the lung functionality in Asthma and Lung Patients, targeted at children in the age range of 2-7 years.

Improving the quality of the examination execution at the time the children is at the hospital to be examined and ensure a useful examination result for the periodic monthly examination.

To improve measurements through biofeedback in the form of a game that is played during the examination and by that, achieve better patient history data about the disease progression and better diagnostics for the patient.

Contents

Contents	ii
Introduction	ix
project Introduction	ix
Motivation	ix
Issues	x
Ideas	x
Goals	x
Proposed solution	x
Project call name: <i>Spiro</i>	x
I Analysis	1
1 Terminology & concepts	3
1.1 Spirometry & Android terminology	4
1.2 Project terminology	5
2 Time plan	7
2.1 Initial planning elements	8
2.2 Revised plan	10
2.3 Plan and time plan discussion	12
2.4 Requirements from requestors RH	13
2.4.1 Revised requirements	14
2.4.2 Additional requirements	14
3 Analysis design Methodologies and tools	15
3.1 Analysis Methods	15
3.1.1 Analysis, design and planning	16
3.2 Development Tools chosen	17
3.3 Technology evaluation	17

3.3.1	Choice of game/display device	17
3.3.2	Choice of tablet	18
3.4	Libraries for android: Charting Graphics Game frameworks, Android API . .	19
3.4.1	Data persistence, storage and exchange	19
3.4.2	Game visualization technology	20
	Descriptions	20
	Evaluation	20
3.4.3	Game libraries	21
	Game libraries evaluation	21
	Decision	21
	Charts API's & libraries	21
3.4.4	Android Charting decision	21
3.5	Table of chosen technologies	22
3.6	The RH environment	23
3.6.1	Technical setup of spirometer measurement system	23
3.6.2	Development setup	26
3.7	Patient Examination observed protocol	27
3.8	Use cases and actors	28
3.8.1	Actors	28
3.8.2	Use case diagram	29
3.8.3	Use case descriptions : S1:Patient Examination system	29
3.9	Use cases & actors S2:SpiroGame	34
3.9.1	Actors S2	34
3.9.2	System Observer actor	35
3.9.3	Use cases S2: SpiroGame	35
3.10	Games scenarios	38
3.10.1	Game interactions	38
3.10.2	Questions for game and interactions	39
4	Project scope: System to be developed	41
4.1	Project formulation	41
4.2	Scope of project	42
4.2.1	Parallel tasks	42
4.2.2	Delimiting	42
4.2.3	Scope	43
4.2.4	Integration with external systems	43
4.3	Splitting up project in system parts	43
4.3.1	Component block diagram	45
4.4	Revised requirements	46
4.5	Domain model	48
4.6	Domain class diagrams	48
5	Android concepts	53
5.1	Android: Activity - Intent - Application	53
5.1.1	Activity	53

5.1.2	Intents	55
	Implicit Intent	56
	Explicit intent	56
5.1.3	Intent Filters	57
5.2	Activity life cycle	58
5.3	Manifest file : manifest.xml	60
5.4	Android Structure, Assets resources and layouts	62
	5.4.1 Assets	63
	5.4.2 Resources	65
	5.4.3 XML layout to view instance	66
5.5	The Android Pattern	66

II Design 69

6	Design	71
6.1	System overview	71
6.2	User interface & Mockups	71
	6.2.1 Mockup drawings	71
6.3	Screen navigation diagram	73
	6.3.1 Basic navigation flow concept	73
	6.3.2 Mockup lists	74
6.4	Separating and isolating system parts	75
	6.4.1 Game and examination separation	76
6.5	Generic and specific game concepts	76
6.6	Input and conversion	78
	6.6.1 Limits for patients: patient configuration	80
	6.6.2 Events state and cues	82
	6.6.3 Input data game event and the System observer	83
6.7	Design architecture	86
	6.7.1 Architectural requirements	86
6.8	S1:Spiro Patient Examination System	89
6.9	Data modelling	91
6.10	Data	91
	6.10.1 First table layout sketch	91
	6.10.2 database 1. Normal form	92
	6.10.3 database 2. Normal form	92
	6.10.4 Database 3. normal form	94
6.11	Game design : S2:SpiroGame	95
6.12	Game flow descriptions	97
6.13	Game events and state	98
6.14	Screen division	98
6.15	Prototype 1 description: SpyroFox	99
6.16	Prototype 2 description: Spiro PaperBoat	99

III	Implementation & Test	101
7	Implementation intro	103
7.1	First prototype	103
7.1.1	Prototype graphics bitmaps & and animation	104
7.1.2	Collision	106
7.2	DataInput Generator description	108
7.3	Implementation issues	110
7.3.1	Scrolling	110
7.3.2	Active game objects	110
7.3.3	The GameFieldMap	111
7.3.4	Setup game patient constraints	112
7.3.5	Updating game state	113
7.3.6	Game View	114
7.3.7	Game Init	115
7.3.8	Refactoring	115
7.3.9	Timing and events	117
7.3.10	Data feed	117
7.3.11	Performance and architecture	117
7.3.12	Implementation status	117
7.4	Test	119
7.4.1	Test1: Collision test	119
7.4.2	Test2: Active inactive list test	120
7.4.3	Graphics artifacts error and stars	121
8	Evaluation	123
8.1	Planning	123
8.2	Problems during the project and workarounds	124
8.3	Project goals	124
8.4	Extensions	124
8.5	Conclusion	125
A	Terminology & concepts	127
A.1	Spirometry terminology System Glossary	128
A.2	Android Terminology	129
A.3	Project terminology	130
B	Risk assessment	131
C	Use case collection : Patient Examination system S1 : Game System S2	135
C.1	Actors & use cases	136
C.1.1	Actors	136
C.1.2	Use case diagram	138
C.1.3	Use case descriptions	138
C.1.4	Use case candidate list	146
C.2	Use case tech memos	146

C.3	Requirements note, physical	152
D	Generator S3: simulated input data diagrams	155
D.1	Generator modelling	155
E	Generator and generator design	157
E.1	Generator design	157
E.1.1	randomized value point interval	157
E.2	Generator Use cases	158
E.3	Generator class diagram	159
F	Class responsibility descriptions	161
F.1	S1: PatientExamination System	161
F.1.1	Examination part	161
ExaminationData	161
ExaminationSession	161
ExaminationProtocol	161
F.1.2	Patient Part	162
Patient	162
PatientAdministration	162
PatientHistory	162
F.2	S2: Spiro Game System	162
F.2.1	InputEvent	162
F.2.2	GameController	162
F.2.3	GameFieldMap	163
F.2.4	Game Objects	163
F.2.5	GameEvents	163
F.2.6	Cue	163
F.2.7	GameState	163
F.3	S3: DataInput System : Probe Measurement Part	163
F.3.1	SpirometerMeasurementDevice	163
F.3.2	ProbeDataCalculation	163
F.4	S1-S2-S3: DataFeeder	164
G	Selected sequence diagrams	165
G.0.1	Patient examination sequence diagrams	165
G.0.2	Game sequence diagrams	165
H	Colour styling of diagram architecture and figures	169
H.1	Colouring legends	169
H.1.1	Spiro classification system	169
Analyzing systems and class systems	169
Peter Coad's classification method	170
Classification method from author:mkj	170
I	Xml for patient data transfer between RH system interfaces	173

I.1	JSON or XML	173
J	Test data	177
J.1	Test1: Collision	177
J.2	Test2: Active inactive	178
J.3	Entity relationship for Spiro	180
J.3.1	Keys, candidate, primary, foreign	180
K	Research documentation	181
K.1	Tablet growth	181
K.2	Source code and project site	183
	List of Figures	185
	List of Tables	187
	Index	189
	Bibliography	191

Introduction

project Introduction

The project is a cooperation between the DTU Software developments department and Rigshospitalet (RH).

The doctor Kim Gjerum and his coworkers at RigsHospitalet thought it would be of great benefit to examine the young patients in the form of a game. Especially children in the age range 2-7. In that way the children could play a game and at the same time complete an examination trial. Helped by biofeedback.

The patients are each monitored for their health and disease progression/regression once each month.

Motivation

The hospital have several games but they do not have a game for the type of examination used for the younger children aged 2-7.

RigsHospitalet have some games already for related examinations but the authors are for different reasons not able to/interested in developing the games further for this type of examination.

The game with biofeedback should make it easier to make good quality examinations, because many times an examination must run it's course, but with no usable examination results. This is because the children are not able to complete the examination test in a way that gives a meaningful and correct picture of the patients current health situation and lung function.

If the patient can play a game, that unconsciously make them perform a better examination, then there will be less waste of time and resources and importantly a better and more precise survey of the patients health progression.

Issues

The younger children doesn't have a fully developed perception of how they move and react on "input".

It is difficult to explain to the children AND make them follow the directions of the explanations, even though they might fully understand the explanation, they might not be able to translate it to the correct actions in the examination trial, breathe slower or faster or harder or softer, have correct body posture. Timing might be lacking or delayed when to start breathing, how much force, how fast and so on.

Ideas

The idea is to use the device to give different forms of biofeedback. Shaking, sound, colour, light and animation. To "guide" the child to react in the proper way to satisfy the examination procedure.

Goals

The goal of the project is to make a game that helps the young children, alter their behaviour when breathing, that makes them perform a better and more reliable examination test.

That way a patient history can be build up over time, so progression and regression of lung function and health can be monitored reliably over time.

Proposed solution

A game prototype will be created to run on an Android tablet, The native Android API will be used to program The game. A USB probe is acquired if possible to replace the legacy RH system as the measuring hardware for the system.

Project call name: *Spiro*

The project are internally called Spiro, it is used as top-level/qualifier names, package names.

Part I
Analysis

Chapter 1

Terminology & concepts

1.1 Spirometry & Android terminology

The Spirometry A.1 on page 128 and Android A.2 on page 129 terminology is described in the Terminology appendix A on page 128.

1.2 Project terminology

Terminology and concepts used in the project. The terminology might not always match the one used in the hospital domain, or the Android terminology.

This section should make it clear what is meant in the context of this report and tries to stay as close as possible to terminology used in Android(TM) and at Rigshospitalet (**RH**).

Project terminology		
Spiro	The internal project name	Used in packages, and for the market id
Probe	The measurement device that the patient breathes into	
Examination	trial, test, examination	Measurement procedure of the patients lung functionality
Patient examination System	Responsible for the patient history & analysis	S1
SpiroGame System	The game system that should be able to support different game prototypes	S2
DataInput System	Representing input to game, from the probe hardware, or from data generator	S3
Cue	(bio) feedback concept	time for the patient to do X...

Table 1.1: Project terminology

Chapter 2

Time plan

2.1 Initial planning elements

Time plan tasks:

1. Research project assignment, domain etc.
2. Initial planning
3. Planning concepts
4. Analysis
5. Dividing project tasks
6. Design
7. Development (Thinking about project)
8. Documentation & report
9. Implementation
10. Test
11. Project assignment

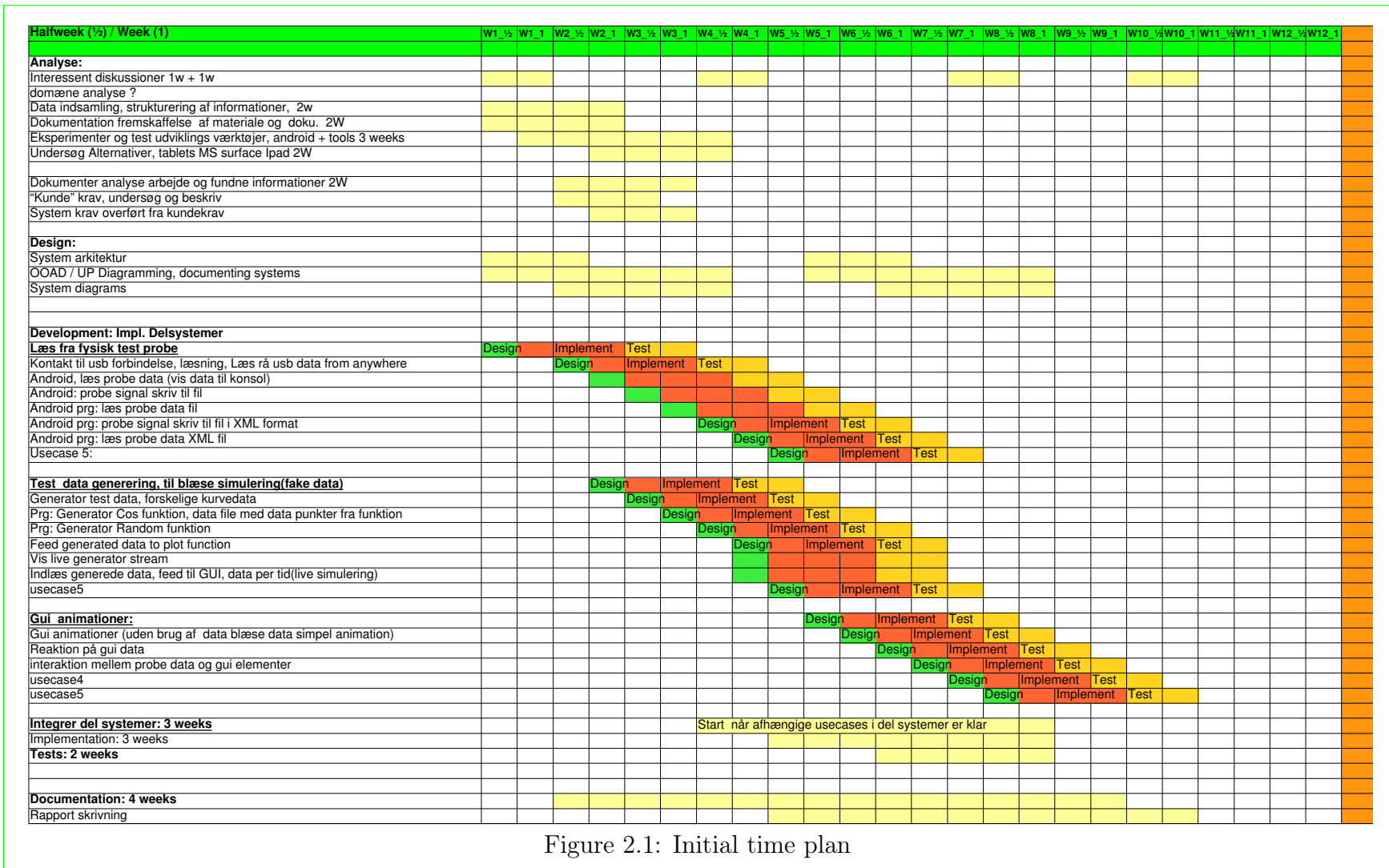


Figure 2.1: Initial time plan

2.2 Revised plan

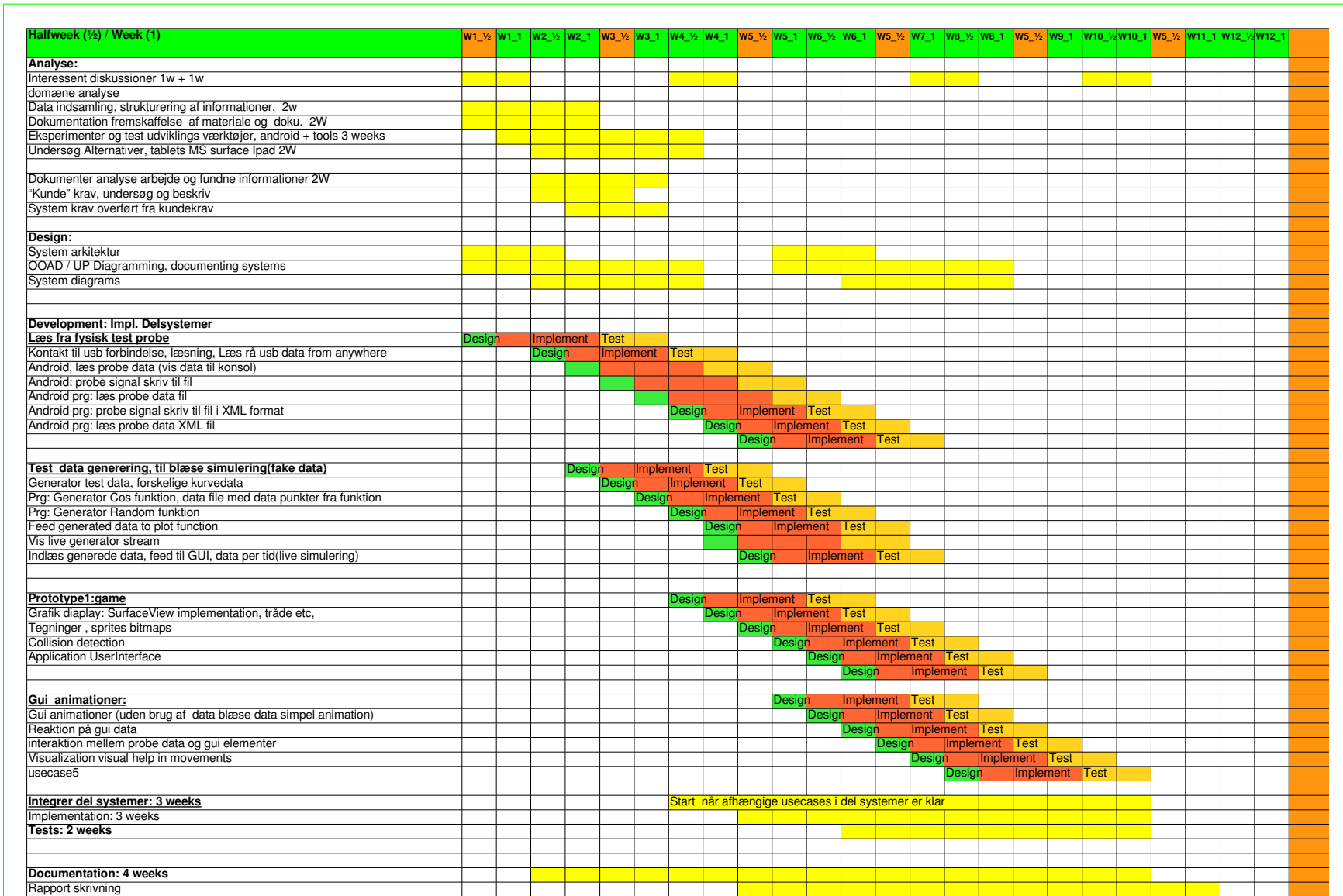


Figure 2.2: Revised time plan

The revised time plan has added prototype tasks and added tasks conceived from the project scope discussion. 4 on page 41 First the plan tasks was planned to be sub tasks for completing use cases but instead the tasks was derived from the systems view of S1-S2-S3 and the subsystems they were divided into. 6.1 on page 71

2.3 Plan and time plan discussion

The revised plan is updated with the new task issues. The set duration of 1 sprint of 2 weeks is marked on the revised time line. 2.2 on page 10

At the start of the project the work flow with sprints of 2 weeks wasn't decided on, but it turned out to work very well, it matched the work flow that was "naturally" used when working efficiently. In Sprint 3 it was formally introduced. It worked very well with a relatively short time period using one subject focus. Before it was introduced there was a tendency of to many focus items at the same time.

2.4 Requirements from requestors RH

<u>Nr:</u>	<u>Requirement</u>	<u>Comment</u>	<u>priority</u>
2)	Biofeedback in a game/visual feedback, that helps small children age 2-7, to respire in a way that creates correct measurement results.	Test 1: Breath in and out with a rhythmical frequency about 35 respirations per minute(in and out). Test 2: Expire (blow) and keep pressure continually (6 sec).	1
3)	Getting more precise and consistent measurements NF	Measurements of patient lung functionality sometimes gives unusable results, because the patient breathes improperly. This leads to unusable results. In those cases it can be necessary to wait until next appointment date for the patient. Which is about one month, and that breaks the patient history.	1
1)	Collect live data feedback from probe and visualize in real time.		2
4)	Game to be distributed via Android/iPad market etc.		2
5)	The system must function with different measurement systems(max lung capacity CT.). There are 4 different measurement systems <i>Airtight box test. Open door box test, for children able to complete it. Bike test(test under physical strain). CT Scanner Max/Min lung capacity.</i> NF	The Examination types and protocols look much the same, the interactions when playing can be very different.	3
6)	Tiny mouthpiece for mobile phone, used for home testing	The mouthpiece ensures a specific distance to the microphone, so each measurement has the same parameters. If the distance to the microphone changed all the time for each test, the measurements would be highly unstable. The mouthpiece should also reduce turbulence. C.3	5
Requirement priority table page 154 : Non-Functional requirement NF			

Table 2.1: Initial requirement list

2.4.1 Revised requirements

The revised requirements are based on the initial requirements gathered from the requirements from the hospital and our project "requirements" from the *developer team* (Author:mkj + Counsellors). This is revised during the project, learning from the experience and better understanding of the project issues that is obtained. The revised requirements is listed in the project scope chapter 4 on page 41

This project is run as a development pilot project and doesn't need to live up to the strict requirements and certification that might otherwise be needed in an public service environment like that on Rigshospitalet Some of the needed extra requirement for that can be seen in the section **additional requirements**.

Revised requirements with priorities set is described in the section: Project scope, formulation and delimiting 4.2.2 on page 42

2.4.2 Additional requirements

At Rigshospitalet there are additional requirements.

1. Storing patient history
2. Analysing patient examination test results(numbers + graphs)
3. Data transfer from system to system(Patient Information, cooperation)
4. Patient data security ("Data tilsynet" laws)
5. Data security, policies, technologies
6. Safe storage of patient data. Encryption, sand boxing
7. Patient interaction support "*research*"
 - a) Audio Sound, Music
 - b) Visuals
 - c) Light
 - d) Animations
 - e) Haptic feedback(Shaking)

The requirements developed against in the project scope is discussed in chapter 4 on page 41 and section 4.2.2 on page 42.

Chapter 3

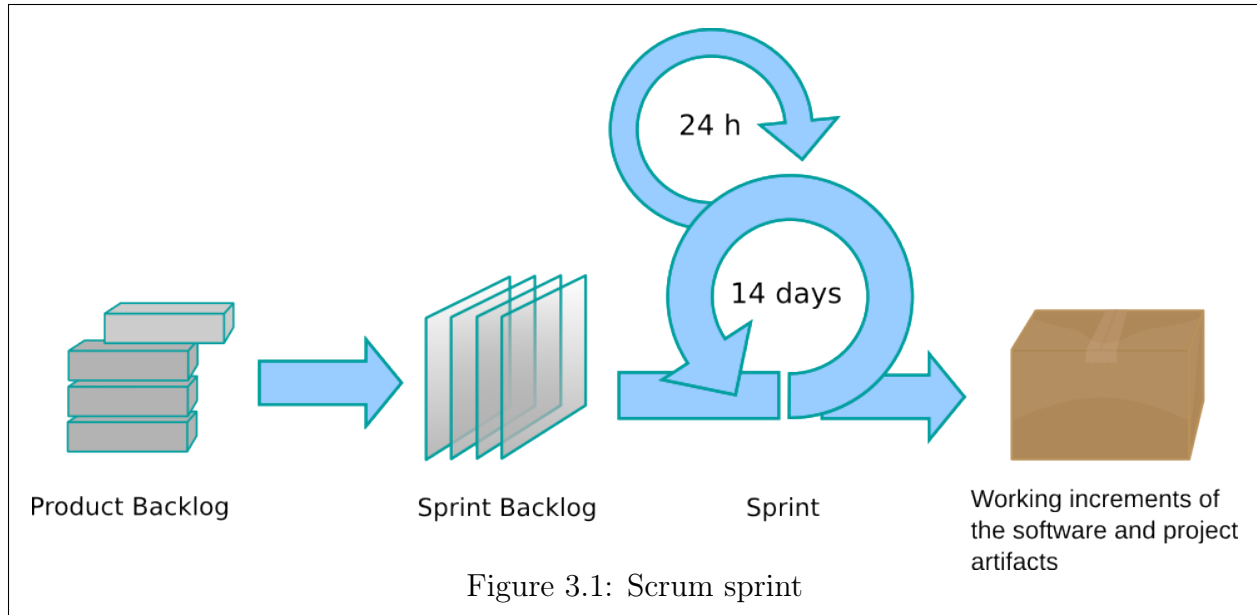
Analysis design Methodologies and tools

3.1 Analysis Methods

In this chapter the project process and method is described. How *steps/iterations* are *planned*. and how the *work* and *resources* are *structured*.

- Use of a subset of Unified Process artifacts.
 - Class diagrams
 - Use cases
 - Sequence diagrams
 - ..
- Scrum Sprints
 - Product Backlog - Set of tasks for the project scope to be complete
 - Sprint Backlog - The set of tasks chosen from the focus area tasks, needed to complete the sprint goal
 - Sprint goal - The focus of the sprint

A scrum sprint can be seen above, the project uses a slightly different terminology. The table above the image show the mapping to scrum terminology. 3.1 on page 15



3.1.1 Analysis, design and planning

The project is developed using object oriented analysis & design and Unified Process artifacts. We are working with small Scrum like sprints for each iteration.

For each iteration the main focus area is defined for that sprint. The main focus is then divided up into parts or subsystems. Each part is divided into small tasks that can be ticked off to keep track of progress. After each sprint, important missing items/features are written down for the next sprint with the same focus. This functions much like the backlog in Scrum, except that we might remove tasks in a focus area if time constraints are considered to get into play.

Punch small holes to breakthrough with functionality: We try to make small breakthroughs to get small pools of working functionality in small development steps.

An iterations focus can be in the following areas:

1. Research
2. Design (Design in the abstract sense of constructing a project)
3. Documentation
4. Implementation & Test

Sprint focus:

Sprint1: focus area: Implementation: Idea:Status , gain new knowledge Implement User interface and model Android concepts: Parts: Android UI classes, Android events.

Sprint2: Focus: Analysis & design Idea: model domain, and system Find Domain objects, model DM, requirements and use cases Content: Analysis RH domain and language, use cases use case diagrams+descriptions, requirements revised.

Sprint3: Focus: Implement prototype

Sprint4: Focus report

Sprint5: Focus Implement

Sprint6: Focus : Tie ends together, report documentation.

3.2 Development Tools chosen

We use the android ADT Eclipse plugin, that is developed by Google that stands behind Android.

Our CASE tool of choice is Visual paradigm. Which is a CASE tool with a large set of analysis and design features, diagramming and much more.

Eclipse is used because of the plugin from Google. Frankly it is used because it is easier to use it than not to use it. but unfortunately there are a lot of inconsistencies in the tool. and awkward inconsistently changing behaviour.

1. Android tablet technology
2. Visual paradigm CASE tool for analysis & design
3. Eclipse + ADT plugin

3.3 Technology evaluation

3.3.1 Choice of game/display device

Tablet availability: tablets are a technology that are growing steadily and the availability will probably be better as time goes on for a considerable number of years, so in that way it is a very safe technology for the future.

Tablets Features and capabilities comparison table			
Feature / measure	IPad™	MS Surface™	Android™
Availability:	High	Medium/coming	High
support:	Great	Medium	Good/(great community)
libraries	good	Coming	Great
tool support:	Great	OK	good
USB Support Host/Accessory:	Great	NA	Varying(Android V3.1->good)
Programming resources	Great	Coming	GREAT
Enhancements and flexibility:	good	NA	GREAT
platform extensions:	You got what you got	NA	Large open community with projects
Availability:	Great	Good	Great
Language:	Object-C	?	Java

Table 3.1: Tablets features and capabilities comparison table

Tablets of size about 10" or above are likely to be good for an immersive experience for games and visualization.

Any tablet could be chosen and be a good candidate.

3.3.2 Choice of tablet

For the project we have chosen the android tablet. For the following reasons.

1. Along with the IPad it has been on the market for some time and has matured a bit.
2. There is a large community around, developing for android and many applications.
3. Personal interest in open source development and communities makes it a good choice for the project.
4. The Java language along with the Android API ensures that the framework is large and of robust quality.

Statistics for tablet growth can be seen in the appendix

K.1 on page 181

As we can see the market share is shifting for the IPad and Android tablets to be close to each other in market share.

Both Apple and Android total devices is estimated to have more than a $< \frac{1}{2}$ billion units out. Most importantly we can gather that tablets is estimated to rise to almost double figures in 2016. Other sources varies a lot, but all predict large growth in tablets.

3.4 Libraries for android: Charting Graphics Game frameworks, Android API

IBM (2010)

3.4.1 Data persistence, storage and exchange

To store the data for the test we have a few different solutions.

XML, JSON, SQLite.

XML is the "most used" and commonplace, JSON is a simpler more compact way to structure the data.

Android have large support for XML there are a lot of API's for manipulating XML, documentation seems to be a bit lacking, especially since many of the API's has several ways of doing the same thing. Which is confusing since most XML tasks are pretty basic..

It is not documented why one approach is used instead of another, in several of the most frequently used libraries, Dom4J XmlPullParser, KxmlPullParser, Sax Parser.

JSON is frequently used instead of XML in web services. SOA, Service oriented Architectures and similar technologies. and is often preferred in the python community.

XML is used in many many places for many types of data exchange. Among other the danish public institutes use it for interchange between their multitude of systems. SKAT.dk, exchange between tax departments f.ex. Hospitals use it to communicate to other public institutions and to connect different IT system together.

SQLite is used for saving data on android and have inbuilt support to make it easy to manipulate data to and from an SQLite database.

Data storage for android			
Feature / measure	XML	JSON	SQLite
API real world usage	Extremely high	Good	Good
Size (for transfer)	Large	Small	Large / inflexible
Exchange capability	Fantastic	Good	not build with exchange in mind
Library manipulation support	Great	OK	Great

Table 3.2: Data Storage & persistence

SQLite is chosen for saving data, because of it's ease of use on the Android systems.

XML would be the best choose for a genuine hospital product. Because it is used for many data exchange tasks. If the system later needs to integrate with other systems at Righospitalet or other systems for example in the public service, XML would be the exchange technology of choose.

3.4.2 Game visualization technology

Descriptions

Canvas and View:

Views is the elements used to make up user interfaces. For that reason it is not very light weight. Because of all the functionality attached to these visual elements. Also the view "by default" are running in the application main thread also known as the UI thread. UI thread, in Android description ?? on page ?? . All views are drawn upon the *canvas* and all views have rectangle shaped boundaries. The Views are placed "inside" a layout, and the smallest unit of screen update is the bounding rectangle of the view.

SurfaceView and Canvas: SurfaceView is a more "intelligent" version of the standard views and canvas. It knows more about which part of the screen is updated, so a view that changes partly doesn't have to be completely updated. The SurfaceView runs in it's own thread.

?? on page ??

OpenGL Java API

Has almost all capabilities of the OpenGL lib's. Exposed through an relatively easy to use Java API.

there is some setup code to be included, that makes it's ease of use less than using the view and SurfaceView.

RenderScript RenderScript are C scripts written to be executed from the android java code. it is OpenGL code in C. And can be used to optimize to a specific architecture and GPU. this off course makes it very specialized and might not be needed but for the most demanding applications of graphics.

Evaluation

It was decided to use SurfaceView as implementation API since it is close to standard UI android programming. But has better performance, because of running in its own thread. And more "intelligent" partial screen update compared to standard Canvas and view use.

LibGDX is the most extensive framework of high quality. It is a collection of frameworks, for sound, graphics, physics engine. etcetera. LibGDX (2013) Box2D (2013) OpenAL (2013) OpenGL (2013)

Game Visualization technology				
Game Visualization	Canvas View	SurfaceView	OpenGL (Java API)	RenderScript
Ease of use:	Extremely high	High	OK	bad
Speed	OK	Good	Great / accelerated	GREAT
Capabilities	not so good	OK	Great	great for it's purpose
API manipulation support	Great	Great	Great	not so good

Figure 3.2: Game Visualization technology

3.4.3 Game libraries

Graphics libraries & frameworks			
Library/ Framework	LibGDX	AppInventor	SurfaceView Android Graphics API
Ease of use:	High	Very high	High, using knowledge from standard android programming
Performance	Great (OpenGL behind)	OK	Good
Music & sound support	Great	OK	Supported via Android sound API

Figure 3.3: Graphics and Game libraries

Game libraries evaluation

LibGDX. Interesting libraries, where all parts of games music sound graphics, etcetera is gathered through one cohesive Framework & API's. Has very high performance. it uses OpenGL for graphics, and whole set of well known longtime supported libraries, OpenAL, OpenGL. Java lightweight game framework LWJGL, Box2D. and more.

Decision

The decision is to use the native SurfaceView framework with bitmap graphics. That way we can use our previous known Android Java API's and libraries.

Charts API's & libraries

3.4.4 Android Charting decision

Android API for charting is chosen for initial simple charting. Chosen because of the familiarity with the Android API.

Charting				
Library/ Framework	AndroidChart	AiChart	ChartDroid	Android Canvas view API
Free / Open source:	Yes/Yes LGPL	150\$/No	Yes/Apache License 2.0	Yes/User space: Apache License 2.0, Kernel & internals: ASL2.0
Clickable parts	Yes	Yes	no	need implementation
Live updating	Yes	Yes	Maybe/not documented	needs implementation
Language	JavaScript	Java	Java Library and Android Intent based	Android API

Figure 3.4: Chart API & Libraries

The best library choice is evaluated to be AChartEngine.

AChartEngine:

1. it is free AND open Source 2. It has large selection of different charts 3. it has documented live update of data

If needs arise for more than very simple charting this is the library of choice. AChartEngine (2013)

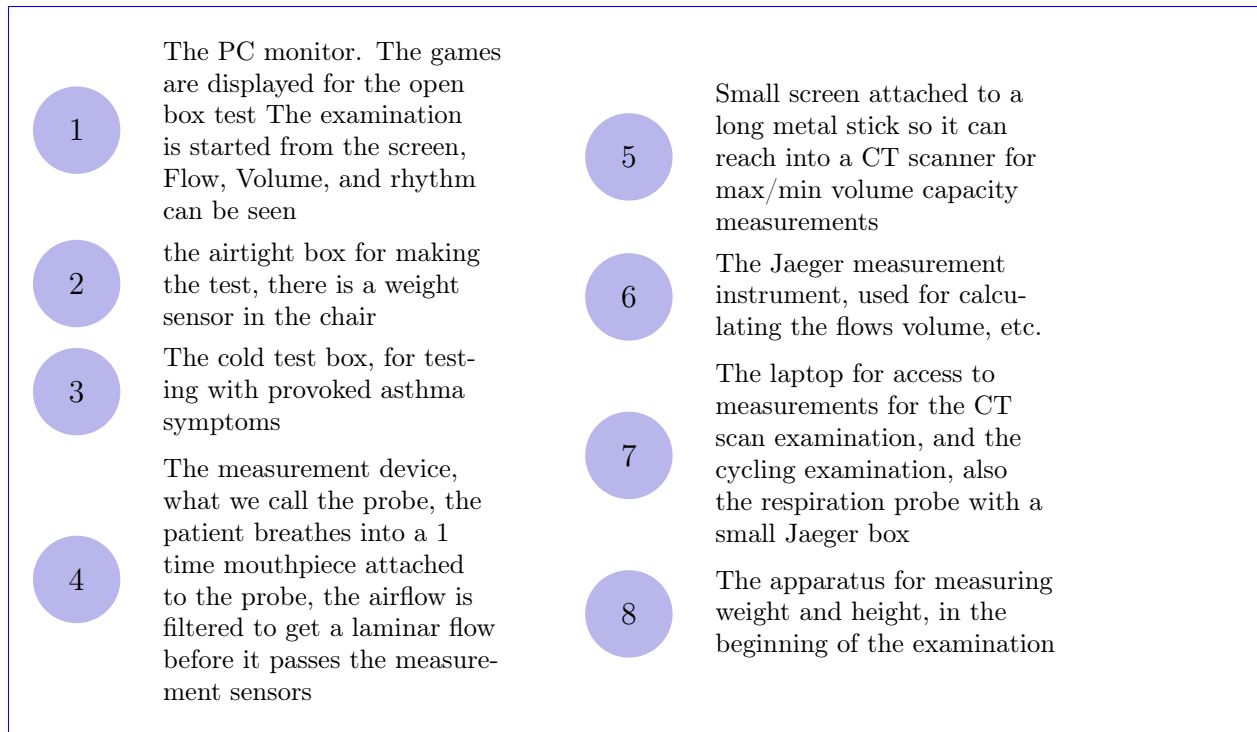
3.5 Table of chosen technologies

3.6 The RH environment

3.6.1 Technical setup of spirometer measurement system



Figure 3.5: Visual elements of the RH measurement system



In the top figure we can see the legacy pin outs for the hospital. The hospital use the 37, 15 and 9 pin.

The bottom schematics we can see the current system as it is at Rigshospitalet, the *system box* outlines the components and functionality that the tablet will replace. The connectors should all be considered outside the systembox.

The USB converter point is where the placement of an adapter or a conversion plug can be placed. IT is possible to only create one converter and that way connect with all the RH examination systems described in the line drawing of the system.

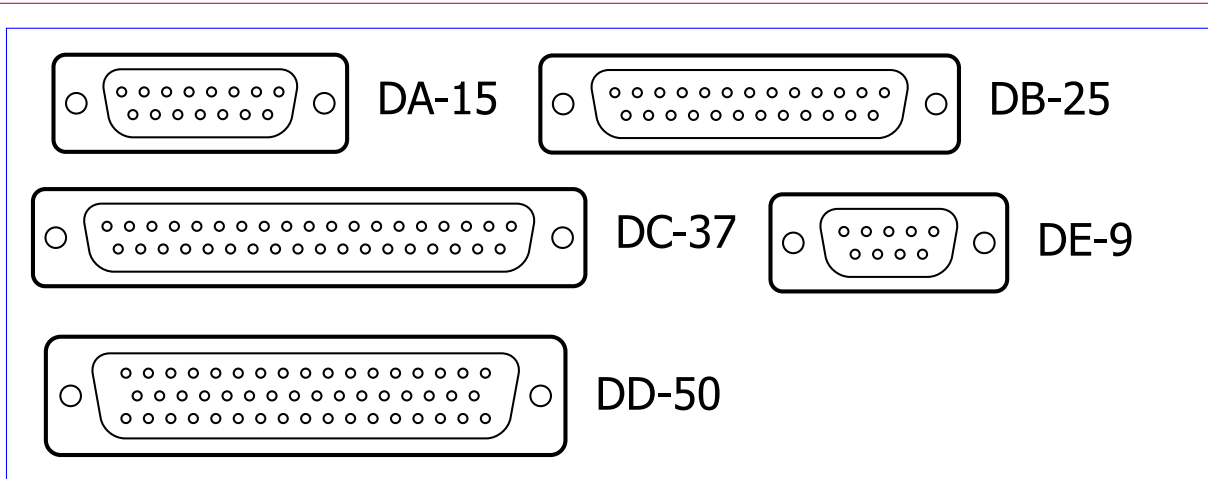


Figure 3.6: Connectors

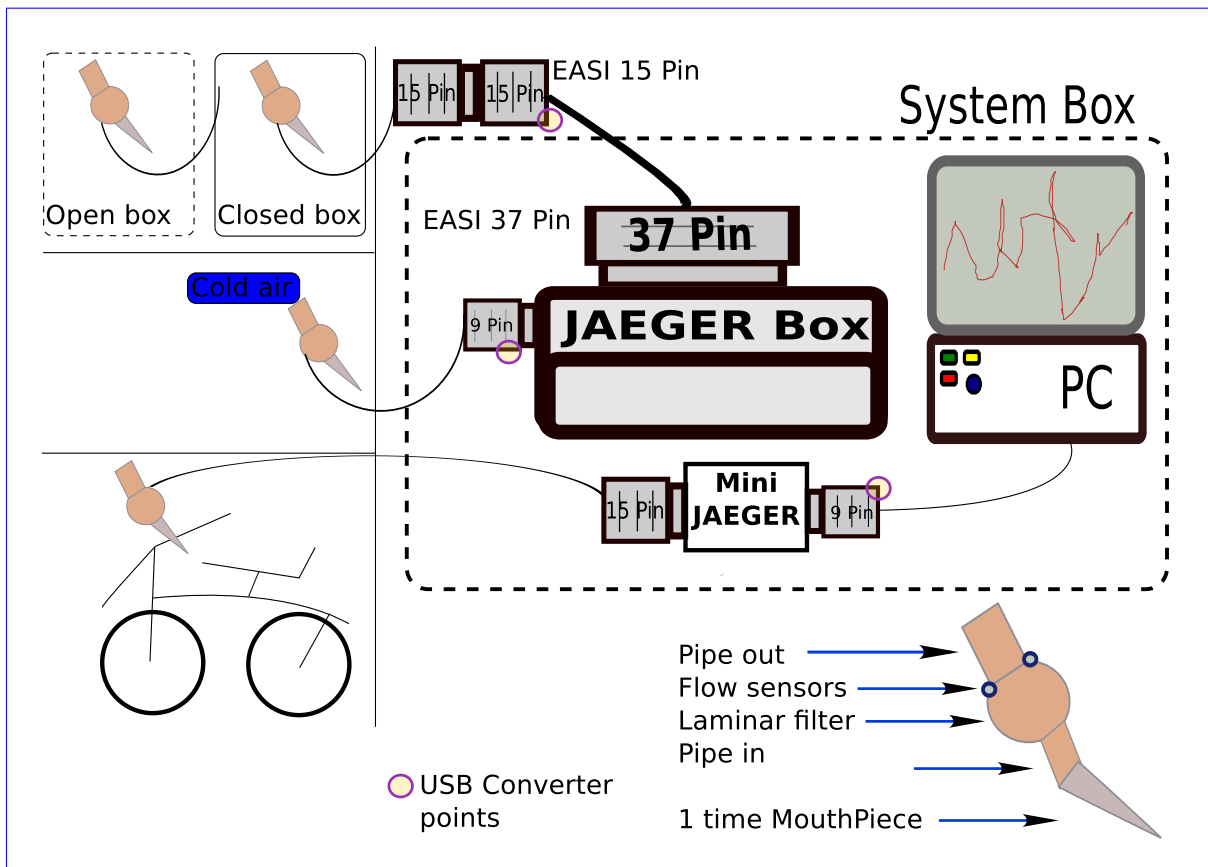


Figure 3.7: Line drawing of the Spirometry measurement system, current system

3.6.2 Development setup

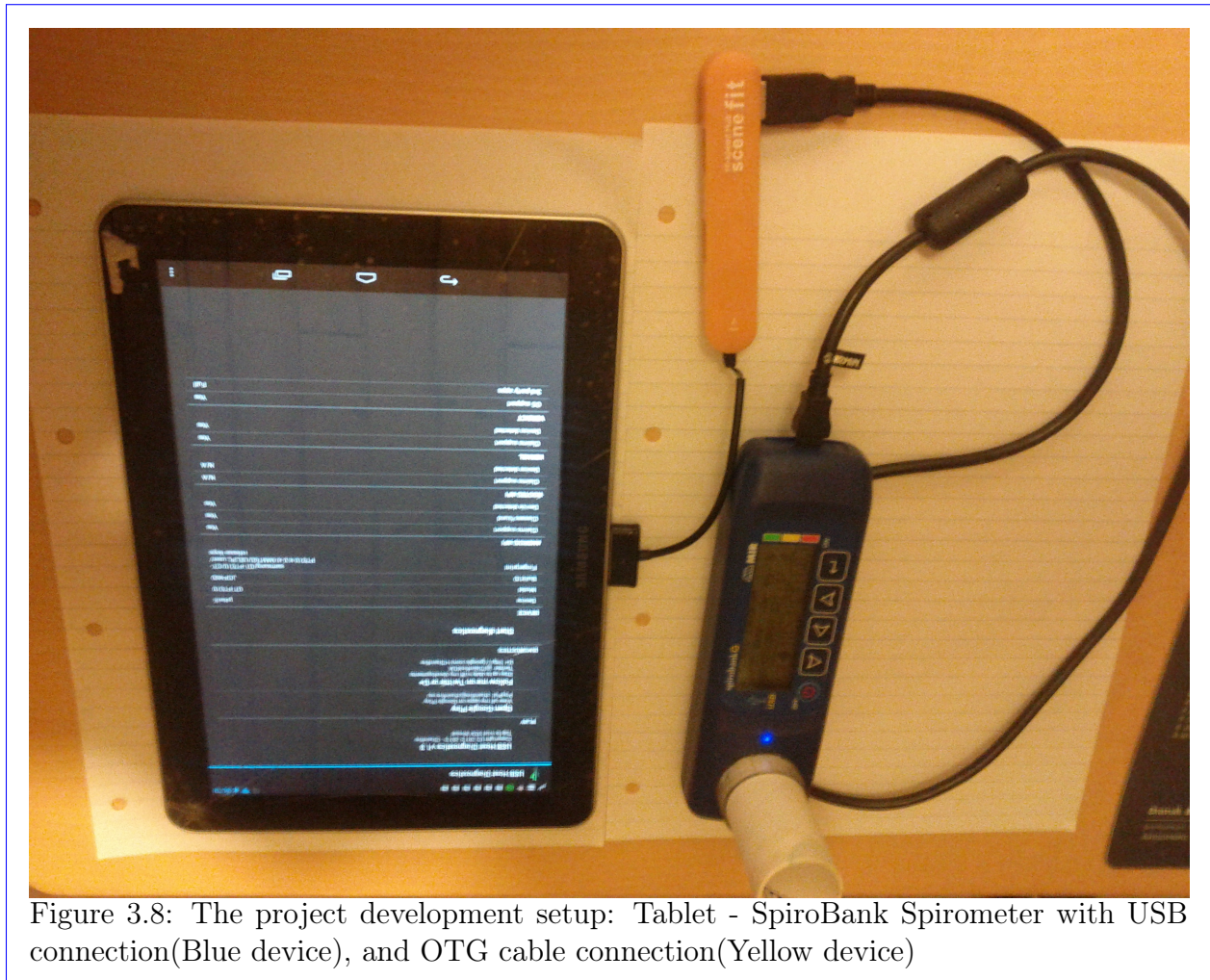


Figure 3.8: The project development setup: Tablet - SpiroBank Spirometer with USB connection(Blue device), and OTG cable connection(Yellow device)

The development setup consists of a spirometer SpiroBank G, Android Galaxy tablet v.1. And OTG cable to enable the USB Device host and make the spirometer an *accessory* device controlled by the tablet.

SpiroBank G: the white tube is where the patient breathes into. Several different examination types can be chosen. The OTG cable connects a pin in the connector attaching to the tablet that enables the USB Host device controller(The tablet is host for the SpiroBank device and controls the USB connection and power).

The development setup makes us able to work independently of the hospitals equipment. Unfortunately this probe doesn't work with the tablet because it has a USB version 1.0. Precisely this makes it incompatible with the tablet. Any other USB version should work, 1.1 2.0 3.0. The Android USB controller and USB connection 1.0 has an undocumented incompatibility.

The Spirometer can't be accessed from the tablet so we cannot get a live stream. Also it seems the live stream data are not recorded on the SpiroBank device, only the calculated

figures. This mean that another spirometer device is needed to test live stream and that we need to use the data generator.

The spirometer can be used for testing to get a feel for the examination procedure. So it can be used for helping with game concepts.

The Spirometer is sponsored from Maribo Medico. <http://www.maribomedico.dk/>
<http://webshop.maribomedico.dk/en/spirometry-15/mir-spirobank-g-68.html>

3.7 Patient Examination observed protocol

An informal protocol description derived after observing several tests at the hospital, and after talking to nurses and doctors.

Informal examination protocol:

- **step1:** patient and follower(Parent) is arriving, Examination supervisor/nurse gets ready.
- **step2:** patient is led/coming into the the examination room, and are weighted (Weight) and measured in height(Height). Age and birthday is noted.
- **step3:** The equipment is readied, new clean tube is put on the probe. Patients are instructed. System is readied(button click on test type)
- **step4:** The patient executes the test(test observed was the open box test, where the patient must keep the airflow steady)
- **step5:** The test is executed several times, the difficulty is adjusted, difficulty is how hard the patient must exhale to progress.
- **step6:** the Examination session is ended, the patient is offered a toy, goes to the waiting room.
- **step7:** A doctor comes after the test and talks with the patient and relatives about the examination and treatment.

Table 3.3: Informal examination protocol

3.8 Use cases and actors

3.8.1 Actors

Actor:	Test person/Patient
Description:	The patient that are about to be tested and measured.
Goal:	To measure the current state of lung function/health and to add that to a data collection, so that progress can be followed. Creating a history of health progression/regression.

Actor:	Supervisor
Description:	The doctor or nurse, that supervises the test, initializes the machine, instructs the test person and observes that the test is valid, useful and done correct.
Goal:	To make it easier to "instruct" or "guide" the patient through an Examination test

Actor:	Treatment analyzer
Description:	The doctor / nurse, check test data, compare patient progress, browses and analyzes patient statistics
Goal:	To see if the patients treatment is giving progress, analyzing the data over time to see progression/regression.

Actor:	Information browser
Description:	The test person: can check the data visualized for the test, and see progress from earlier test measurements
Goal:	To let the patient follow and their relatives(followers) their progression and status without requiring deep knowledge of the subject

NOTE: the test person gets access to a simplified view of the data, or maybe a replay of test ie. the game replays the session.

Actor:	Advisor
Description:	Follows the patient and comes with input, advice and support to the patient, has access to all historic data, and might also be the observer(?), so that s/he can counsel the patient
Goal:	The advisor is the specialist/doctor etc, that follows the patient and gives advice on progress and status.

3.8.2 Use case diagram

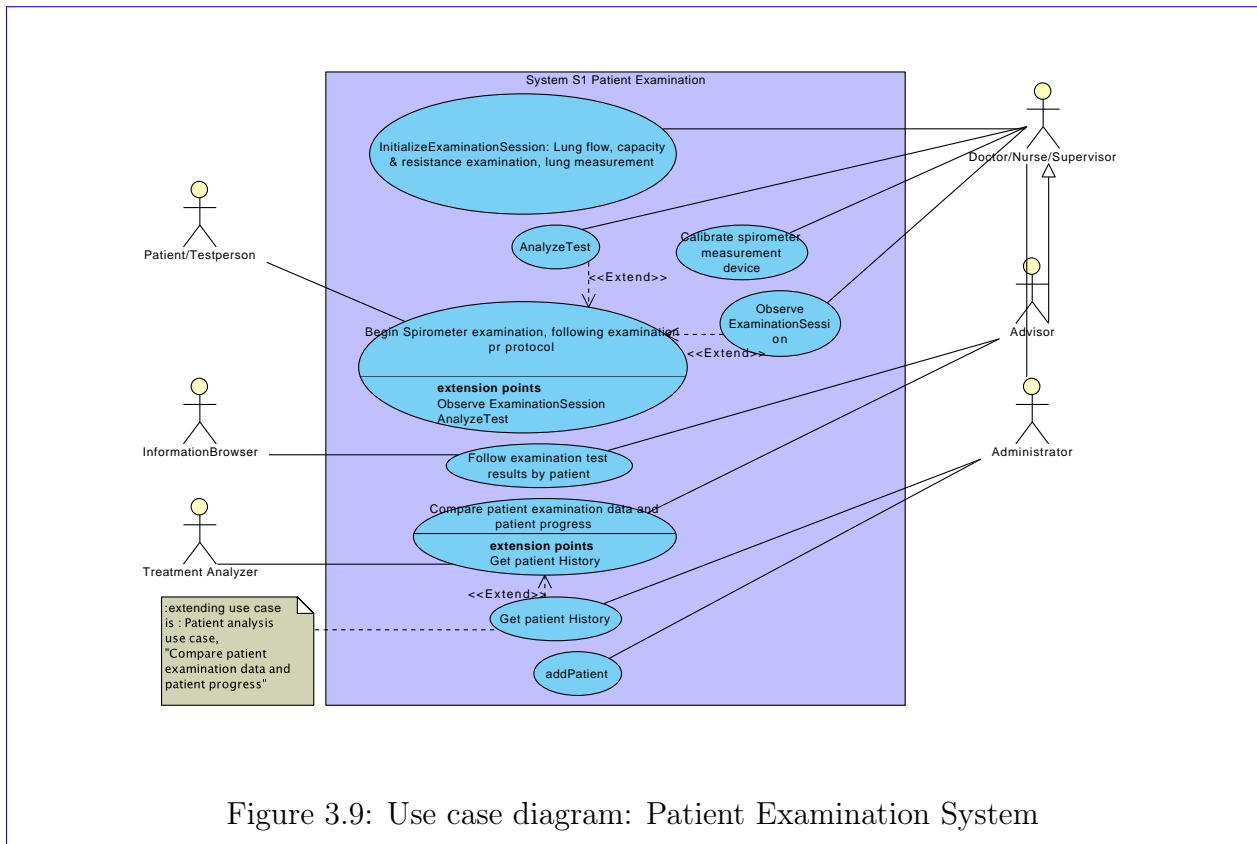


Figure 3.9: Use case diagram: Patient Examination System

3.8.3 Use case descriptions : S1:Patient Examination system

A collection of all use cases can be found in the appendix. App: C on page 135

There are not use cases for each examination type. The work flow is similar. If work flow differs significantly, named extensions is used.

Use case Id: UC:PE1	
Use Case Title: Examination <i>initialization</i>	
Primary actor:	Supervisor
Level:	wip: Work in progress
Stakeholders:	Doctors, Nurses, Patients
Precondition:	Patient has arrived
Success Scenario	System has existing patient data and lung capacity and flow limits for the patient. Meta data is recorded, (metadata: name, birthday, ID data, weight, height). The examination is ready to proceed.
Trigger:	
Standard Scenario:	<ol style="list-style-type: none"> 1. Patient Id data and height, weight, name are registered in to the application. 2. Patient flow and capacity limits are set or read from earlier examination. 3. The examination is started according to protocol (see extensions): 4. Patient breathes in to the probe, recording, biofeedback and game is started(See Use case for Examination: ?? on page 140)
Extensions:	<ol style="list-style-type: none"> 1. Faulty equipment: test is aborted: 2. Other measurements and calculations not yet thought of might occur. 3. Closed Box lung resistance, Cold air, Cycle test, CT Max/min Scan.
Comments:	This use case is extended by the different examination types available. ex: use case "Lung flow and capacity examination, lung resistance examination test(closed Box)". (@all examination type use cases)

Figure 3.10: UC PE1: Initialise examination

Use case Id: UC:PE2	
Use Case Title: Closed Box Examination (lung resistance examination)	
Level:	wip
Stakeholders:	Doctors, Nurses, Patients, Follower
Precondition:	Use case Examination <i>initialization</i> is done
Minimal Guarantee:	
Success Guarantee:	The patient keeps an steady respiration rhythm and a sufficient strength amplitude, so that the measurement is usable. The measurements are recorded in detail and the data are stored away.
Trigger:	...
Standard Scenario:	<p>Patient breathes in to the probe, recording is started,</p> <ol style="list-style-type: none"> 1. - The game tries to adjust the test persons breathing if it is to slow or to fast. 2. - Feedback to test person, to ensure the correct strength-/amplitude of the breathing. 3. - The game tries to help the user to correct breathing with feedback. 4. - When the patient is doing well the game can adjust difficulty up, or down if the patient errors again. 5. (@other detection? of bad measurements.)
Extensions:	<ol style="list-style-type: none"> 1. If the game is unsuccessful at correcting the respiration of the patient the game can restart, and reintroduce the patient to the execution of the examination. 2. Patient breathes to fast or to slow, and can't adjust to correct breathing pattern, the session times out and ends with an overview, The game signals for manual intervention.
Comments:	<p>This use case provides a description for examination. But it doesn't specify the exact nature of the feedback, also use cases like <i>use visualization</i> or <i>give visual hint</i> doesn't give information of the system. @see Game Scenarios section.</p> <p>NOTE: this is a use case for general types of examinations.</p>

Figure 3.11: fig:Lung resistance test, closed box

Use case Id: UC:PE3		
Use Case Title: Use Case Title: Analyze examination data & patient progress		
Primary actor:	Treatment analyzer, Advisor	
Level:	wip	-
Stakeholders:	Doctors, Nurses, Supervisors, treatment analyzer	
Precondition:	Minimum one examination test is executed before.	
Minimal Guarantee:		
Success Guarantee:		
Trigger:		
Standard Scenario:	<ol style="list-style-type: none"> 1. The actor searches and compares examination data. 2. The Treatment analyzer chooses, <i>Analyse Patient</i>. 3. Chooses 1 patient. 4. Chooses a time period. 5. The patient data for that time period is displayed on screen (graph). 6. The data and progression can be visualized and data can be chosen or added. 	
Extensions:	<ol style="list-style-type: none"> 1. The Analyser can go back and pick patient and examination and a new time period. 	
Comments:	This use case are not directly involved with the system "the Game" we want to develop. But it is important to be able to use the data after, and the way the data can be assessed and evaluated can be important for the system to be a success. That is why we want to make the game.	

Figure 3.12: Compare data and patient progress

Use Case Title: Choose existing patient for examination		Use case Id: UC:PE7
Primary actor:	Supervisor, Nurse	
Level:	wip	
Stakeholders:	Patient, Supervisor, Advisor, Follower.	
Precondition:	The patient MUST exist in the system.	
Minimal Guarantee:		
Success Guarantee:	The chosen patients data comes on the screen and is ready to start examination.	
Trigger:	Examination started	
Standard Scenario:	<ol style="list-style-type: none"> 1. Supervisor chooses a patient in the system, by name or Id 2. Patient data appear <ol style="list-style-type: none"> a) PatientID (CPR) b) Name c) Birthday d) Weight e) Height f) Test date g) Limits (lung capacity, amplitude/strength etc.) are set or adjusted. 3. Supervisor confirms patient and examination is ready to start. 	
Extensions:	1. Limits are altered, because of patients condition (took medicine recently, sick, etc.) This is examination initializing with existing user, this includes the initialize examination use case ?? on page 139 it describes the first 2 steps in more detail	
Comments:		

Figure 3.13: Choose existing patient for examination

3.9 Use cases & actors S2: SpiroGame

C.2 on page 147

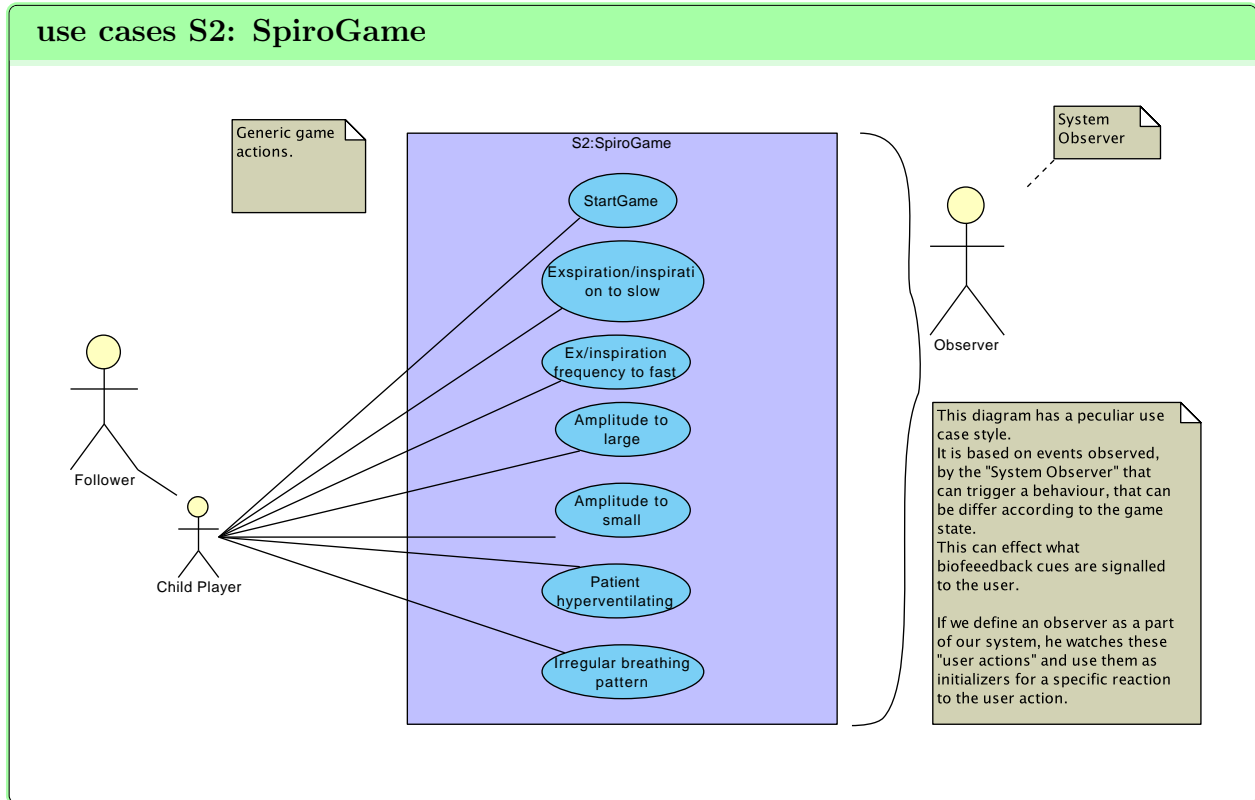


Figure 3.14: Game Generic use cases

3.9.1 Actors S2

Actor:	Child player
Description:	The patient to be examined and measured, the human player.
Goal:	To complete an examination, by playing a game,

Actor:	Observer (System observer)
Description:	The System observer, watches actions and events generated by the human player.
Goal:	To react to events that the player generates

The use case diagram shows a *actor* called *follower*, this is often the parent. And is displayed on the diagram, because it is an important person in the process. But not active in the process, the follower works as a guide to the children patients.

3.9.2 System Observer actor

The System observer is a special actor that watches all the input event that are initiated by the patient actor.

The system observer do not initiate any action itself, but reacts to the events and actions that it observes.

3.9.3 Use cases S2: SpiroGame

Use Case Title: Init game	
Primary actor:	Patient
Level:	wip
Stakeholders:	Patient, Supervisor, Advisor, Follower.
Precondition:	Patient Examination Session is started.
Minimal Guarantee:	
Success Guarantee:	
Trigger:	
Standard Scenario:	<ol style="list-style-type: none"> 1. Patient chose start game. 2. The game starts Init setup. 3. Objects are placed on the screen. 4. The game map is initialized 5. Game State is set to running state. 6. The input data from the probe is read throughout the game.
Extensions:	
Comments:	

Figure 3.15: Game is initialized,ready to start

SpiroGame Use Case list & suggestions:

1. Existing list:

- a) StartGame
- b) Expiration/Inspiration to slow
- c) Expiration/Inspiration to fast
- d) Amplitude to large
- e) Amplitude to small
- f) Patient Hyperventilating
- g) Irregular breathing pattern.

2. Suggestions:

Use case Id: UC:SG3	
Use Case Title: Patient amplitude to large(exercise to much force in breathing)	
Primary actor:	Patient
Level:	wip
Stakeholders:	Patient, Supervisor, Advisor, Follower.
Precondition:	game is active.
Minimal Guarantee:	
Success Guarantee:	game on
Trigger:	
Standard Scenario:	<ol style="list-style-type: none"> 1. Patient Breathes into the probe. 2. The patient breathes with excessive force into the probe. getting above optimal path(). 3. The game detects that the patient breathes above capacity/flow limits (How?) 4. The Patient is signalled to breathe in a relaxed fashion (only, closed box ex.?) 5. Waiting for patient to react to signal and be stable for period of time 6. Patient has reacted to the signal and have been breathing with the correct speed for some time. 7. Signals?
Extensions:	<p>The patient does not react to the biofeedback cue, and doesn't speed up the respiration.</p> <p>The Game sends a new signal/Event to try get the user to react.</p> <ol style="list-style-type: none"> 1. Player figure gets different colour (blue) 2. Visual hints (Eyes are tilting in the direction it want to go) 3. player figure tries to signal that it want to speed up(fidgeting) 4. The path or something on the path begins to lighten up 5. music plays faster 6. Voice says to blow
Comments:	

Figure 3.16: UC:Child Player breathing rhythm to slow

Use case Id: UC:SG5	
Use Case Title: Patient breathes correctly	
Primary actor:	Patient
Level:	wip
Stakeholders:	Patient, Supervisor, Advisor, Follower.
Precondition:	game is active.
Minimal Guarantee:	
Success Guarantee:	game on
Trigger:	
Standard Scenario:	<ol style="list-style-type: none"> 1. Patient Breathes into the probe. 2. Game encourages the path 3. Points are gained, 4. Game visible Objects display cue for correct behavior 5. Adjust difficulty ? 6. System observe capacity limits 7. System observe flow limits 8. Patient has maybe reacted to the signal and have been breathing properly 9. How do we find if encouragement works, and keeps the patient in range 10. Cue Signals?
Extensions:	The misbehaviour Use cases are all extensions of this one
Comments:	This us case is important because it touches on what the user gets from the system when doing good, continuously encouraging correct behavior, but it is hard to know when it is needed. A flow diagram might be a good way to show this)

Figure 3.17: UC:Child Player breathing rhythm perfect

a)

Other SpiroGame Use Case game scenario suggestions:

1. Generic Basic game scenarios:

- a) Collision
- b) Direction mark lights up
- c) Collected X objects

2. Scenario:

- a) To far from optimal path
- b) irregular rhythm for to long
- c) No reaction to biofeedback clue

3.10 Games scenarios

Use cases have a non-optimal format for description of some of the game interactions. See Game scenarios describe in details how the game is to react in certain circumstances to support the patient interaction with the game. These scenarios change from game concept to game concept. While use cases are better suited for generally describing the goal for the user.

The reactions to and interactions between system and the patient is developed and examined via the game prototypes and their game play and concepts.

Craig Larman on using Use cases and alternatives for finding requirements etc. Larman (2004) page 63-66.

However there is a base part of the interactions that concentrates on the examination issues, that the game should support.

This means that there is a set of use cases that deals with executing the examination correctly, and another part where we are more game centric, and create game scenarios, that might not directly be of help to the examination process and result, but frees us to think in game terms and think about human behaviour. These game scenarios actions and components should off course support a better measurement technique and create useful biofeedback concepts or items.

3.10.1 Game interactions

In the games we define an optimal breathing pattern, amplitude and frequency. For design only *Closed Box examination* is used because that is the most interesting for RH.

Time durations are defined, with states. For example if the state *'breathing to slowly'* is kept. and we send biofeedback clues back, but that doesn't bring the patient back on the correct track we can enhance the feed back clues, try other clues, sound, blinking objects and so on.

The game environment can be affected too f.ex. by moving the player back on the right track nearer the optimal path. The game could slow down so there is more time to react. Let the patient go through some steps we know that they can do before the game / examination continues measurements.

We measure how far we been away from an optimal path, (this is the openBox scenario and it is dependent on the examination type/protocol). Also we measure if the rhythm is unsteady for to much time. A score could be made for the quality of the measurements of the examination results.

3.10.2 Questions for game and interactions

Tech memo:

Questions asked for game interactions.

- Girl Boy differences?
- Easy slow rhythm.
- Fast involving.
- Hiding the test from the patient.
- Enhancing the test for the patient. (Visualized the test and add the patient to follow the correct test procedure).
- Age mobility dexterity levels

List of considerations that went in to the R&D for the game prototype concept designs.

Chapter 4

Project scope: System to be developed

4.1 Project formulation

Children in the age range 2-7 have difficulties following instructions from the supervisor. It is difficult to explain children at that age range, in words how to "do" the examination procedure.

The patients have an appointment once each month and sometimes an examination is executed poorly so that no usable results are obtained. It can be because the patient get sad, afraid, scared or because the examination is carried out but "incorrectly leading to invalid results".

The patients are measured examined and tracked for Flow and lung volume capacity and strength:

1. Test where cold air is blown into the lung while the child breathes(respirate) through the measurement device(spirometer). This can provoke asthma symptoms.
2. Lung function with controlled rhythmic respiration.
3. Lung Volume capacity, measuring max. expiration and max inspiration of air volume.

In test 1 and 2 the test person must breathe with a amplitude /strength, near normal breathing amplitude, with a relaxed rhythm and frequency.

Then our system should measure data and records it live and gives biofeedback to the patient via the tablet unit.

The biofeedback is provided by developing a game, that provides clues that tells the patient to adjust his/her breathing in some way. Until the patient is in to a stable and correct breathing pattern again.

The clues tells the patient if they should breathe with more power or breathe with a faster or slower speed(frequency).

The **objective** is to increase the quality of the examination tests, so the results are more stable and provides a solid picture of the patient actual health status.

To avoid a inappropriate respiration pattern that could affect the measurements results, without getting detected, so we can initiate a new test.

With a more reliable measurement procedure, the patients development(progression) over time can be monitored and show the correct development.

A game prototype will be developed on a tablet unit. And will be published on the tablets respective markets or distribution channel.

The set of requirements gathered from the analysis from the initial talks with stakeholders from the hospital.

Using the prioritized requirements, we can define a scope for the project time and resource constraints.

4.2 Scope of project

4.2.1 Parallel tasks

The project tasks are sorted and split in to sub systems

USB access, Graphics, Data storage. User interface. Game design.

That way the tasks are not dependent and if progress stops on one task, another part of the system can be developed.

a task usually is worked on 2-5 days, with some design, implementation. then we work on a another task in the focus area of the sprint.

4.2.2 Delimiting

We have isolated the RH system the hospitals "legacy" hardware and abstracted from that, That means we do not try to integrate our game and tablet system into the RH system in the scope of this project and the time allotted.

This is done so we can develop our system independently from the hospital and do not have to rely on their equipment for testing.

The drawback from this is that we get a separate system, that doesn't integrate with the hospitals other systems.

Early in the project we will develop a data generator to be used for simulation, that way we can test the game, even if access to the probe doesn't succeed.

The focus area is on the game and how it interacts with the patient, to be successful.

The next focus area is to have access to usable data, to be able to visualize the test.

We try to reach these focus areas by developing several mini apps Each developing one piece of functionality.

App1: input user data, transfer to new activity(think the game) here it is just the input values from one activity displayed in activity 2.

App2: Views and Layouts, play.

app3: animation, touch control, collision detection. Simple object movement.

App4: prototype1: Android SurfaceView for efficient screen updating, running in own thread, game state. plus graphics and bitmaps, animation.

4.2.3 Scope

4.2.4 Integration with external systems

Ideally it would be nice to integrate the game/examination system with the systems at RH and have information sharing between systems. Integration with patient journal, the examination data could be automatically stored/referenced from the patient journal. That would have the added benefit, that if the hospitals external system began to share information from hospital to hospital, then examination and measurement data information could also propagate with it automatically.

We have decided not to store the data via XML as a time constraint issue.

XML is the standard for coupling independent systems data transfer and sharing in the public service institutions. Each new system that is developed to be used isolated from the other equipment, burdens the personnel focus ie. their knowledge and time is used on making devices and systems work together, instead of concentrating on the patient and "*doctoring*" tasks. Therefore integration at some point is important.

4.3 Splitting up project in system parts

Systems: S1 S2 S3

S1: Patient Examination system - Hardware respiration instruments
- Reading data USB connection.

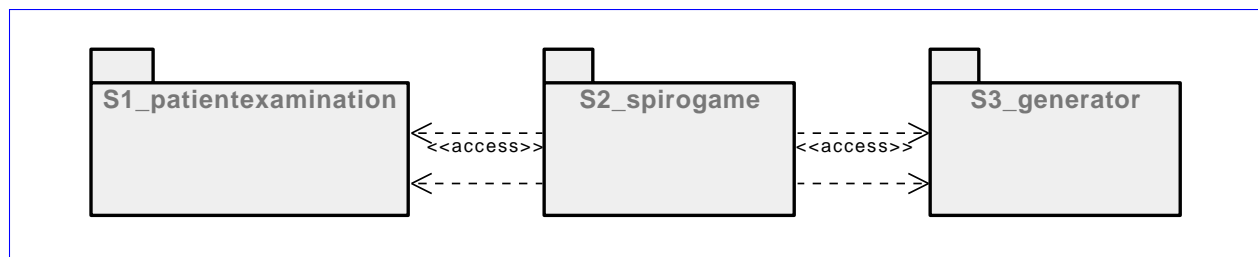
- Data Feeder
- Patient Examination Analysis
- Patient Information Storage

S2: Game System

- Generic game parts
- Specific game parts.
- Cue

S3: Generator System

- Input data
- File input
- Function input
- Data converted to discrete list of values.



S1: Patient Examination System: Has responsibility for the patient data, it is the part that later could be integrated with parts of the patients journal system.

The **PatientExamination** part of the system stores the patients data- It is the system from where the doctors and nurses add new patients, look up patient examinations, set up examination, and input patient information such as weight height and age before the examination itself begins.

The Patient Examination System part takes care of the persistence, while the SpiroGame part just uses the input values from the probe or the data *Generator*.

S2: Game System: The game system is the most important part of the system. The other systems is supporting the game part.

2. The **Game**(SpiroGame) part, where the patent play the game and without noticing it to much! do the lung capacity test or other test, while playing the game.

The Game receives input from the patient via the probe and reacts to it, If the patient needs guidance The game system sends a *cue*, the biofeedback to try and get the patients breathing pattern adjusted. The adjustment depends on the examination type.

S3: DataInput responsible for the input, both the hardware input from the probe and the generator

DataInput is a abstraction for the way data input enters the system.

The generator can create different data sets from a mathematical function. It also is responsible for delivering simulated respiration data, should we not have access to the hardware spirometer instrument.

The **need for the generator** was discovered early in the project to mitigate the risk of not getting access to either the hospital probe or the more modern USB probe that we later acquired. That's why it is already in the domain model, but a case could be made for not including the *Generator* as part of the domain model.

The game might also only use a subset of the read data for manipulation for performance issue reasons. But all the probe data input should be recorded so it can be used in later analysis.

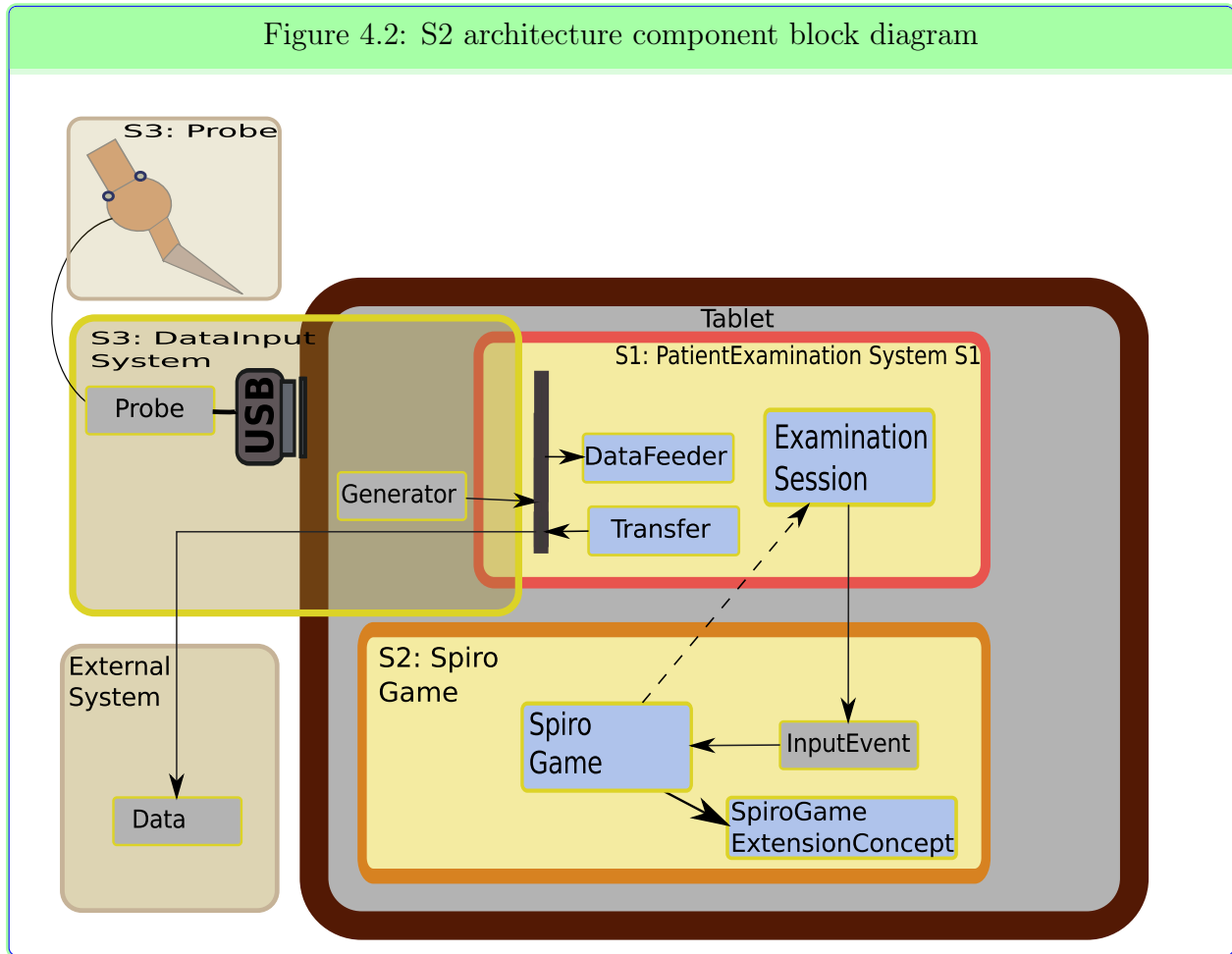
The model shows the system integrated with the Hospital domain.

View the requirements 2.1 on page 13 or the navigation chart in ?? on page ??

4.3.1 Component block diagram

The block diagram shows the systems components and their placements, accord....

Figure 4.1: title



4.4 Revised requirements

See the revised requirements list in appendix: C.3 on page 154

See next section (ref:section domain model)for the system overview.

Tasks in project scope & time.**Implement small prototype for game**

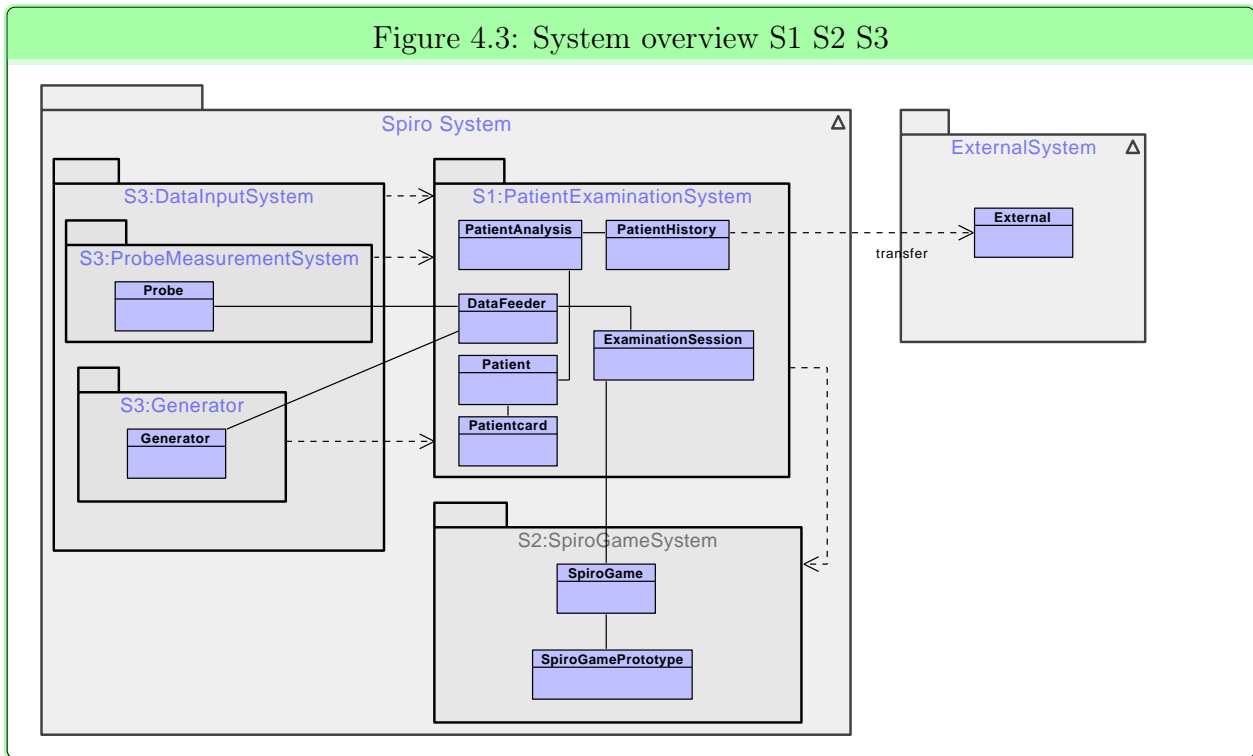
1. Animation
2. Moving position on the screen
3. Draw sprites and background
4. Collision detection

Major tasks in the project scope

1. Measure patient lung function: Flow:
 - a) capacity
 - b) volume
2. Control and adjust to the patients reaction, help the patient breathe in the correct way.
3. Access USB spirometer.
4. Measure Flow with USB equipment.
5. Display Graph visualization of examination for perusal by doctors and nurses
6. Comments for each Examination.
7. Game prototype for the Closed Box examination
8. Publish to Android™ market
9. Integration concerns

4.5 Domain model

?? on page ?? Peter Coad (1999)



Showing the systems with each responsibility: **S1:PatientExamination** is responsible for the patient information, examination procedure and storage. **S2:SpiroGame** responsibility for the game, **S3:DataInput** is responsible for the input data.

The most significant class concepts are shown in the system overview. They can be referenced in the Class description index. F.1 on page 161

The DataInput is divided into the live stream from the probe and a generated and simulated output. Both are sharing data *to* the **DataFeeder**. The DataFeeder supplies data to the ExaminationSession and the Game. The Patient has a PatientCard with information for each examination session or examination date. The patients data is analyzed via the PatientAnalysis. The collected analysis of the patient history is stored in the PatientHistory. The PatientHistory can be transferred to an external system, maybe for better analysis or integration with other information systems at RH.

4.6 Domain class diagrams

Figure 4.4: S1:Patient Examination System Domain

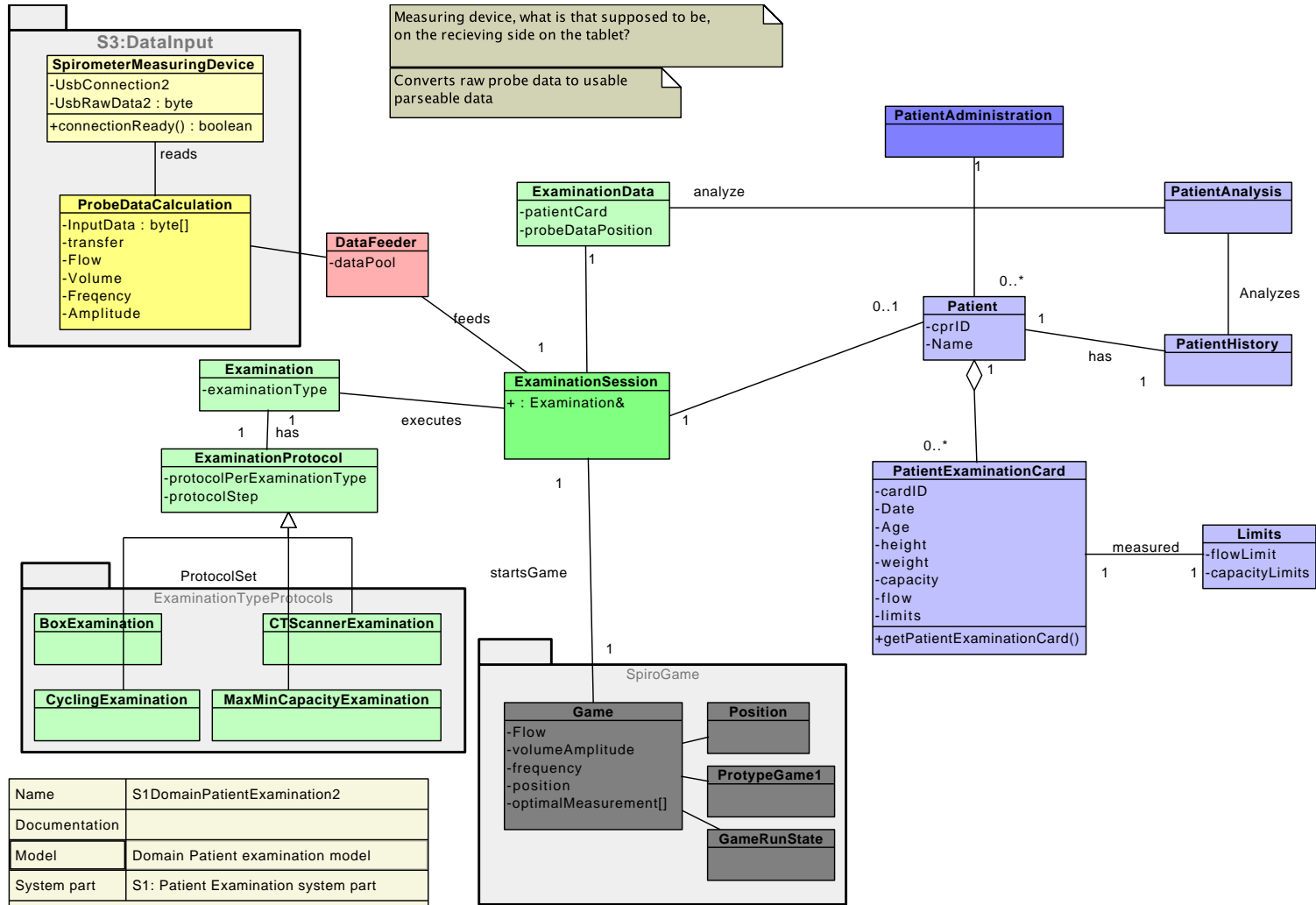
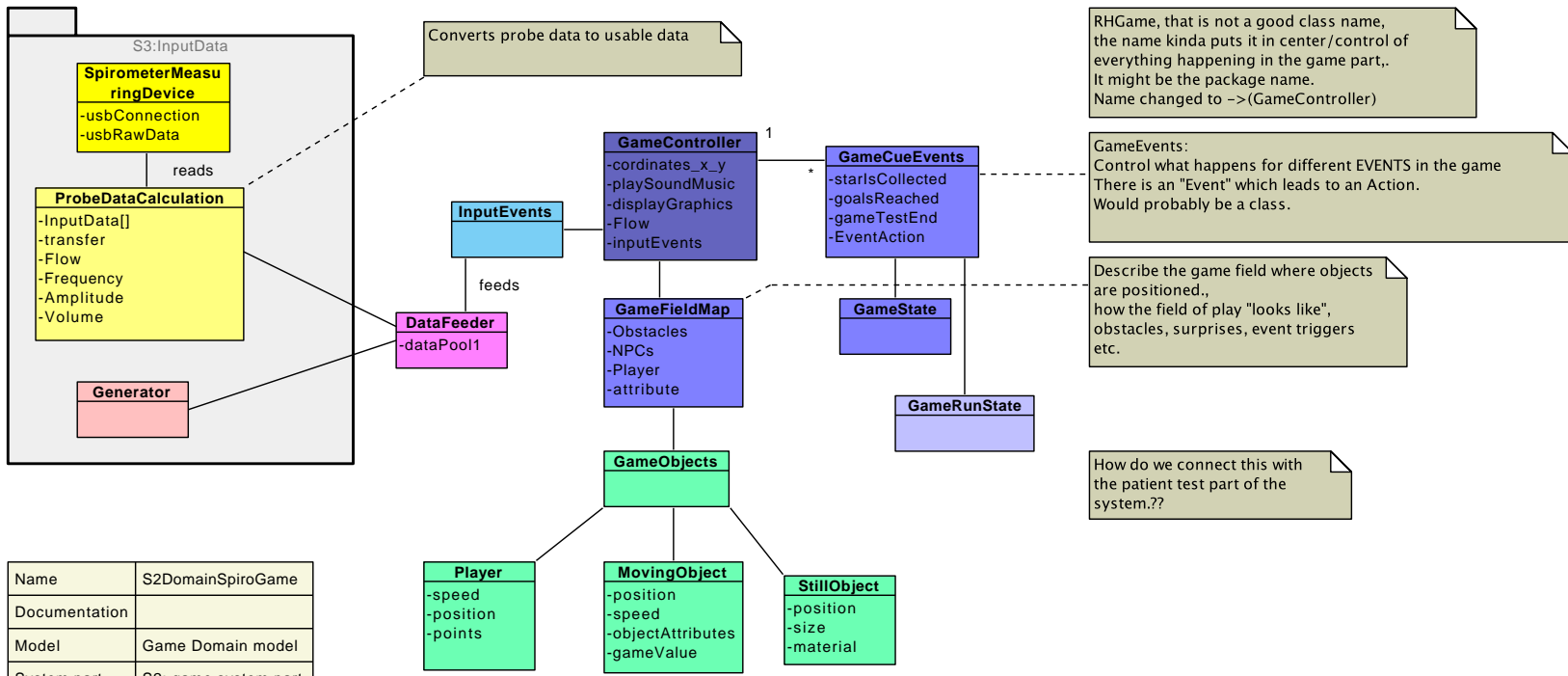


Figure 4.5: S2: SpiroGame Domain



Name	S2DomainSpiroGame
Documentation	
Model	Game Domain model
System part	S2: game system part

The class diagrams shows the concepts used in the project. The **patient system part** is the entry of the patient, the journal and and patient information. After that the examination can start and the examination game starts..

In both the Patient Examination and SpiroGame system we have added the DataInput system S3 to the diagram because the Input affects both systems. The DataFeeder class appears in both S1 and S2 because the systems are developed in parallel.

Coloring: The coloring divides a system after responsibility and behaviour, controlling, description. The description of the color scheme can be found in appendix H on page 169

Chapter 5

Android concepts

5.1 Android: Activity - Intent - Application

Application - Activities - Intents - Context

The 2 most important Android concepts to understand in Android development is *Activities* and *Intents*.

An *application* on Android is generally made of a set of *activities*, where each activity can fulfill an *intent*

We need an activity to run for the user to interact with and for controlling the application, without the activity, there is no application life cycle to take responsibility for resuming and pausing applications. 5.4

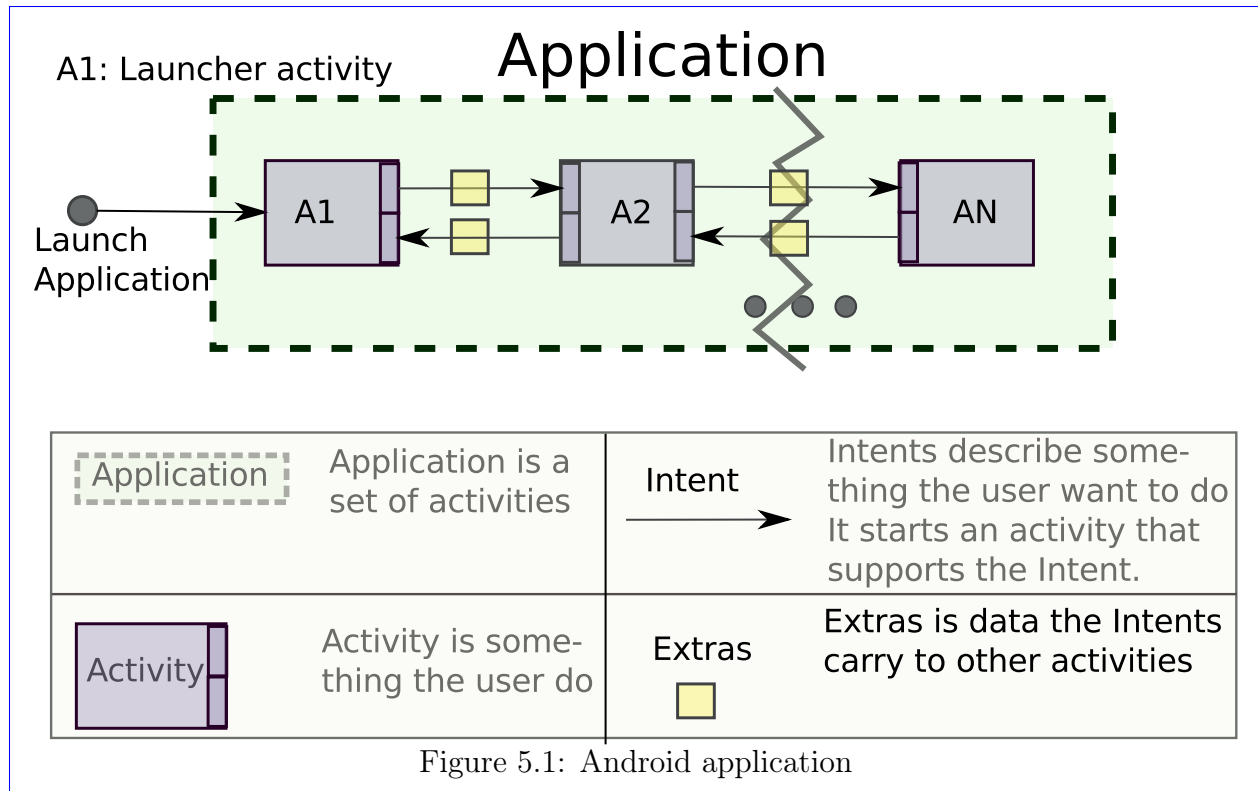
An Intent can be used to communicate between activities to send data between activities, and to start a new activity.

Each activity has a *context*, Which can be seen as the applications environment. Via the *context* there is access to the applications *resources* and *assets*.

5.1.1 Activity

An activity services the user via a UI associated to the activity, through which the user interacts with the application.

The activity is responsible for displaying the UI to the user and for controlling the activity when the user navigates to other activities and it goes to the background.



When an **activity** is displayed on the screen it must be running otherwise it would be unresponsive. When a new activity comes to the foreground of the screen the "old" activity pauses. when the activity is paused the OS can decide to stop the activity.

The activity is responsible for starting, pausing, resuming the app and for saving state of the application. i.e. when the app is closed down, or the device is turned off.

The high level abstraction of an **Activity**, would be asking the question, "what activity your kid doing now", **possible answers:** he is playing, he is reading, he is bathing.

On the phone or tablet devices the activities are called reading mail, reading pdf, phoning, videoing, i mean watching videos.

It is about the user do-*ing* something. So the user have the intent of reading mail, the intent starts up an activity that is capable of reading mail. The user express the intent of phoning somebody and a phone dialer activity is started.

The intents will then start a specific activity, *Playing music* intent could start the *Winamp Music player* activity.

Other Intents could be:

1. Finding public transport
2. Reading feeder
3. Playing music
4. Texting an SMS

Figure 5.2: fig:andr:Intent type examples

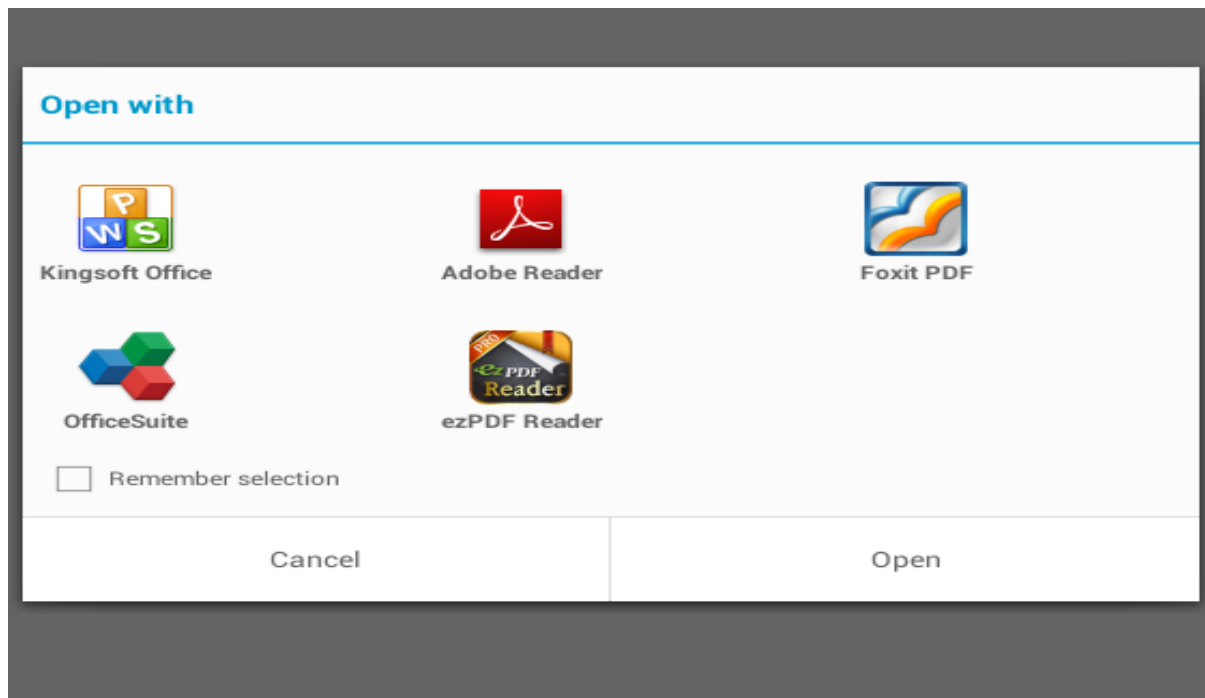


Figure 5.3: Opening a pdf file, the app shows all apps that can work with pdf files, more precisely, the window shows all apps that have registered a VIEW action for pdf types in the manifest file

5.1.2 Intents

Intents are used to start up a new activity, send data to a new activity and for asking for results by another activity. The Intents system is used to decouple applications and activities from each other, this makes it easy to use another installed application for a task, that way the developer don't need to develop a feature for his own app but can reuse other existing activities. One of the most reused components is Google maps which can be invoked via an intent with a URL with geo position data. Google maps are reused in countless applications on the market.

In Android what the user wants to do is expressed via Intents. If the user wants to be *reading* mail, the user has the *intent* of reading mail, so if it is the users Intent to read the mail from

inside an application, he chooses the applications *send mail* option. Then all the applications that can send email is presented to the user via a list displayed on the screen for the user to choose between. The user can choose a default application on the list, then next time he chooses to send mail, the list will not be presented, the application will be routed to chosen default email sending **activity**.

Implicit Intent

Implicit intent is when we request an action" but not a specific application or activity to execute it.

```

1 Intent intent = new Intent("android.content.Intent.ACTION_VIEW" ,
2   "http://www.facebook.com");
3 startActivity(intent);

```

Listing 5.1: implicit intent, calling the URL, with any activity installed ,that can do the action ACTION_VIEW, with a web URL

Here the first parameter is an action request, action "VIEW" is requested on the URL <http://www.facebook.com> Any application that has registered an intent-filter in their manifest that matches the action "view". and also matches the URL full fills the requirements to display the web page, for example the web page can be shown in the default browser on the device, or maybe the user want to display it in a window of another application, for example an application that show the src in one part of the screen and the web page on the second part of the screen.

The activity/activities suggested or chosen are based on the type of activity the category and the data type.

Example: suppose that you have 2 document readers.

Document reader app1, can read, txt, docx and pdf Document reader app2 can read, txt and pdf.

If the user clicks on a pdf "link" both app1 and app2 are suggested. If the user picks a docx document only app1 is suggested.

When you create an application, the capabilities of the application is registered. This is done via the manifest file for each application. described later. 5.3 on page 60

This process is called intent filtering. and (0000)

Explicit intent

An explicit Intent is when a specific activity is called via the Intent. While a implicit Intent could display a list of apps, that can fulfill the implicit Intent.

```

1 Intent intent = getIntent( this , MyEmailClientActivity.class );
2 startActivity( intent );

```

Listing 5.2: Explicit intent, calling my own (MyEmail) email client

The intent is called from an Activity, where the the *this* parameter gives the context of the activity to the intent.

```

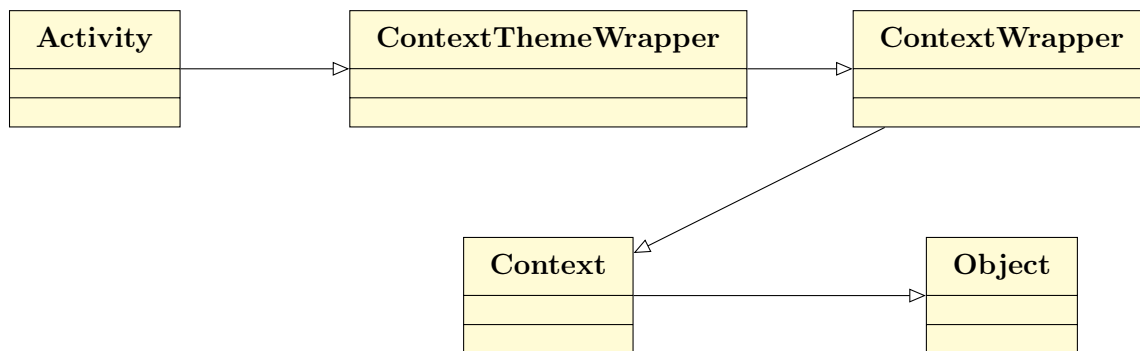
1 Intent intent = getIntent( app.packackage.name.AppClass );
2 startActivity( intent );

```

Listing 5.3: Explicit intent, calling my own (MyEmail) email client

Here you are calling a specific application on the device, already installed. However you have to know the package name and class.

This is not how you would normally call the activity and the method might not be supported because the application developers can change the naming from version to version.



The intent can carry additional information, this information is called *extras*. The extras are used to bundle and package data to and from activities.

5.1.3 Intent Filters

An intent is broadcast to the system, application are registered to the system via their manifest.xml file. The intents that can be handled are described in the manifest file in the `<intent-filter>` section for the activity.

```

1 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2     package="com.example.android.notepad">
3     <application android:icon="@drawable/app_notes"
4         android:label="@string/app_name" >
5
6         <provider android:name="NotePadProvider"
7             android:authorities="com.google.provider.NotePad" />
8     ...
9     <activity android:name="TitleEditor"
10         android:label="@string/title_edit_title"
11         android:theme="@android:style/Theme.Dialog">

```

```

12     <intent-filter android:label="@string/resolve_title">
13         <action android:name="com.android.notepad.action.EDIT_TITLE" /
14             >
15         <category android:name="android.intent.category.DEFAULT" />
16         <category android:name="android.intent.category.ALTERNATIVE" /
17             >
18         <category android:name="android.intent.category.
19             SELECTED_ALTERNATIVE" />
20         <data android:mimeType="vnd.android.cursor.item/vnd.google.
21             note" />
22     </intent-filter>
23 </activity>

```

Listing 5.4: Intent filter

As we can see each registered activity has an intent-filter this manifest is from a notepad application.

The intent-filter nested in the activity "TitleEditor" describes what the activity can do. the Intent filter describes action categories and data types that the activity can work with.

When an (implicit)intent object is send it is filtered by the intent filters registered to the activity in the manifest files. The intent object is first checked if it can full fill the action f.ex. ACTION_CALL \Leftarrow can the activity phone out. Then category is checked, and implicit intent must have the DEFAULT category set otherwise all implicit intents are rejected.

5.2 Activity life cycle

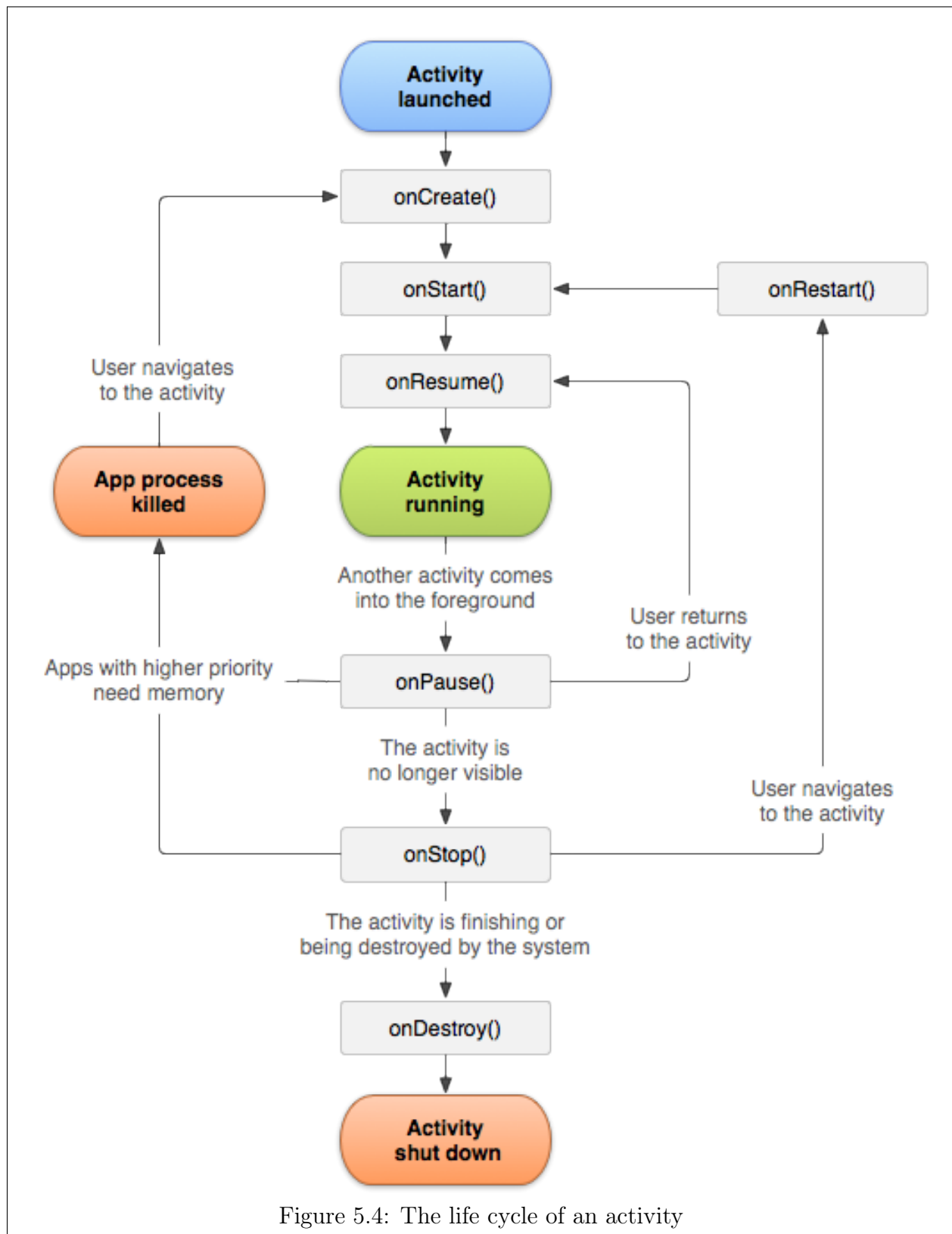
The Activity is the basis for your application, it controls what happens when your app is started for the first time on your device.

The idea of the activity , is based on the assumption that the mobile device has a relative small screen. and therefore each activity must be optimized for the related task.

If your application looses focus because a new application/activity starts up, the activity is responsible for saving needed state.

The Activity is responsible for saving the state of the application, f.ex when a calls comes in your application loses focus, and might be teared down by the OS. i.e. if you talk for a really long time. and when you end the call you have lost some of your work, it could be you were in the middle of entering a new contact to your contact book and only confirmation of the data were left when the phone ringed, when you get back the data you entered in the contact page is gone.

Another example is if the phone or tablet is flipped and change view mode between portrait and landscape the Activity is actually teared down and restored, and entered data is erased. this is seldom what the user wants.



If views are designated an ID it should automatically be saved and restored by the life cycle system.

When the application is started the first time the following methods are called.

onCreate - onStart - onResume

if the application is paused onPause is called just before System pauses. So state information must be saved here. After the system is paused, the Android OS can prioritize what should happen to the application. the OS can stop the app if the app hasn't run for some time or if resources are needed.

If the app is paused the OS can choose to stop it. Which means it is no longer scheduled to run, it is removed from the run queue.

OnDestroy If the app is stopped, it can be destroyed. which means its process is removed from the system.

The important thing to note is that you are guaranteed to run through the described callbacks in order.

If onCreate is called, onStart is called, after that onResume is called. which means you can setup your app in these steps. if your app is running you cannot destroy the running app. you have to pause it first, then stop it and then destroy.

This is guaranteed, so you can save your state before your app is removed, and that you can restart sound and external connections when resuming(onResume).

5.3 Manifest file : manifest.xml

Each application has a manifest file, the manifest.xml file setup the the application. Defines the activities and intents the app exists of. The actions it can handle.

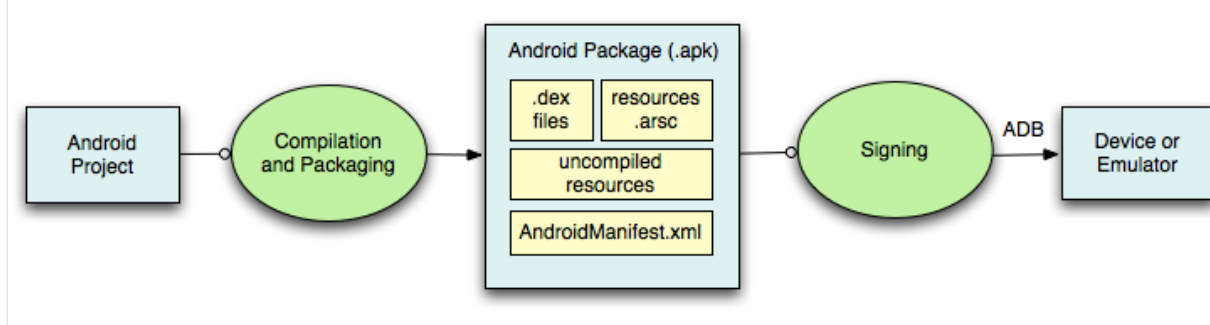
The documentation it a bit vague on how the manifest is used when transferred from the development workstation over to the Android device, not explaining exactly *how* is mapping from the manifest file to the resources. However each element used in the setup is well documented.

The manifest can be seen as a filter, where the application registers it's capabilities: where the capabilities can be anything, :

Permissions, what action the application handle, phone, write pdf, read pdf, display WWW links, access to external storage(SD card), what android versions to use and be compatible with, screen orientation, access to sensor, USB access and more.

are compiled

Figure 5.5: Android build apk package



```

1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="net.micrun.animationscrollsurfaceview"
4     android:versionCode="1"
5     android:versionName="1.0" >
6
7     <uses-sdk
8         android:minSdkVersion="16"
9         android:targetSdkVersion="16" />
10
11     <!-- android:screenOrientation="landscape" : landscape all application -->
12     <application
13         android:allowBackup="true"
14         android:icon="@drawable/ic_launcher"
15         android:label="@string/app_name"
16         android:theme="@style/AppTheme" >
17         <!-- <activity android:screenOrientation="landscape" : landscape for
18             this activity -->
19         <activity android:screenOrientation="landscape"
20             android:name="net.micrun.animationscrollsurfaceview.FlyActivity"
21             android:configChanges="orientation|keyboardHidden|screenSize"
22             android:label="@string/app_name"
23             android:theme="@style/FullscreenTheme" >
24             <intent-filter>
25                 <action android:name="android.intent.action.MAIN" />
26
27                 <category android:name="android.intent.category.LAUNCHER" />
28             </intent-filter>
29         </activity>
30     </application>
31 </manifest>
  
```

Listing 5.5: Manifest example: AndroidManifest.xml

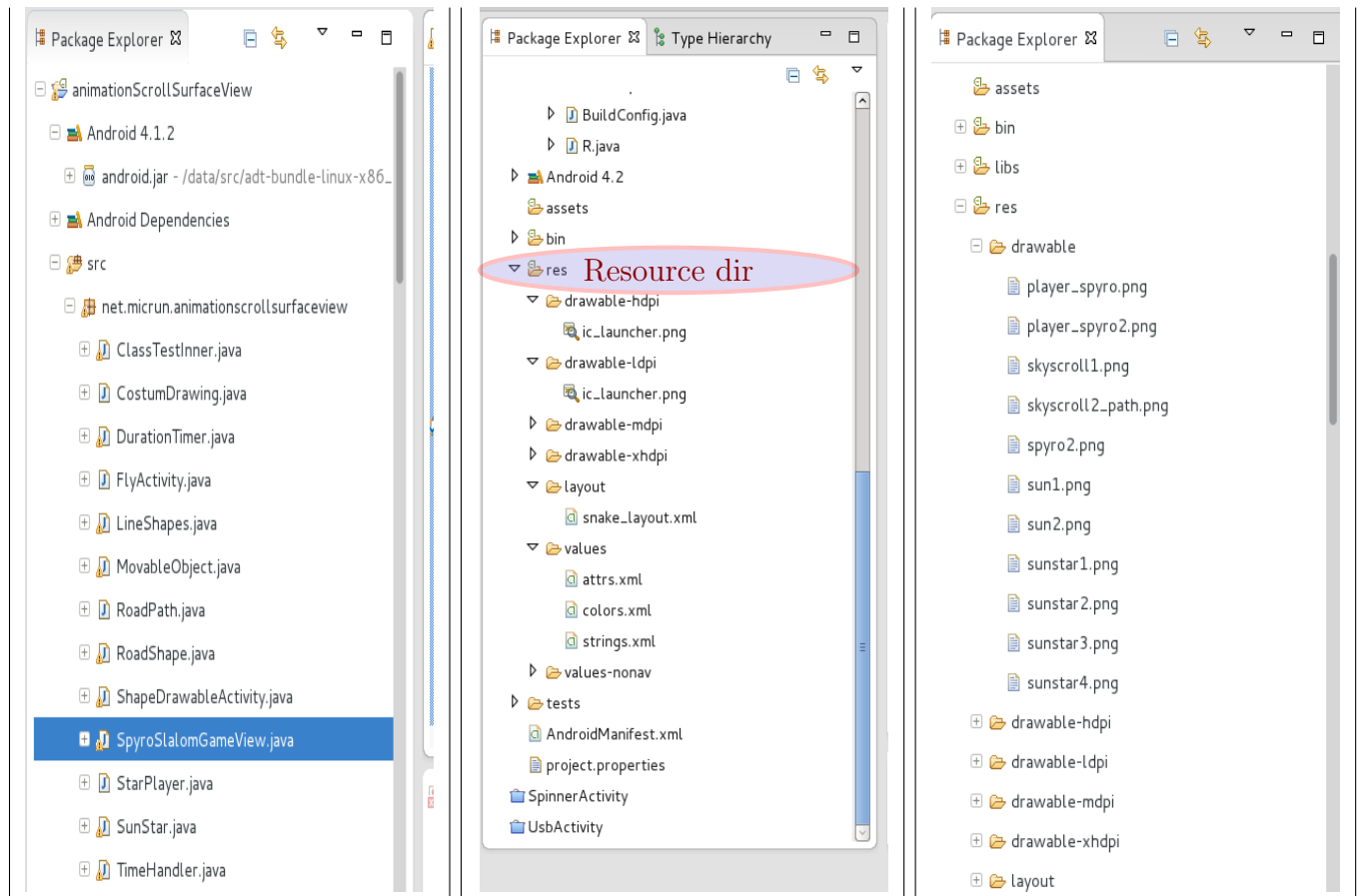
All the resource and assets files and class files and the manifest is packaged into an apk file.

The figure shows how the compiled **apk** packaged is structured. This happens before the apk is transferred to the device. The Manifest is the "map" that shows where and what resources

are available for the application.

5.4 Android Structure, Assets resources and layouts

The Android setup for developing has a flat file tree structure for the types of resources, so the file structure cannot be nested. If a nested file structure is needed it can be placed under the assets folder.



(a) a) the src directory and files folders (b) the assets and resource directory (c) a list of the basic drawables for the games (bitmaps etc) structure

Figure 5.6: The structure of the project tree, directories and files

5.4.1 Assets

An asset is a raw resource, there is no packaging/compression of files as the files under the *res* resource folder. So raw data files that doesn't need to be processed can be placed here.

Assets are accessed via the assets manager. Official (2013) Assets are accessed by file name.

EXAMPLE:

```

1 %Resources.getAssets
2 public String readFromAssetsFolder(Context c, String dir, String filename){
3
4     if (dir.equals("") && filename.equals("")) {
5         dir = "probedata";
6         filename="spirobank_trials_export.txt";
7     }
8
9     AssetManager am = getAssets();
10
11     String [] listFiles=null;
12     try {

```

```

13     listFiles = am.list(dir);
14 } catch (IOException e) {
15     e.printStackTrace();
16 }
17 ....
18 /// get Assets via Assets Manager : am
19 String str = listFilesBuilder.toString();
20 InputStream is = null;
21 try {
22     is = am.open(dir + "/" + filename);
23 } catch (IOException e) {
24     e.printStackTrace();
25 }
26
27 BufferedReader br = new BufferedReader(new InputStreamReader(is));
28 StringBuffer sb = new StringBuffer();
29 String readLineString;
30 try {
31     while ((readLineString = br.readLine()) != null){
32         sb.append(readLineString);
33     }
34 } catch (IOException e) {
35     e.printStackTrace();
36 }
37 ...
38 /// Return assets manager file as string list
39 return sb.toString();
40 }

```

Listing 5.6: Accessing assets

On line 9 the assets manager object is created, line 13 listing files in directory probedata/ (line 5), Line 22 the file assets/probedata/spirobank_trials_export.txt is opened, on line 27-33 its appended to a StringBuffer.

```

1 /// read from raw resources under res/raw
2 public String[] readRawResourceFile(Context context, Resources res, String
   filename) {
3     int rId = 0x7f090001; /// cheating by looking it up in R.java file
4     String resName = context.getResources().getResourceName(rId);
5     String resPkgName = context.getResources().getResourcePackageName(rId);
6     String resEntryName = context.getResources().getResourceEntryName(rId);
7     String resTypeName = context.getResources().getResourceTypeName(rId);
8     String pkgName = context.getApplicationContext().getPackageName();
9     String pkgResPath = context.getApplicationContext().
   getPackageResourcePath();
10
11     Log.v("DisplayData", pkgName + "\n" + pkgResPath);
12     Log.v("DisplayData", resName + "\n" + resPkgName + "\n" + resEntryName + "\n"
   "+ resTypeName);
13
14     /// omit extension of filename
15     /// :TODO: make function that strips extension if the String has it.
16

```

```

17 // filename var for spiro
18 int id = res.getIdentifier(pkgName + ":raw/" + filename , null , null); //
    defType, defPackage); //
19
20 // Should be inefficient accessing via name , should be accessing by id.
21 InputStream is = res.openRawResource(id);
22
23 byte[] buffer = new byte[1024];
24
25 int i=0;
26 BufferedReader br = new BufferedReader(new InputStreamReader(is));
27
28 // StringBuffer is synchronized
29 //StringBuffer sb = new StringBuffer();
30 // StringBuilder is **NOT** synchronized
31 StringBuilder sb = new StringBuilder();
32 ArrayList<String> lines = new ArrayList<String>(100);
33 String readLineString = null;
34 try {
35     // CHK: TODO: we are not using StringBuilders Strengths here?
36     while ((readLineString = br.readLine()) != null){
37         lines.add(readLineString);
38     }
39 } catch (IOException e) {
40     e.printStackTrace();
41 }
42 //List<String> list = ..;
43 String[] strArray = lines.toArray(new String[lines.size()]);
44 return strArray;
45 }

```

Listing 5.7: Accessing raw resources

Reading a raw resource, simulating that only the filename of the resource is know, the context is used to get the resources for the application. Then we get the package names form the resources append the filename and then get the resourceId.

5.4.2 Resources

Resources are assets that are "compiled" and packaged into the application, the exception is files under the *xml/* and *raw/* folders.

- Images
- Sound bits
- Music
- Strings resources
- localisation
- **etcetera**

They are accessed via the Resource manager.

```
1 // from an activity class
2 Resources res = getResources();
3 ...
4 // from a non-activity class, with the context passed
5 Resources res = context.getResources();
```

Listing 5.8: Accessing resources, and finding a resource by ID

From the resource reference *res* we now have access to all resources associated with the application.

5.4.3 XML layout to view instance

A User Interface can be build declaratively via the xml files. the layout files are stored in the resource folder *res/layouts* When the application launches, *onCreate* callback is executed. The method *setContentView(layoutView)*; is called, this reads the packaged layout xml files and instantiate them as java objects. and are ready for display. The views cannot be manipulated before *setContentView* is called because the java objects doesn't exist before that time.

5.5 The Android Pattern

The Android pattern resembles MVP : Model View Presenter or MVC. The Android framework has it own structure that divides the responsibilities

The activity is the life cycle control and fit into part of the role as the controller/presenter in a Model-View-Presenter pattern.

The Views are usually instantiated from the Activity class, this means that the separation of concerns is somewhat restricted in the standard practice of how a project is build.

The Model is then the data model to be presented in the view. The data model updates the view data via the presenter/controller and back again.

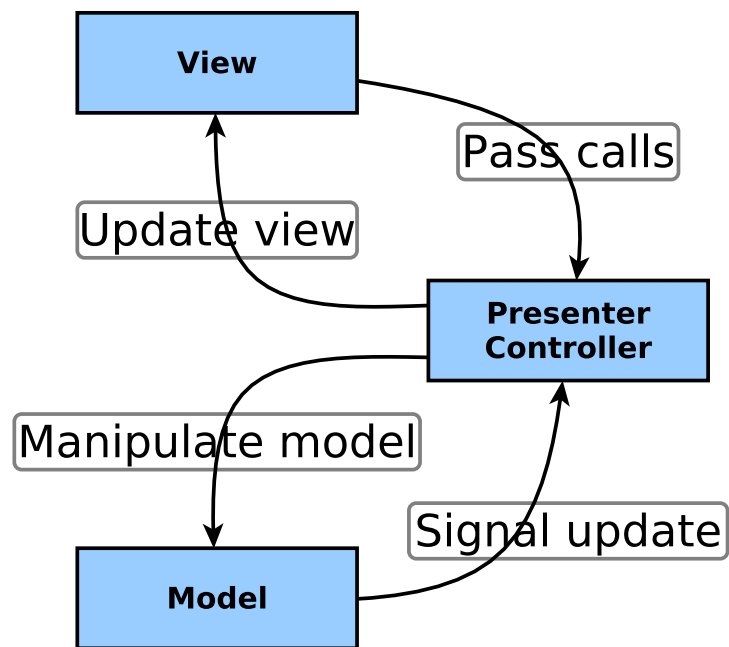


Figure 5.7: Model-View-Presenter/Controller

Part II

Design

Chapter 6

Design

6.1 System overview

The 3 systems S1 S2 and S3 are described in the analysis section. ?? on page 48

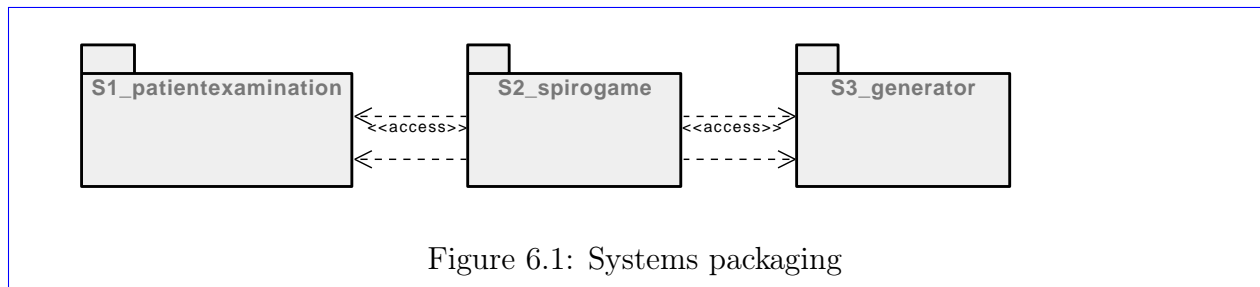


Figure 6.1: Systems packaging

The systems S1-S2-S3 that makes up the complete project. The reason for system division is explained in the project scope chapter where the project is partitioned in a hopefully efficient way, that separates concerns and isolate errors or other mishaps from influencing the continued development on the other systems. The systems should in large part be able to be developed individually.

:ref:

6.2 User interface & Mockups

6.2.1 Mockup drawings

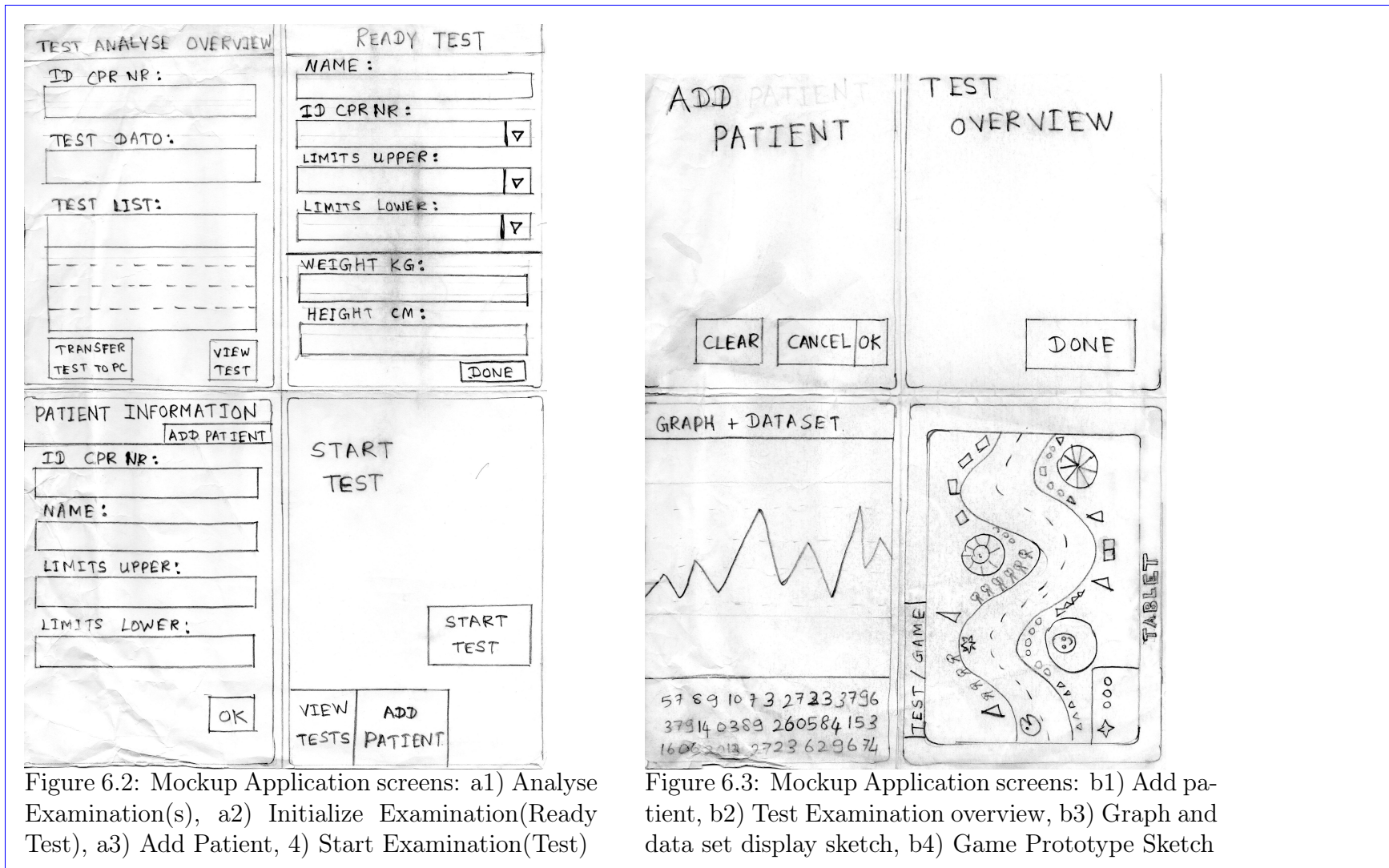


Figure 6.4: The figures are numbered from top left to bottom right

12
34

6.2 on page 72 a2) Ready Test/Examination, is the application start page, Starting a test is the default user activity, because most of the time the user wants start an examination, next screen is the examination session/game b4).

6.2 on page 72 in a1) *Test Analyse overview* patient and examination can be chosen from a list of examinations sorted by date. When the examination to view/analyse is chosen we move to graph display window.

6.3 on page 72 b3) This is the graph display to visualize and print data recorded from the patient respiration in to the probe(flow measurements). The graph display is created so we can visualize data early in the development phase, and help us to analyse behaviour in the game prototype later, with regards to the input received. We can reuse this without the first screen.

The *patient add page a3*), and *confirmation page b1*) that is used to visualize the special steps needed in the system to add a new patient. this should help us understand what the "requirements" and the needs are when a new patient arrives at the hospital.

The *confirmation screen pattern* is useful for touch devices, after the data for the patient is entered an overview page with the supplied info is displayed, and the operator can quickly see if things are correct before confirming the data, or go back to change information. the confirmations screens are a4) Start Test overview, b1) Add patient overview, b2) Test(Examination) overview.

6.3 Screen navigation diagram

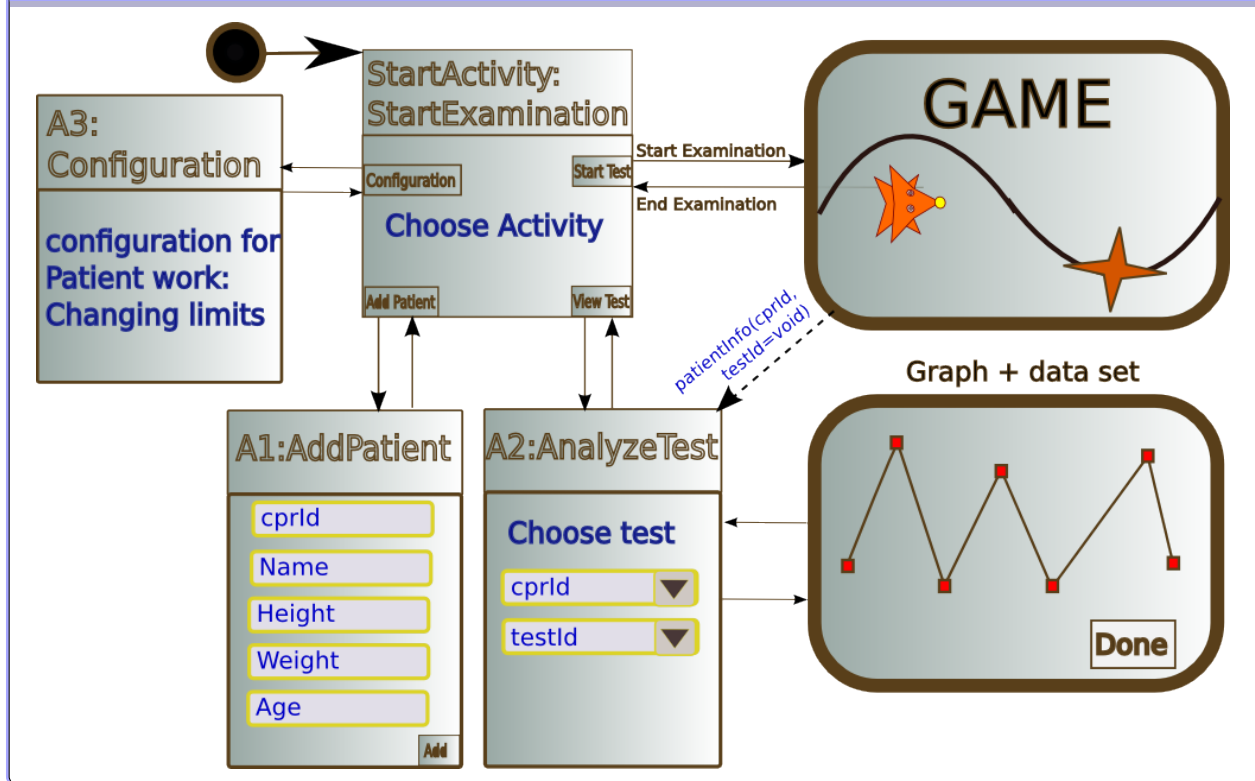
The main activity is where the application starts up (Launcher activity), from here we can navigate to our designated activity, *adding new patient (A1)*, looking at patient history from the *AnalyseTest (examination) activity (A2)*. *Change configuration (A3)*, change flow and capacity limits for patient. So the examination can use these measurements to ensure challenging but not too hard tests/examinations.

6.3.1 Basic navigation flow concept

The application follows a simple navigation flow pattern.

- The user navigates from the main screen,
- choose the wanted activity, on the chosen activity screen,
- choose object and set constraints (Object=the patient, constraints=upper and lower flow limits)
- A confirmation screen, overview of the chosen input data.
- then the activity that uses the input object and constraints, the game, and the graph data set display.
- When finished the user returns the same way back, and then reaches the MainActivity where a new action can be chosen.

Figure 6.5: Application UI navigation diagram: Basic navigation mockup screens: a) StartActivity, b) A1: addpatient activity, c) A2: Examination analysis, d) Game screen(S2) e) Graph & data set screen



There is another alternative flow after an examination game. Going directly to analyse test to the examination results, this can be used to check if the examination results are valid and useful.

6.3.2 Mockup lists

1. AddPatient: Add New patient
2. PatientConfiguration: Change patient information limits (Flow, Capacity limits)
3. GameConfiguration: Game difficulty speeds , extra: how would that work
4. FindPatient: find patient history : Analysis
5. StartExaminationGame : start Examination : StartGame
 - a) StartGame Screen
 - b) Game Information help screens(plural)

6.4 Separating and isolating system parts

6.4.1 Game and examination separation

The **game** is part of the examination protocol, the **Examination Session** is where the *examination protocol* begins and ends. 4.4 on page 47

Figure 6.6: ExaminationSession to GameController

Protocol start - > initial examination preparation - > ready to start test
 - > go to game ready - > start test/game, record probe data from patient
 - > end examination - > confirm - > end game.

6.5 Generic and specific game concepts

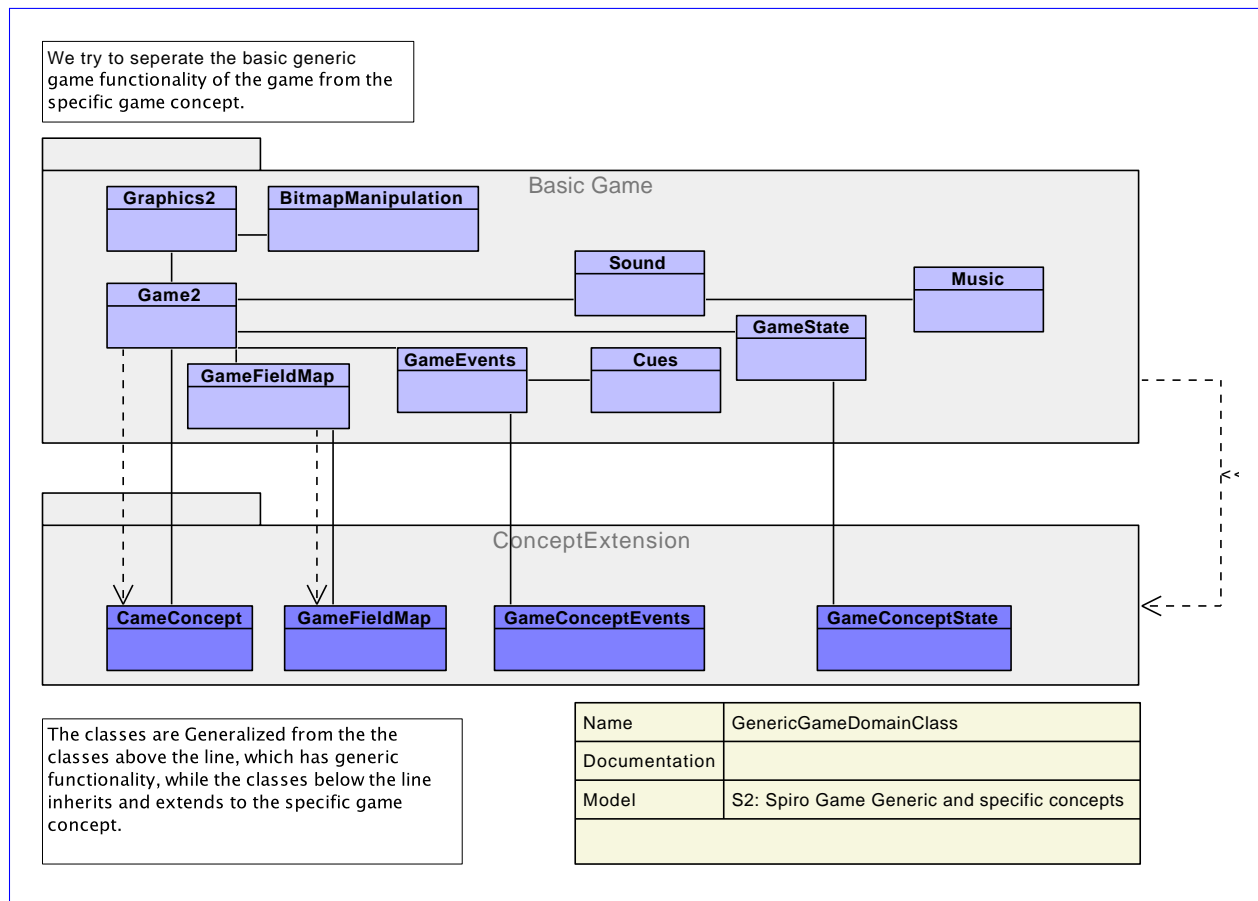


Figure 6.7: Game Top level class: generic and extended specific game concepts

The Game design is split up so specific prototypes can build upon the basic game concepts. Each prototype is an extension of the basic game concept design/impl. The diagram shows the classes that are suited to be extended and provide Functionality to the extension games.

The basic game implementation deals mostly with the game part that directly supports the examination of the patients.

All prototypes or games are an abstraction of one(or more) of the examination types. The extensions should make it possible to add almost any type of game on top of the generic system. Providing a passable variety of games both for new examination types but also games where the input probe is just working as another type of input, like a joystick, screen touch input and so on.

The basic game skeleton provides object position, speed, events from probe input, events and Points etc in the game.

6.6 Input and conversion

The **DataInput** system (**S3**) is responsible for providing data from the probe or the generator, The data input is then available for the DataFeeder class.

it would be natural to put the **DataFeeder** in the **DataInput** system. But it is placed both in the patient examination system and in the game system.

The reasons are:

1. The DataFeeder is not directly related to the hardware probe or the generator. It is there for providing the input at the correct intervals and provide pre-calculated data like frequency, amplitude and other values to the examination session & game.
2. The systems are developed separately and the DataFeeder concept is just on the edge of the system and for simplicity it's concept is used at each developed subsystem, also because the subsystems are developed in parallel

We need to measure:

- Frequency
- Amplitude
- Direction
- Speed (speed vector)
- Acceleration -> for what
- Changing direction, bottom top(local minimum maximum on a curve)

Values are derived from the probe data and calculated into the specific value type(speed, ampl)

The data from the probe device are converted and recalculated into frequency, Amplitude and the other values in the list.

Taking multiple samples over time to decide flow. Some action and events much be watched over time, observing if patient interaction tendency continues. If the patient just gets a out of rhythm for an instance the **Cue** might be unnecessary or might even confuse or disturb the patient.

When some times has passed and a tendency has kept on for some time the System Observer decide it is a pattern. For example if the patient breathes to slowly we don't want to interfere before we see that the patient doesn't get back on track.

If the patient performs stable and good performance the game must encourage the patient to continue that behaviour

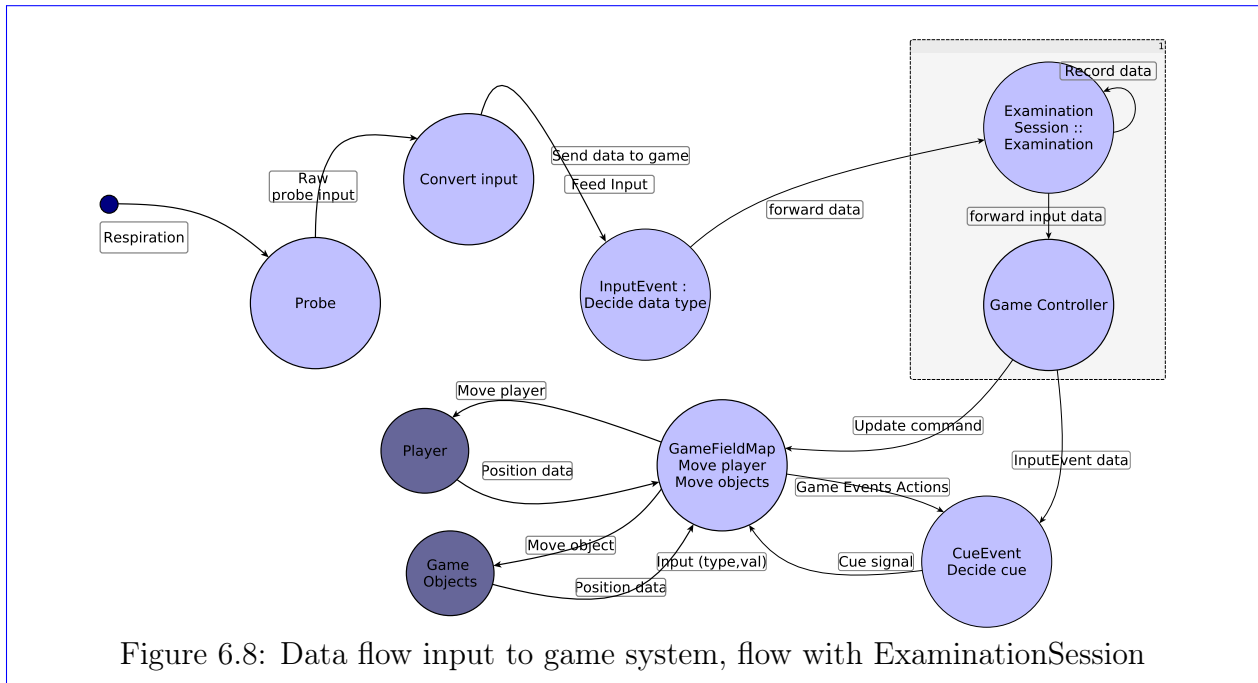


Figure 6.8: Data flow input to game system, flow with ExaminationSession

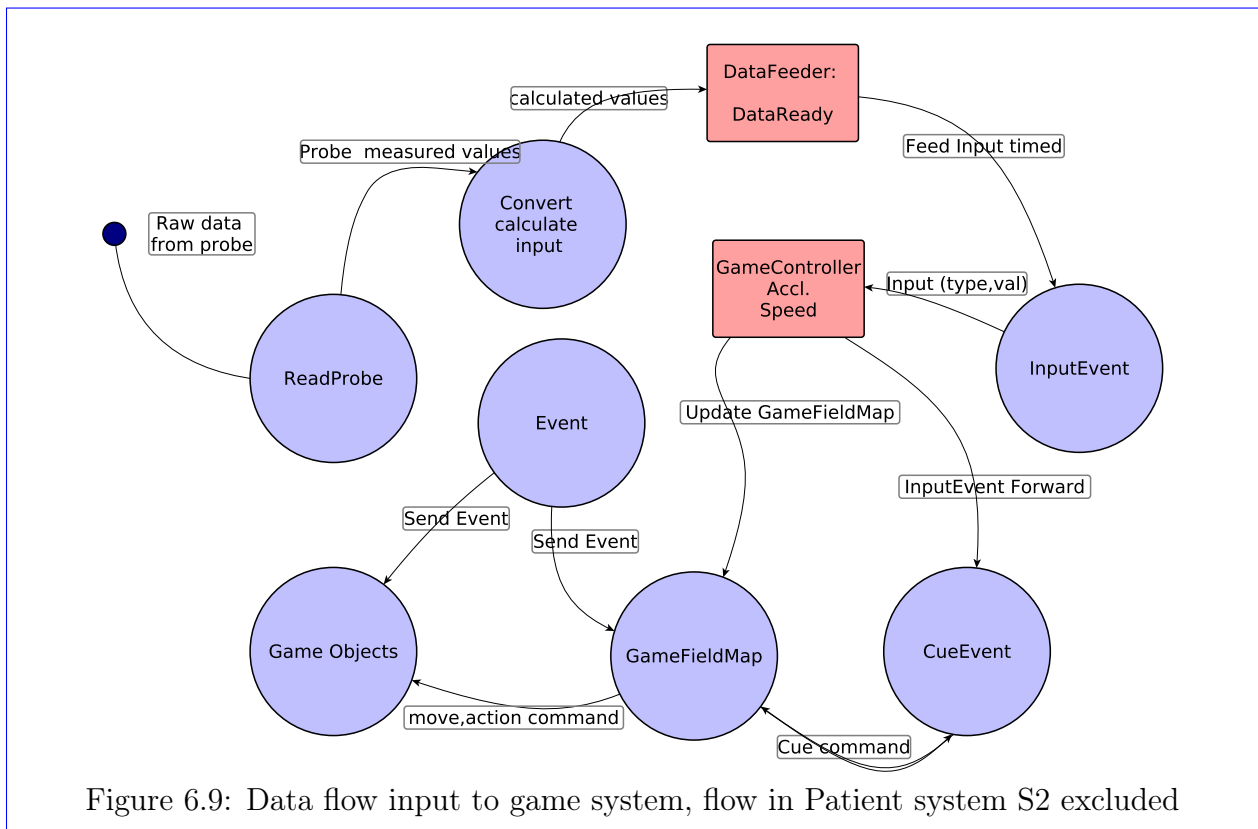
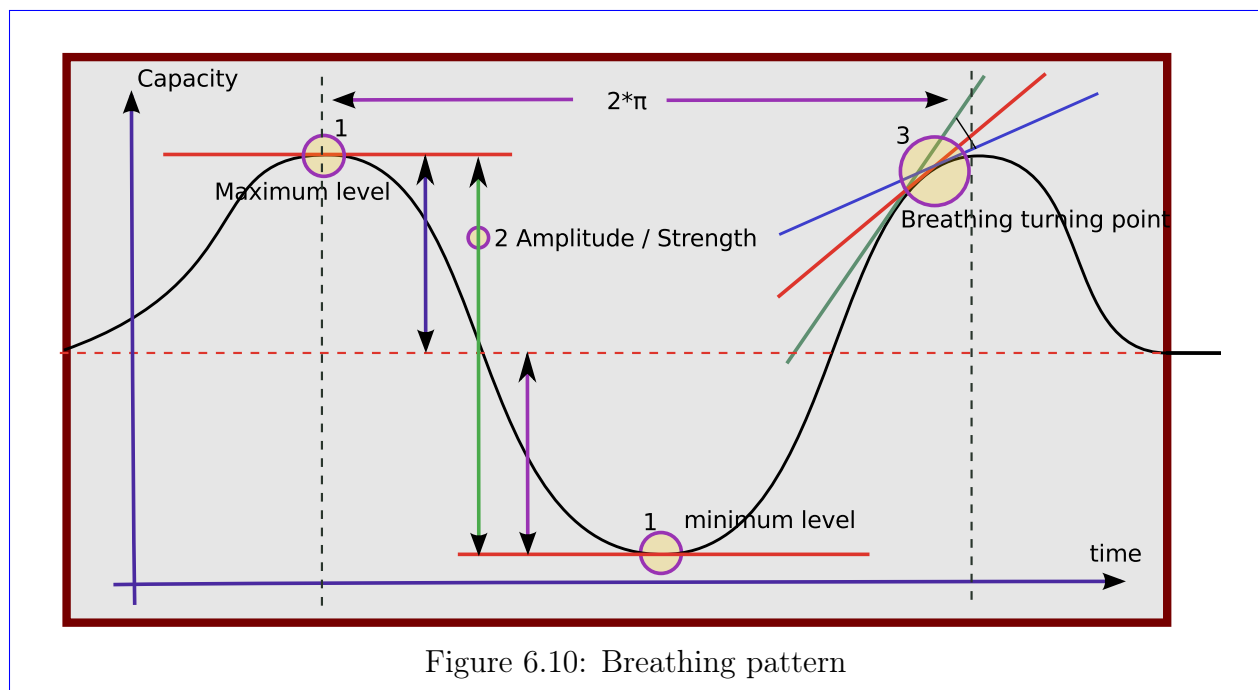


Figure 6.9: Data flow input to game system, flow in Patient system S2 excluded

The data flows from the probe and into the *game* and *examination session*. Conversion of raw bytes from probe to flow frequency and amplitude numbers. followed by calculation of speed, acceleration and timing information. After those calculation the data "arrive" at the DataFeeder. The DataFeeder supplies the InputEvent entity with data at timed intervals. The InputEvent entity supply the game with specific value,types (acc, speed, frequency amplitude). The GameController compares the InputEvent data values with the game state and send a cue to GameFieldMap process that forward it to the correct game object.

6.6.1 Limits for patients: patient configuration

The game must use the flow and capacity limits of the patient to set the boundaries for the optimal path. The limits are then adjusted to the screen size of the tablet. all players move on the same space on the screen the limits of the patient are scaled to fit the screen dimensions and division.



The graph is showing a breathing pattern. On the Y-Axis is the breathing volume capacity on the X-Axis is the time passed.

The graph shows the maximum top and the minimum bottom at the two small yellow circles. To the right the larger yellow circle shows 3 lines indicating the slope, this can be used to calculate the turning point before it happens. Another way is to sample and see when the values change, but that would be after the fact, with the slope approach we can calculate the turning point time before it happens when the slope approaches zero.

The specification states 35 BPM(Breaths Per Minute) for the examination is an optimal breathing frequency.

$$T = \frac{60s}{35BPM} \quad (6.1)$$

$$(6.2)$$

$$\frac{2 * \pi}{T} = f \quad f = \frac{2 * \pi * 35}{60} \quad (6.3)$$

$$(6.4)$$

$$f = \frac{70\pi}{60} \quad f = 1.1667 * \pi = 3.66 \quad (6.5)$$

$$(6.6)$$

$$f = 0.5833hz \quad (6.7)$$

$$(6.8)$$

The breathing frequency is about 0.58. That corresponds to a a high scrolling speed in prototype 1. which means that the prototype 1 might have a weak point because the patient must move very fast from top to bottom of the screen to pick up the sunstars.

6.6.2 Events state and cues

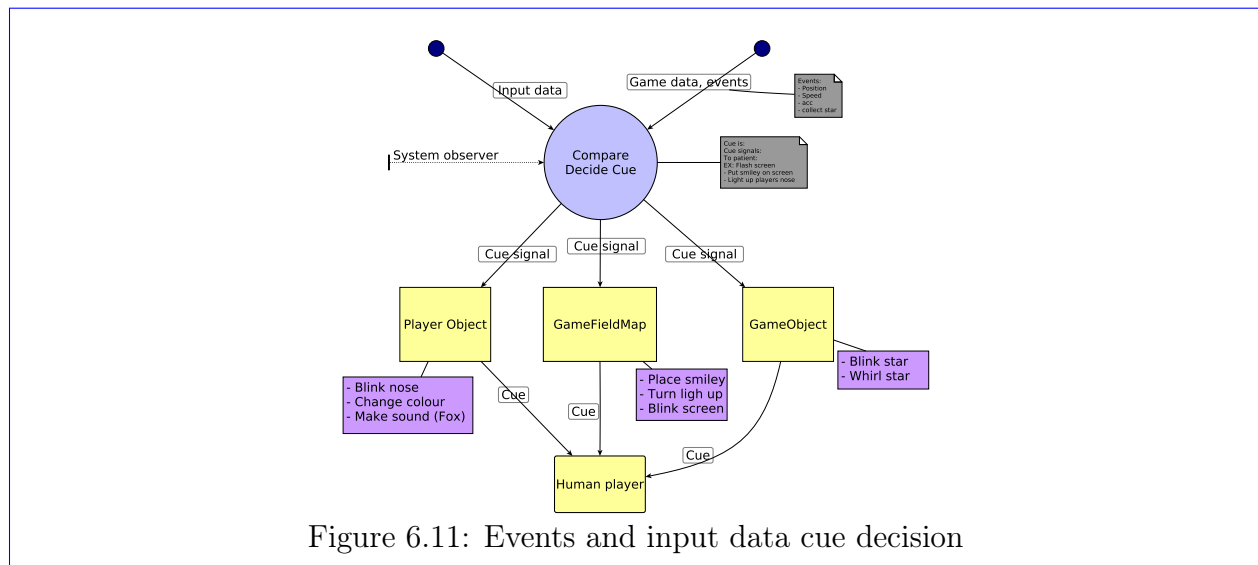
The *Concept* cue is defined as: A cue is like the a hint to do something, on theatres the actors waiting to get on stage are waiting for a Cue, "That's my cue". Thus an appropriate *cue* is selected according to the recent behaviour and actions of the patient/player.

First some basis cues are defined later new Cue types can be added by extending the table of Cues and events or by extending the **Cue** class.

The design of *Cues* is not completely finished, the first cue model is a simple event model in the game.

Cues:

- Blow faster
- Blow Slower
- Blow softer
- Blow Harder
- Breathing irregularly, relax
- Not Breathing, go go go
- Doing Fine, keep going.



6.6.3 Input data game event and the System observer

We can see that the input data, the actions from the patient and the game actions and event data, what happens in the game, flows into a process that compares the incoming data and decides if a cue is necessary, if it is a cue signal is send to the corresponding object.

The process that observes this, is the *system observer* concept, the *System observer* watches the incoming actions and decide if a cue is needed.

Cue table				
Cue	Cue signal	Reaction	Time/or repetition	Text description
go back to path	Twist eyes	Yes/no		
Good behavior	Blink nose, flip tail	Yes/no	time=60	
Breathing to fast	Sound(Slow down)	Yes/no		
Breathing to hard	Sound(take it easy)	Yes/no		

Table 6.1: Cue table

Table Legend:

Cue The purpose of the cue

Cue signal The cue "appearance"

Reaction Is the patient behavior changing

Time or reception Set a time for the cue trigger event

Text description

The Cue table is used to check if events in the game or for the player requires correction, it checks if a behaviour of the player continues for some time or a behaviour repeats, including "good" behaviour. The Cue table decides if a **cue** for the user is needed, if it decides that a cue signal is needed, it is send to the game/player object or the GameFieldMap, which is then displayed for the patient player.

The flow diagram shows how input data from the patient and games events are going into the **cue process** and used to decide the cue signal to be send.

6.7 Design architecture

6.7.1 Architectural requirements

Table 6.2: FURPS+ Architecture requirements

Extensibility	Extensibility and ease of use for building new games	
Reliability	No	
Scalability	Not needed	
Integration	Integration of data exchange with other RH systems desired	
Storage	Storage of data on SD card is not protected	Needs encryption
Security	Patient data is confidential, and must be kept secure	There a laws hat needs to be followed

Figure 6.12: Architecture overview diagram: MVP

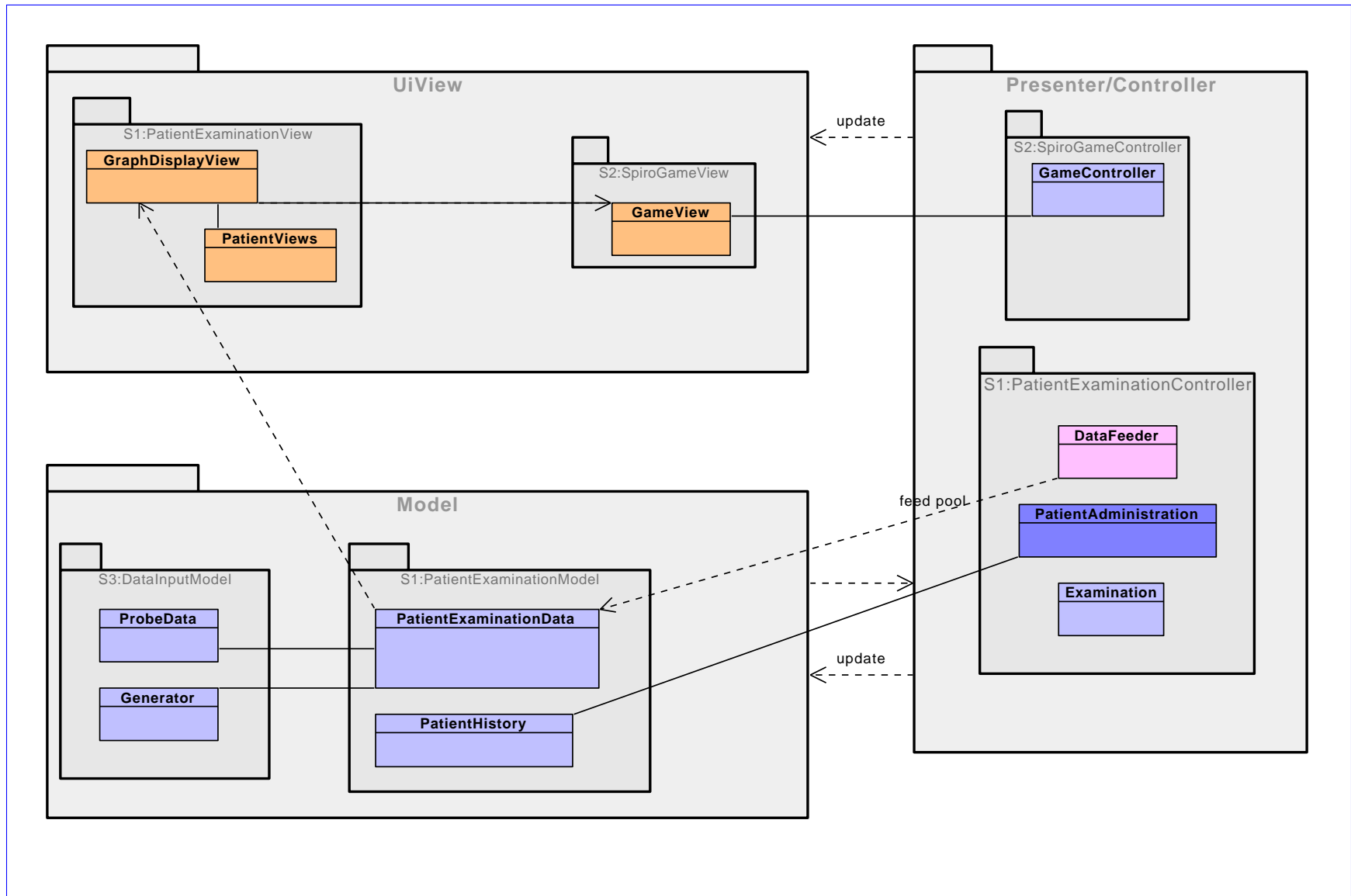


Figure 6.13: System architecture

The architecture follows the MVP/MVC design pattern architecture as described in the **Android concepts** chapter. 5.5 on page 66

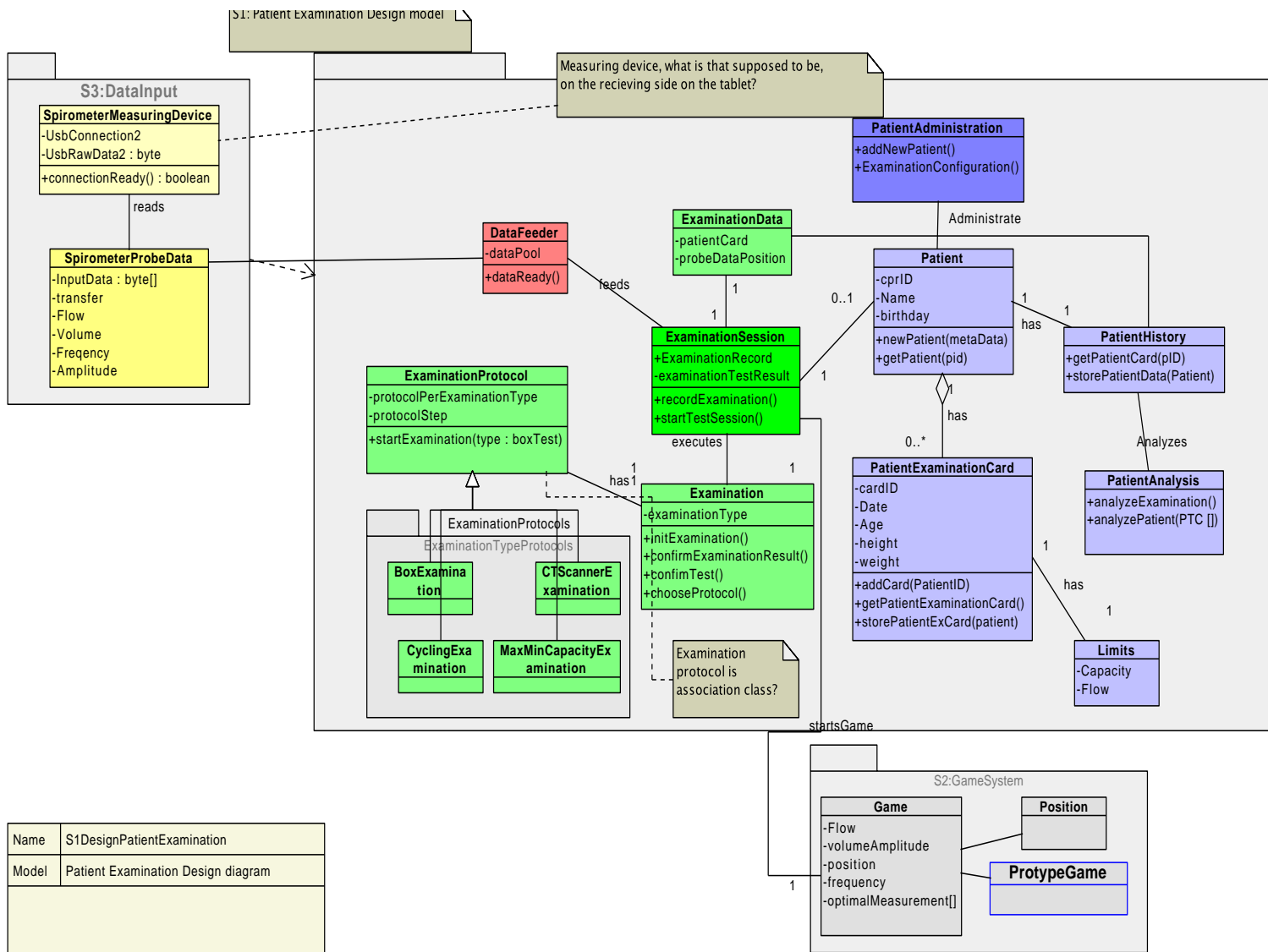
The **orange** package system is the **View**, **Blue** package is the **presenter/controller** package and the **Green** package is the **Model**.

The architecture make the application communication simple, **Model** updates view via the **presenter/controller** that tells the **View**, the **View** asks the **Presenter/Controller** to update the **Model**. The **Presenter/Controller** has the business logic, it manipulates the data and controls how to store and update the model and how to represent the data to the view.

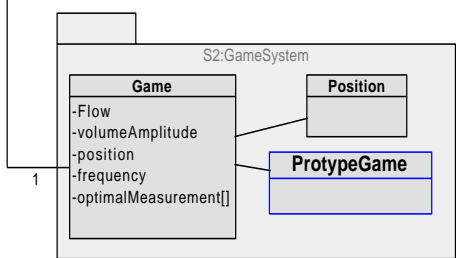
6.8 S1: Spiro Patient Examination System

Model S1: Patient Examination System

Figure 6.14: S1:Patient Examination diagram



Name	S1DesignPatientExamination
Model	Patient Examination Design diagram



The diagrams are coloured according to the rules described in appendix H on page 169

An alternative design class diagram ?? on page ?? created with analysis colouring, exploring COAD's color rules described at H.1.1 on page 170 See also a selected set of sequence diagrams created use for the use case realization and extension into design diagrams. G on page 165

6.9 Data modelling

The data model comes from the patient system *DataFeeder* system.

The *DataFeeder* and the probe data we have made part of the same system, it could be argued that the hardware access and data reading could be another part of the system. But it doesn't provide any benefits for the analysis the design or the project overall.

However there is a set of classes responsible only for communicating with the probe and the USB access, (which could later be substituted with the RH legacy system component.

The reason we haven't bothered with that is because USB is a technology that is used more and more also in the Medico business. Probably the Hospital might be equipped with USB equipment in the "near" future. This can be verified by looking in ware catalogues for current hospital equipment.

Furthermore, in the research phase it was determined that we could probably couple all the measurements system we were presented with via some wiring, into the existing system, into a standard RS Serial cable and from there, connect it with a USB adapter. To be clear an adapter plugin and wiring could be made so all Examination types, Open/Closed Box, Cold air test, CT scanner test, cycling test could be connected and connected to USB adapter, if that is the case we can with a small single effort in creating the adapter and wiring tap into all the systems and connect our device, to work with all the systems, off course the games might need to be adapted to each examination type.

That means that our development most probably could be independent on the specific hardware system that is measured on in the near future.

6.10 Data

Data modelling and databases.

6.10.1 First table layout sketch

A number of tables with column attributes that are seemingly needed, filled out informally, PatientHistory is filled up with all information about patient history that was likely. The PatientHistory table has more duplicates with other table than any other table, because almost everything is in it.

1. Patient tables
 - a) PatientInfo
 - b) PatientExamintionCard
 - c) PatientLimits
 - d) Patient
2. Examination tables
 - a) ExaminationInfo
 - b) ExaminationSession
 - c) ExaminationProbeData
 - d) ExaminationData
3. Game tables
 - a) GameInfo
 - b) GameCuesDispatched

GameCues should count what cues are used and at what time.

GameInfo Collects info about point scoring and collected objects. There is examination quality, which should express how well the examination was done.

PatientLimits table, lists the limit we could think of at the time.

6.10.2 database 1. Normal form

Now there exists set of tables is created with a logical and topically division, the duplicate elements are removed and the tables are cleaned up. At last 1. NF. is applied, All attribute must be atomic.

6.10.3 database 2. Normal form

Define or identify pieces of information in each table. Find out how the tables are related.

Finding candidate keys:

A patient is uniquely identified by his/her social security number so that is an easy and logic choice.

PatientExaminationCard, what identifies that? this is much harder or easier if more choices means it easier. *examinationId* is invented, but what will it look like, should it uniquely identify or should it be combined with something else, should it be unique for all records or just fro records for one patient.

We could also identify by data and or time.

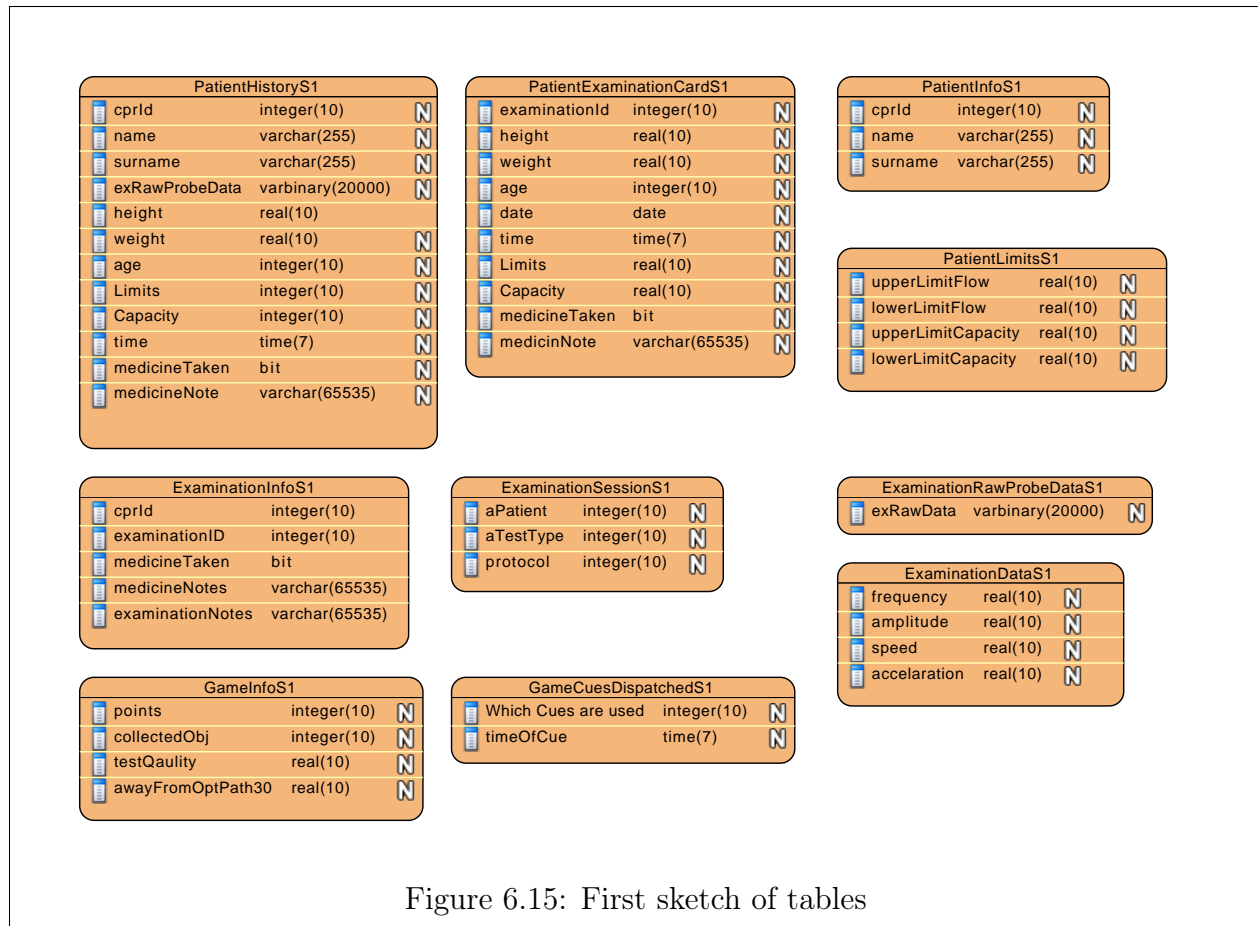


Figure 6.15: First sketch of tables

Define rules for examinations:

rule1: a patient is registered only once when they meet for the examination summoning"

rule2a: a patient might need more than one examination rule2b: a patient might take an examination over again, or do something repeatedly which constitutes multiple examinations

with those two simple rules, information can be divided between the information that are registered when they arrive at the hospital, and information gathered for each examination

ID's:

Patient: cpId; PatientExaminationCard: examinationId (date + nr) Still cpId is needed to determine the record in the cardTable, but does cpId need to be primary key.

How the tuples are identified depends on how we define the examinationId and cardId. Because the names suggest identification or uniqueness, they are renamed to cardNum and examinationNum.

6.10.4 Database 3. normal form

The tuples must not depend on part of a non candidate key. Any column not part of a key must not uniquely identify a row in the table.

Studying the set of created and manipulated tables it seems that we have no such dependencies. This is probable because of the initial table partitioning, where we intuitively created separate tables with information "logically" belonging to a specific table. Patient personal data, examination info, patient limits game score.

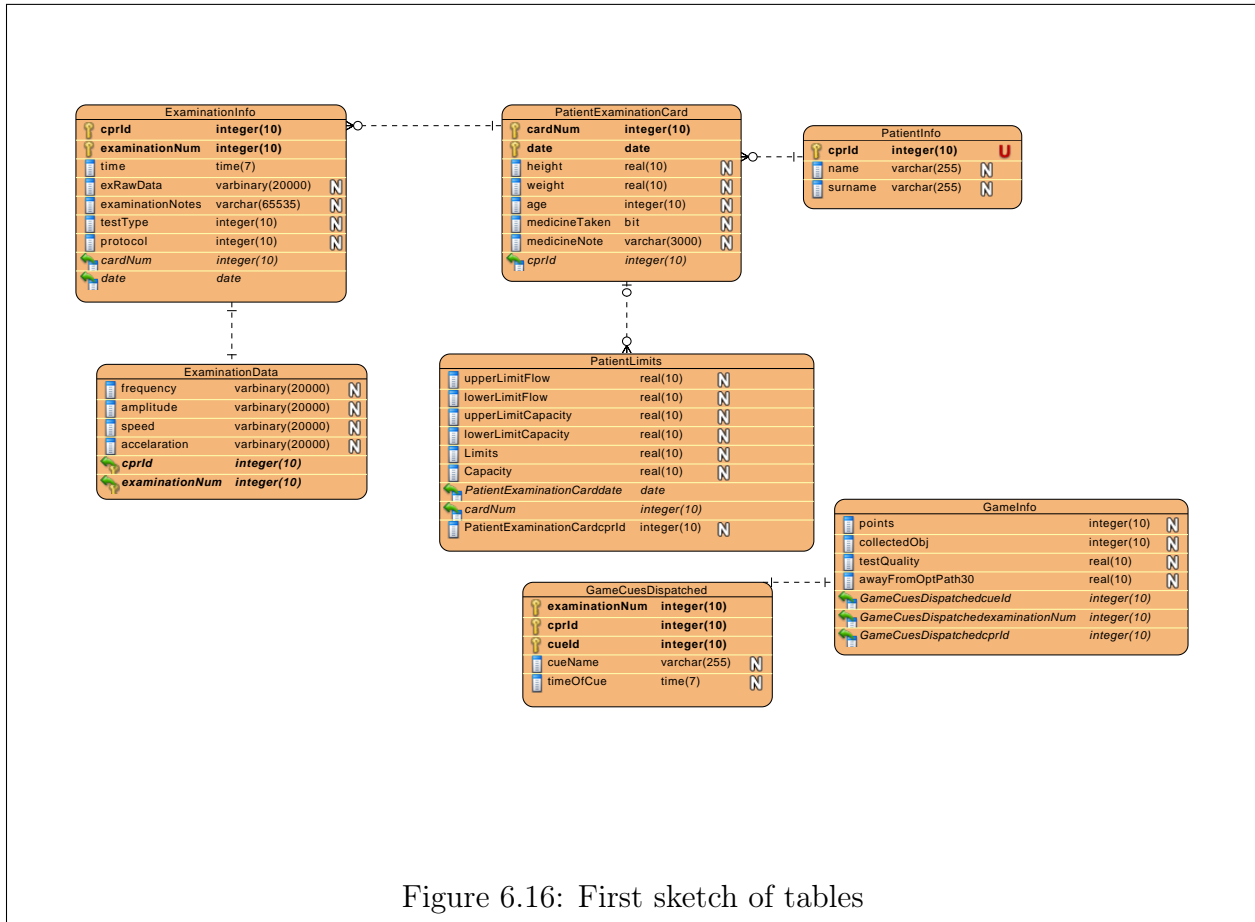


Figure 6.16: First sketch of tables

Some information that could uniquely be in one table is split out in several tables, this will simplify some queries, it is expected that the information that will be queried fits the table schema that is designed.

6.11 Game design : S2:SpiroGame

Model S2: Spiro Game System

SpiroGame is the skeleton for the game. It is partly build as the game itself and partly built to be extensible and provide functionality for the new games to be developed.

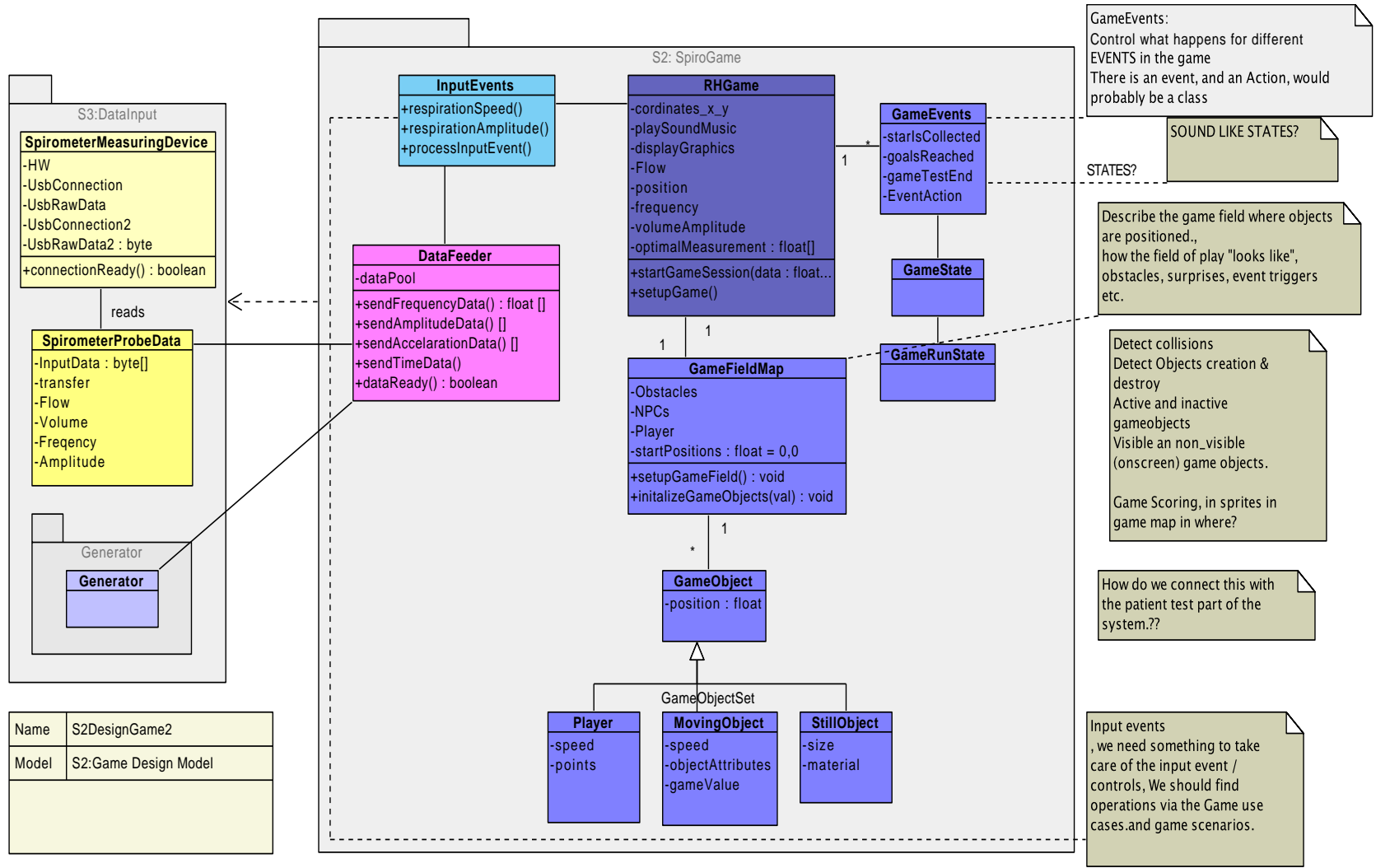
The game is designed to be extensible, as seen in diagram at 6.7 on page 76.

Description of game input and control:

- GameInputEvents: Receives the data input and translate it to a specific type of input event.
- GameController: this is a large block of functionality, that coordinates the rest of the game functionality, it is a object might be given to much responsibility if not careful. The GameController also contains the *System observer* that oversees input and game events and control the cues to be send to the player.

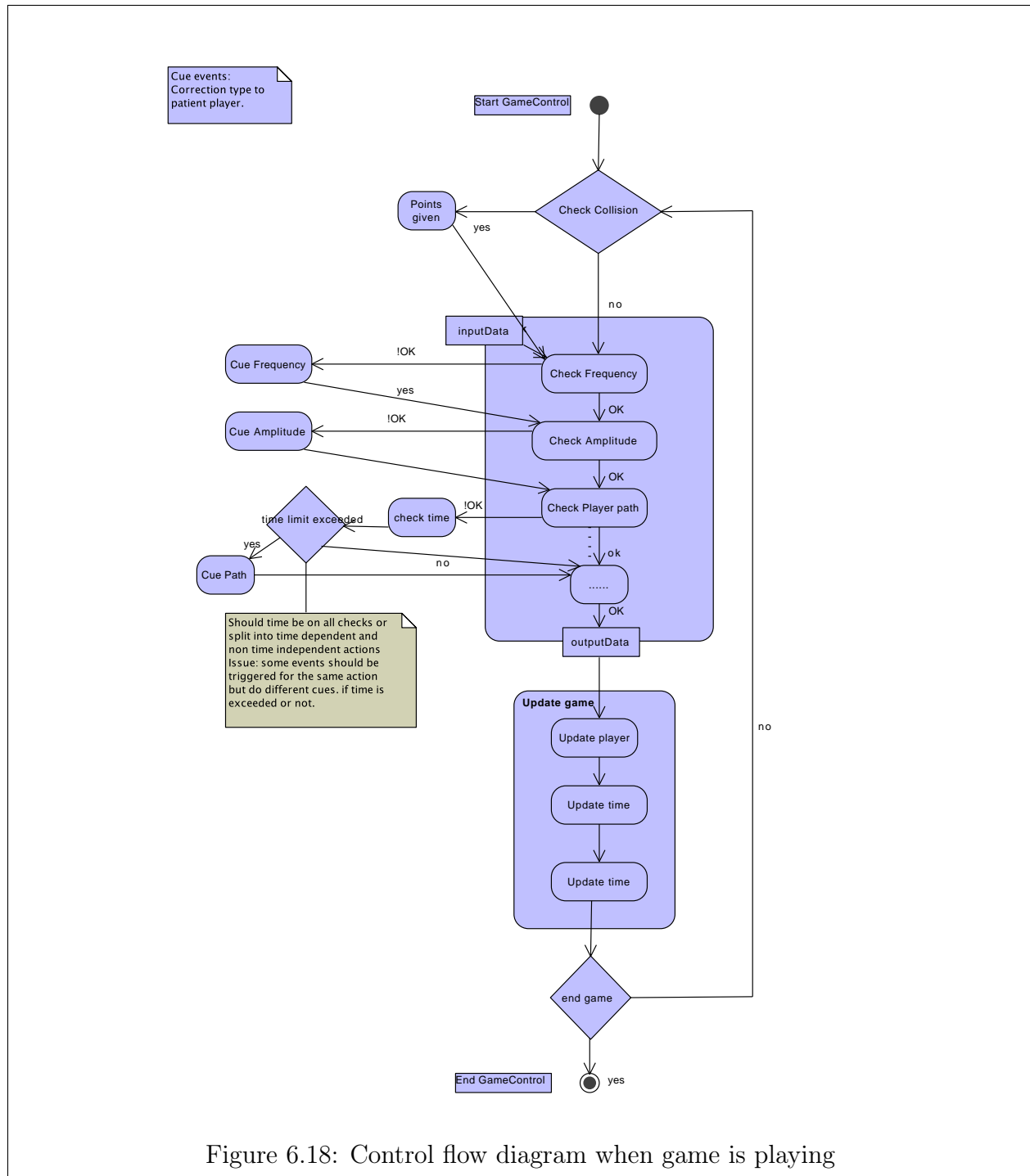
The design is preparing to move some of the responsibilities to the InputEvent and GameFieldMap classes. as seen in the diagrams. The prototype(s) are working towards that design goal, but also focuses on getting working functionality.

Figure 6.17: S2: Spiro game prototype design



Name	S2DesignGame2
Model	S2:Game Design Model

6.12 Game flow descriptions



A flow control diagram of the ingame flow. The flow diagram shows that there is a game loop, for each loop the game tests for collision, *if collision*, pick up the object(Sunstar) and get points awarded. Return to check input data. Different kinds of input data is tested (*type, val*) where type can be (*frequency, amplitude, player path, etc*) If the value of the type

is not ok (!OK), if there is a problem then send a cue and continue to next test. repeat for all tests. Some cues are time based not only does the *type value check* need to fail, but a time restraint much be considered also before a cue is send. After that the game updates all the different game objects etc. and the tests are redone again until the game loop is interrupted.

The flow control diagram uses prototype 1 for finding which actions need done. So it lacks generality and would have to be updated for new games or for generality.

6.13 Game events and state

The event system is developed to react to game events. The first idea was to complex for a short project The idea was to create a Game event for each object type, and also for the game and the GameFieldMap. this is to complex I wanted the game to create events based on input and on position. and on object type. that would mean each object, the GameFieldMap, and the input. Would have a kind of event action matrix, the matrices could then be put on top of each other. And the actions possible would be where the matrix masks fitted.

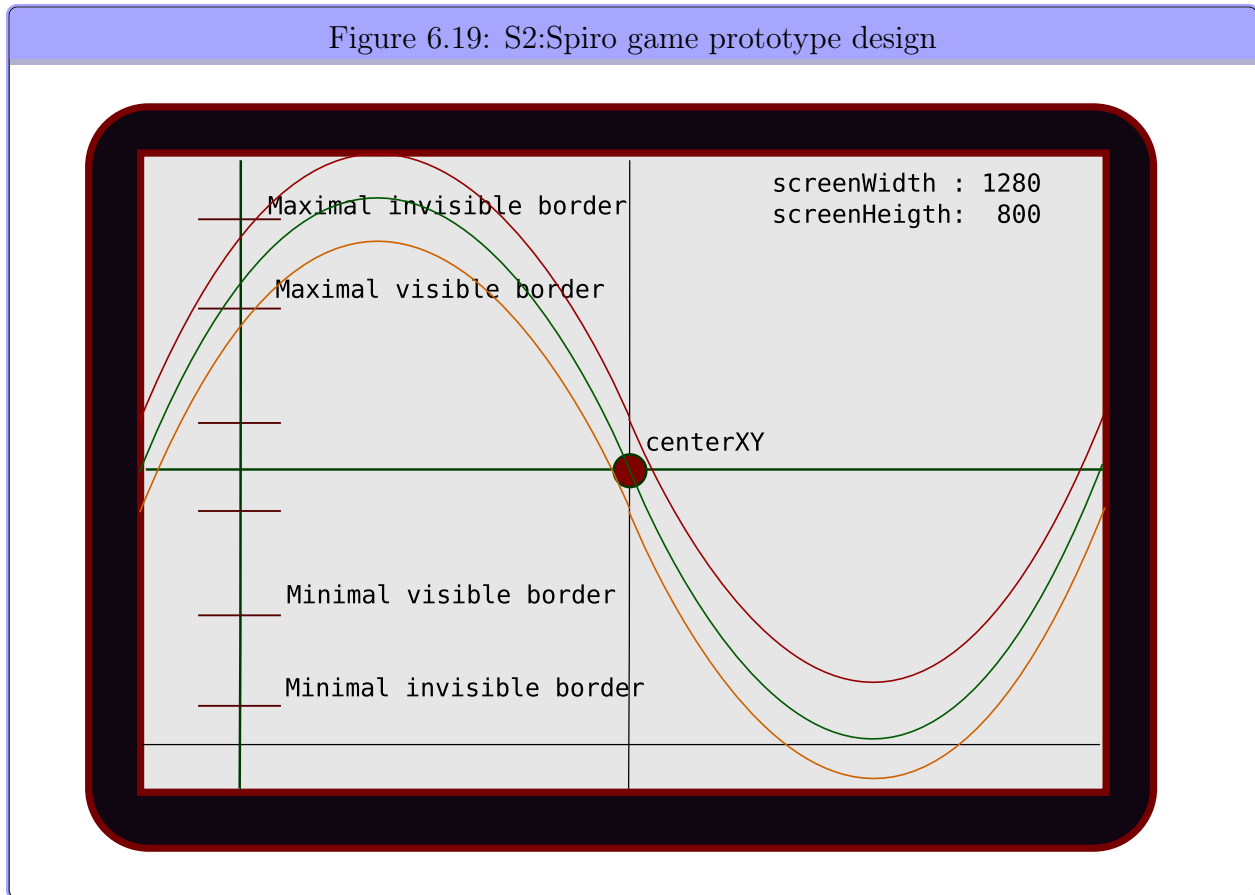
This would be very complex($(ACTIONS*EVENT)^3$) very flexible and probably not needed.

Instead a more simple approach is giving the **GameFieldMap** or the **GameController** an Event table. when something that triggers an event in the game, the GameFieldMap that *knows* the game objects can propagate and execute the event.

6.14 Screen division

The screen division for the tablet is shown we divide the screen into sections, the left axis shows the division of the tablets, we will not go to near the edges of the screen.

Drawing a approximated curve of the breathing pattern of the patient we have a path of a certain width(Max/min visual border). The Maximal and minimal visual border should give visual hints for the patient to follow, that is the width of the "road" the patient should keep inside. The Maximal and Minimal invisible borders are the area that the patient should keep inside to maintain a good breathing rhythm and strength. If outside the invisible borders the game might intervene in a drastic way to keep the patient breathing properly. This could f.ex. be halting the game and guide the patients breathing, before continuing the game.(Examination continue recording under pause)



6.15 Prototype 1 description: SpyroFox

SpyroFox: a little foxy hero is trying to get to the end of the field, and while doing it collection as many sunny smiley's as possible.

The smileys comes in the top and in the bottom of the screen, if you blow to hard you will pass the Sunstar over the visible max border or under visible min border.

The idea is to collect as many sunstars as possible, if you miss your outside a good breathing pattern. And the game will send a cue to the player. If you collect a certain amount of stars in a row with out missing one, there will be an encouragement cue.

6.16 Prototype 2 description:Spiro PaperBoat

A paper boat sails on the big garden tub, it has a sail. Blow out to blow wind in the sail, blowing near the best limit for the patient, gives the fastest sailing , which mean you will catch another boat/fish, seagull , something and overtake, if you blow to hard the water Resistance gets so high that you slow down faster, if patient blows under limit the boat doesn't sail as fast as possible. When breathing in, i don't have a clear idea, but paper boats take in water, because it made of paper. So maybe the patient should shovel water out or maybe the sail slacks or something...

Part III

Implementation & Test

Chapter 7

Implementation intro

7.1 First prototype

Prototype 1 is designed so that the game movements of the player resembles the optimal breathing curve. The player must move along the optimal breathing curve as closely as possible. If the player follows the optimal path all the Sunstar objects is collected. The optimal path can be seen as the apex of the "road" that must be followed, the road has a certain width and if the player keeps inside the width of the road all Sunstars Will be picked up.

If a graph is drawn for the breathing of the patient that equals the position of the player. In an optimally" performed examination the breathing graph of the patient and the graph of the optimal path will overlap. This prototype helps in development because the abstraction of the game and the breathing pattern is close.

Prototype1 is a game that scrolls from right to left, the player starts in the middle of the screen, and must collect Sunstars The Sunstars appear on the right side of the screen with even timing intervals, matching the breathing frequency wanted. the appear on top and bottom of the imaginary curve that the optimal" breathing pattern would draw, the player must move to pick them up and get points for picking them up.

The Sunstars are always at the same height on the screen, but the effort to move to pick them up is adjusted by the flow and capacity limits read from the patients data. That how the game is configured for each patient.

Issue: The game speed might be fast or restricted by this pattern More game prototypes are needed, so we can test out what work on the patients.

?? on ??

7.1.1 Prototype graphics bitmaps & and animation

Graphics is designed simply, since there are no artist "resources" assigned to the project.

The drawings are inspired in its simplicity from the drawings of danish cartoonist Martin Strid. <http://www.strid.dk/>

The graphics is being designed to be simple but expressive. The graphics of the player figure can then assist the human player by letting the eyes move to the direction it wants to move. Also a tail should be added to assist the feeling of the movement direction, the tail will bend to follow the path movement.

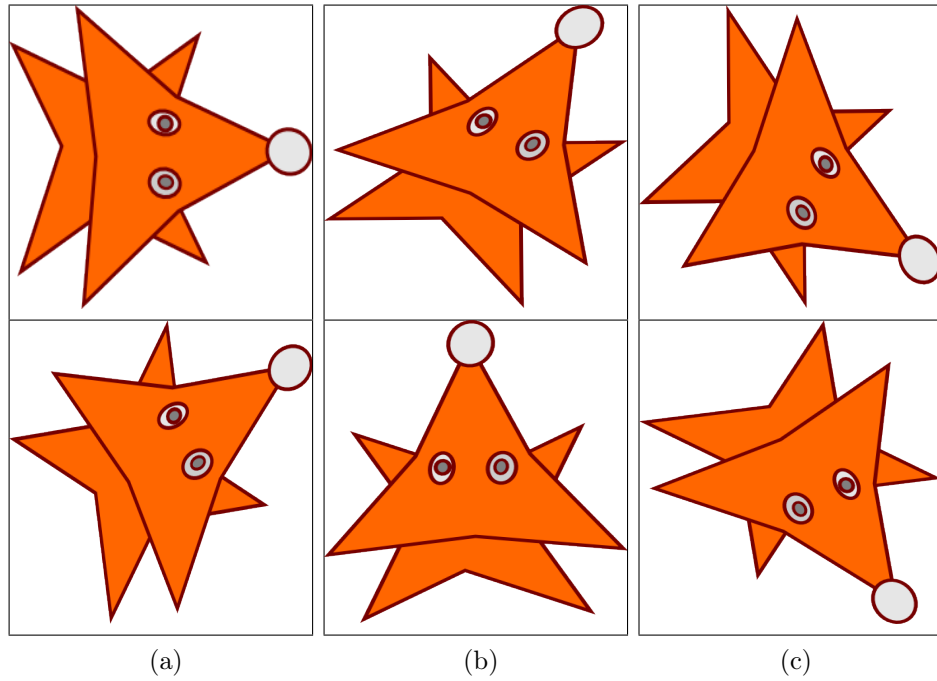


Figure 7.1: The player figure, simple but expressive drawing

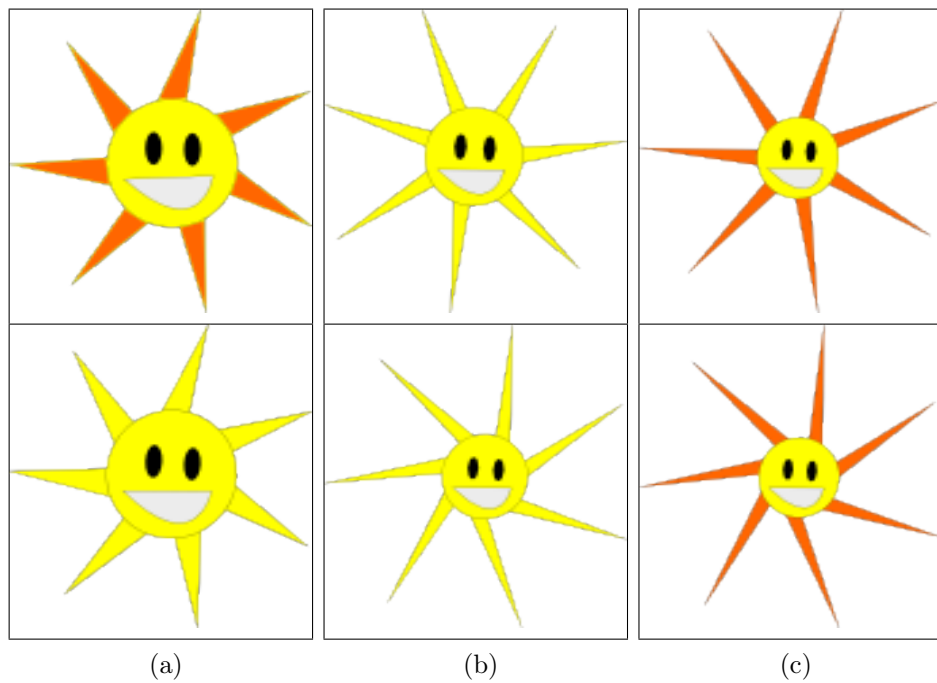
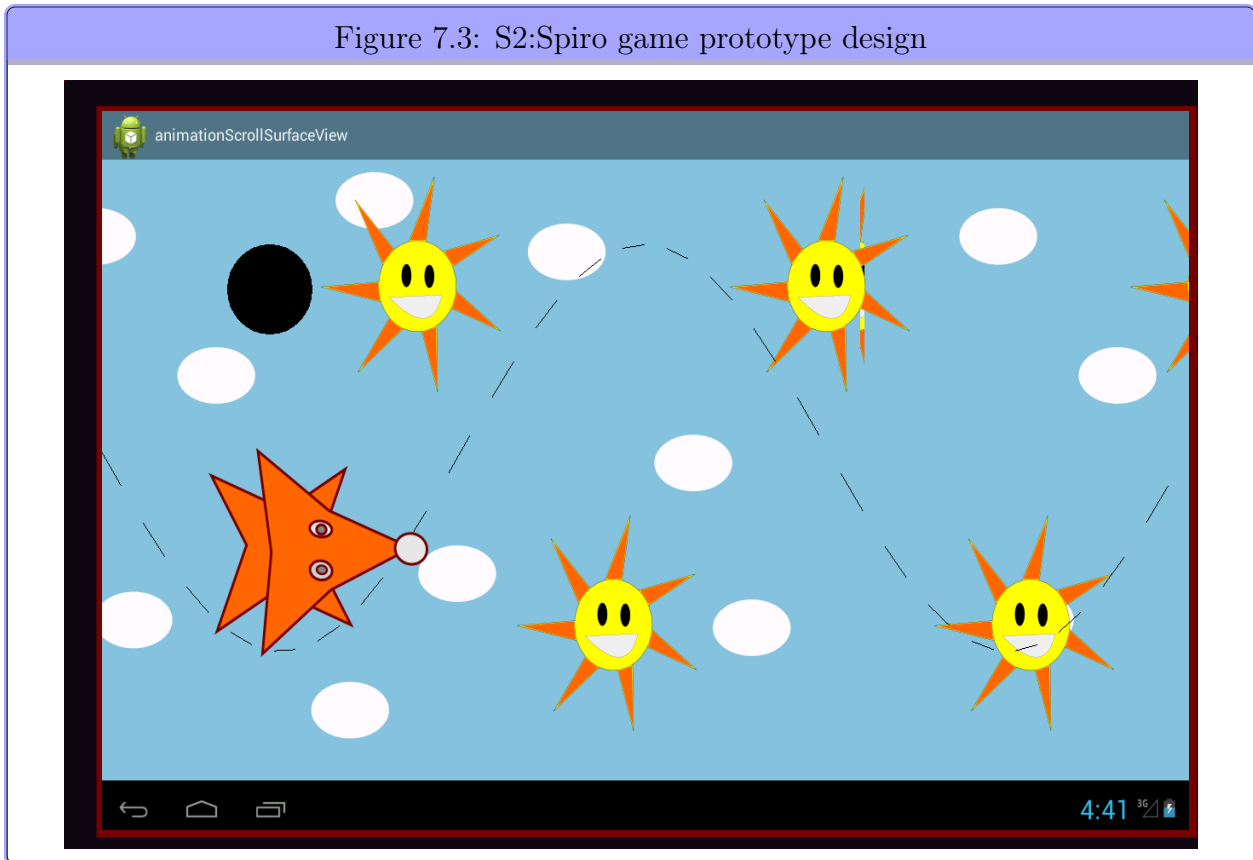


Figure 7.2: A selection of the Sunstars

The sprite in figure 7.3 to the left in the middle of the screen is the player. The player must pick up the Sunstars, that is the yellow and orange smiley with rays. If the player breathes correctly all Sunstars will be picked up by the player.

Figure 7.3: S2:Spiro game prototype design



7.1.2 Collision

```

1
2 scrollPosBG+=5;
3 if (newBgPos <= 0){
4
5     scrollPosBG=0;
6 }
7 }
8
9
10 // What should updateStars() do
11 public void updateStars(){
12 synchronized (gameThread) {
13 //Log.v("updateStars3", Float.toString(sunstar3.objectPosition.x) + " :
14 //     " + Float.toString(sunstar3.objectSpeed.x));
15 //         inactiveArray.add(activeArray.get(index));
16 //         activeArray.remove(index);
17 //     }
18     gameState=GAME_STATE_RUNNING;
19 }
20
21 // CHK: put in gameObject handler class
22 public boolean collisionDetection(MovableObject object, MovableObject
    otherObject){

```

```
23 // CHK: TODO: get center position of object
24 float xdist = Math.abs(object.objectPosition.x - otherObject.
    objectPosition.x);
25 float ydist = Math.abs(object.objectPosition.y - otherObject.
    objectPosition.y);
26
27 Log.v("collisionDetection", "x1,y1 " + Float.toString(object.
    objectPosition.x) + " " +
28     Float.toString(object.objectPosition.y) );
29
30 Log.v("collisionDetection", "x2,y2 " + Float.toString(otherObject.
    objectPosition.x) + " " +
31     Float.toString(otherObject.objectPosition.y) );
32
33
34 Log.v("collisionDetection", "xdist, ydist " + Float.toString(xdist) + "
    " + Float.toString(ydist));
35 Log.v("collisionDetection", "width,height " + Float.toString(object.
    moWidth/2) + " " + Float.toString(object.moHeight/2));
36 if (xdist < object.moWidth/2+0 && ydist < object.moHeight/2+0){
37     return true;
38 }else{
39     return false;
```

Listing 7.1: Collision detection file

After each collision between the player and the Sunstar, the Sunstar is removed from the active to the inactive list as described earlier.

7.2 DataInput Generator description

Figure 7.4: S1:graph curve

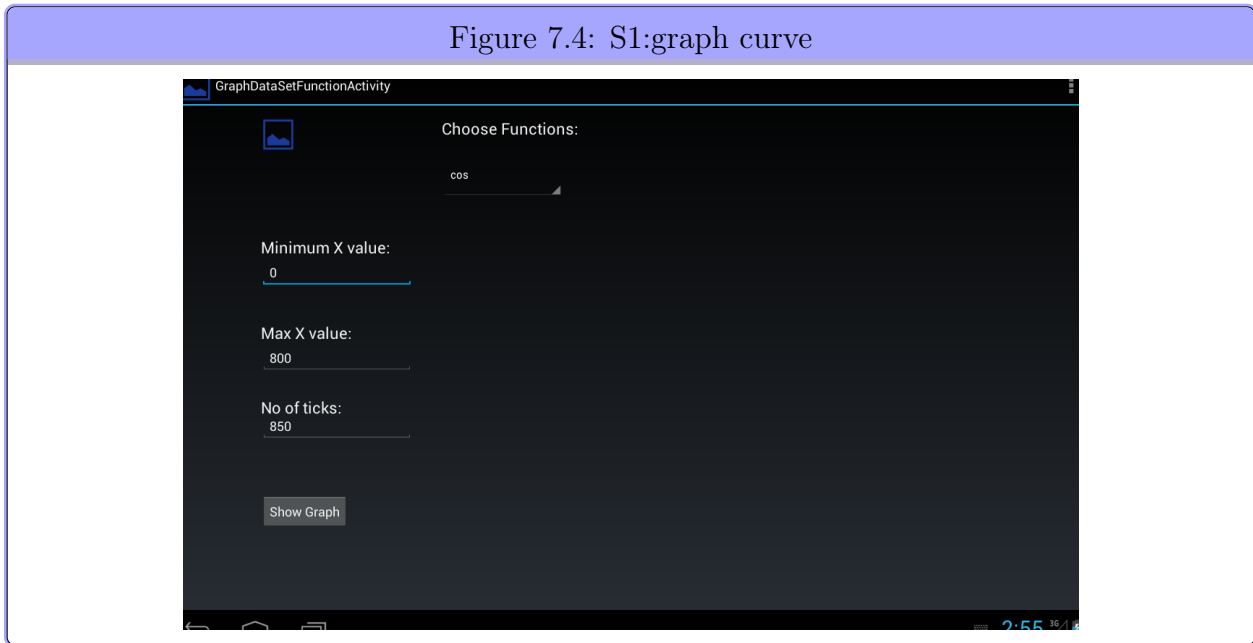
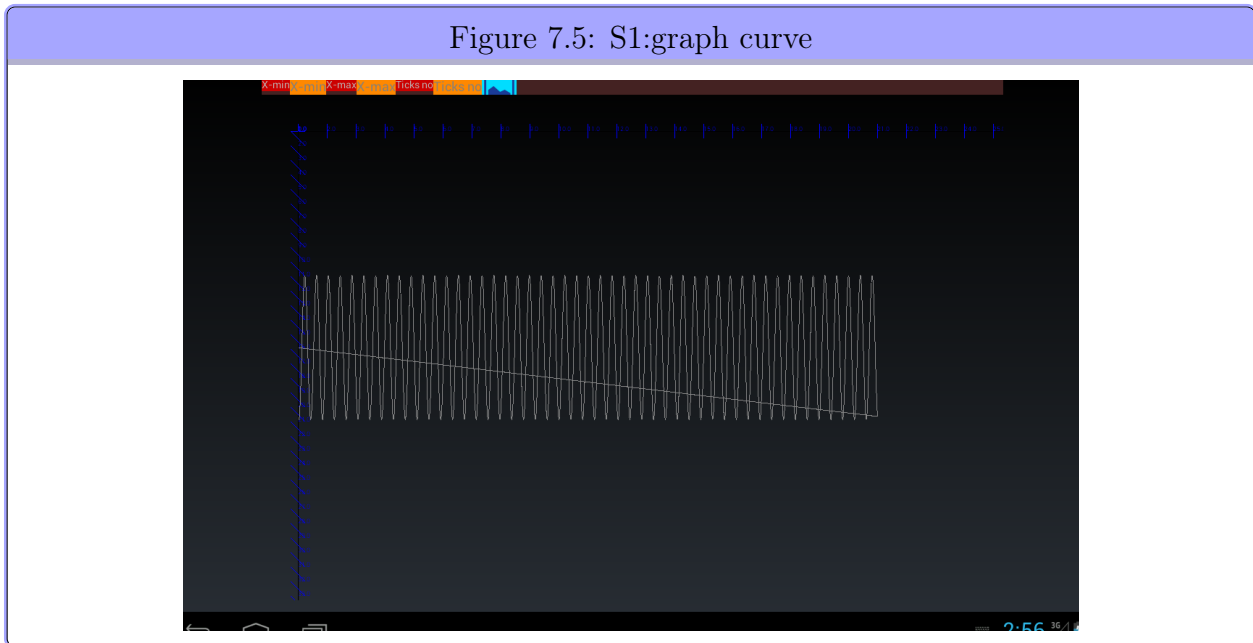


Figure 7.5: S1:graph curve



Images taken from the application development of the simulated DataInput via the generator. The 1. image ?? is of the input values screen, The range along the X-axis is chosen, function and resolution of the graph is chosen.

On the second image ?? the graph for the chosen values, resolution and function is displayed.

The data is generated by the **Generator** system as described in D on page 155. On the diagram an artifact on the graph display can be seen, a line from the point at the rightmost

x value to point 0,0. This is an error in the conversion of the graph data where the last value in the conversion doesn't get assigned a value, and the default value 0,0 is displayed for the last point, because the graph is put together of small line fragments is drawn across the screen.

g

7.3 Implementation issues

7.3.1 Scrolling

The scrolling (Prototype 1) is implemented by splicing 2 of the same background image side by side, each sized to match the screen resolution. It works like a torus that repeats itself. The spliced image is moved from right to left with the calculated speed that match 35 breaths per minute.

7.3.2 Active game objects

The GameFieldMap class is responsible for control of which game objects are active at any moment in prototype 1 "SunFox", an active objects is any object visible on the screen area, when the game objects in this case the Sunstars leave the screen they are moved to the inactiveObject list.

One issue is synchronization because a new Sunstar appearing on the screen is just added to the active list. We have created 5 Sunstars, because that's the most that are needed to be displayed on the screen in our game. See description at ?? on page ?? screen division section.

The first prototype defined the screen width length as $4 * \pi$. The Sunstars are placed with a distance of

$$\pi. 0\pi \leftrightarrow 1\pi \leftrightarrow 2\pi \leftrightarrow 3\pi \leftrightarrow 4\pi$$

As we can see 5 stars is the most that can be displayed on screen at an instance when the Sunstars are placed with a distance of π including the edge of the screen.

```

1 public void updateStars(){
2   synchronized (gameThread) {
3     MovableObject moDel;
4
5     // update active star positions
6     for (MovableObject mo: gfm.activeGameObj) {
7       Log.v(LOG_SPYRO_V + "updateStars:activeStars", Integer.toString(mo.
8         sunNumber) + " : " + Float.toString(mo.objectPosition.x)
9         + ":" + Float.toString(mo.objectPosition.y) );
10      mo.objectPosition.x -= mo.objectSpeed.x;
11      Log.v(LOG_SPYRO_V + "updateStars:activeStars", Integer.toString(mo.
12        sunNumber) + " : " + Float.toString(mo.objectPosition.x)
13        + ":" + Float.toString(mo.objectPosition.y) );
14    }
15    // check positions of game objects :
16    // TMPFIX;

```

```

15     int index=-1;
16     for (int i =0 ; i < gfm.activeGameObj.size() ;i++) {
17         moDel = gfm.activeGameObj.get(i);
18
19         // disappear on the left of the game area : remove from active obj
20         if (moDel.objectPosition.x + moDel.objectBitmap.getWidth() < 5 ){
21             Log.v(LOG_SPYRO_V + "updateStars:forRemoval", Integer.toString(moDel
22                 .sunNumber) + " : " + Float.toString(moDel.objectPosition.x) + ":"
23                 + Float.toString(moDel.objectPosition.y) );
24             // move to Inactive Queue and remove from active Queue.
25             index=i;
26         }
27     }
28     // :FIX: only one star can be removed per round, : Remove correctly from
29     // arrayList: @chk
30     // This change seems to have fixed something now we have stars on the
31     // screen, before no stars appeared
32     if (index != -1 ) {
33         moDel = gfm.activeGameObj.remove(index);
34         gfm.inactiveGameObj.add(moDel);
35     }
36 }

```

Listing 7.2: Active and inactive game objects

A fix for the active inactive list is underway, the activeGame list seems correct, but the inactive list keeps growing. See test2. ?? on page ??.

The test data shows that new objects are created instead of changing between the active and inactive list. The cause is probably because of they way ArrayList is used we add and remove while we should use a index on the range 0-4 instead. Looking at the references to the objects in the inactiveList we will probably see that there are only 5 unique references, let's look... As it shows we have multiple references to the same object in the list. Solution is, using a closed index set 0-4 or choose another List type. There has also been Some Concurrent modifications errors, so maybe a Concurrent container can be used. Maybe the game object bitmap flickering is related to a thread issue. Thats were we look next.

7.3.3 The GameFieldMap

```

1 public class GameFieldMap {
2     public static final String LOG_GAMEFIELD = "GameField:";
3     // :CHK: ScrDiv | GameFieldMap
4     Context context;
5     // game Field Map
6     int yStart=300; int xStart=100; // should be calculated via screenDivision
7     , height, width
8
9     // these are not performance critical, use setter and make non public
10    public int screenXResolution;
11    public int screenYResolution;

```

```

12 // game object on the screen to be displayed
13 public ArrayList<MovableObject> inactiveGameObj = new ArrayList<
    MovableObject>();
14 // game objects that goes off screen/ or are inactive but are ready to
    be used
15 public ArrayList<MovableObject> activeGameObj = new ArrayList<
    MovableObject>();
16 //! TODO: inactive objects are deleted ... when!
17
18 //! Creating the player and sprite objects on the field inclusive initial
    position etc
19 public StarPlayer starPlayer;
20 private int movePlayer=0;
21
22 public SunStar sunstar1;
23 public SunStar sunstar2;
24 public SunStar sunstar3;
25 public SunStar sunstar4;
26 public SunStar sunstar5;
27
28 // chk:TODO:maybe this should be inside GameFieldMap
29 ScreenDivision screenDivision = new ScreenDivision(screenXResolution ,
    screenYResolution , context);

```

Listing 7.3: GameFieldMap

The GameFieldMap determines placement and state of all type of game objects, it should control the events and user interactions it gets from the **GameController** and process the received game event and input event data and process execute or delegate the decided action.

It responsible for collision control and must know where the objects are on the screen and the screen resolution to place them properly on different Android devices(Using theScreenDivision class).

7.3.4 Setup game patient constraints

```

1 // @chk : whats the difference in implementing runnable and not do it
2 // run() still exists
3 // runnable doesn't need to extend thread, you just have to create a thread
4 // and send the runnable as parameter to the Thread constructor
5 public class SpyroSlalomGameViewThread extends Thread //implements Runnable
6 {
7     public final int BLOWS_PER_MINUTE = 35; //
8     public final int INV_BPM= 1/35; //2*1714L
9     public final int TIME_MILLIS_PER_BLOW = 35 / 60 * 1000;
10
11     public final int GAME_STATE_RUNNING=0; // Game is started
12     public final int GAME_STATE_INIT=1; // game pieces and state are setup
13     public final int GAME_STATE_PLAY=2; // game play is started, the game is
    going on
14     public final int GAME_STATE_PAUSE=3;
15     public final int GAME_STATE_LOSE=4;

```

```

16 public final int GAME_STATE_WIN=5;
17 private int gameState=GAME_STATE_INIT;
18
19 private SurfaceHolder msHolder;
20 private Handler handler;
21 private Context context;
22 private SpyroSlalomGameView spyroSGameView;
23
24 //GameFieldMap gfm = new GameFieldMap(gameThread.context);
25 GameFieldMap gfm;
26
27 ScreenDivision screenDivision;
28
29 boolean isRunning=false;
30
31 //! Access to resources etc
32 Resources res; // what is it we get here
33
34 // Background parts
35 public Bitmap mBackgroundSky;
36 public int scrollPosBG=0;
37 public int newBgPos=0;
38
39 private Paint mPaint;
40 private Point mScreenCenter;
41
42 Canvas gameCanvas=null;

```

Listing 7.4: Patient constraints

This is the game thread, where the state variables for the running game state is defined and controlled. The optimal blow frequency and the calculated time in milliseconds between each top and bottom point. The GAME_STATE that controls initialization and Game running, pause and ending. The GameFieldMap reference and ScreenDivision.

The ScreenDivision are going to be refactored. into the GameFieldMap, but needs information that is announced via callbacks from the GameView(SurfaceView Android class).

The Patients constraint should be injected here from the PatientExaminationSystem **S1**.

7.3.5 Updating game state

```

1 public void run() {
2     gameCanvas = null;
3     while (isRunning){ // While game is running, including on pause
4         //try {
5             gameCanvas = msHolder.lockCanvas(null);
6             synchronized (gameCanvas) {
7
8                 if (gameCanvas != null){
9                     doDraw(gameCanvas);
10                }

```

```

11     }//Synchronized
12
13     // ready to be updated
14     msHolder.unlockCanvasAndPost(gameCanvas);
15
16     // updating game before reposting for drawing
17         stateChange();
18     updatePlayer(5);
19     updateStars();
20
21     scrollBackground();
22     ...

```

Listing 7.5: Updating game state

This code *locks* the drawing canvas so the SurfaceView thread can update the graphics values. When the canvas is unlocked the updated graphics is displayed on the screen. While the canvas is locked the screen will not be updated, when the canvas is unlocked the drawn canvas is displayed to the screen. *doDraw* is a method where all the Android drawing methods is executed inside the canvas lock. After the synchronized section, we unlock the canvas and call all game update methods, updating game objects and scroll the background. Test for collision and events. and are ready to draw again. This is the general game loop.

All the update methods should be moved to the GameFieldMap class. Then only one public call or interface GameFieldMap.update() from GameController initiates all the specific update methods in the GameFieldMap class. this way our game is upholding the data flow for the game system and the design architecture. 6.9 on page 79

7.3.6 Game View

```

1 public class SpyroSlalomGameView extends SurfaceView implements SurfaceHolder.
   Callback {
2     public final static String LOG_SPYRO_V="SpyroView: ";
3
4     EditText playerPosition;
5     Bitmap iconBitmap; // :CHK <- why is this bitmap here
6     SpyroSlalomGameViewThread gameThread;
7
8     GameInfoView giv;
9
10    // Screen max min
11    public int screenXResolution;
12    public int screenYResolution;
13
14    RoadShape rs;
15
16    public SpyroSlalomGameView(Context context){//, AttributeSet attrs) {
17        super(context);//, attrs);
18
19        setFocusable(true);
20        requestFocus();

```

```

21     giv = new GameInfoView(context);
22
23     final Rect gameViewRect = this.getHolder().getSurfaceFrame();

```

Listing 7.6: Game initialization

This is the `GameView` that is updated via the secondary nested thread class. This part runs on the main thread (UI thread) where the graphics 'state' is drawn. `ScreenXYResolution` is here since we can first access the screen size information after the callback `surfaceCreated` is executed, that is a method in the `SurfaceView` class (`SurfaceHolder.Callback`)

7.3.7 Game Init

```

1     case GAME_STATE_INIT:
2         initSetGameField();
3
4         canvas.drawBitmap(mBackgroundSky, +newBgPos, 0, null); // :chk: why
5             can it be null
6         canvas.drawBitmap(mBackgroundSky, -scrollPosBG, 0, null);
7         canvas.drawBitmap(iconBitmap, 10, 10, null);
8
9         Log.v("onDraw: init:gameobj_size", Integer.toString(gfm.activeGameObj.
10             size()) + " " + Integer.toString(gfm.inactiveGameObj.size()));
11         for(MovableObject mo : gfm.inactiveGameObj){
12             Log.v("onDraw: init:gameobj_num", Integer.toString(mo.sunNumber));
13         }
14         int i=0;
15         while (!gfm.inactiveGameObj.isEmpty()) {
16             MovableObject mo;
17             gfm.activeGameObj.add(mo = gfm.inactiveGameObj.remove(0));
18             canvas.drawBitmap(mo.objectBitmap, mo.objectPosition.x, mo.
19                 objectPosition.y, null);
20             Log.v("onDraw: INIT", Float.toString(mo.objectPosition.x) + " " +
21                 Float.toString(mo.objectPosition.y));
22             i++;
23         }
24         Log.v("onDraw: init:gameobj_size", Integer.toString(gfm.activeGameObj.
25             size()) + " " + Integer.toString(gfm.inactiveGameObj.size()));
26         for(MovableObject mo : gfm.activeGameObj){
27             Log.v("onDraw: init:gameobj_num", Integer.toString(mo.sunNumber));
28         }
29         gameState=GAME_STATE_PLAY;
30         ...

```

Listing 7.7: game Init

The `GAME_STATE_INIT` state sets up the gaming field and places the game objects on the field. The background is drawn. All game objects are transferred to the `activeGameObj` list(this is wrong!), only one game object should move into the `activeGameObj` list.

7.3.8 Refactoring

```

1 @Override
2   public void surfaceChanged(SurfaceHolder holder, int format, int width,
3     int height) {
4     screenXResolution = width;
5     screenYResolution = height;
6     // transfer screen resolution to gamemap
7     gameThread.gmf.screenXResolution=screenXResolution;
8     gameThread.gmf.screenYResolution=screenYResolution;
9     // chk:ScrDiv:update here for changed surface?
10  }

```

Listing 7.8: Refactor from GameController to GameFieldmap

This is a callback from SurfaceView, it is called when the canvas has changed structurally NOT after a canvas draw update(*doDraw*), it is always called on creation of the GameView(class name SpyroSlamomGameView) when the game starts. In the game system the game view is mixed with the GameController. *gmf* is the GameFieldMap reference and because the GameFieldMap has to know the screenXresolution when it places objects on the screen(the usable screen space) it has to update via the thread since GameFieldMap is created in the thread for the SurfaceView.

screenXYResolution might be redundant in the GameView but until refactoring is finished we have it in two places. This is ongoing work with refactoring as much as possible away from the GameView so the achitectural design can be maintained where feasible.

```

1     // CHK: TODO: load these in inactive, and then put in active when some
2     // condition is set.
3     // Move to gamefield map
4     gfm.inactiveGameObj.add(gfm.sunstar1);
5     gfm.inactiveGameObj.add(gfm.sunstar2);
6     gfm.inactiveGameObj.add(gfm.sunstar3);
7     gfm.inactiveGameObj.add(gfm.sunstar4);
8     gfm.inactiveGameObj.add(gfm.sunstar5);

```

Listing 7.9: Before refactoring from GameController to GameFieldMap

Here is an example of code for refactoring, first code try placed the removal and adding of active and inactive objects in the GameView - > *GameController*. Looking at this code it is obvious that it can be moved to the GameFieldMap That is repnsible for adding game objects. The code in GameFieldMap would look like this.

```

1     inactiveGameObj.add(sunstar1);
2     inactiveGameObj.add(sunstar2);
3     inactiveGameObj.add(sunstar3);
4     inactiveGameObj.add(sunstar4);
5     inactiveGameObj.add(sunstar5);

```

Listing 7.10: After refactoring from GameController to GameFieldMap

Simpler and following architecture responsibility design.

7.3.9 Timing and events

A timer is set for running with a frequency of 35 Breaths per minute. The system timer is used to create a game timer.

```

1  // CHK: MKJ: TODO: i want to make a timer class so i can create classes
2  // with different timing uses.
3  // functionTiming
4  if (timing < SystemClock.uptimeMillis() - PI_INTERVAL) {
5      Log.v("timing", "time for new gameObject");
6
7      //! belong to own class StarGameClass StarGame{ Sunstar s[]; ...
8      gfm.placeNewStar(new Point());
9      timing = SystemClock.uptimeMillis();
10 }

```

Listing 7.11: 1. Timer code

The 1. working timer code. Tested out several timer and events implementations The quick fix approach is to package the timer code into class and add *startTimer*, *stopTimer*, *timeStamp* methods. So we can time actions in the game object. *objectTime* : time since object creation, *actionTime* : time since some action started Then the game objects could have a timer object as general timer and action event timer. A similar idea is worked on for events in game objects to work with the *cue* system.

7.3.10 Data feed

DataFeeder:

DataFeeder supplying data to game and player object is not implemented.

7.3.11 Performance and architecture

In games performance is important. The Dalvik Java Virtual Machine for Android has a bad performance characteristics, when using getter and setters, the standard java runtime from OracleTM optimizes getters and setters to work in most cases as fast as direct variable manipulation (Write/read address). This makes some of the architecture decisions and separation problematic from a performance perspective Package partitioning can be a problem because we need public access to write directly to variables if they are in different packages, this breaks encapsulation.

Some methods and Fields are public for efficient access mostly position and velocity updates.

7.3.12 Implementation status

1. Modules need to be integrated

2. Systems and application parts
3. GraphDisplay doesn't work through DataFeeder
4. Architecture division

Mini apps for system parts functionality		
Name	Description	Development status
SpiroGame	Prototype1 game	keyboard and screen touch input, SurfaceView and thread done
InfoInputOutput	Accessing raw resources, resources and assets	
SpiroPatientExamination	The patient system, storing patient data in DB	Simple information stored, needs extended queries done (simple program)
MkjAndroid	Sends message with Extras information	
TimerTool	Class for keeping game time	Repeatable task problem, almost done
BouncingBall	Moving and collision testing graphics updating	Works, some touch and direction problem
GraphCharting	Display graphs of function generated data	Almost done
Generator	Generate different function values	Some problems with the scaling to screen sometimes, works for cos function
CustomView	Bitmap object custom view	Done
USB Access	Access USB as Host and accessory	Lacks testing

7.4 Test

1. Active Inactive Objects
2. Collision test
3. Graphics error test, disappearing stars + many stars in "wave"
4. Movement speed
5. Events read
6. graphics test

7.4.1 Test1: Collision test

Collision test: test coordinates with player object vs game object.

Testnr 1.1 player lower than game object:

Testnr 1.2 player higher than game object:

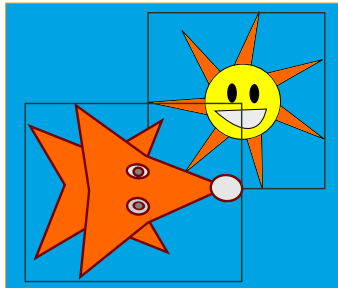


Figure 7.6: The overlap of the game objects for collision

Figure of two overlapping game objects, the player and a Sunstar, if the 2 figures are of equal size they will cover about 25 percent of their bounding box area, before a collision is detected, this makes the collision seem natural. The SunStars with very thin rays are be hard to make look like they collide because of the large area of the bounding rectangle that is not part of the Sunstar drawing, later this will be improved by tracing closer to the shape of the bitmap picture.

```

1  public boolean collisionDetection(MovableObject object , MovableObject
      otherObject){
2  // CHK: TODO: get center position of object
3  float xdist = Math.abs(object.objectPosition.x - otherObject.
      objectPosition.x);
4  float ydist = Math.abs(object.objectPosition.y - otherObject.
      objectPosition.y);
5
6  Log.v("collisionDetection", "x1,y1 " + Float.toString(object.
      objectPosition.x) + " " +
7  Float.toString(object.objectPosition.y) );

```

```

8      Log.v("collisionDetection", "x2,y2 " + Float.toString(otherObject.
9          objectPosition.x) + " " +
10         Float.toString(otherObject.objectPosition.y) );
11
12     Log.v("collisionDetection", "xdist, ydist " + Float.toString(xdist) + "
13         " + Float.toString(ydist));
14     Log.v("collisionDetection", "width,height " + Float.toString(object.
15         moWidth/2) + " " + Float.toString(object.moHeight/2));
16     if (xdist < object.moWidth/2 && ydist < object.moHeight/2){
17         //if (xdist < object.moWidth/2+63 && ydist < object.moHeight/2+63){
18         return true;
19     }else{
20         return false;
21     }
22 }

```

Listing 7.12: Collision center

The *collisionDetection* method calculates bitmap center of the game objects from the top left position, otherwise the collision is skewed

Test1: Collision		
TestName	Expected result	Actual result
collison player above object	25 percent area overlap $\frac{1}{2}x$ $\frac{1}{2}y$	Pass
collison player above object	25 percent area overlap $\frac{1}{2}x$ $\frac{1}{2}y$	Pass

In the test i will show how close the objects align and check that it looks like the objects collide properly on the screen. After a code change in collisionDetection line 13 the collision fitted the 25 percent area coverage tha wa predicted.

7.4.2 Test2: Active inactive list test

```

1      public void updateStars(){
2          synchronized (gameThread) {
3              MovableObject moDel;
4              // update active star positions
5              for (MovableObject mo: gfm.activeGameObj) {
6                  Log.v(LOG_SPYRO_V + "updateStars:activeStars", Integer.
7                      toString(mo.sunNumber) + " : " + Float.toString(mo.
8                          objectPosition.x)
9                      + ":"+ Float.toString(mo.objectPosition.y) );
10                 mo.objectPosition.x -= mo.objectSpeed.x;
11                 Log.v(LOG_SPYRO_V + "updateStars:activeStars", Integer.
12                     toString(mo.sunNumber) + " : " + Float.toString(mo.
13                         objectPosition.x)
14                     + ":"+ Float.toString(mo.objectPosition.y) );
15             }
16             // :TMPFIX: check positions of game objects :
17             int index=-1;

```

```

14     for (int i =0 ; i < gfm.activeGameObj.size() ;i++) {
15         moDel = gfm.activeGameObj.get(i);
16
17         // disappear on the left of the game area : remove from active
           obj
18         if (moDel.objectPosition.x + moDel.objectBitmap.getWidth() < 5 ){
19             Log.v(LOG_SPYRO_V + "updateStars:forRemoval", Integer.
               toString(moDel.sunNumber) + " : " + Float.toString(moDel
                 .objectPosition.x) + ":"+ Float.toString(moDel.
                 objectPosition.y) );
20             // move to Inactive Queue and remove from active Queue.
21             index=i;
22         }
23     }
24     // only one star can be removed per round, : Remove correctly from
           arrayList: @chk
25     // This change seems to have fixed something, now we have stars on
           the screen, before no stars appeared
26     if (index != -1 ) {
27         moDel = gfm.activeGameObj.remove(index);
28         gfm.inactiveGameObj.add(moDel);
29     }

```

Listing 7.13: active Inactive game object lists

In the listing the Game Objects are updated

Test the list list size $act + list\ size\ inact = 5$

After object removal $act_{listsize_before} - 1 = act_{listsize_after}$; $inact_{listsize_before} + 1 = inact_{listsize_after}$;

After object add $act_{listsize_before} + 1 = act_{listsize_after}$; $inact_{listsize_before} - 1 = inact_{listsize_after}$;

Test2: Active inactive game obejct list re- moval and add		
TestName	Expected result	Actual result
start game: inactive ac- tive list size	inactive 5 active 0	Failed
after object removal:		
after object add:	$act_{listsize_before} + 1$ $= act_{listsize_after}$; $inact_{listsize_before} - 1$ $= inact_{listsize_after}$;	Failed

Both tests failed. The inactive list keeps growing linear in time. While the active list goes up to 5 items and stop there. A discussion as to why can be found here. 7.3.2 on page 110

7.4.3 Graphics artifacts error and stars

In the game new Sunstars should appear at the start of the screen to right and move to the left one at the time. There is an error in the game where the stars comes in a close row

almost on top of each other.

```

1 public Point placeNewStar(Point position , ArrayList<MovableObject> activeArray
2   ){
3     // create new sunstar , set its position
4     //SunStar s = new SunStar(context); //
5     // PROBLEM : Sunstar might have some position , we must add a non used
6     // MO object (inactive)
7     SunStar s = gmf.sunstar1;
8
9     if (updown == -1){
10      y = y*0.6f ; //screenYResolution*0.6f;
11    }else if (updown == 1){
12      y = y*0.6f ; //screenYResolution*0.1f;
13    }
14    s.objectPosition.x = x;
15    s.objectPosition.y = y;
16    s.objectSpeed.x = 10f;
17
18    updown = updown*-1;
19
20    float roadXLength = (float)Math.PI*2*3; // 3 waves
21    float stepLengthX = roadXLength/((float)Math.PI*6);
22    float xPerPixel = roadXLength/x;
23    float relSize = x/roadXLength;
24
25    activeArray.add(s);
26    return new Point((int)0,(int)0);
  }

```

This code explains the error with the disappearing Sunstars. The active inactive object lists aren't used in this place and only `sunstar1` is reused and updated. (residual bug from testing 1 Sunstars behaviour). The commented solution above on line 3, creating a new sunstar won't work either since this will create a new Sunstar each time a new sunstar enters the screen instead of using those already created and stored away in the *inactiveGameObj* *activeGameObj* list, this happens continuously through the game so that would create more and more objects as time passes.

The solution is to use the inactive list and move the objects to the active object list.

Test:

Chapter 8

Evaluation

8.1 Planning

Before the project started it was decided to use **UP Unified Process** and planning with vision, elaboration, construction and transition phases. Shortly into the project it was decided to change to a scrum like process instead. The danger of UP is to include too many artifacts, which seems to be a common mistake. Scrum methodology is easier, the difference in my perception is that in UP you have a large pool to choose from and deciding what is necessary to use can be overwhelming, Scrum on the other hand is a work planning method at the start there nothing in it(artifacts). You then find out what you need to use for for specific tasks.

Planning in small sprint iterations with a specific focus worked well, especially in the beginning and middle phase.

The 14 days of one sprint makes sure that a satisfactory progress can be made and some detailed work can be done.

There is time to reflect on the subject, do research and test out solutions. There is time to accomplish a level of completeness, so progress is easily evident. This creates energy for the next sprint that you can see a clear progress on each item or focus area.

When reaching the last third of the project period the planning sprint began to slide. Causes for that is that i had to focus on to many subjects in one sprint in the later phases of the project, which made me do to many things at once. Especially sprint 5 simply went wrong after a good start, this happened because of multiple focus on report writing while the sprint focus was implementing and extending the prototype.

8.2 Problems during the project and workarounds

Eclipse development environment: The Eclipse development system created several problems during the project. For every new feature that was used, consistently there was a learning curve of how to use it and how not to use it to get it to work properly. Examples are that you need to clean files in some circumstances but it is not clear how to know when, sometimes the application wouldn't run unless the log entries were removed from the console and or the logcat log window. The biggest fault was that debugging didn't work, breakpoints in the eclipse debugger, plain did not work, this is an important feature as the application grows in size and complexity. Also the game runs with more than 1 thread for performance reasons It much more efficient if you can use a thread aware debugger. ??

USB access: A USB spirometer probe was sponsored by **Maribo Medico**. Unfortunate it seemed to not work with the USB controller. A newer probe with USB 2.0 was ordered but did not arrive in time for development.

USB to Android connection and rooting: Enabling root on the Android was tried to enable and install USB drivers to be able to work with USB 1.0. The tablet was run the newest Android version was tried without success. Because of that the development had to go ahead with simulated data.

It is not possible to test the system on the patients before we have a probe, preferably USB 2.0. An alternative could be building the adapter and test on the RH equipment.

8.3 Project goals

The project goals still lacks test and user feedback. Some examinations has been observed. But the game concepts needs to be tested on the patients to ensure the interaction ideas are valid.

There is a growing *mini* framework to build new games on. And a sound basis knowledge of android is obtained.

8.4 Extensions

The parallel development of functionality makes the integration of the different modules a little more complex, the focus on working functionality cause the Implementation and the architecture to be out of sync. The architecture is the ideal it is not the goal.

In **S2** the game system part, only few of the classes are designed between generic game functionality and specific game concepts. To be able to extend the Generic system with new games easily more of the game system must be refactored. The GameFieldMap and Game objects and game player classes are ready to be extended. Other classes need more work.

Implementation features:

The cue system used to signal the user and observing if signals are reacted to, is early in design/implementation stage. I expect this system to generalize the interaction and observation of the users and hereby make it easier to extend new interactions as we learn more about the interaction with child patients.

The graph chart is not very flexible and the chart cannot be manipulated, The user can create a new graph with a new data set, but cannot manipulate the selected data set interactively. This could be developed with one of the researched Chart libraries for Android.

8.5 Conclusion

I was very happy with how the partitioning of the system was done, it made a clear separation of the different major tasks and made it clear to me how the systems should work together, Actually the patient examination was first introduced to improve the visualization of what input would enter into the *game system*.

A problem was that too much time was used on the supporting systems related to the game. More work should have been used to create more game system functionality.

There is in my opinion very well structured documentation for development support for working on and extending the project.

During the project i rediscovered some methods and tools and discovered certain tools that i had derided earlier, especially the data flow diagrams, which i always thought to be quite useless, those helped me a lot to find out where responsibility for different tasks should be in the system.

I wanted the implementation more complete, but there a lot of small applications that are are developed and can be used a as base for further development(see 7.3.12 on page 117).

Developed functionality includes: Touch screen, moving objects, direction change, collision detection, patient database(small but it is mostly more SQL queries and their supporting query classes used with SQLite). There is an application for plotting graph data, there is code for access to raw resources, standard resources, assets and more.

A lot is learned about Android development not just coding, but how it to build good applications following the Android Google developer guidelines and create applications that works efficiently for the user. For example input types for helping the user to be more efficient when entering data. Internationalization and more. This is achieved by using the Android API basis and not use any 3rd party libraries. Now that the basic knowledge is founded, development speed and features can be speed up by using specialized libraries for some of the new or extended features.

The extension design of the generic game concepts still needs to be validated by producing more prototypes. Also the first prototype seems to be unsuited for real testing, the patient simply has to move to fast up and down the screen.

A lot of preparation for later integration with the hospital system is done. With the large amount of design of the patient Examination system. an XML schema is developed.

Appendix A

Terminology & concepts

A.1 Spirometry terminology System Glossary

Spirometry terminology System Glossary	
VC	Vital capacity
RV	Residual capacity
ITGV	Intra Thoracal Gas Volume ($VC + RV = ITGV$) The famous circle graph
PEF	FEV_1 & FVC
FEV_1	Force expiration volume
FVC	Force Vital Capacity
BF	Breathing Frequency (exhalations and inhalations per minute)
Plethysmograph	Whole body measurement , can measure resistance in the complete respiratory system
BPM	Breaths per minute
Probe Spirometer	Measurement device: the device that the patient breathes into, and that measures flow and capacity.
Examination	an examination is the evaluation of a patient, sometimes called <i>test</i> or <i>trial</i> (The trial and test the complete examination, but a tool in the examination, however sometimes in the text, the words are used interchangeably)
Inspire	inhale, breath in.
expire	exhale, breathe out.
respiration	inspire + expire, to breathe.
Asthma	Illness that makes the lung thick or inflexible
Cystic fibrosis	Illness that scars the lunge tissue and affects other organs also

Table A.1: Terminology for spirometry measurements

A.2 Android Terminology

Android Terminology		
AndroidManifest	The setup of the application, capabilities, permissions, required features on device. etc.	
Activity	The activity a user want to perform	
Intent	The intent to do something via an activity, also sending messages and data between Activities	
Intent-filter	a filter in the manifest file, that specify among other which actions the application can fulfill, f.ex reading mail	
Resources	The strings, images, files, databases etc. that the application uses	
Broadcast service	Broadcast for an activity, all apps that can fulfill the request will get notified.	
Context	Each application has a context. The context gives access to the resources, assets and much more	
Services	A service can be made if something need to be run while not having screen focus (music etc.)	
Views	The graphics elements used to create user interfaces & layouts	
UI thread	The application MAIN thread, all views are displayed in this thread	

Table A.2: Android terminology

A.3 Project terminology

Terminology and concepts used in the project. The terminology might not always match the one used in the hospital domain, or the Android terminology.

This section should make it clear what is meant in the context of this report and tries to stay as close as possible to terminology used in Android(TM) and at Righospitalet (**RH**).

Project terminology		
Spiro	The internal project name	Used in packages, and for the market id
Probe	The measurement device that the patient breathes into	
Examination	trial, test, examination	Measurement procedure of the patients lung functionality
Patient examination System	Responsible for the patient history & analysis	S1
SpiroGame System	The game system that should be able to support different game prototypes	S2
DataInput System	Representing input to game, from the probe hardware, or from data generator	S3
Cue	(bio) feedback concept	time for the patient to do X...

Table A.3: Project terminology

Appendix B

Risk assessment

Risk	Description	Issue	Risk level	Solution
Sickness	Have previous longtime illness	Causes tiredness and pain, and decreases the time work can be done on project, also treatments take time out of the day	High	Keep tabs on tiredness level, organize treatments ahead, regular intervals,
Cannot access RH probe hardware	RH has legacy hardware	We cannot borrow for extended time, because the have only the equipment that is in use at the hospital	Medium	We try to isolate this problem away.

Cannot access USB	For some reason we might not be able to access the probe and get data	Because of fragmentation of the Android hardware, some things might work on on device but not on other, USB has a history of trouble on the android device	Low	Make a generator early on to simulate probe measurements. Android Os version 3.1+ should have improved USB support, so run with newest possible, that is 4.2
Android tablet crashes		If the Android tablet crashes we will be unable to test with the probe and test with children. Also there is no money to buy new tablet.	Low	Can use own phone for device test, not optimal because of screen size.

Computer loses some or all project data			Medium / low	Backup to central server. Use version control of source code and report.

Appendix C

Use case collection : Patient
Examination system S1 : Game System
S2

C.1 Actors & use cases

C.1.1 Actors

Overview of actors:

- - Patient/test person : the person to be tested for lunge function
- - Doctor : The Evaluator of the examinations.
- - Nurse: Performs the tests and observes the execution and correctness of the test oversees the procedure of the examination session.

Actor:	Test person/Patient
Description:	The patient that are about to be tested and measured, for lung functionality.
Goal:	To measure the current state of lung function/health and to add that to a data collection, so that progress can be followed. Creating a history of health progression/regression.

Actor:	Supervisor
Description:	The doctor or nurse, that supervises the test, initializes the equipment, instructs the patient and observes that the examination is valid, useful and done correct.
Goal:	To make it easier to "instruct" / "guide" the patient through an Examination test

Actor:	Treatment analyser
Description:	The doctor / nurse, check test data, compare patient progress, browses and analyses patient statistics
Goal:	To see if the patients treatment is giving progress, analysing the data over time to see progression/regression.

Actor:	Information browser
Description:	The test person: can check the data visualized for the test, and see progress from earlier test measurements
Goal:	To let the patient follow and their relatives(followers) their progression and status without requiring deep knowledge of the subject

NOTE: the test person gets access to a simplified view of the data, or maybe a replay of test i.e. the game replays the session.

Actor:	Advisor
Description:	Follows the patient and comes with input advice and support to the patient. Has access to all historic data, and might also be the observer(?), so that s/he can counsel the patient
Goal:	The advisor is the specialist/doctor, that follows the patient and gives advice on progress and status.

C.1.2 Use case diagram

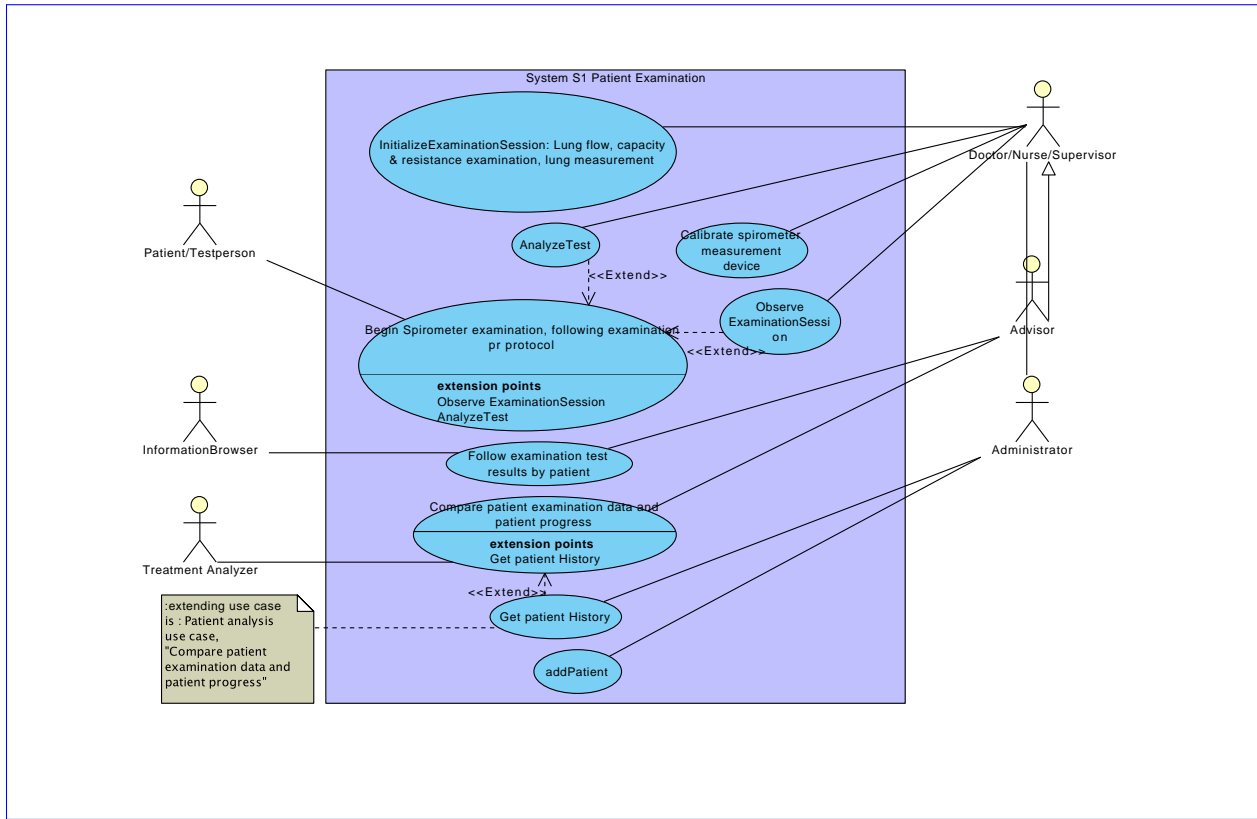


Figure C.1: Use case diagram: Patient Examination System

C.1.3 Use case descriptions

NOTE: this is a sub use case from general types of tests. Use cases for each test is not needed, since the work flow is almost the same. If the work flow differs significantly it's extended with named extensions.

Use case Id: UC:PE1	
Use Case Title: Examination <i>initialization</i>	
Primary actor:	Supervisor
Level:	wip: Work in progress
Stakeholders:	Doctors, Nurses, Patients
Precondition:	Patient has arrived
Success Scenario	System has existing patient data and lung capacity and flow limits for the patient. Meta data is recorded, (metadata: name, birthday, ID data, weight, height). The examination is ready to proceed.
Trigger:	
Standard Scenario:	<ol style="list-style-type: none"> 1. Patient Id data and height, weight, name are registered in to the application. 2. Patient flow and capacity limits are set or read from earlier examination. 3. The examination is started according to protocol (see extensions): 4. Patient breathes in to the probe, recording, biofeedback and game is started(See Use case for Examination: ?? on page 140)
Extensions:	<ol style="list-style-type: none"> 1. Faulty equipment: test is aborted: 2. Other measurements and calculations not yet thought of might occur. 3. Closed Box lung resistance, Cold air, Cycle test, CT Max/min Scan.
Comments:	This use case is extended by the different examination types available. ex: use case "Lung flow and capacity examination, lung resistance examination test(closed Box)". (@all examination type use cases)

Figure C.2: UC PE1: Initialise examination

Use case Id: UC:PE2	
Use Case Title: Closed Box Examination (lung resistance examination)	
Level:	wip
Stakeholders:	Doctors, Nurses, Patients, Follower
Precondition:	Use case Examination <i>initialization</i> is done
Minimal Guarantee:	
Success Guarantee:	The patient keeps an steady respiration rhythm and a sufficient strength amplitude, so that the measurement is usable. The measurements are recorded in detail and the data are stored away.
Trigger:	...
Standard Scenario:	<p>Patient breathes in to the probe, recording is started,</p> <ol style="list-style-type: none"> 1. - The game tries to adjust the test persons breathing if it is to slow or to fast. 2. - Feedback to test person, to ensure the correct strength-/amplitude of the breathing. 3. - The game tries to help the user to correct breathing with feedback. 4. - When the patient is doing well the game can adjust difficulty up, or down if the patient errors again. 5. (@other detection? of bad measurements.)
Extensions:	<ol style="list-style-type: none"> 1. If the game is unsuccessful at correcting the respiration of the patient the game can restart, and reintroduce the patient to the execution of the examination. 2. Patient breathes to fast or to slow, and can't adjust to correct breathing pattern, the session times out and ends with an overview, The game signals for manual intervention.
Comments:	<p>This use case provides a description for examination. But it doesn't specify the exact nature of the feedback, also use cases like <i>use visualization</i> or <i>give visual hint</i> doesn't give information of the system. @see Game Scenarios section.</p> <p>NOTE: this is a use case for general types of examinations.</p>

Figure C.3: fig:Lung resistance test, closed box

Use case Id: UC:PE3		
Use Case Title: Use Case Title: Analyze examination data & patient progress		
Primary actor:	Treatment analyzer, Advisor	
Level:	wip	-
Stakeholders:	Doctors, Nurses, Supervisors, treatment analyzer	
Precondition:	Minimum one examination test is executed before.	
Minimal Guarantee:		
Success Guarantee:		
Trigger:		
Standard Scenario:	<ol style="list-style-type: none"> 1. The actor searches and compares examination data. 2. The Treatment analyzer chooses, <i>Analyse Patient</i>. 3. Chooses 1 patient. 4. Chooses a time period. 5. The patient data for that time period is displayed on screen (graph). 6. The data and progression can be visualized and data can be chosen or added. 	
Extensions:	<ol style="list-style-type: none"> 1. The Analyser can go back and pick patient and examination and a new time period. 	
Comments:	This use case are not directly involved with the system "the Game" we want to develop. But it is important to be able to use the data after, and the way the data can be assessed and evaluated can be important for the system to be a success. That is why we want to make the game.	

Figure C.4: Compare data and patient progress

Use Case Title: Observe spirometry examination trial		Use case Id: UC:PE4
Primary actor:	Supervisor	
Level:	wip	
Stakeholders:	Supervisor, Advisor, Patient, Observer	
Precondition:	Use Case for Examination is started.	
Trigger:		
Standard Scenario:	The supervisor can see the screen with the measurement data. He observes the patient, and the biofeedback game. The supervisor can watch if the test is executed correct by watching the measurements screen and the patient. generally the supervisor aids the patient, all through the examination.	
Extensions:	If the test is not done correctly and the game has unsuccessfully tried to change the way the patient breathes. The supervisor can take note of that or intervene, do a new test or other decisions.	
Comments:	The game can return to a base line behaviour, reintroducing the patient to the game/test. i.e. the active game stops, asks the patient to relax. gives introductory instructions to start up again. (should be in extension!) The game should be forgiving in the first while the patient settles in. This use case will be explored under game use cases (?? on page ??) Because this use case describes a mostly manual process we will see where the system can aid or take over some of the tasks. The idea is that the patient should be able to play and during play as a side effect, "execute" the examination trial with high quality in measurement reliability etc.	

Figure C.5: Observe spirometry tests

Use Case Title: Follow (watch) test results		Use case Id: UC:PE5
Primary actor:	Patient, advisor	
Level:	wip	
Stakeholders:	Patient, advisor	
Precondition:		
Minimal Guarantee:		
Success Guarantee:	Test person can follow his her measurements and progress in a comprehensible way for a layman.	
Trigger:		
Standard Scenario:	The Patient or test person, want to follow his her own progress, starts the application chooses the wanted examination and can watch it.	
Extensions:	None	
Comments:	examples: The user might want to see how the last test was executed, or how well the lung functioned. A time period might be chosen.	

Figure C.6: Follow test results

Use Case Title: Add patient to system	
Primary actor:	Supervisor, Nurse
Level:	wip
Stakeholders:	Patient, Supervisor, Advisor, Follower.
Precondition:	The patient cannot already exist.
Minimal Guarantee:	
Success Guarantee:	A patient is added to the patient information database, and can now start a testing program
Trigger:	New patient coming
Standard Scenario:	<ol style="list-style-type: none"> 1. Supervisor chooses add patient 2. Patient data is entered <ol style="list-style-type: none"> a) PatientID (CPR) b) Name c) Birthday 3. Patient Initial data is registered. (First patient examination card is registered.) <ol style="list-style-type: none"> a) Weight b) Height c) Date of registering/examination 4. Supervisor confirms patient data 5. Data is stored
Extensions:	
Comments:	

Figure C.7: Add patient data

Use Case Title: Choose existing patient for examination		Use case Id: UC:PE7
Primary actor:	Supervisor, Nurse	
Level:	wip	
Stakeholders:	Patient, Supervisor, Advisor, Follower.	
Precondition:	The patient MUST exist in the system.	
Minimal Guarantee:		
Success Guarantee:	The chosen patients data comes on the screen and is ready to start examination.	
Trigger:	Examination started	
Standard Scenario:	<ol style="list-style-type: none"> 1. Supervisor chooses a patient in the system, by name or Id 2. Patient data appear <ol style="list-style-type: none"> a) PatientID (CPR) b) Name c) Birthday d) Weight e) Height f) Test date g) Limits (lung capacity, amplitude/strength etc.) are set or adjusted. 3. Supervisor confirms patient and examination is ready to start. 	
Extensions:	1. Limits are altered, because of patients condition (took medicine recently, sick, etc.) This is examination initializing with existing user, this includes the initialize examination use case ?? on page 139 it describes the first 2 steps in more detail	
Comments:		

Figure C.8: Choose existing patient for examination

C.1.4 Use case candidate list

1. Add Patient
2. Choose existing patient in system
3. Accept test session: after a test, the observer can OK the test, or discard it. if it is discarded it will be thrown away otherwise it is stored.
4. Compare patient data: alt. scenarios how well the test is performed, patient progression (how is it measured), flow, limits, volume
5. Analyse patient measurement just performed.(check it the test can be OK'ed).

C.2 Use case tech memos

Technote: Test observation Emilie, test types

test type question for Kim Gjerum

Note: there must be two test types Kim described one test where you breath slowly, with a respiratory speed of about 30-35 breaths per minute Also the breathing should have a specific strength / amplitude.

The Test i was watching with nurse Nadia and test person Emillie(8 years)(?) was another test, here the box was open and the test was done like this: the child is ready an breathes normally, easily, at some point in time the nurse tells the patient to exhale, maintaining the exhalation for some extended time. The difficulty can be changed So that the patient have to apply more respiratory strength, to keep up. if the patient stops in the middle to let in breath, the test must be redone.

include this form included file in appendix

Use Case Title: Init game	
Primary actor:	Patient
Level:	wip
Stakeholders:	Patient, Supervisor, Advisor, Follower.
Precondition:	Patient Examination Session is started.
Minimal Guarantee:	
Success Guarantee:	
Trigger:	
Standard Scenario:	<ol style="list-style-type: none"> 1. Patient chose start game. 2. The game starts Init setup. 3. Objects are placed on the screen. 4. The game map is initialized 5. Game State is set to running state. 6. The input data from the probe is read throughout the game.
Extensions:	
Comments:	

Figure C.9: Game is initialized,ready to start

Use Case Title: Patient breathes to slowly	
Primary actor:	Patient
Level:	wip
Stakeholders:	Patient, Supervisor, Advisor, Follower.
Precondition:	game is active. UC: start game is executed
Minimal Guarantee:	
Success Guarantee:	game on
Trigger:	
Standard Scenario:	<ol style="list-style-type: none"> 1. Patient Breathes into the probe. 2. The patient breathes to slowly. Getting behind the optimal path. 3. The game detects to slow breathing 4. The Patient is signalled to breathe faster. 5. Waiting for patient to react to signal and be stable for period of time 6. Patient has reacted to the signal and have been breathing with the correct speed for some time. 7. Signals?
Extensions:	<p>The patient does not react to the biofeedback cue, and doesn't speed up the respiration. The Game sends a new signal/Event to try get the user to react.</p> <ol style="list-style-type: none"> 1. Player figure gets different colour (blue) 2. Visual hints (Eyes are tilting in the direction it want to go) 3. player figure tries to signal that it want to speed up(fidgeting) 4. The path or something on the path begins to lighten up 5. music plays faster 6. Voice says to go faster
Comments:	These are hard to describe without a concrete game, it can be so general or abstract, that it is non descript.

Figure C.10: UC:Child Player breathing rhythm to slow

Use Case Title: Patient amplitude to large(exercise to much force in breathing)	
Primary actor:	Patient
Level:	wip
Stakeholders:	Patient, Supervisor, Advisor, Follower.
Precondition:	game is active.
Minimal Guarantee:	
Success Guarantee:	game on
Trigger:	
Standard Scenario:	<ol style="list-style-type: none"> 1. Patient Breathes into the probe. 2. The patient breathes with excessive force into the probe. getting above optimal path(). 3. The game detects that the patient breathes above capacity/flow limits (How?) 4. The Patient is signalled to breathe in a relaxed fashion (only, closed box ex.?) 5. Waiting for patient to react to signal and be stable for period of time 6. Patient has reacted to the signal and have been breathing with the correct speed for some time. 7. Signals?
Extensions:	<p>The patient does not react to the biofeedback cue, and doesn't speed up the respiration. The Game sends a new signal/Event to try get the user to react.</p> <ol style="list-style-type: none"> 1. Player figure gets different colour (blue) 2. Visual hints (Eyes are tilting in the direction it want to go) 3. player figure tries to signal that it want to speed up(fidgeting) 4. The path or something on the path begins to lighten up 5. music plays faster 6. Voice says to blow
Comments:	

Figure C.11: UC:Child Player breathing rhythm to slow

Use Case Title: Patient breathes irregularly	
Primary actor:	Patient
Level:	wip
Stakeholders:	Patient, Supervisor.
Precondition:	game is active.
Minimal Guarantee:	
Success Guarantee:	
Trigger:	
Standard Scenario:	<ol style="list-style-type: none"> 1. Patient Breathes into the probe. 2. The patient is observed over time 3. The game detects irregular breathing pattern 4. Cue's triggered. 5. Initialize patient, slow down/pause , and instruct, starting over, or re establish in "quiet" game environment(suggestions@) 6. Waiting for patient to react to signal and be stable for period of time 7. Patient has reacted to the signal and have been breathing with the good rhythm for some time. 8.
Extensions:	The patient does not react to the biofeedback <i>cues</i> , change to regular breathing pattern.
Comments:	

Figure C.12: UC:Child Player breathing rhythm irregular

Use Case Title: Patient breathes correctly	
Primary actor:	Patient
Level:	wip
Stakeholders:	Patient, Supervisor, Advisor, Follower.
Precondition:	game is active.
Minimal Guarantee:	
Success Guarantee:	game on
Trigger:	
Standard Scenario:	<ol style="list-style-type: none"> 1. Patient Breathes into the probe. 2. Game encourages the path 3. Points are gained, 4. Game visible Objects display cue for correct behavior 5. Adjust difficulty ? 6. System observe capacity limits 7. System observe flow limits 8. Patient has maybe reacted to the signal and have been breathing properly 9. How do we find if encouragement works, and keeps the patient in range 10. Cue Signals?
Extensions:	The misbehaviour Use cases are all extensions of this one
Comments:	This us case is important because it touches on what the user gets from the system when doing good, continuously encouraging correct behavior, but it is hard to know when it is needed. A flow diagram might be a good way to show this)

Figure C.13: UC:Child Player breathing rhythm perfect

C.3 Requirements note, physical

Note about requirement 6: Testing on own phone / tablet device:

Thomas(Doctor): stated it would be nice if the game could be downloaded on the users own device.

Then they could get a little plastic pipe to ensure uniform distance to sensor.

The microphone sensor on the device would then be the probe flow sensor.

Their phone + pipe would be calibrated on the hospital.

Then it would be nice if the users could also see their test data

However, there might be an issue of how we can secure that data? (must only be read by patient and followers).

Questions: Is the microphone stable enough, is the air flow steady enough.

Is it feasible(would it work).

Number	Requirement	Description	Comment	Priority
1)	Opsamle live data feed fra probe og visualisere i realtid			1
1)	Visual feedback for children, guiding interaction from patient		On reaction from patient we must react appropriately to help guiding.	1
1)	Adjust speed and rhythm to fit the users level (size, physique and age)			3
1.3)	Set limits on volume according to each patient			2
1.4)	Adjust limit, in game, to decrease / increase games difficulty to help the patient		Helping the patient not getting frustrated, the difficulty level can be made easier, to ensure we don't bore the patient so they lose concentration, we can increase difficulty	1
1.5)	Hint patient to slow or fasten speed rhythm (frequency)			1
1.6)	Hint patient to expire inspire harder or softer -> the Amplitude.			2
2)	Store Patients examination session (what happens under the game.)			2
3,1)	Save patient examination session in PatientHistory .			2
2.2)	After a test confirm(todo: use case extension scenario) that the test is valid, attach comment to it, or reject and delete it.			2
3)	Define test protocols. ?(TODO: is this a requirements?)			3
4)	Add new Patients to the Patient database			2

5)	View patient history(ie earlier test versions)	1
6)	Make sure stored data is secure, since there are data access laws, that must be followed for actual patient data(todo half is comment)	3
7)	Share /n transfer data, via XML, securely	2
9)	Measure Flow and Volume for patients	2
9.1)	Record adjust limits for the patient according to the ExaminationSession (A Use Case..), confirm by **supervisor** .	1

Priorities		
1	Very high	The core of the system, this is the area we work to implement
2	High	Important supporting requirements
3	Medium	Bread and butter functionality, QA work(on scope system), polishing, more/extended functionality
4	Low	Nice things to have but not necessary fro core scope project
5	Very low	Items not belonging to the core project, but has an association, that might be important for other requestors, maybe done as another project

Table C.2: Priorities

Appendix D

Generator S3: simulated input data diagrams

Content:

1. Generator
2. Use case for generators
3. Class diagrams

D.1 Generator modelling

Appendix E

Generator and generator design

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots \quad (\text{E.1})$$

$$= \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}, \quad (\text{E.2})$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots \quad (\text{E.3})$$

$$= \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}. \quad (\text{E.4})$$

E.1 Generator design

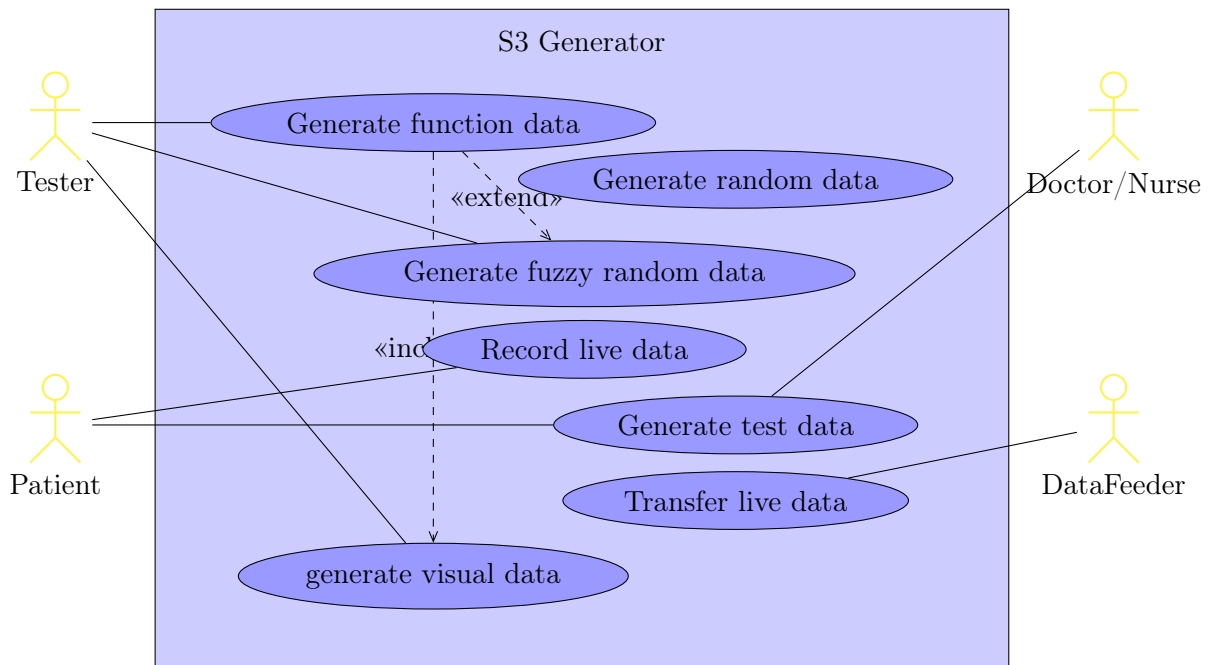
The generator is designed to take over from the USB spirometer we do not succeed in getting access to the spirometer

First of it will generate different outputs of mathematical formulas. exponential, random, cosine, sine. Random function.

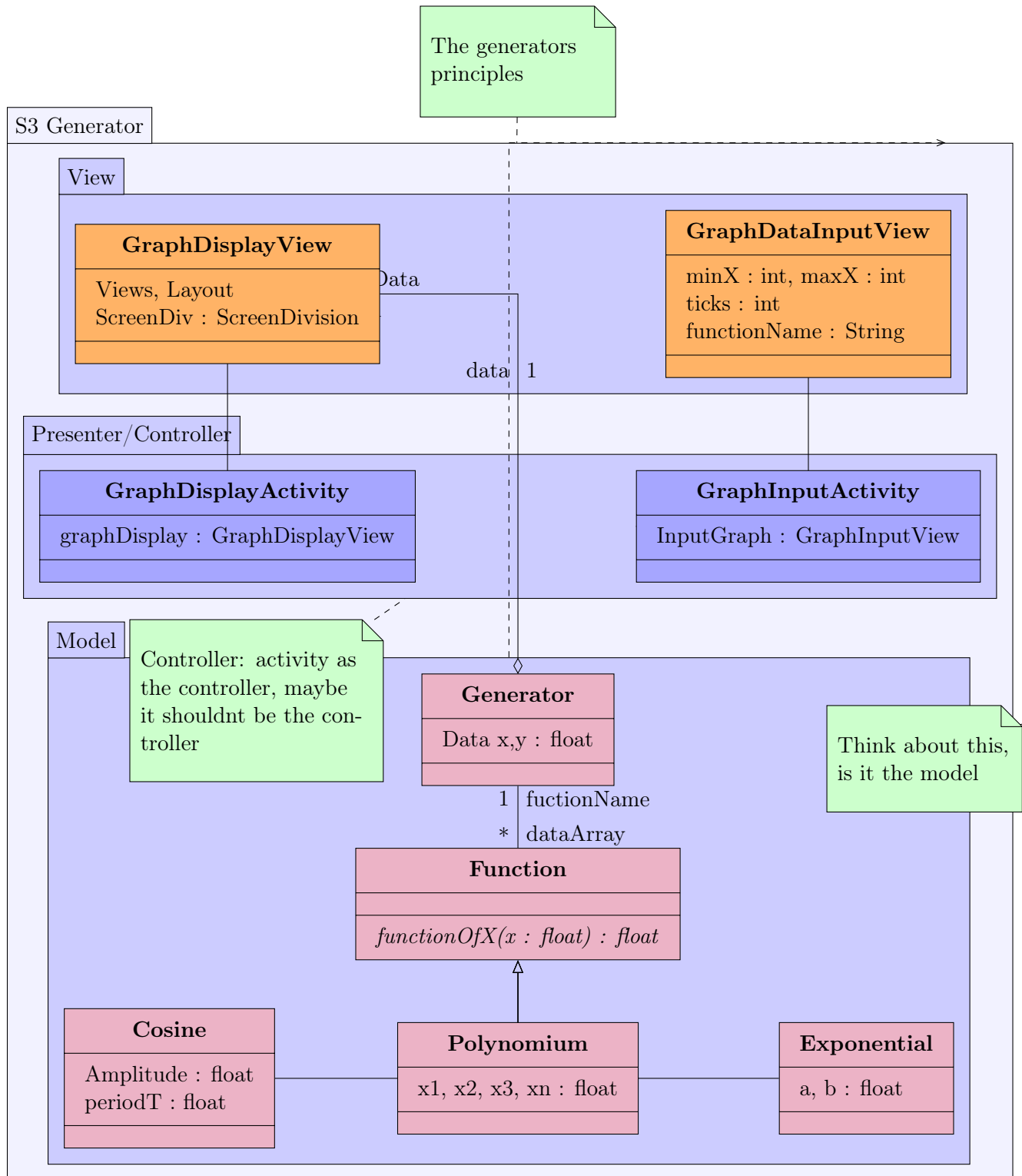
E.1.1 randomized value point interval

A feature that is usable is a random feature on top of the functions, so that each function value will be inside a randomized value, interval a each point.

E.2 Generator Use cases



E.3 Generator class diagram



Appendix F

Class responsibility descriptions

F.1 S1: PatientExamination System

F.1.1 Examination part

ExaminationData

ExaminationData a class for the recorded test. The class tracks the data from the probe. It recounts flow volume and and capacity. And records all the input data.

The recorded data from the examination can be saved on a *patient card*, that will be stored as part of the patient history.

ExaminationSession

ExaminationSession is responsible for, initialization, protocol, execution, and finalization of the examination.

The examination session runs behind the game Saves the Data calculated and recorded during the examination session.

ExaminationProtocol

ExaminationProtocol Is the business logic for the execution of tests. Part of it is out of the control of the system, measuring weight and height of the patient before the test f.ex. However the application could have some check points, before allowing next part of protocol to be executed.

In the future, some of the “external” systems might be integrated with the application.

F.1.2 Patient Part

Patient

Patient is describes the patient personal data.

PatientAdministration

PatientAdministration responsibilities.

Adding patient + transfer data to external systems.

The patient administration is a kind of concept class for all the things that the doctors and nurses could think to do with the patient patient history and patient data.

- Adding new patients.
- Conclusions about treatments, adjusting limits for each patient
- observe the test data
- Journal access, and update.
- move the examination data to PC for analysis
- backup of external data

Many of these things are not going into the build system, so it in the diagram more to remember these parts of the system should be there. if it is a non research /development system

PatientHistory

PatientHistory The concept of a Patient history like a container class. It should be both patient data and analysis data with notes and analysis comments. For now it really doesn't contain more than the patient Information and the patient examination cards.

F.2 S2: Spiro Game System

F.2.1 InputEvent

InputEvent receives the data from the Probe or the data generator via the DataFeeder, it translates the data to an appropriate event or action.

F.2.2 GameController

The *GameController* has the job of distributing the different aspects of the game, events to action, input, calculations of speed, screen sizing etc. Passing events to other objects. Distribute *Cue* events.

The GameController class act as controller in the controller pattern according to **GRASP**¹ terminology. This works fine in smaller systems.

F.2.3 GameFieldMap

The GameFieldMap is the map of the game, it sets the positions of the player objects. Controls collision and removal and addition of game objects.

In First iterations of development it also have the Event control.

F.2.4 Game Objects

Game objects is a class for all the common behaviour and attributes of the game sprites. Positions, bitmap used, Actions it can do, movement and more.

F.2.5 GameEvents

The events that happens in the game.

F.2.6 Cue

Cues defines different types of feedback to the player. The conditions, and trigger for when the *cue* should initiate.

F.2.7 GameState

GameState controls the state in the game. There are 2 different states. 1 state control if the game is running, paused, stopped or in initialize state.

F.3 S3: DataInput System : Probe Measurement Part

F.3.1 SpirometerMeasurementDevice

Responsible for the hardware and hardware near systems. Measurement hardware, Enable UsbConnection, Raw data.

F.3.2 ProbeDataCalculation

Getting the data from the probe in a more representable form. Takes raw data and turn it values of acceleration, speed, flow.

F.4 S1-S2-S3: DataFeeder

DataFeeder has the responsibility of Supplying the game with input data from the probe or generator at timed intervals.

While the systems are developed the DataFeeder exists in all System parts. This must be integrated later.

Appendix G

Selected sequence diagrams

G.0.1 Patient examination sequence diagrams

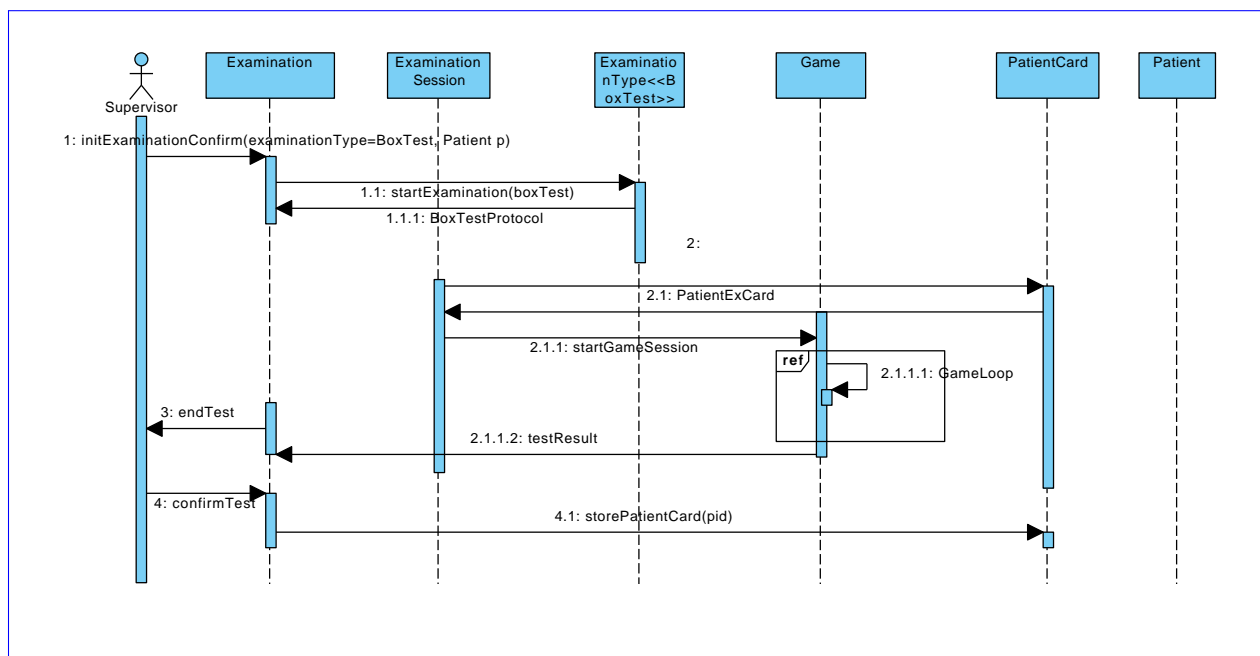


Figure G.1: Make box examination sequence diagram

G.0.2 Game sequence diagrams

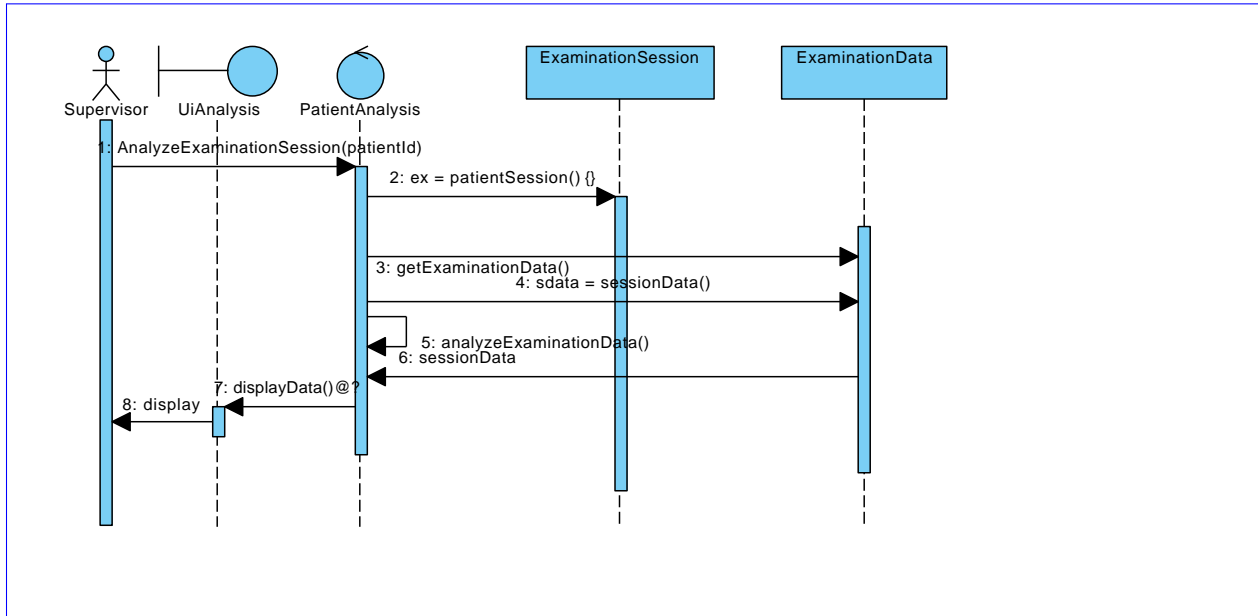


Figure G.2: Analyse examination sequence diagram

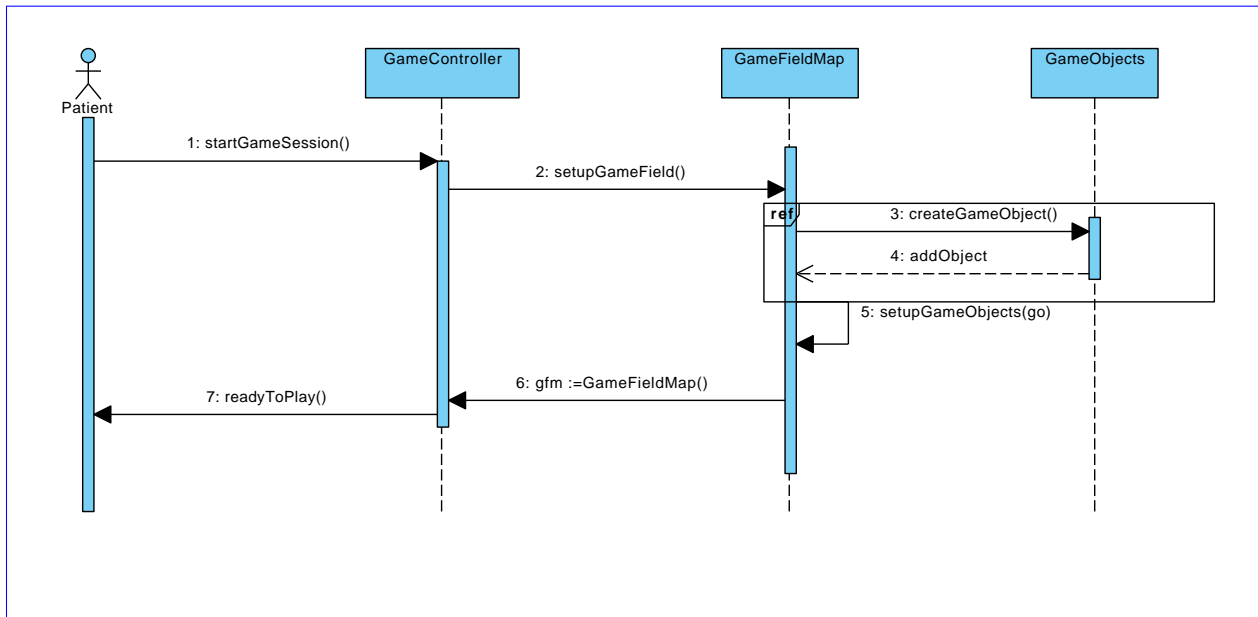


Figure G.3: Init game sequence diagrams

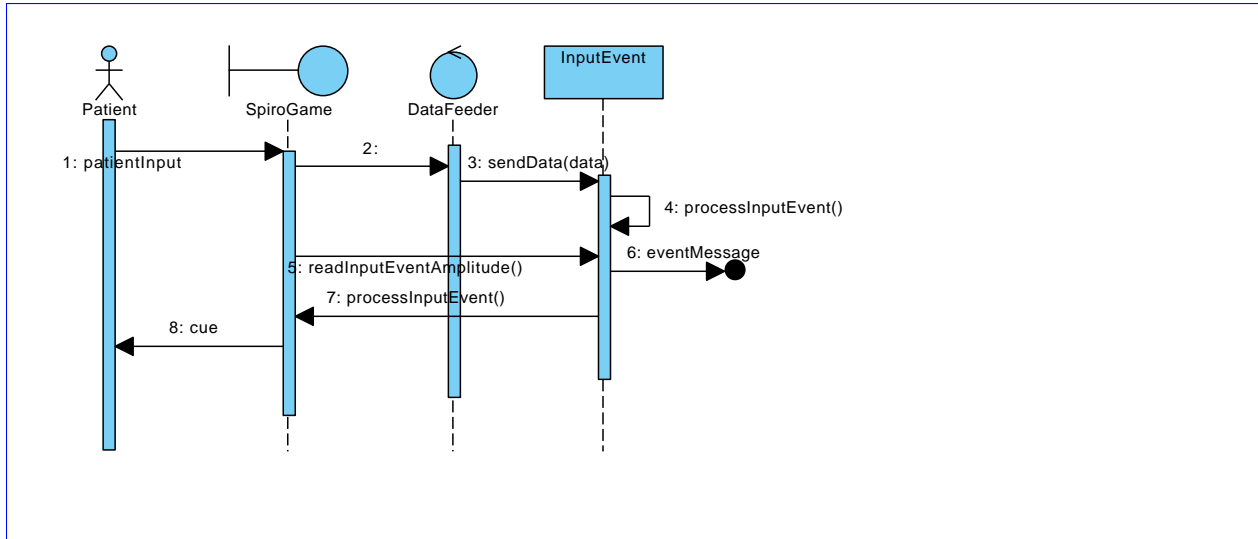


Figure G.4: Input game loop sequence diagrams

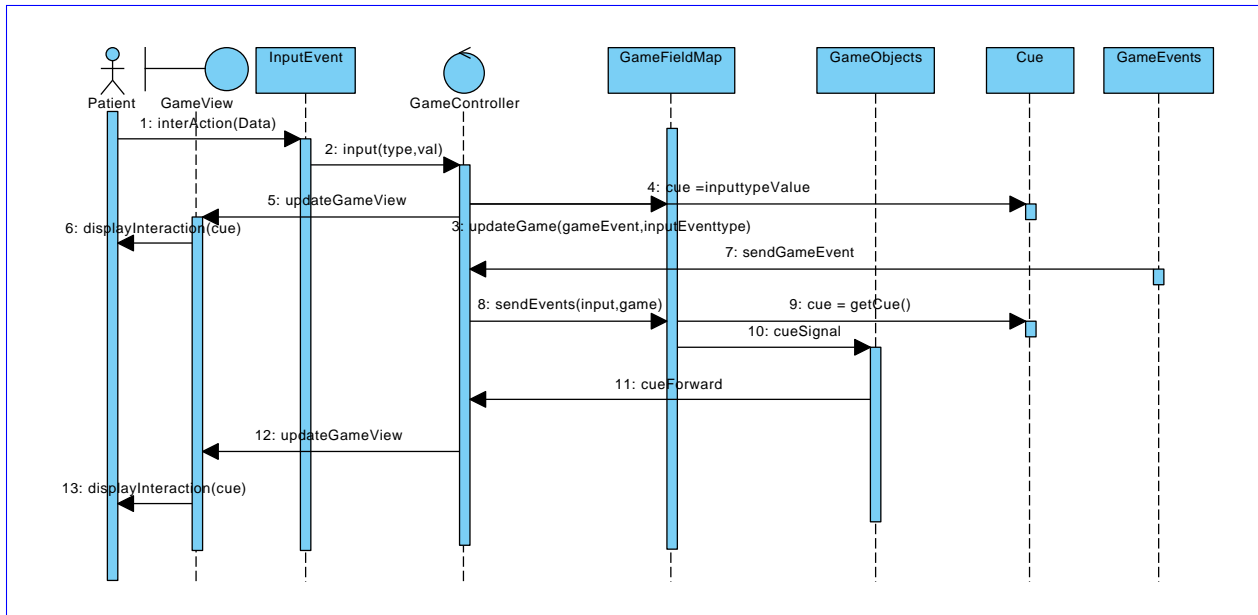


Figure G.5: In game sequence diagrams

Appendix H

Colour styling of diagram architecture and figures

H.1 Colouring legends

Colour legend: Peter Coad

1. Roles class
2. Description class()
3. Party, thing
4. Moment-interval

Figure H.1: Colouring legend

Colour legend: author:mkj

1. Controlling object(Ex: PatientAdministration)
2. is-a-thing object (Patient)
3. Manipulating object, (Ex: InputEvent)

Figure H.2: Colouring legend

H.1.1 Spiro classification system

Analyzing systems and class systems

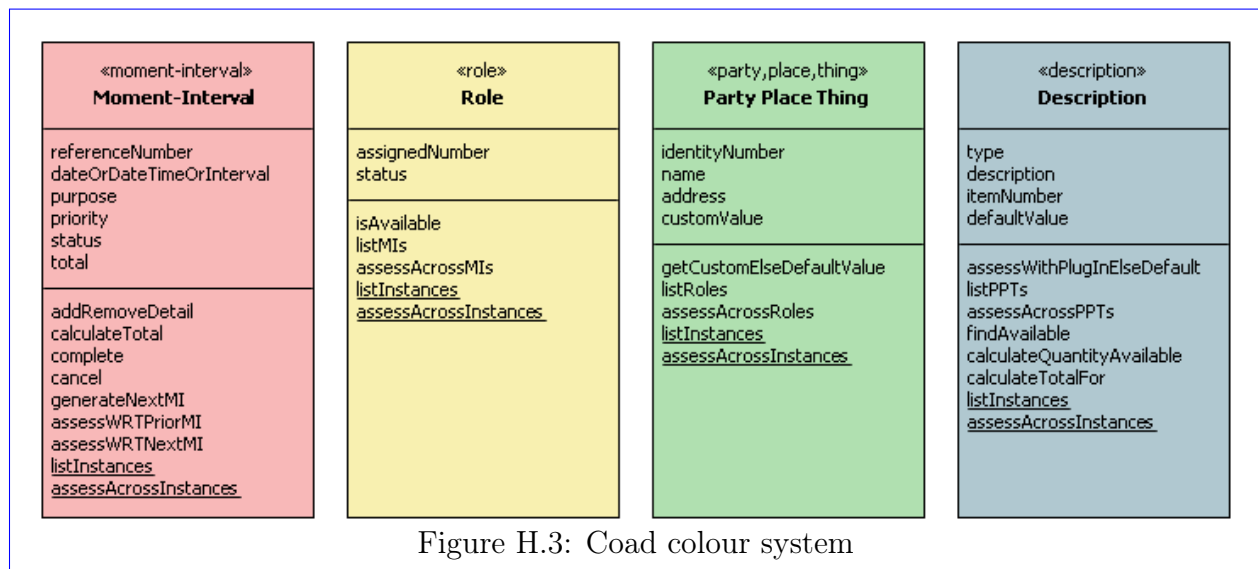
We use colour styling in our report, chapters figures and especially class diagrams etc.

The colour items have significance, because the partition the classes in to different meta types or behaviours etc.

We have 2 colouring methods, one created by the author of this report(no peer review). 1 created by Peter Coad et al. Described in the book *Java Modelling In Colour With UML: Enterprise Components and Process*. Coad

The colouring was researched and one of them created because of the authors wish to have a tool, for thinking about classes and draw what was thinking, furthermore create something that the eyes can snap onto, basically, remove the orientation coordinates, without adding more artifacts in the class diagrams, instead of using energy on finding stuff on the map,we create something that helps follow the

Peter Coad’s classification method



Classification method from author:mkj

the first classification system was a concept invented by the author to help analysing the classes and get a grip on what the system *is*, The classes are coloured According to groups the belong to and inside the group how they relate to each other. how the relate to each other. Classes are grouped into groups, that together naturally form a system task or a responsibility area, f.ex. the classes that are directly responsible for the Examination test, is considered one unit, while the unit that have patient responsibilities are coloured another colour

When looking down on the diagram it is immediately visible, what responsibility area you are looking at, because the areas are clearly divided by their colour

We have classes the represent and object, then we have controlling object that imposes something on the object

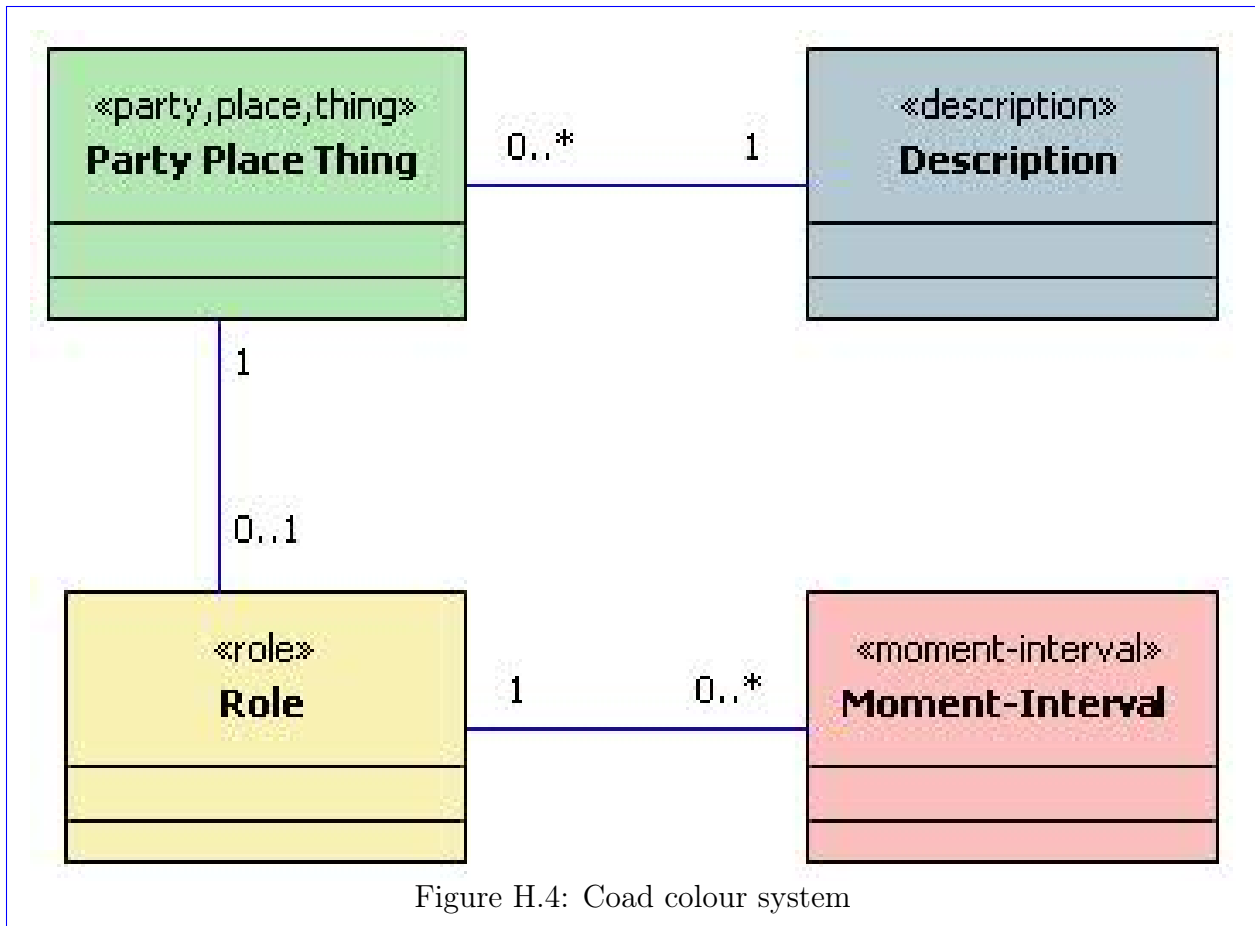


Figure H.4: Coad colour system

there is also other classes that looks like control classes but that don't actually do exercise control.

Example could be a **house** as the object, then we have a maintenance class let's call it HouseMaintenance, that controls maintenance, we also have a House Painter class. Even though the HousePainter class "imposes" on the **House** by painting it some colour, it is not considered to impose control over the house. More likely the HouseMaintenance class initiated a new house painting. and *told* the housePainter class to paint the house.

Then there a two types of action or behaviour classes, and they will be coloured differently, because of that difference.

Appendix I

Xml for patient data transfer between RH system interfaces

To integrate and communicate with other Hospital systems the information benefits from being in an XML format.

Rough draft of patient information in hierarchical format:

```
TestPersondata:  
Name  
Birthday  
limits  
testdate  
Age  
Height  
Weight  
Limits[]  
TestCollection[]
```

I.1 JSON or XML

JSON:

```
1 {"TestData": {  
2   "name": "Santa",  
3   "Id": "_cprnr",  
4   "heigth": "117",
```

```

5  "weight": "28",
6  .. more here ....
7  }}

```

XML: With DTD definition

```

1  <?xml version="1.0"?>
2  <!DOCTYPE TestPersonData [
3      <!ELEMENT TestPersonData (name|Id|height|weight|limitsList|
4          TestCollection)*>
5      <!ELEMENT name (#PCDATA)>
6      <!ELEMENT Id (#PCDATA)>
7      <!ELEMENT height (#PCDATA)>
8      <!ELEMENT weight (#PCDATA)>
9      <!ELEMENT limitsList (limit)*>
10     <!ELEMENT limit (_limitname1|_limitname2|_limitname3)*>
11     <!ELEMENT _limitname1 (lmvalue)*>
12     <!ELEMENT lmvalue (#PCDATA)>
13     <!ELEMENT _limitname2 (lmvalue)*>
14     <!ELEMENT _limitname3 (lmvalue)*>
15     <!ELEMENT TestCollection (testdataId|dataSrc)*>
16     <!ELEMENT testdataId (#PCDATA)>
17     <!ELEMENT dataSrc (#PCDATA)>
18 ]>
19 <TestpersonData>
20     <name>Santa</name>
21     <Id>_Cprnr</Id>
22     <height>115</height>
23     <weight>27</weight>
24     <limitsList><limit>
25         <_limitname1><lmvalue>_value</lmvalue></_limitname1>
26         <_limitname2><lmvalue>_value</lmvalue></_limitname2>
27         <_limitname3><lmvalue>_value</lmvalue></_limitname3>
28     </limit></limitsList> // might not need to be list
29     <TestCollection>
30         <testdataId>_date_n_x
31         </testdataId>
32         <dataSrc>
33             Whole lot of data here or is that not correct xml, a tag for each data
34             point seems
35             a huge waste.
36         </dataSrc>
37     </TestCollection>
</TestpersonData>

```

XML has more tools available than JSON. JSON more compact for mobile devices, but not really need compact since we don't send messages between mobile objects. Much more

likely to send to Workstation computer if need to be analyzed. XML can be packaged to be of roughly the same size as JSON for network transfer etc.

JSON JQUERY: Xml alternative

<http://iviewsource.com/coding/getting-started-with-javascript-object-notation-json-for-a>

XML Tutorial: <http://www.w3schools.com/xml/>

XML: xmlserializer http://www.anddev.org/write_a_simple_xml_file_in_the_sd_card_using_xmlserializer-t8350.html

Appendix J

Test data

J.1 Test1: Collision

Subtract 64 from the Dy: this give the collision range. Because of a constant that was erroneously introduced adding 63.

```
1 Player Above:
2 05-17 06:42:57.877: VERBOSE/GameField:doCollSize(2876): 125 102
3 05-17 06:42:57.877: VERBOSE/GameField:doCollSize(2876): 112 125
4 05-17 06:43:04.657: VERBOSE/GameField:doCollSize(2876): 125 102
5 05-17 06:43:32.447: VERBOSE/GameField:doCollSize(2876): 125 102
6 05-17 06:44:00.337: VERBOSE/GameField:doCollSize(2876): 125 102
7
8 05-17 04:33:55.850: VERBOSE/GameField:doCollPos:playerObj__(16929): 0.0 128.0
   478.0
9 05-17 04:33:55.850: VERBOSE/GameField:doCollPos:gameObject(16929): 10.0 240.0
   480.00003
10 Dx: 112 Dy: 2
11 05-17 06:42:57.817: VERBOSE/GameField:doCollPos:playerObj__(2876): 0.0 128.0
   438.0
12 05-17 06:42:57.817: VERBOSE/GameField:doCollPos:gameObject(2876): 10.0 240.0
   480.00003
13 Dx: 112 Dy: 42
14 05-17 07:14:36.747: VERBOSE/GameField:doCollPos:playerObj__(4175): 0.0 128.0
   383.0
15 05-17 07:14:36.827: VERBOSE/GameField:doCollPos:gameObject(4175): 10.0 220.0
   480.00003
16
17 05-17 07:16:51.977: VERBOSE/GameField:doCollPos:playerObj__(4175): 0.0 128.0
   368.0
18 05-17 07:16:51.977: VERBOSE/GameField:doCollPos:gameObject(4175): 10.0 220.0
   480.00003
19 Dx: 92 Dy: 112
```

```

20
21 Player below :
22 05-17 07:21:41.627: VERBOSE/GameField:doCollPos:playerObj__(5047): 0.0 128.0
    578.0
23 05-17 07:21:41.627: VERBOSE/GameField:doCollPos:gameObject(5047): 10.0 250.0
    480.00003
24 Dx: 122 Dy: 98
25 05-17 07:39:55.967: VERBOSE/GameField:doCollPos:playerObj__(6050): 0.0 128.0
    583.0
26 05-17 07:39:55.967: VERBOSE/GameField:doCollPos:gameObject(6050): 10.0 230.0
    480.00003
27 Dx: 102 Dy: 103
28 05-17 07:25:12.087: VERBOSE/GameField:doCollPos:playerObj__(5047): 0.0 128.0
    588.0
29 05-17 07:25:12.087: VERBOSE/GameField:doCollPos:gameObject(5047): 10.0 220.0
    480.00003
30 Dx: 92 Dy: 108

```

Listing J.1: Test1 Collsion

J.2 Test2: Active inactive

The test data shows that new objects are created instead of changing between the active and inactive list. So that must be FIX'ed.

```

1 05-17 10:58:47.828: VERBOSE/SpyroView:updateStars:activeStars(11123): 7 :
    1110.0:480.00003
2 05-17 10:58:47.828: VERBOSE/SpyroView:updateStars:activeStars(11123): 7 :
    1100.0:480.00003
3 05-17 10:58:47.858: VERBOSE/onDraw:PLAY:Active(11123): active objects: 5
4 05-17 10:58:47.858: VERBOSE/onDraw:PLAY:InActive(11123): inactive objects: 7
5
6 05-17 10:59:24.268: VERBOSE/SpyroView:updateStars:activeStars(11123): 7 :
    310.0:480.00003
7 05-17 10:59:24.268: VERBOSE/SpyroView:updateStars:activeStars(11123): 7 :
    300.0:480.00003
8 05-17 10:59:24.288: VERBOSE/onDraw:PLAY:Active(11123): active objects: 5
9 05-17 10:59:24.288: VERBOSE/onDraw:PLAY:InActive(11123): inactive objects: 17
10
11 05-17 11:01:32.528: VERBOSE/SpyroView:updateStars:activeStars(11123): 7 :
    -150.0:480.00003
12 05-17 11:01:32.538: VERBOSE/SpyroView:updateStars:activeStars(11123): 7 :
    -160.0:480.00003
13 05-17 11:01:32.568: VERBOSE/onDraw:PLAY:Active(11123): active objects: 6
14 05-17 11:01:32.568: VERBOSE/onDraw:PLAY:InActive(11123): inactive objects: 53
15
16
17 05-17 11:01:54.768: VERBOSE/onDraw:PLAY:Active(11123): active objects: 5
18 05-17 11:01:54.768: VERBOSE/onDraw:PLAY:InActive(11123): inactive objects: 61
19 05-17 11:01:54.828: VERBOSE/SpyroView:updateStars:activeStars(11123): 6 :
    1117.3335:81.899994
20 05-17 11:01:54.828: VERBOSE/SpyroView:updateStars:activeStars(11123): 6 :
    1117.3335:81.899994

```

Listing J.2: Test2: Active inactive game objects

J.3 Entity relationship for Spiro

Tables:

PatientHistory:

```
# examinationId + cprid + Name + surname + Examination data
```

PatientExaminationCard:

```
# cprid + Limit + flow + capacity
```

Examination info:

blue 4pt

t1: Examination table

Examination id [date;testnr]: cprid(int) : time : examination data(blob)

t2: Patient table

tx: gameinfo table

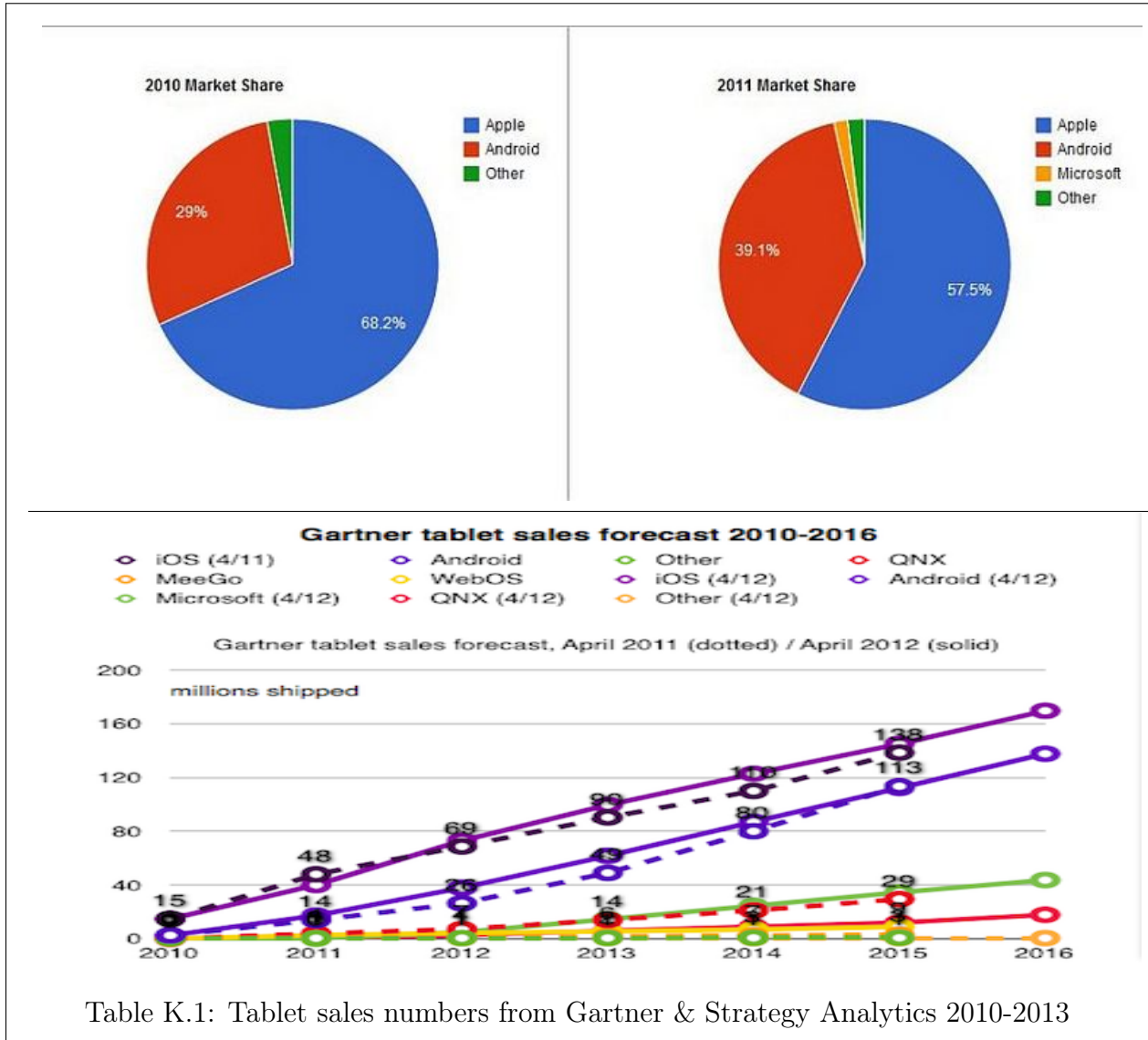
J.3.1 Keys, candidate, primary, foreign

Keys Examination id

Appendix K

Research documentation

K.1 Tablet growth



K.2 Source code and project site

The source code can be found on the cd with the project report.

Newest development can be fetched vi Mercurial HG.

```
hg clone http://repos.micrun.net:8081/hg/spiro/spiro_rh_childrespiration
```


The project site is at:

```
http://trac.micrun.net:8081/trac/spiro
```

Where the source code will be available along with tickets and project documentation.

List of Figures

2.1	Initial time plan	9
2.2	Revised time plan	11
3.1	Scrum sprint	16
3.2	Game Visualization technology	21
3.3	Graphics and Game libraries	21
3.4	Chart API & Libraries	22
3.5	Visual elements of the RH measurement system	23
3.6	Connectors	25
3.7	Line drawing of the Spirometry measurement system, current system	25
3.8	The project development setup: Tablet - SpiroBank Spirometer with USB connection(Blue device), and OTG cable connection(Yellow device)	26
3.9	Use case diagram: Patient Examination System	29
3.10	UC PE1: Initialise examination	30
3.11	fig:Lung resistance test, closed box	31
3.12	Compare data and patient progress	32
3.13	Choose existing patient for examination	33
3.14	Game Generic use cases	34
3.15	Game is initialized,ready to start	35
3.16	UC:Child Player breathing rhythm to slow	36
3.17	UC:Child Player breathing rhythm perfect	37
4.1	title	46
4.2	S2 architecture component block diagram	46
4.3	System overview S1 S2 S3	48
4.4	S1:Patient Examination System Domain	49
4.5	S2:SpiroGame Domain	50
5.1	Android application	54
5.2	fig:andr:Intent type examples	55

5.3	Opening a pdf file, the app shows all apps that can work with pdf files, more precisely, the window shows all apps that have registered a VIEW action for pdf types in the manifest file	55
5.4	The life cycle of an activity	59
5.5	Android build apk package	61
5.6	The structure of the project tree, directories and files	63
5.7	Model-View-Presenter/Controller	67
6.1	Systems packaging	71
6.2	Mockup Application screens: a1) Analyse Examination(s), a2) Initialize Examination(Ready Test), a3) Add Patient, 4) Start Examination(Test)	72
6.3	Mockup Application screens: b1) Add patient, b2) Test Examination overview, b3) Graph and data set display sketch, b4) Game Prototype Sketch	72
6.4	The figures are numbered from top left to bottom right 	72
6.5	Application UI navigation diagram: Basic navigation mockup screens: a) StartActivity, b) A1: addpatient activity, c) A2: Examination analysis, d) Game screen(S2) e) Graph & data set screen	74
6.6	ExaminationSession to GameController	76
6.7	Game Top level class: generic and extended specific game concepts	76
6.8	Data flow input to game system, flow with ExaminationSession	79
6.9	Data flow input to game system, flow in Patient system S2 excluded	79
6.10	Breathing pattern	80
6.11	Events and input data cue decision	83
6.12	Architecture overview diagram: MVP	87
6.13	System architecture	87
6.14	S1:Patient Examination diagram	90
6.15	First sketch of tables	93
6.16	First sketch of tables	94
6.17	S2:Spiro game prototype design	96
6.18	Control flow diagram when game is playing	97
6.19	S2:Spiro game prototype design	99
7.1	The player figure, simple but expressive drawing	105
7.2	A selection of the Sunstars	105
7.3	S2:Spiro game prototype design	106
7.4	S1:graph curve	108
7.5	S1:graph curve	108
7.6	The overlap of the game objects for collision	119
C.1	Use case diagram: Patient Examination System	138
C.2	UC PE1: Initialise examination	139
C.3	fig:Lung resistance test, closed box	140
C.4	Compare data and patient progress	141
C.5	Observe spirometry tests	142
C.6	Follow test results	143

C.7	Add patient data	144
C.8	Choose existing patient for examination	145
C.9	Game is initialized,ready to start	147
C.10	UC:Child Player breathing rhythm to slow	148
C.11	UC:Child Player breathing rhythm to slow	149
C.12	UC:Child Player breathing rhythm irregular	150
C.13	UC:Child Player breathing rhythm perfect	151
G.1	Make box examination sequence diagram	165
G.2	Analyse examination sequence diagram	166
G.3	Init game sequence diagrams	166
G.4	Input game loop sequence diagrams	167
G.5	In game sequence diagrams	167
H.1	Colouring legend	169
H.2	Colouring legend	169
H.3	Coad colour system	170
H.4	Coad colour system	171

List of Tables

1.1	Project terminology	5
2.1	Initial requirement list	13
3.1	Tablets features and capabilities comparison table	18
3.2	Data Storage & persistence	19
3.3	Informal examination protocol	27
6.1	Cue table	84
6.2	FURPS+ Architecture requirements	86
A.1	Terminology for spirometry measurements	128
A.2	Android terminology	129
A.3	Project terminology	130

C.2	Priorities	154
K.1	Tablet sales numbers from Gartner & Strategy Analytics 2010-2013	182

Index

- Abstract
 - Danish, i
 - English, i
- Actors
 - S2:Spiro game, 34
- Analysis
 - Iteration and sprint focus, 16
 - Methods, 15
 - Tools, 17
- app:Use cases
 - Collection SpiroGame, 147
- Appendix
 - Entity relationship, 180
- Architecture block diagram
 - Components block diagram, 45
- Class diagrams
 - Domain S1:PatientExamination, 50
 - Domain S2:SpiroGame, 51
- Design, 71
 - Basic navigation flow, 73
 - Design architecture
 - Architectural requirements, 86
 - Design architecture, 86
 - Game and examination separation, 76
 - Game design, 95
 - Game events and state, 98
 - Prototype 1 PaperBoat, 99
 - Prototype 1 SpyroFox, 99
 - Screen division, 98
 - Game flow description, 97
 - Generic and specific game concepts, 76
 - Input conversion, 78
 - Input events, Game events and System Observer, 83
 - Patient configuration, 80
 - Screen navigation, 73
 - System overview, 71
 - System Partitioning, 75
 - User interface, Mockups, 71
- Evaluation and Conclusion, 123
 - Conclusion, 125
 - Extensions, 124
 - Planning, 123
 - Problems and workarounds, 124
- Game
 - Visualization, 20
- Implementation, 103
 - Active game objects, 110
 - Collision, 106
 - First prototype, 103
 - Game Init, 115
 - Game set patient constraints, 112
 - GameFieldMap, 111
 - GameView, 114
 - Impl. issues, 110
 - Performance and architecture, 117
 - Prototype graphics and animation, 104
 - Scrolling, 110
 - Status of implementation, 117
 - Timing and events, 117

- Updating game state, 113
- Project Introduction, ix
 - Goals, x
 - Issues, x
 - Motivation, ix
 - Proposed solution, x
- Project planning steps
 - Scrum like, 17
- Project scope, 41
 - Delimiting, 42
 - Parallel tasks, 42
 - Scope, 42
 - System Parts S1-S2-S3, 43
- Requirements, 13
 - Additional requirements, 14
- Requirements:Revised requirements, 14
- System overview
 - S1 S2 S3, 48
- Terminology, 4
 - Android, 4
 - Project Terminology, 5
 - Spirometry, 4
- Testing, 119
 - Collision test, 119
 - Graphics artifacts error, 121
- Time plan, 7
 - Discussion, 12
 - Initial plan, 8
 - Revised, 12
 - Revised plan, 10
- time plan, 10
- Use cases
 - S1:Patient Examination, 28
 - S2:Spiro game, 34, 35

Bibliography

- Android Developer Official Intent and actions@ONLINE, January 0000. URL http://developer.android.com/intent_actions/.
- AChartEngine. Achartengine website, 2013. URL <http://www.achartengine.com>.
- Box2D. Box2d website, 2013. URL <http://www.box2d.org>.
- Peter Coad. Modellingincolour@ONLINE. URL <http://www.step-10.com/SoftwareDesign/ModellingInColour/>.
- IBM. Json vs xml sizes, 2010. URL <http://www.ibm.com/developerworks/xml/library/x-dataAndroid/>.
- Craig Larman. *Applying UML and patterns*. Prentice Hall PTR Upper Saddle River, NJ, USA ©1999, 2004. ISBN ISBN:0131489062.
- LibGDX. Libgdx website, 2013. URL <http://www.libgdx.org>.
- Android Developer Official. Assetmanager api@ONLINE, 2013. URL <http://developer.android.com/reference/android/content/res/AssetManager.html>.
- OpenAL. Openal website, 2013. URL <http://www.openal.org>.
- OpenGL. Opengl development website, 2013. URL <http://www.opengl.org>.
- Eric Lefebvre Peter Coad, Jeff de Luca. *Java Modeling In Color With UML: Enterprise Components and Process*. Prentice Hall PTR Upper Saddle River, NJ, USA ©1999, 1999. ISBN ISBN:013011510X.