# DV1435 - Fördjupning i objektorienterade tekniker

*L2: Detaljerad Design - 2012-04-27*
*Team Oppenheimer*

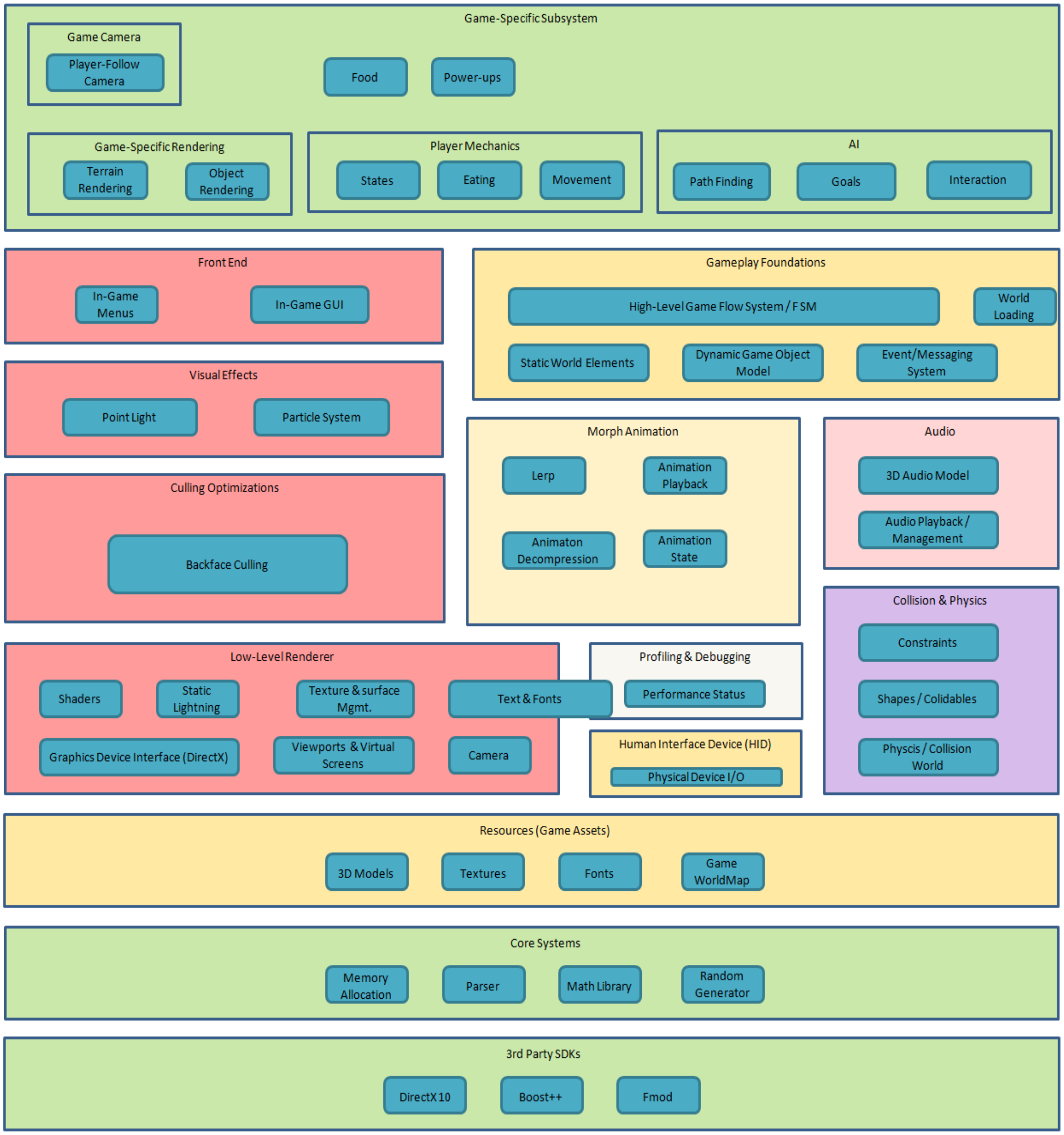
| | |
|---|---|
| Ossian Petri | ossian.p@gmail.com |
| Fredrik Cronqvist | cronqvist_fredrik@hotmail.com |
| Sebastian Larsson | sjereq@gmail.com |
| Robin Karlsson | rokc09@student.bth.se |
| Christian Lindås | chlf09@student.bth.se |

## Contents

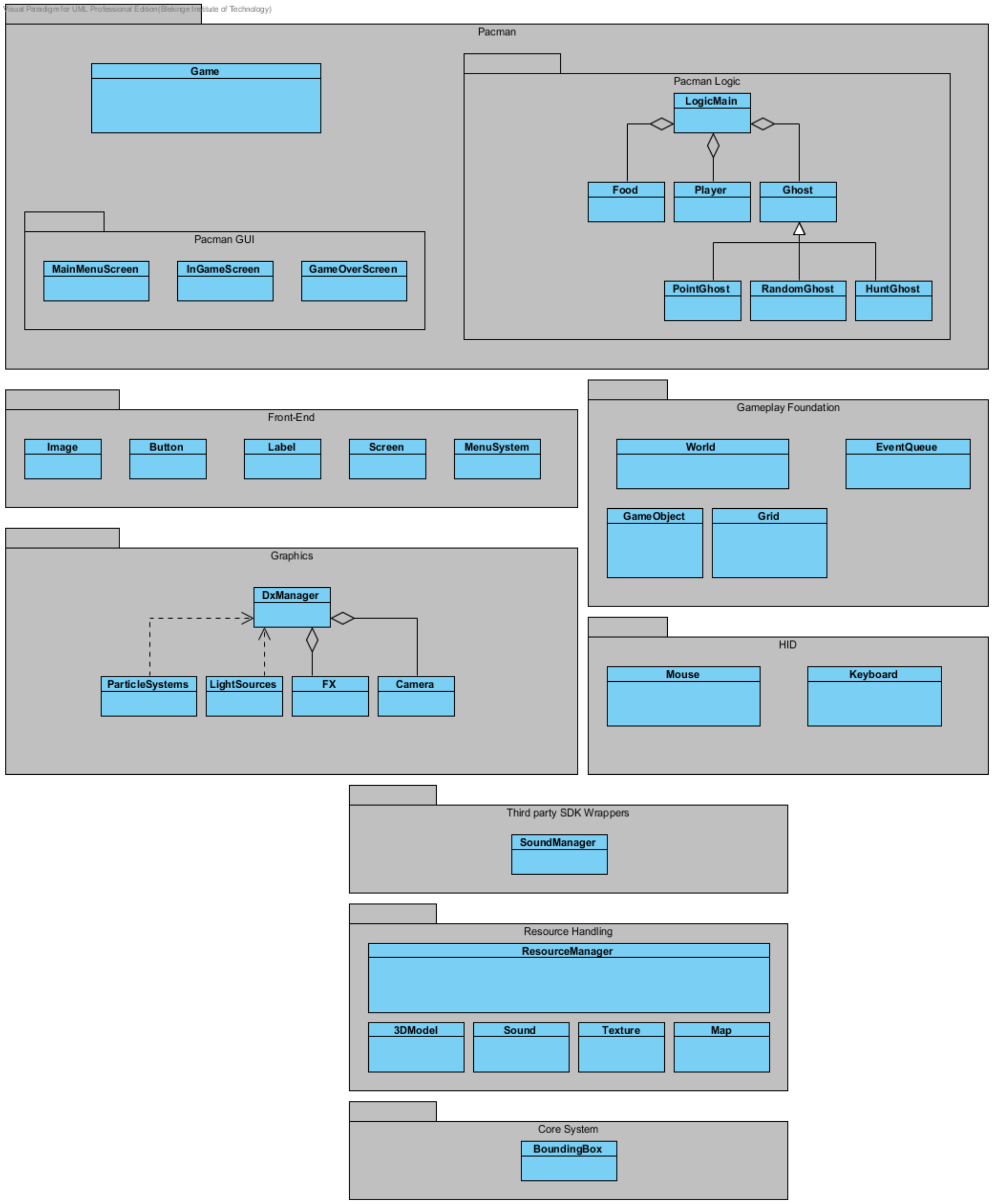
## Summary of the High-level design

Most of the decisions made under L1 are kept as they are. The architecture is still largely based on the example from the book “Run-time Game Engine Architecture”, where game specific details are dependent on more independent components. Major parts are the the low-level renderer, the front-end and the game-specific subsystem.

We will develop it on and to PC with the help from some programs, Visual Studio 2010 to make the game in, Git to help us have the same version of the project all the time, JIRA to help us see what’s left to do and what’s done.


Game-Specific Subsystem
Game Camera
Player-Follow Camera
Food
Power-ups
Game-Specific Rendering
Terrain Rendering
Object Rendering
Player Mechanics
States
Eating
Movement
AI
Path Finding
Goals
Interaction
Front End
In-Game Menus
In-Game GUI
Gameplay Foundations
High-Level Game Flow System / FSM
World Loading
Static World Elements
Dynamic Game Object Model
Event/Messaging System
Visual Effects
Point Light
Particle System
Morph Animation
Lerp
Animation Playback
Animaton Decompression
Animation State
Audio
3D Audio Model
Audio Playback / Management
Culling Optimizations
Backface Culling
Collision & Physics
Constraints
Shapes / Colidables
Physcis / Collision World
Low-Level Renderer
Shaders
Static Lightning
Texture & surface Mgmt.
Text & Fonts
Graphics Device Interface (DirectX)
Viewports & Virtual Screens
Camera
Profiling & Debugging
Performance Status
Human Interface Device (HID)
Physical Device I/O
Resources (Game Assets)
3D Models
Textures
Fonts
Game WorldMap
Core Systems
Memory Allocation
Parser
Math Library
Random Generator
3rd Party SDKs
DirectX 10
Boost++
Fmod

## Class overview, UML



## Class Diagram Breakdown

*Pacman*

- **Pacman Logic:** Package name
  - LogicMain: This class acts as a connection class that ties Logic together with the InGameScreen.
    - Ghost: Abstract class representing the different  AI:s in the game. If the player has eaten special food the AI will run away. All AI subclasses will  inherit from this class. Might be referenced as an actor further down this document.
      - PointGhost: This AI will navigate to a specific point on the map and when he reaches his destination it will get a new point to navigate to. The points will be placed out randomly.
      - RandomGhost: This AI will move around completely random and will not care about where the player is. Every time this AI reaches a crossing it will get a random direction.
      - HuntGhost: This AI will follow the player if the player hasn’t eaten special food.
    - Food: Normal food gives score and Pacman needs to collect all of them to finish a level. Some of the food are special and will give Pacman the ability to eat ghosts, this will make the AI switch states from their normal to a running away state instead.
    - Player: Player specific behaviour is stored in this class. It also holds score and other player specific data. The player might be referenced as an actor further down this document.

- **Pacman GUI:** Package name
  - MainMenuScreen: This represents the main menu. You can start and exit the game.
  - InGameScreen: Here is where the game takes place and HUD is displayed with the vital information.
  - GameOverScreen: When Pacman dies or the game exits. The player score is displayed.

*Front-End*

- Image: Holds a texture and a position to draw the texture to.
- Button: A class with button textures and a changeable text.
- Label: Holds a text and a position to draw the text to.
- Screen: This is a base class for all the different screens.
- MenuSystem: A fancy class that manage the menu systems.

*Gameplay Foundation*

- World: A class that will build up the world, with help from the grid system and the loaded map.
- EventQueue: A simple event system for events like mouse- and keyboard presses.
- Game Object: Is a base class for game objects, like the player, the ghosts and the food.
- Grid: Creates and manage the grid which all actors use to navigate. Grid contains an 2D array of GridNodes. When an actor navigate the grid it uses the index from the 2D array as a multiplier to get the position.

*Graphics*

- DxManager: Handles the initiations of the DirectX API. This will be the main graphical class, which means it handles all draw calls. Most graphical classes depends on this class to work.
    - FX: This represents the .fx file, which contains all the shaders.
    - Camera: Creation and behaviour of the camera will be stored in this class.
- LightSources: Will contain data about the Light source in the game. We will use a point light.
- ParticleSystems: Describes the creation and control of the particle systems.

*HID*

- Mouse: This class handles the mouse input using the Win32 library.
- Keyboard: This class handles the keyboard input using the Win32 library.

*Third party SDK Wrappers*

- SoundManager: All the game sound is managed from here by using Fmod sound engine.

*Resource Handling*

- ResourceManager: Connection point of all resources.
- 3DModel: Responsible for parsing the model files and creating ID3D10Meshes that the DxManager can use.
- Sound: Handles sound resources.
- Texture: Reading the texture files and saving them as ID3D10Texture2Ds.
- Map: Reads an image file and creates the map mesh, starting positions of the ghosts and Pacman, depending on the colors of the image.

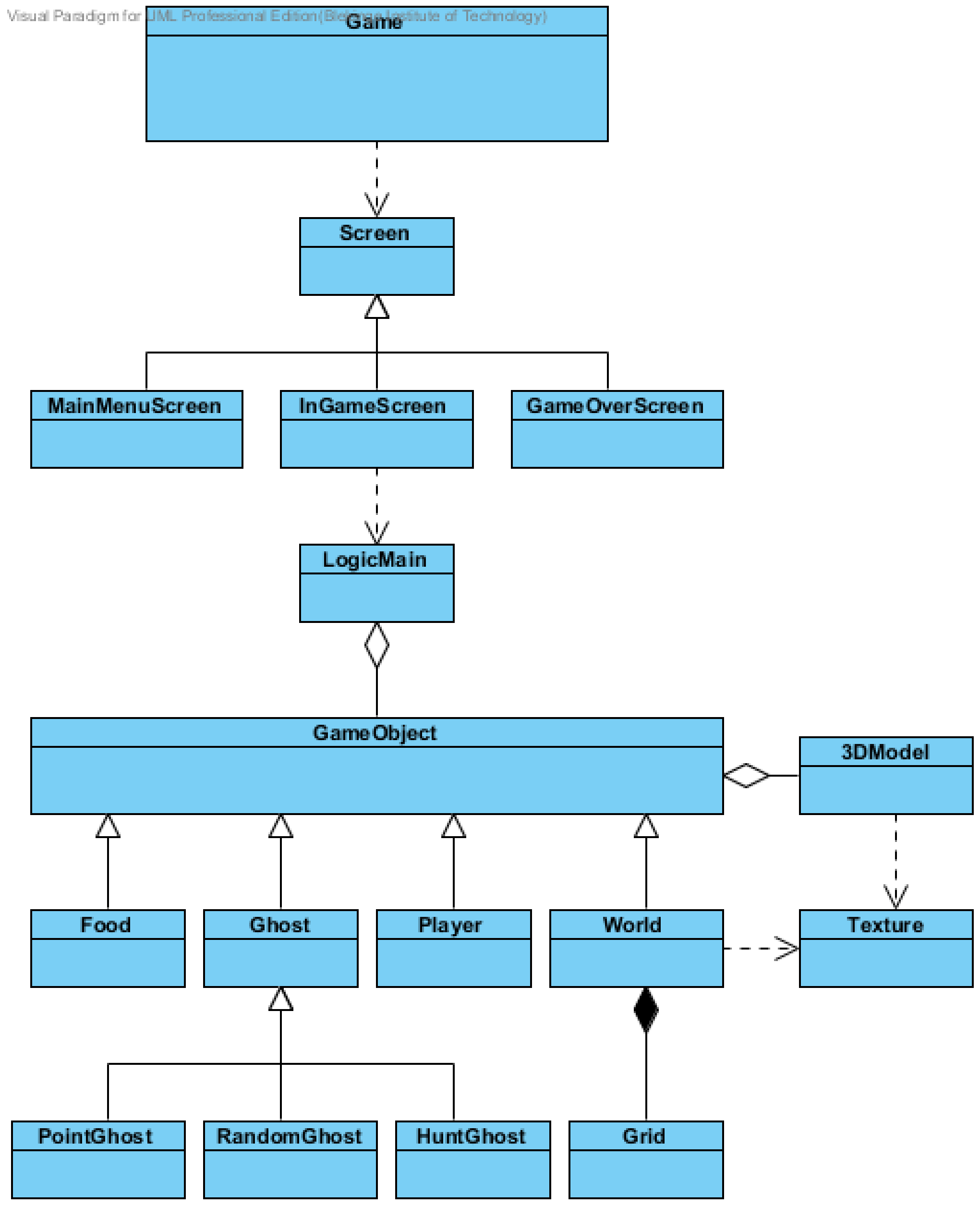
*Core System*

- BoundingBox: Each actor has their own bounding box represented by this class. It contains all the necessary data and functionality required for collision detection between actors.

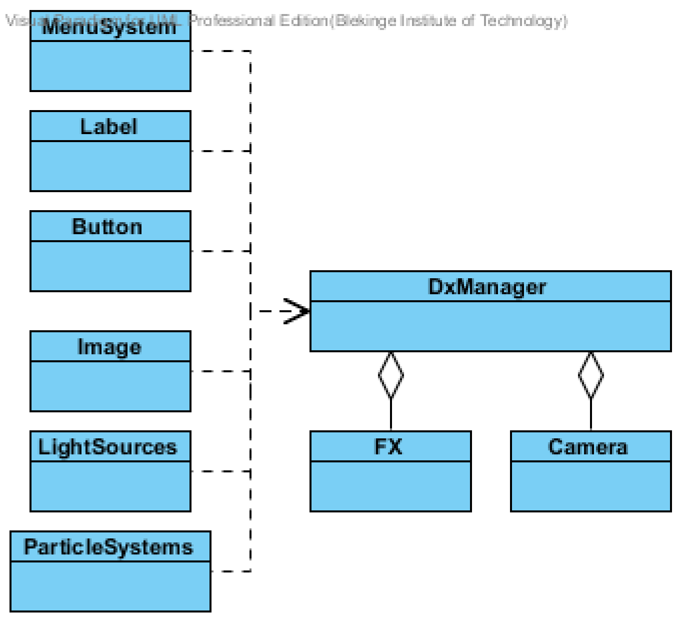
## How Classes are connected to each other

We made two parts of UML diagrams to easier follow how our classes talk and communicates with each other. The first diagram are the highest point in the program, the main and where graphics- and main-parts connect to make the full program. The main part uses the other parts and the only part we found out were sufficiently large to have a separate UML, was the graphics part.

*Main part:*



*Graphics part:*


Visual Paradigm for UML Professional Edition(Blekinge Institute of Technology)
MenuSystem
Label
Button
Image
LightSources
ParticleSystems
DxManager
FX
Camera

## Develop new, change old or completely reuse?

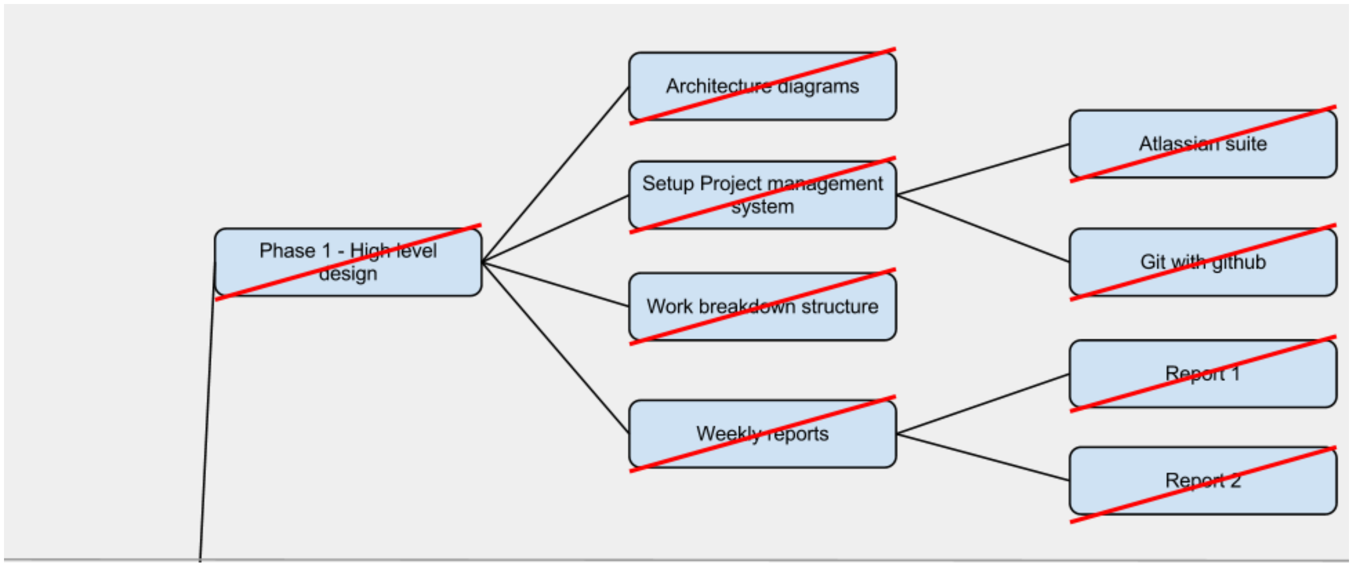
Most of the system for this project is newly developed mostly because we don’t have the right code or what code we have is too integrated into its own projects and therefore not usable in other projects. But some parts are close to what we need and some classes will be used to see what we could need when we make the new one.

- Button class : Change an existing class. The old class was for static buttons and we are going to use dynamic buttons with a label on it and not flipping between different textures.
- 3DSound: Reuse, mostly because the class we will use is developed recently and its use will be similar in this project.
- Grid: Reuse ideas. We are going to reuse the idea of a grid system and change the old class from another project so it will fit in our new.
- Mouse and Keyboard: Develop new, we are going to develop a new class, mostly because we want a mouse/keyboard class that we can reuse without changing it too much to fit in nearly any project we make after this.
- New file format .mta (ModelTexureAnimation): Develop new, we are going to make a new file format because we want to have morph animations in the same file instead of having it in several files.
- OBJ Parser: Develop new. A small program that will parse OBJ files that hold one animation pose each of the 3D model to only one file (our own file format (.mta)), with a GUI.

## Work Breakdown Structure

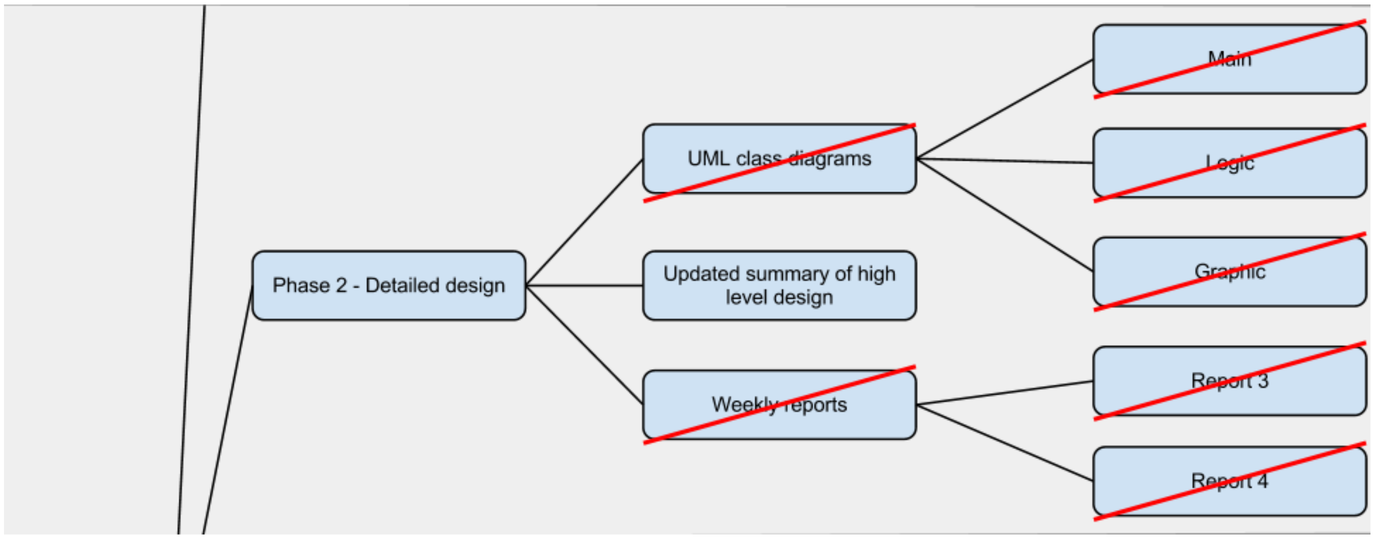
The project is broken down into four different phases, one for every greater milestone and since L1 we have made some progress on it. We have marked in the new WBS what we have done in the highest overview it is. And by looking at only the WBS we are more or less in time with everything and it is true in the case of turning in reports and other documents.

### Phase 1:


~~Phase 1 - High level design~~
~~Architecture diagrams~~
~~Setup Project management system~~
~~Work breakdown structure~~
~~Weekly reports~~
~~Atlassian suite~~
~~Git with github~~
~~Report 1~~
~~Report 2~~

Phase 1 are all done in the WBS and all the tasks we have put up in JIRA. All that means we have closed that phase and could move on with the next phase of the project.

### Phase 2:



As for phase 2 we are nearly done with all the greater tasks, all that's left to do is this document and some smaller tasks in JIRA but mostly all in phase 2 is done and we are slowly moving over to the next phase in the project, phase 3 the programming phase.

We will cross off the last task(Updated summary of …) when it's been evaluated.

### Phase 3 and 4:

As for both phase 3 and 4, we haven't done too much to show it in here (a.k.a. the same pictures as in L1 for phase 3 and 4) and this is because we haven't had too much time in them to make any greater progresses to show it in here as for phase 1 and 2. But we are in phase 3 and working as this document is written.