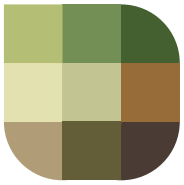

Breensoft

Nine shades of breen

Eye-Controlled Game for Young Individuals

Architecture Design Document

Ahlgren, Joel
Ahumada Juntunen, Marco
Erneholm, Carl-Oscar
Eklund, Viktor
Forsberg, John
Lindh, Fredrik
Mattson, Lukas
Mikkola, Mattias
Nycander, Martin

Version 1.4-FINAL
May 5, 2010

Abstract

This document is the architectural design document group for the project group Breensoft. It intends to provide the reader with a full understanding of the final implementation of the "Re:Cellection" strategy game being developed for Tobii Technology. It's main purpose is to give the developers a specification to work with when the development phase begins.

Contents

Document Change Record

Version	**Date**	**Affected Sections**	**Changes**
0.1	03/03/2010	All	Document hastily Created.
0.2	03/10/2010	§2,§3	Added first drafts.
0.3	03/14/2010	§4	Added first component drafts.
0.4	03/20/2010	All	Revisions and more component drafts.
0.5	03/22/2010	§6	Added draft.
0.6	03/23/2010	All	Revisions.
1.0-FINAL	03/23/2010	All	Final version of document.
1.1	03/30/2010	§6	Revised tables.
1.2	04/21/2010	§4.1, §5	Revised all components. Added Observer/Publisher pattern to design.
1.4-FINAL	04/25/2010	§4.1, §5	Further revisions of components

This page intentionally left blank.

1 Introduction

1.1 Purpose

This document intends to give the reader a thorough and detailed specification of the implementation of the software. This specification will be be followed when the developers work on the project. It is part of a degree project at the Royal Institute of Technology. The intended readers are Tobii Technology, the course leaders, and the project members. It is advised that you make sure you have read the User Requirements Document and Software Requirements Document before reading this document, as it contains lots of information derived from those documents.

1.2 Scope of the Software

The software will form a prototype of a game designed for Tobii's eye tracking hardware. It will utilize the full potential of the Eye tracking hardware and is developed towards the hardware specifications of Tobii's C12 computer.

1.3 Definitions, Acronyms and Abbreviations

CEye 9
Tobii Technology's eye tracking module.

SDK 9
Abbreviation for Software Development Kit, a collection of tools and information to assist a developer in using a certain method or technology.

WPF 9
Windows Presentation Format, a graphical subsystem supplied by Mircosoft for rendering user interfaces in Windows-based applications.

XNA 10
A game programming framework using C# developed by Microsoft.

OSI 11
Open System Interconnection Reference Model is an abstract design for network protocols.

MVC 11
The Model-View-Controller is an architectural paradigm.

1.4 References

[1] Breensoft. Eye-controlled game for young individuals: Software requirements document. 2010.

[2] S. Burbeck. Applications Programming in Smalltalk-80: How to use Model-View-Controller (MVC), 1992.

[3] J. Perry, J. Perry, et al. *RPG Programming with Xna Game Studio 3.0*. Wordware, 2009.

[4] Wikipedia. Layer (object-oriented design) — Wikipedia, the free encyclopedia, 2010. URL `http://en.wikipedia.org/wiki/Layer_(object-oriented_design)`. [Online; accessed 15-March-2010].

[5] Wikipedia. Model–view–controller — Wikipedia, the free encyclopedia, 2010.

[6] Wikipedia. Multitier architecture — Wikipedia, the free encyclopedia, 2010. URL `http://en.wikipedia.org/wiki/Multitier_architecture`. [Online; accessed 15-March-2010].

[7] Wikipedia. Naked objects — Wikipedia, the free encyclopedia, 2010. URL `http://en.wikipedia.org/wiki/Naked_objects`. [Online; accessed 15-March-2010].

[8] Wikipedia. Visitor pattern — Wikipedia, the free encyclopedia, 2010. URL `http://en.wikipedia.org/wiki/Visitor_pattern`. [Online; accessed 20-March-2010].

1.5 Overview of the Document

This document contains the design specifications for the "Re:Cellection" game being developed for Tobii Technology inc. The first section contains some useful information to get the reader acquainted with the project and the document.

The second section, System Overview, will briefly introduce the reader to the project. The general outline of the Architecture Design will be given here, in a somewhat abstracted level.

Section three, the System Context, will put the system into context relating to other systems.

Section four will go into more detail concerning the different components and how they relate to the chosen design design method.

In section five all components of the software will be listed, along with detailed descriptions of their implementation, interfaces etc.

Section six will go over the management concerns regarding the project. Here you will find scheduling for the coding phase and risk estimations.

Finally, section seven will cover the relationship between the software requirements and the components.

This page intentionally left blank.

2 System Overview

This project, in addition to being our bachelors project, presents an incredible opportunity for every member of our group. Not counting possibilities for future collaboration with Tobii, it also gives us the fortuity to familiarize ourselves with some new and exciting SDKs and working techniques. These include the Tobii C12 eye tracker and its associated calibration suite "CEye", as well as the C12 library itself. We will also get the chance to create something original and exciting as a group, using the Microsoft XNA game studio which has become increasingly popular the last few years. As stated in earlier reports, the reason for choosing XNA is primarily that of the inherent alacrity that comes with using XNA over other possible frameworks. Since the prototype is due in late May, we simply have to take every measure possible to speed up the development process.

We have also chosen to comply with the MVC design method since it suits our needs to be able to split work, and consequently reintegrate it seamlessly fast and easy. As it is not the native design method recommended when using XNA, which means some extra time will have to be committed to designing the implementation of the MVC model into the XNA pipeline. However, the increased modularization will be worth the extra effort in the end.

This page intentionally left blank.

3 System Context

This system is designed to be used with the Tobii CEye hardware as its only source of input. In order to receive and utilize the input from this hardware we have been supplied an SDK from Tobii and then translate this input into actions within the game.

Furthermore, the audiographical output of the game will solely be handled by the Microsoft XNA framework and its various subsystems.

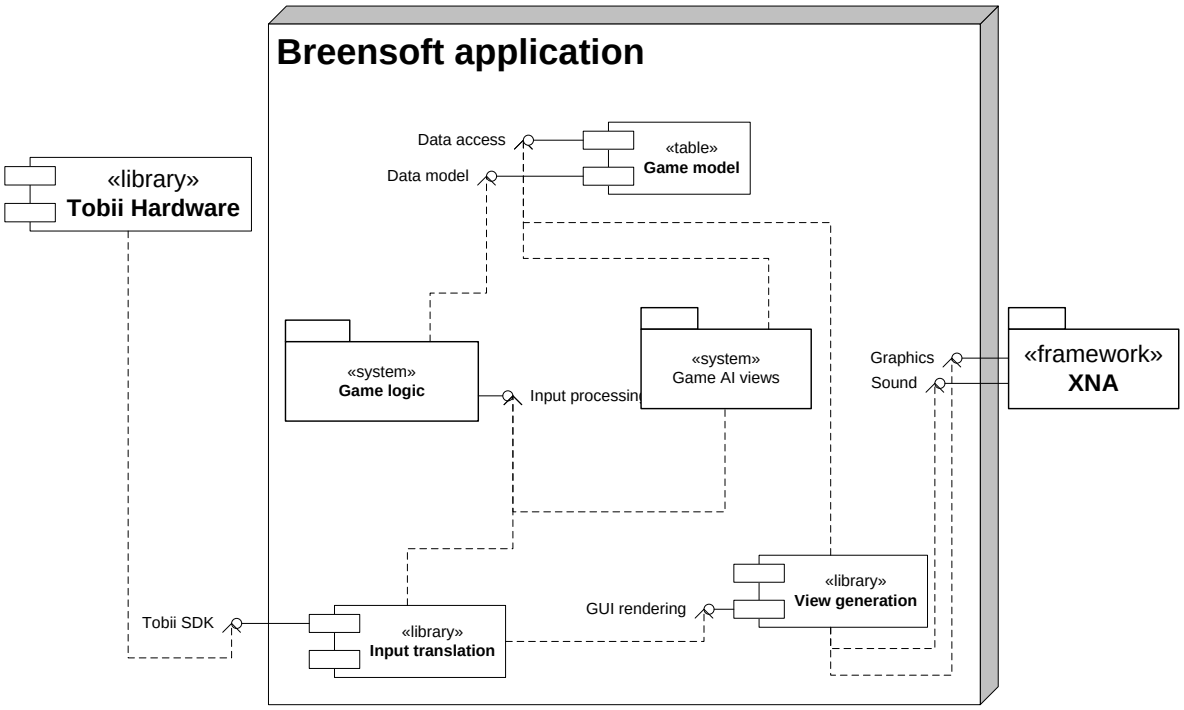

Figure 1: System context diagram

3.1 Tobii Hardware and SDK

The Tobii C12 is a portable computer designed to be used together with the Tobii CEye to allow for a compact mobile solution. It also features a touch sensitive display which however won't be utilized in our implementation.

To enable the interaction of our software with the Tobii CEye we will be utilizing an SDK

provided by Tobii. The SDK adds functionality which will ease the interaction with the eye tracker considerably as well as making our code exchangable and allow for future upgrades to the SDK or new versions of the eye tracker without requiring considerable reworking of our code.

3.2 XNA Framework

Our code will utilize Microsoft's game programming framework to spare us from the time-consuming task of creating a game engine from scratch. To translate our game data to XNA entities we will use an in-house developed "adapter". In addition to the engine, XNA also provides us with a plethora of static tool classes for math, content and media especially made for game design. It will also enable us to quickly port the software to other Microsoft platforms such as Zune and the XBOX360.

4 System Design

4.1 Design method

There are several established architectural design methods. This section will summarize the available options, how applicable they could be to our project; and finally motivate our choice of the Model-View-Controller (MVC) pattern.

Layers [4]

In this approach each component has a set of reusable dependencies which means the components will most likely conform to a hierarchy tree, and thus forming *layers*. A very strict example of this is the OSI-model, wherein the different protocols depend on each other. This approach could suit us, but in a game there are generally many dependencies which in turn leads to low parallelization in the work flow. Being able to parallelize work is important for larger development groups such as ours and we have thus abandoned this approach.

Naked objects [7]

An approach targeted to the sole use of domain objects. The rule is that all business logic is encapsulated in the so-called domain objects, and the user interface is a direct representation of the domain object. We could have chosen this approach if it had not been for our need of a complex user interface which could hardly have been automatically generated from such domain objects.

Model-View-Controller (MVC) [2;5]

The MVC approach is an attempt to separate the model, the behavior and the visual interface from each other. The *model* is the domain-specific data objects upon which the application operates. The *view* renders the model in a form suitable for human interaction. The *controller* receives input and responds through *model* components. This approach has been found to suit our needs well.

Three-tier architecture [6]

Similar to the MVC-approach in the sense that it splits responsibilities. The fundamental difference is that the *view* will only communicate with the *model* through the *controller*. This is a real option for this project, but since it will require more work on the controllers;

it has been decided to be abandoned.

As can be seen in figure 4.1, we have decided to follow the principle of MVC in a rather unmodified fashion. Our GUI will be the views and the controllers will handle the input and game logic. Furthermore the model will hold all the domain objects for the game and hold rules for how these are allowed to be modified.

Whether or not the controller may tell the view to render itself is an optional feature in MVC. We have decided to allow this for cases where the input has to be validated, or is incorrect (such as “Are you sure you want to quit?”).

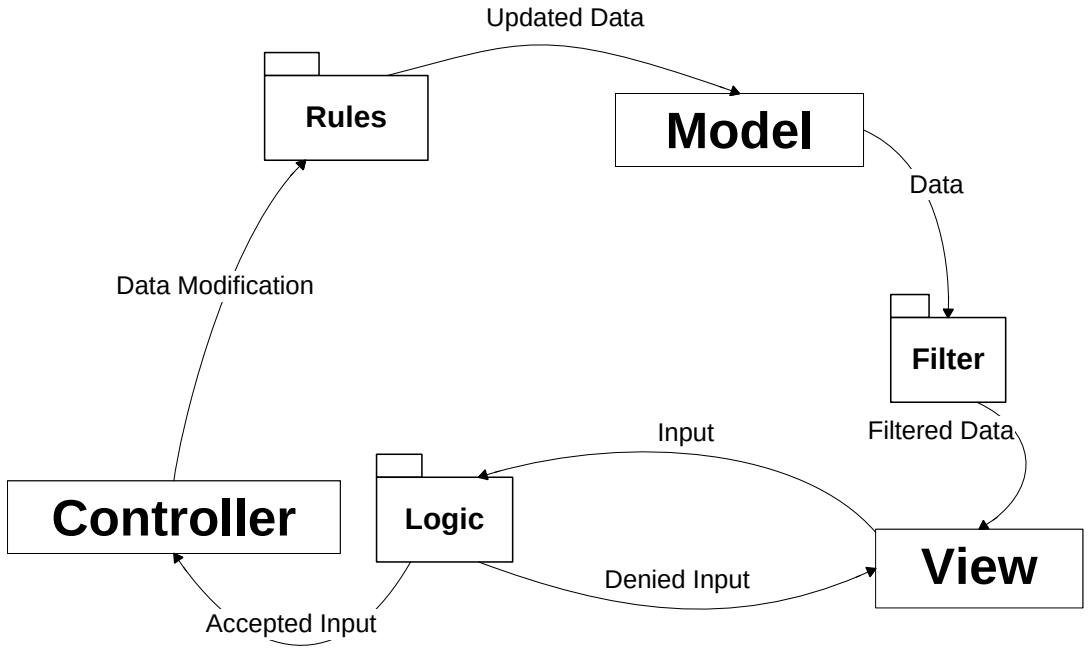

Figure 2: A top level description on our MVC approach.

4.1.1 Observer pattern

In general all models are 'Observable', which means any number of 'Observers' may subscribe to changes in the model. These observers will then be notified of any updates and the nature of the update as it happens eliminating the need of constantly polling the Observables for updates. This also allows for more flexibility and the reduced coupling will make the system easier to extend because of the decreased need for a common interface. This means we follow the Hollywood Principle of "Don't call us, we'll call you"

The Observer pattern will in our application be implemented with the event system of C#. Objects can provide events to which so-called delegates can be attached. Delegates are callback-objects which means that when an event occurs all attached delegates are invoked. This in turn runs the associated methods for all registered observers.

Generally all views are Observers, and are updated with data from any events they are subscribed to in order to stay up to date.

4.2 Decomposition description

4.2.1 Decomposition View

The following table will describe what our components in the software do, they are grouped by which part they belong to in the MVC architecture.

Controller

Name of Component	Description
AI Player	• The Artificial Intelligence used as an opponent in the game.
Building Controller	• Used when building, selling and destroying buildings. • Will provide the functionality of each building type.
Configurator	• Handles state management for the component Game Options.
Game Initializer	• Will first call World Generator, then add players to the world and place the required buildings and units to start a new game.
Graph Controller	• Controls the unit dispersion in graphs.

Name of Component	**Description**
Initializer	• Handles the initiation of the application.
Loader	• Will use the Load View to load a previous saved game.
Localizer	• Will only be instantiated once. • Will handle all the text strings and provide functionality to switch language.
Menu Controller	• Controls all menu logic in the game.
Saver	• Will send the current game state to file to be loaded later.
Tobii Input Translator	• Will only be instantiated once. • Handles and translates input from the Tobii control suite.
Victor Turner	• Keeps track of whose turn it is. • Gives control to the player/actor when it's their turn.

Name of Component	**Description**
Unit Accountant	• When units are used, killed or earned this controller updates the owners number of units.
Unit Controller	• Provides the main functionality for units, like movement and attack.
World Controller	• Will be the controller that alters the world model.
World Generator	• Will randomly generate a map to play on.

Model

Name of Component	Description
Building	• Placed on tiles. • Has a type which defines their function.
Aggressive Building	• May have a unit that is targeted for an attack.
Base Building	• Holds a reference to each child building.
Barrier Building	• Has a bonus to the amount of damage Units can sustain before termination.
Resource Building	• Holds the amount of resources produced every turn.
Game Option	• Keeps track of configurations/options in the application.
Graph	• Contains a dictionary that maps a building to a number that specifics the number of units that should be at that building.
GUI Region	• Has a `GenericInteractionRegion`.

Name of Component	**Description**
Language	• Holds all required text strings for the game menus. • Loads the text from a formated file. • Knows which language currently in use.
Menu Model	• Has data defining the appearance and function of the menus in the game.
Player	• A player has a colour and that colour will affect the players buildings and units. • Will keep track of the level of upgrade of the players units. • Also holds a list of the players building graphs.
Sounds	• Will map an identifier to a specific sound.
Terrain Type	• A terrain type is defined by how much it affects a units movement. • Has a name. • Has a texture. • Has an amount of resources.

Name of Component	**Description**
Texture	• Will contain references to sprites.
Tile	• Is a square of pixels defining a small area on the map. • Has a terrain type. • Might have a building on it.
Unit	• A unit can move around in the world. • Can attack enemy buildings. • Can attack enemy units.
World	• Consists of tiles. • Has a number of playing players.
Menu Manager	• Contains the current state of the GUI.

View

Name of Component	Description
AI View	• This view will provide information for the AI Player, it will read from the models containing information about the game.
Help View	• Renders the help slides.
Menu View	• Renders generic menus.
Music Player	• Will play the background music.
World View	• Renders the GUI inside the game. • Makes sure Graphic Rendering renders the required objects.

Other

Name of Component	Description
Graphic Rendering	• Keeps tracks of what objects to render.
Main	• The XNA-thread.

4.2.2 Dependency View

The previous showed figures will describe how our components in the software depend on each other. The first figure will describe briefly on how our MVC architecture will be implemented. Seen in figure 4.1, each of the MVC components, the model, the view and the controller.

As described in section 4.1 each component has a specific assignment: The model stores all the data, the view presents all the data and the controller changes the data when it has received input from the view. The figure also presents three additional components, the filter, the logic and the policy. These will provide some control for each of the MVC components. The filter will format the data from the model to the view so it can be viewed. The logic component will control if the specific view could use a specific controller at a given time, if not it will fall back to the view and the view will present why it failed. Between the model and the controller are rules, these rules has to be followed if a controller wants to change the model.

The following figures will describe a more in depth view of the dependencies, if a component has no arrows going out from it to another component it is a component that depends on no other component. First are the model components in figure 4.2.2, they have the least dependencies as they are models and should only contain data. Next comes the more complex figure 4.2.2, it shows all the controllers and some model and view components. The reason that most of the controllers depend on other components is simply because a controller needs a model to have something to alter. The last figure (figure 4.2.2) contains all of our views and because not all views have an interface to a model component not each one of them depends on them. For example the Music Player only requires some files on the hard drive.

The components that are not presented in the figure is the Main component, the Graphic renderer and the Debugger, these do not belong to any of the MVC parts and are therefore not in the figures. The Main component will not depend on any other component and the Debugger will need other components to be implemented as it needs code to debug but what component that is required is not specified.

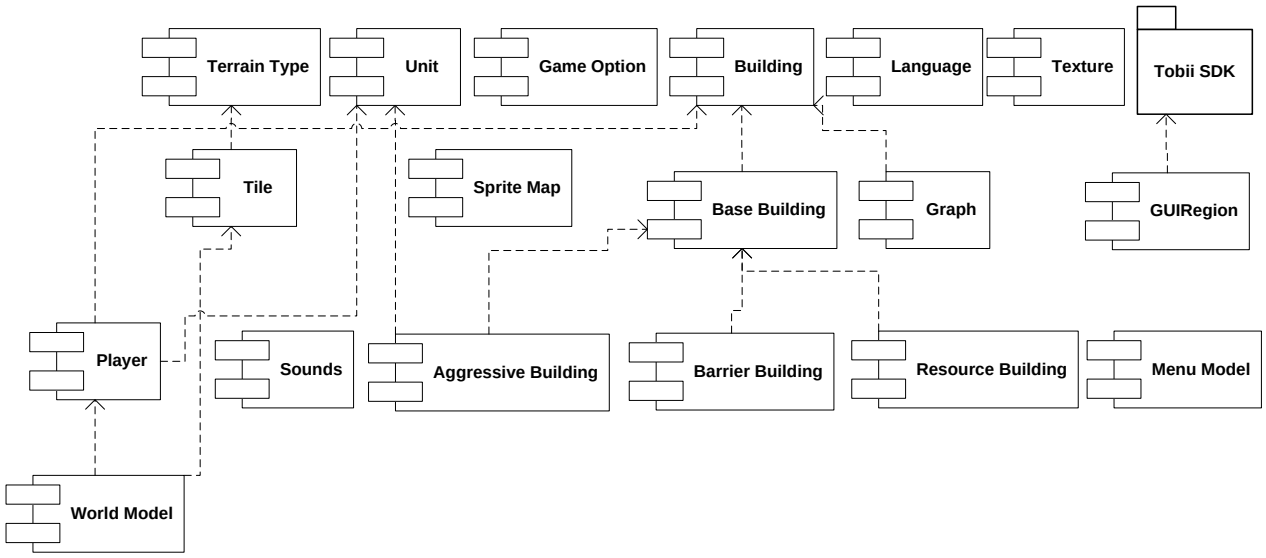

Figure 3: This figure shows every model component in the software and how they depend on other model components.

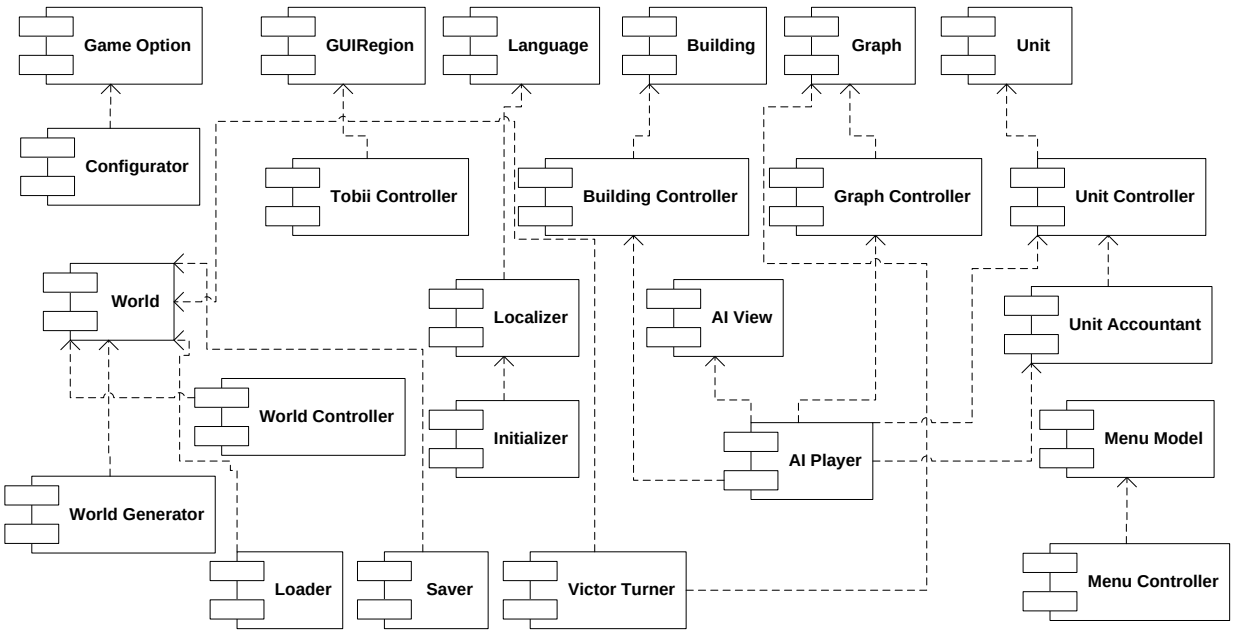

Figure 4: This figure shows every controller component in the software and how they depend on other model, view and control components.

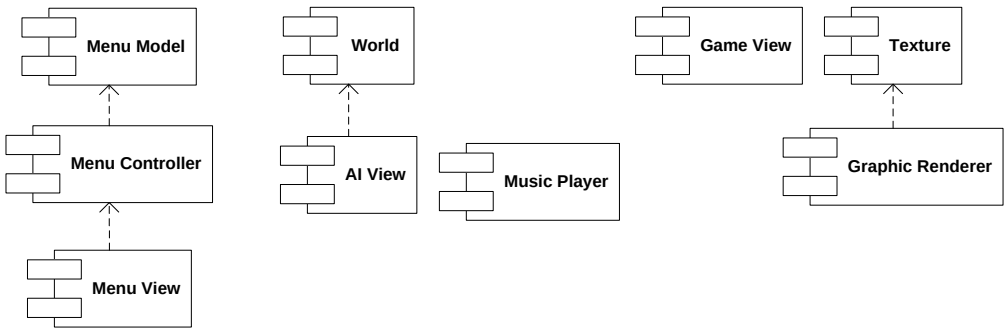

Figure 5: This figure shows every view component in the software and how they depend on other model, view and controller components.

This page intentionally left blank.

5 Component Description

5.1 Building

5.1.1 Type

The building component is an **abstract class** belonging to the **model** part of MVC.

5.1.2 Purpose

As described in the software requirement SR1.3 the buildings will have different kinds of properties. The purpose of this component is to implement a common interface for those properties.

5.1.3 Function

Stores the position of a Building as well as its visual properties such as name and sprites.

This component and subordinates will implement the *Visitor design pattern*[8]. This means that when a Building has been created, it will copy a reference to a Base building from it's creator. It will then "visit" the Base building; the Base building can then register the Building as a part of the graph component.

Since this component is a model it will implement the publisher pattern. It will provide one or several *event*s for observers to register themselves to. This component will then publish any updates to the observers when one of its properties has changed by calling its *event*.

```
// Part of visitor pattern
public void Accept(BaseBuilding visitor);
public Player GetPlayer();
public Unit[] GetUnits();
public void AddUnits(Unit[] units);

// Properties
public string GetName();
```

```
public Globals.BuildingType GetType();
public Globals.Texture GetSprite();
public int GetX();
public int GetY();
public BaseBuilding GetBase();
public int GetHealth();
public int GetHealthMax();
public int GetHealthPercentage();

// Modifiers
public void damage(int dmghealth);
public void repair(int health);
```

5.1.4 Subordinates

- 5.4. Base Building
- 5.5. Resource Building
- 5.2. Aggressive Building
- 5.3. Barrier Building

5.1.5 Dependencies

Requires Base Building to exists for the Visitor pattern to work.

5.1.6 Interfaces

Caller/Callee	Header
5.39. World View	`public string GetName();`
5.39. World View	`public Globals.BuildingType GetType();`
5.39. World View	`public Texture2D GetSprite();`

continued on next page

continued from previous page

Caller/Callee	Header
5.39. World View	`public int GetX();`
5.39. World View	`public int GetY();`
5.39. World View	`public BaseBuilding GetBase();`
5.39. World View	`public int GetHealth();`
5.39. World View	`public int GetHealthMax();`
5.39. World View	`public int GetHealthPercentage();`
5.34. World Controller	`public Player GetOwner();`
5.20. Building Controller	`public void Damage(int dmghealth);`
5.20. Building Controller	`public void Repair(int health);`
Any View	`public event Publish<Building> healthChanged;`
Any View	`public event Publish<Building> unitsChanged;`

5.1.7 Resources

This components does not require any resources.

5.1.8 References

The Visitor pattern has to be understood to understand the relationship with the Base building, please refer to wikipedia[8] for this.

5.1.9 Processing

This component act as a collection of mutual functions and forms a standard of what properties a building has. It will not process anything since it is a model.

To follow the *Visitor pattern* the accept-method specified in the Interface-section should look like this.

```
public void Accept(BaseBuilding visitor)
```

```
{
    visitor.Visit(this);
}
```

5.1.10 Data

Data	Structure	Description
name	`string`	The name of the building
type	`Globals.BuildingType`	The type of building.
baseBuilding	`BaseBuilding`	A reference to a base building.
x	`int`	Which tile-column in the world the building exists.
y	`int`	Which tile-row in the world the building exists.
health	`int`	How much health this building has.
maxHealth	`int`	The health-capacity of this building.
units	`Unit[]`	The units belonging to this building.
controlZone	`Tile[]`	The nine tiles surrounding the building and the tile the building st

5.2 Aggressive Building

5.2.1 Type

This component is a **subclass** of Building belonging to the **model** part of MVC.

5.2.2 Purpose

The purpose of this module is to implement the data storage of the Aggressive buildings described in SR1.3, along with policies on how to retrieve and update the data.

5.2.3 Function

Inherits the function of Building.

5.2.4 Subordinates

This component has no subordinates.

5.2.5 Dependencies

Requires the superclass Building.

5.2.6 Interfaces

Caller/Callee	Header
5.20. Building Controller	`public void SetTarget(Unit unit);`
5.20. Building Controller , 5.39. World View	`public Unit GetTarget();`

5.2.7 Resources

Sprite Map has to exist for the buildings visual representation.

5.2.8 References

This component requires no references.

5.2.9 Processing

As a model, the Aggressive building does not process anything.

5.2.10 Data

Data	Structure	Description
currentTarget	`Unit`	The current unit that is targeted for attack.

5.3 Barrier Building

5.3.1 Type

This component is a **subclass** of Building belonging to the **model** part of MVC.

5.3.2 Purpose

The purpose of this module is to implement the data storage of the Barrier buildings described in SR1.3, along with policies on how to retrieve and update the data.

5.3.3 Function

Inherits the function of Building and affects the Units in its vicinity.

5.3.4 Subordinates

This component has no subordinates.

5.3.5 Dependencies

Requires the superclass Building.

5.3.6 Interfaces

This component has no interfaces.

5.3.7 Resources

Sprite Map has to exist for the buildings visual representation to be correct.

5.3.8 References

This component requires no references.

5.3.9 Processing

As a model, the Barrier building does not process anything.

5.3.10 Data

Data	Structure	Description
No additional data is stored in Barrier building.		

5.4 Base Building

5.4.1 Type

This component is a **subclass** of Building belonging to the **model** part of MVC.

5.4.2 Purpose

The purpose of this module is to implement the data storage of the Base buildings described in SR1.3, along with policies on how to retrieve and update the data.

5.4.3 Function

Mainly inherits the function of Building, but adds the following functionality:

Keeps track of attached buildings. Is the base of a component in the graph, meaning it will "accept" building visitors (see description of the *Visitor pattern* in Building) and register them to an array of Buildings. It will also provide functionality to traverse the graph-component and to produce units for the graph-component.

```
public LinkedList.Enumerator GetBuildings();
public int Production();
public void Visit(AggressiveBuilding building);
public void Visit(BarrierBuilding building);
public void Visit(ResourceBuilding building);
public void Visit(BaseBuilding building);
```

5.4.4 Subordinates

This component has no subordinates.

5.4.5 Dependencies

Requires the superclass Building.

5.4.6 Interfaces

Here follows a list of interface-methods that is provided for other Buildings and the Building Controller.

Caller/Callee	Header
Building Controller	`public LinkedList.Enumerator GetBuildings();`
Building Controller	`public int Production();`
Aggressive Building	`public void Visit(AggressiveBuilding building);`
Barrier Building	`public void Visit(BarrierBuilding building);`
Resource Building	`public void Visit(ResourceBuilding building);`
Base Building	`public void Visit(BaseBuilding building);`

5.4.7 Resources

Sprite Map has to exist for the buildings visual representation.

5.4.8 References

The Visitor pattern has to be understood to understand the relationship with the Base building, please refer to wikipedia[8] for this.

5.4.9 Processing

As a model, the Base building does not process much. It does however override the default building behaviour of copying a reference of a Base building. Instead it copies a reference of itself, thus ignoring inheritance and creates a new graph-component.

5.4.10 Data

Data	Structure	Description
childBuildings	`LinkedList(Of Building)`	A list of child buildings.

5.5 Resource Building

5.5.1 Type

This component is a **subclass** of Building belonging to the **model** part of MVC.

5.5.2 Purpose

The purpose of this module is to implement the data storage of the Resource buildings described in SR1.3, along with policies on how to retrieve and update the data. Implements SR1.10

5.5.3 Function

Inherits the function of Building, but additionally it keeps track of how many units the building will produce every round.

5.5.4 Subordinates

This component has no subordinates.

5.5.5 Dependencies

Requires the superclass Building.

5.5.6 Interfaces

Caller/Callee	Header
5.20. Building Controller	`public void SetProductionRate(int rate);`
5.4. Base Building	`public int GetProductionRate();`

5.5.7 Resources

Sprite Map has to exist for the buildings visual representation.

5.5.8 References

This component requires no references.

5.5.9 Processing

As a model, the Resource building does not process anything.

5.5.10 Data

Data	Structure	Description
rateOfProduction	`int`	How many units this building produces per round.

5.6 Game Options

5.6.1 Type

Game options is a **class** in the **model** part of the MVC.

5.6.2 Purpose

The purpose of this component is to keep track of configurations/options in the application. It implements SR1.8.

5.6.3 Function

Keeps track of all options availiable in the game, both during gameplay and in menus.

Possible options:

- Sound volume
- Screen resolution
- Game difficulty
- ... more are likely to be added during development.

Since this component is a model it will implement the publisher pattern. It will provide one or several *event*s for observers to register themselves to. This component will then publish any updates to the observers when one of its properties has changed by calling its *event*.

5.6.4 Subordinates

This component doesn't have any subordinates.

5.6.5 Dependencies

Does not have any dependencies and as such it should probably be one of the first objects constructed.

5.6.6 Interfaces

Caller/Callee	Header
5.21. Configurator	`public void SetValue(string option);`
5.9. Language	`public Object GetValue(string option);`
5.30. Menu Controller	`public Object GetValue(string option);`
5.34. World Controller	`public Object GetValue(string option);`
5.36. Graphic Rendering	`public Object GetValue(string option);`
5.38. Music Player	`public Object GetValue(string option);`
All above components	`public event Publish<GameOption> OptionHasChanged;`

5.6.7 Resources

The application should save the set options to an external file upon exit, and load the values from said file upon startup. The file would be a simple text-file along the lines of a *.ini-file.

5.6.8 References

This component has no required references.

5.6.9 Processing

```
private void LoadSettings()
{
    fr = fileReader(settings.ini);
    foreach(line in fr)
    {
        // Line format: descriptor = value
        options.Add(line.descriptor, line.value);
```

```
    }
}

private void SaveSettings()
{
    fs = fileWriter(settings.ini);
    keys = options.GetKeys();
    foreach(key in keys)
    {
        fs.WriteLine(key+" = "+options.GetValue(key));
    }
}
```

5.6.10 Data

Data	Structure	Description
options	`Dictionary<string><Object>`	Keeps track of the values of all options.

5.7 Graph

5.7.1 Type

The Graph component is a **class** belonging to the **model** part of MVC.

5.7.2 Purpose

Stores a set of buildings and their weights for the graph weighting.

5.7.3 Function

The Graph component is a storage class for buildings and their weights.

Since this component is a model it will implement the publisher pattern. It will provide one or several *event*s for observers to register themselves to. This component will then publish any updates to the observers when one of its properties has changed by calling its *event*.

5.7.4 Subordinates

The Graph has no subordinates.

5.7.5 Dependencies

The following things must be available in order for the component to function:'

- 5.1. Building
- 5.4. Base Building

5.7.6 Interfaces

Caller/Callee	Header
Graph Controller	`void Add(Building building);`
Graph Controller	`void Remove(Building building);`
Graph Controller	`void SetWeight(Building building, int weight);`
Graph Controller	`int GetWeight(Building);`
Graph Controller	`float GetWeightFraction(Building b);`
Various Controllers	`Enumerable<Building> GetBuildings();`
Any View	`public event Publish<Building> weightChanged;`

5.7.7 Resources

The Graph needs no resources.

5.7.8 References

The Graph needs no references to be understood.

5.7.9 Processing

Since Graph is a model it lacks processing.

5.7.10 Data

Data	Structure	Description
buildings	`Dictionary<Building, int>`	The Buildings and their weights.
baseBuilding	`BaseBuilding`	The originating building for this graph.

5.8 GUI Region

5.8.1 Type

The GUI Region component is a model component.

5.8.2 Purpose

The purpose of this model component is simply to provide viewable regions to be used by the eye tracking system. It realizes SR5.1.

5.8.3 Function

The GUI Region component is implemented to provide the same functionality as an ordinary clickable button but will instead of (or possibly, as well as) mouse clicks, provide it's event triggers through eye gaze timers.

Since this component is a model it will implement the publisher pattern. It will provide one or several *event*s for observers to register themselves to. This component will then publish any updates to the observers when one of its properties has changed by calling its *event*.

5.8.4 Subordinates

This component has no subordinates.

5.8.5 Dependencies

This model component will depend on the abstract class `GenericInteractionRegion<T>` provided by Tobii that a class needs to inherit from if it wants to be considered by the Tobii SDK as a viewable region.

5.8.6 Interfaces

Each Tile component will in the game be considered a unique GUI Region. In addition to this, each element in the user interface will also be considered a unique GUI Region. Because of this all viewable regions will already be defined when a game starts. Therefor, the GUI Region component will not provide public getters/setters.

Caller/Callee	Header
5.32. Tobii Controller	`public event Publish<GUIRegion> regionActivated;`

5.8.7 Resources

This component has no external resources.

5.8.8 References

See the Tobii SDK for details regarding `GenericInteractionRegion.`

5.8.9 Processing

The GUI Region model component will perform different processing depending on what the `onActivate()` function is designed to do for a specific region. The `onActivate()`-function will mostly call one of the controllers and tell it to do something with the game.

5.8.10 Data

Data	Structure	Description
Bounding geometry	`Geometry`	This structure will hold the bounding regions of this GUI Region.
Dwell Time	TimeSpan	The time required to trigger this Region `onActivate` function.
⋮	⋮	⋮

This model component will, as stated earlier, inherit from the abstract class

`GenericInteractionRegion<T>`. It will therefore at least store all the boolean values and geometry information there.

5.9 Language

5.9.1 Type

This component is a **class** belonging to the **model** part of MVC.

5.9.2 Purpose

The purpose of the Language component is to store different translations for strings in the application.

5.9.3 Function

The Language component provides functionality for getting strings in the correct language. It does this by providing a static function for every other component which handles strings in the application.

```
public static string GetString(string label);

// Adding new strings to the model
public void GetString(string label, Type type, string translation);

// Changing the language
public void SetLanguage(Language.Type newLanguage);
public Language GetLanguage();

// Different language
public enum Type { SWEDISH, ... }
```

Since this component is a model it will implement the publisher pattern. It will provide one or several *event*s for observers to register themselves to. This component will then publish any updates to the observers when one of its properties has changed by calling its *event*.

5.9.4 Subordinates

There are no subordinates to this component.

5.9.5 Dependencies

This component depends on the Localizer to populate it with translations.

5.9.6 Interfaces

Caller/Callee	Header
All Views	`public static string GetString(string label);`
All Views	`public event Publish<Language> CurrentLanguageChanged;`
5.28. Localizer	`public void SetString(string label, Type type, string translation);`
5.28. Localizer	`public void SetLanguage(Language.Type newLanguage);`
5.28. Localizer	`public Language GetLanguage();`

5.9.7 Resources

This component requires no resources.

5.9.8 References

There are no references for this component.

5.9.9 Processing

There is no processing in this component since it's a model.

5.9.10 Data

Data	Structure	Description
currentLanguage	`Type`	Stores the currently selected language.
translations	`Dictionary<Type, <string, string>>`	Stores all translations.

5.10 Menu Model

5.10.1 Type

The Menu Model component is a model component.

5.10.2 Purpose

The purpose of this component is to provide data for the 5.30. Menu Controller and 5.37. Menu View . It assists these components in realizing SR3.4.

5.10.3 Function

This component will provide the Menu View with information about the appearance of a specific menu item, which will be an instance of Menu Model. It will also provide the menu controller with access to the specific menu items "onActivate" function.

Since this component is a model it will implement the publisher pattern. It will provide one or several *event*s for observers to register themselves to. This component will then publish any updates to the observers when one of its properties has changed by calling its *event*.

5.10.4 Subordinates

This component has no subordinates.

5.10.5 Dependencies

- 5.30. Menu Controller
- 5.37. Menu View

5.10.6 Interfaces

Caller/Callee	Header
5.30. Menu Controller	public IEnumerable GetRegieons();
5.37. Menu View	public DrawData GetDrawData();
Any View	`public event Publish<Menu_model> stateChanged;`

5.10.7 Resources

This component requires no external resources.

5.10.8 References

No references are required for this component.

5.10.9 Processing

This component performs no processing, it will only provide data for the Menu View to render, and for the Menu Controller to modify.

5.10.10 Data

Data	Structure	Description
Viewable region	`IEnumerable<GUIRegion>`	This component will hold it's defined GUI Region(s).
Texture	`DrawData`	This is the visual representation of the menu.

5.11 Menu Manager

5.11.1 Type

The Menu Manager component is a **class** belonging to the **model** part of MVC.

5.11.2 Purpose

The Menu Manager keeps track of the current state of the menus, so that input is associated with the correct set of GUIRegions. Needed to fulfill SR3.4.

5.11.3 Function

The Menu manger has an internal stack of 5.10. Menu Model -objects and the position on the screen of all these Menu Objects.

It's interface is defined as follows:

`public void PushMenu(Menu menu)`
Puts the menu on the top of the stack.

`public Menu PopMenu()`
Returns the topmost menu in the stack.

`public void ClearMenus()`
Clears the stack of menus, effectively resetting the state of the menus.

```
public void SetPosition(Menu m, Vector2 position);
```

`Vector2 GetPosition(Menu m)`
Return a vector containing the onscreen position of the menu for rendering purposes.

More methods are to be expected in the implementation phase.

And since this component is a model it will implement the publisher pattern. It will provide one or several *event*s for observers to register themselves to. This component will then publish any updates to the observers when one of its properties has changed by calling its *event*.

5.11.4 Subordinates

The Menu Manager has no subordinates.

5.11.5 Dependencies

- 5.10. Menu Model

5.11.6 Interfaces

	Caller/Callee	Header
The Menu Manager	5.37. Menu View	`public event Publish<Menu> MenuEvent;`
	5.37. Menu View	`public Vector2 GetPosition(Menu m);`
	5.30. Menu Controller	`public Menu PopMenu()`
	5.30. Menu Controller	`public Menu PushMenu(Menu m)`
	5.30. Menu Controller	`public Menu SetPosition(Menu m, Vector2 position)`
	5.30. Menu Controller	`public Menu ClearMenus()`

5.11.7 Resources

This component has no resources.

5.11.8 References

The World needs no references to be understood.

5.11.9 Processing

Since the Menu Manager is a model this section is not applicable.

5.11.10 Data

Data	Structure	Description
menuStack	`Stack<Menu>`	The menustack. The most recently added menu is on top
menuPosition	`Map<Menu, Vector2`	The position of each map

5.12 Player

5.12.1 Type

The Player component is a **class** belonging to the **model** part of MVC.

5.12.2 Purpose

The Player stores information about a player and holds a place for him/her in the turn queue.

5.12.3 Function

```
public string GetName();
public Color GetColor();

public Graph[] GetGraphs();
public int GetUpgradeLevel();
```

Since this component is a model it will implement the publisher pattern. It will provide one or several *event*s for observers to register themselves to. This component will then publish any updates to the observers when one of its properties has changed by calling its *event*.

The `GetColor` function allows outside callers to find out what color to use when representing the Players units and buildings in the game.

The `GetName` function allows outside callers to find out what name to use when represent the Player in various situations.

The `GetGraphs` function returns all subgraphs of the player's networks.

The `GetUpgradeLevel` function returns the upgrade level of the players units.

5.12.4 Subordinates

The Player has no subordinates.

5.12.5 Dependencies

The Player depends on Graph.

5.12.6 Interfaces

The Player component is a storage class for players. It stores data about them and nothing more.

Caller/Callee	Header
5.29. Victor Turner	`Color GetColor();`
5.29. Victor Turner	`String GetName();`
5.29. Victor Turner	`Graph[] GetGraphs();`
5.23. Unit Controller	`int GetUpgradeLevel();`
Unknown	`void SetUpgradeLevel(int);`
Any View	`public event Publish<Graph[]> graphsChanged;`

5.12.7 Resources

The Player needs no resources.

5.12.8 References

The Player needs no references to be understood.

5.12.9 Processing

Since Player is a model it lacks processing.

5.12.10 Data

Data	Structure	Description
color	`Color`	The color of the Players units and buildings in the game.
name	`string`	The name of the Player.
graphs	`Graph[]`	The Graphs belonging to the player.
upgradeLevel	`int`	The upgrade level of this players units.

5.13 Sounds

5.13.1 Type

This component is a **class** belonging to the **model** part of MVC.

5.13.2 Purpose

This class will manage the loading and decoding of all the games sounds. This include background music and sound effects.

5.13.3 Function

The Sounds component will manage all the games sound files. Other controllers that need to play sounds will call for an appropriate sound from the Sounds component and receive a Cue object which can be used to play said sound.

Since this component is a model it will implement the publisher pattern. It will provide one or several *event*s for observers to register themselves to. This component will then publish any updates to the observers when one of its properties has changed by calling its *event*.

5.13.4 Subordinates

The Sounds component has no subcomponents.

5.13.5 Dependencies

The Sounds component will assume that sound files exists, and that they are in the appropriate format. Sound files will be encoded using the XACT tool distributed with the XNA framework.

5.13.6 Interfaces

Caller/Callee	Header
Any controller	`public Cue LoadSound(string sound);`
Any View	`public event Publish<Sounds> soundsCued;`

5.13.7 Resources

XNA Content Pipeline The Sounds Class requires the XNA Content Pipeline in order to load sound files.

5.13.8 References

Microsoft provides guides for using XACT and the XNA Content Pipeline.[1]

5.13.9 Processing

Controllers that need to play sounds will use the `LoadSound()` function of the Sounds component. It takes a string as a parameter that identifies a certain Cue, which is a collection of one or more sounds.[3]

```
public Cue LoadSound(string sound)
{
    return soundBank.GetSong(sound);
}
```

5.13.10 Data

Data	Structure	Description
audioEngine	`AudioEngine`	The Audio Engine XNA object creates WaveBanks and SoundBanks.
soundBank	`SoundBank`	The XNA SoundBank object containing all the Cues.

[1] See `http://msdn.microsoft.com/en-us/library/bb203895\%28XNAGameStudio.20\%29.apx`

5.14 Sprite Texture Map

5.14.1 Type

This component is a **class** belonging to the **model** part of MVC.

5.14.2 Purpose

The class is a wrapper for the content pipeline handles from XNA.

5.14.3 Function

This component will heed requests from view components and deliver textures in response.

```
public Texture2D GetTexture(Globals.Texture texture, int curFrame, int size);
```

5.14.4 Subordinates

This component has no subcomponents.

5.14.5 Dependencies

This component has no dependencies.

5.14.6 Interfaces

Caller/Callee	Header
AnyView	`public Texture2D GetTexture(Globals.Texture texture);`

5.14.7 Resources

XNA Content Pipeline The Sprite Texture Map Class requires the XNA Content Pipeline in order to load images files.

5.14.8 References

This component has no references.

5.14.9 Processing

The wrapper decides from the input which texture to get, and which part of that texture to return (the sprite).

```
public SpriteTextureMap(ContentManager content)
{

    Retrive each texture enum in Globals, retrive its enum name.

    Use the names as file names and load all the images representing the enum.

    Place them in the Texture2D[].

}

public Texture2D GetTexture(Globals.Texture texture)
{
    return textures[(int)texture];
}
```

5.14.10 Data

Data	Structure	Description
textures	`Texture2D[]`	An array of textures, indexed parallel to an enumeration in the sealed global class.

5.15 Terrain Type

5.15.1 Type

The Terrain Type component is a model component.

5.15.2 Purpose

The purpose of this component is to provide the game mechanic of different kinds of terrain.

It realizes SR 1.14

5.15.3 Function

The Terrain Type component provides the software with functions to access information about the type of terrain that is present in a given Tile component. The different kinds of terrain will in the game contribute with various properties for units and buildings.

5.15.4 Subordinates

This component has no subordinates.

5.15.5 Dependencies

This component has no dependencies.

5.15.6 Interfaces

As a terrain type is a subordinate to a Tile, this component provides information about the terrain present on the specified Tile, all data here is accessed with calls through Tile.

Caller/Callee	Header
5.16. Tile	`public float GetSpeedModifier();`
5.16. Tile	`public int GetDamageModifier();`
5.16. Tile	`public int GetResourceModifier();`

```
public float GetSpeedModifier();
PRE: A call is made to retrive the speed modifier present on the Terrain Type
of the specified Tile.
POST: The speed modifier is returned.
```

```
public int GetDamageModifier();
```

Terrain might be poisoned by special abilities or other means.

```
PRE: A call is made to retrieve the damage modifier from this terrain.
(as our units do not have hitpoints, this could represent the rate at
which units perish).
POST: The damage modifier is returned.
```

```
public int GetResourceModifier();
```

The game will provide resource hotspots, as stated in earlier documents.

```
PRE: A call is made to retrive the amount
of available resources / the resource income rate bonus, from this terrain.
POST: Said number is returned.
```

5.15.7 Resources

No external resources are required for this component.

5.15.8 References

No references are needed for this component.

5.15.9 Processing

This component performs no processing.

5.15.10 Data

Data	Structure	Description
Damage modifier	`int`	The damage modifier provided by this terrain
Speed modifier	`int`	The speed modifier provided by this terrain
Resource modifier	`int`	The resource modifier provided by this terrain

5.16 Tile

5.16.1 Type

The Tile component is a model component.

5.16.2 Purpose

The purpose of this component is to represent one square unit of the game field. This component realizes SR 1.7 and SR1.5

5.16.3 Function

A set of Tile components will be what creates a map for the game. It provides the functionality to access information about terrain type, resources and and what players can see this tile.

Since this component is a model it will implement the publisher pattern. It will provide one or several *event*s for observers to register themselves to. This component will then publish any updates to the observers when one of its properties has changed by calling its *event*.

5.16.4 Subordinates

- 5.15. Terrain Type

5.16.5 Dependencies

- 5.15. Terrain Type

5.16.6 Interfaces

This component will provide interfaces to interact with the tile. It will contain getters and setters for the data it contains, mentioned in the Data subsection.

i.e

Caller/Callee	Header
5.39. World View	`Get/SetTerrainType(TerrainType t);`
5.39. World View	`Get/SetUnits(Unit[] u);`
5.39. World View	`Get/SetBuilding(Building b);`
5.39. World View	`CanBeSeenBy(Player[] p);`
Any View	`public event Publish<Player[]> visionChanged;`
Any View	`public event Publish<Unit[]> unitsChanged;`
Any View	`public event Publish<Building> buildingChanged;`

```
Get/SetTerrainType(TerrainType t);
PRE: attempts to get/set the Terrain Type.
POST: the terrain type is retrieved/set.
```

```
Get/SetUnits(Unit[] u);
```

Returns the set of units on this tile. This could possibly return an array of Unit components, as for combat purposes units will be divided into swarms acting as a single unit or "squad".

```
Get/SetBuilding(Building b);
PRE: attempts to add/retrieve a building to/from this tile
POST: The building has been positioned/returned
```

5.16.7 Resources

This component has no external resources.

5.16.8 References

This component has no referances.

5.16.9 Processing

The Tile component performs no processing, it only contains data structures.

5.16.10 Data

Data	Structure	Description
terrain type	`TerrainType`	A terrain type structure with data about the kind of terrain on this tile
position	`Vector2`	The position in the worlds tile map
units	`Unit[]`	The units currently standing on the tile if any
building	`Building`	The building constructed on the tile if any
visibleTo	`Player[]`	An array that indicates to which players the tile is visible

5.17 Unit

5.17.1 Type

The unit is a **class** that belongs in the **model** part of the MVC.

5.17.2 Purpose

The purpose of the unit class is to give a graphical representation of the units. The unit is controlled by the Unit Controller. Each unit has a target that it decide it's actions.

This component is fullfills the SR-requirement 'SR1.2 Autonomous Units'.

5.17.3 Function

Contains the graphical representation of units and some basic properties.

Since this component is a model it will implement the publisher pattern. It will provide one or several *event*s for observers to register themselves to. This component will then publish any updates to the observers when one of its properties has changed by calling its *event*.

```
// Properites
public boolean IsDead();
public Player& GetPlayer();
public int GetTargetX();
public int GetTargetY();
public void SetTargetX(int);
public void SetTargetY(int);
public int GetXTile();
public int GetYTile();
public void SetDispersed(boolean);
public boolean IsDispersed();

// Graphical representation
public Texture GetSprite();
```

```
public int GetXOffset();
public int GetYOffset();

public int GetAngle();

// Modifiers
public boolean Kill();
public void Update(int systemTime);
```

5.17.4 Subordinates

This class has no subordinates.

5.17.5 Dependencies

- 5.14. Sprite Texture Map
- 5.12. Player

5.17.6 Interfaces

Caller/Callee	Header
5.36. Graphic Rendering	`public Sprite GetSprite()`
5.36. Graphic Rendering	`public int GetXOffset()`
5.36. Graphic Rendering	`public int GetYOffset()`
5.36. Graphic Rendering	`public int GetAngle()`
5.23. Unit Controller	`public int GetXTile()`
5.23. Unit Controller	`public int GetYTile()`
5.23. Unit Controller	`public void Kill()`
5.23. Unit Controller	`public void Update(int)`
5.23. Unit Controller	`public Tile GetTargetX()`
5.23. Unit Controller	`public Tile GetTargetY()`
5.23. Unit Controller	`public void SetTargetX(Tile)`
5.23. Unit Controller	`public void SetTargetY(Tile)`
5.23. Unit Controller	`public void SetDispersed(boolean)`
5.23. Unit Controller	`public boolean IsBusy()`
Any View	`public event Publish<Unit> unitUpdated;`

5.17.7 Resources

The Unit component has no resources.

5.17.8 References

The Unit component needs no references.

5.17.9 Processing

With one exception, no processing is done in this component. It mainly serves to provide graphical information of a unit and to see whether or not it is to be drawn.

The only processing done in this component is in the Update() function where the position and angle of the unit is calculated knowing it's properties and the time passes since the last update.

5.17.10 Data

Data	Structure	Description
tileX	`int`	The column of the tile the unit is in
tileY	`int`	The row of the tile the unit is in
targetX	`int`	The column of the tile the unit is in
targetY	`int`	The row of the tile the unit is in
isDispersed	`int`	Whether or not this unit should recieve a new target from the dispersion procedu
x	`int`	The x-offset in pixels of this unit in the world
y	`int`	The y-offest in pixels of this unit in the world
angle	`int`	The angle of the unit in degrees. Used for drawing purposes
isDead	`boolean`	Whether or not the unit is still in play

5.18 World

5.18.1 Type

The World component is a **class** belonging to the **model** part of MVC.

5.18.2 Purpose

The World class represents the current game environment according to SR1.1, SR1.2, SR1.3, SR1.7, SR1.10.

5.18.3 Function

The World model holds variables that define the current screen and the tiled environment. It's interface is defined as follows:

`void SetMap(Map newMap)`
Changes the Worlds map to the given one.

`Map GetCurrentMap()`
Returns the current map of the World.

`void RemovePlayer(Player newPlayer)`
Removes the player from the list of players currently in the game world.

`Player[] GetPlayers()`
Returns a list of all the players currently present in the game world.

`Rectangle CurrentScreen`
C# accessor to get/set a rectangle with the same bounds and positions as the screen.

More methods are to be expected in the implementation phase.

And since this component is a model it will implement the publisher pattern. It will provide one or several *event*s for observers to register themselves to. This component will then publish any updates to the observers when one of its properties has changed by calling its *event*.

5.18.4 Subordinates

The World Model has an inner class Map

5.18.5 Dependencies

- **Map**
- 5.12. Player

5.18.6 Interfaces

The World model is used as a means of holding all relevant world variables in one place for readability's sake, so the only thing that the World provides is Get/Set/Remove-functions for its variables.

It also keeps track of the current screen for the 5.36. Graphic Rendering .

Caller/Callee	Header
5.39. World View	`public event Publish<Tile> TileEvent;`
5.39. World View	`public event Publish<Tile[,]> MapEvent;`
5.39. World View	`public event Publish<Player> PlayerEvent;`
5.33. World Generator	`public void SetMap(Map m)`
5.39. World View	`public Map GetMap()`
5.33. World Generator	`public Map AddPlayer()`
5.29. Victor Turner	`public Map RemovePlayer(Player p)`
Various Components	`public void GetPlayers();`

5.18.7 Resources

This component has no resources.

5.18.8 References

The World needs no references to be understood.

5.18.9 Processing

Since the World is a model this section is not applicable.

5.18.10 Data

Data	Structure	Description
map	`Map`	The current map being played.
players	`List<Player>`	All players in the worl.
currentScreenPosition	`Rectangle`	The current screen bounding rectangle.

5.19 AI Player

5.19.1 Type

The AI Player component is a **module** belonging to the **controller** part of MVC.

5.19.2 Purpose

The purpose of the AI Player is to act as an opponent to the human Player in the game. This is according to SR1.1, SR1.4, SR1.15 and SR1.19.

5.19.3 Function

The AI Player is very simple when viewed from the outside, the only function it has is

```
void MakeMove()
```

which causes the AI Player to evaluate its situation and take some actions in the game.

5.19.4 Subordinates

The AI Player has no subordinates.

5.19.5 Dependencies

The following Models must exist for the AI Player to function:

- 5.35. AI View
- 5.20. Building Controller
- 5.22. Graph Controller
- 5.24. Unit Accountant
- 5.23. Unit Controller

5.19.6 Interfaces

Caller/Callee	Header
5.29. Victor Turner	`void makeMove()`

5.19.7 Resources

AI View is needed for the AI Player to get information about the game world.

5.19.8 References

The AI Player needs no references.

5.19.9 Processing

Once the turner has called the AI Players `MakeMove()` function the AI Player calls the AI View for available information about the world and then proceeds to calling the Controllers to make changes to the world.

5.19.10 Data

Data	Structure	Description
Data variables unknown at this time.		

5.20 Building Controller

5.20.1 Type

This component is a **class** belonging to the **controller** part of MVC.

5.20.2 Purpose

As described in the software requirement SR1.3 the buildings will have different kinds of functionality. The purpose of this component is to implement that functionality by taking relevant input from the *views* and modifying the *models* accordingly.

5.20.3 Function

For each building in the world run one of these functions depending on the type of the building.
5.2. Aggressive Building

```
public void AcquireTarget()
{
    Find a number of targets and fire at them();
}
```

5.5. Resource Building

```
public void SetProductionRate()
{
    Check the amount of resources from the tile beneath it set it as a rate();
}
```

5.20.4 Subordinates

This component has no subordinates.

5.20.5 Dependencies

This component requires the 5.1. Building Building component and all of it's subordinates to exist.

5.20.6 Interfaces

Caller/Callee	Header
Player.Graph	`public void AddBuilding();`
Player.Graph	`public void RemoveBuilding();`
Player.Graph	`public void DamageBuilding();`
Player.Graph	`public void RepairBuilding();`

5.20.7 Resources

The Building Controller has no resources.

5.20.8 References

The Building Controller needs no references.

5.20.9 Processing

The controller creates, removes and modifies building instances held in the player superclass (graph members).

```
public void AddBuilding(Globals.BuildingType type, Player player)
{
    construct a new building depending on the type and if it is not a base building add it t
    if it is a base building construct a new graph.
}
```

5.20.10 Data

Data	Structure	Description
No data is stored in this component.		

5.21 Configurator

5.21.1 Type

The configurator is a **class** belonging to the **controller** part of MVC.

5.21.2 Purpose

Handles state management for the component 5.6. Game Options .

5.21.3 Function

The configurator contains functionality to operate on 5.6. Game Options . It will use the 5.30. Menu Controller to get the menus for changing the options. Depending on the return value of the 5.30. Menu Controller it will change the 5.6. Game Options model in different ways.

A handful of suggested methods:

```
private void BuildMenu();
private void ChangeOptions();
```

5.21.4 Subordinates

This component does not have any subordinates.

5.21.5 Dependencies

Requires an instance of 5.6. Game Options and 5.10. Menu Model .

5.21.6 Interfaces

There is no flow of data to this component, however it makes use of some other components, and it should do this in the following suggested methods.

Caller/Callee	Header
5.10. Menu Model	`private void BuildMenu();`
5.6. Game Options	`private void ChangeOptions();`

5.21.7 Resources

This component requires no resources.

5.21.8 References

This component requires no references.

5.21.9 Processing

There is no data processing delicate enough to deserve an in-depth explanation.

5.21.10 Data

Data	Structure	Description
No data is stored in this component.		

5.22 Graph Controller

5.22.1 Type

This component is a **singleton class** belonging to the **controller** part of MVC.

5.22.2 Purpose

The buildings in the game will be built in clusters known as graph components or subgraphs. The Graph Controller will provide this functionality as well as controlling the dispersion of units in these components according to the software requirement SR1.2.

5.22.3 Function

The Graph Controller organizes Buildings into different groups and makes sure that appropriate numbers of units are kept in each building of the components at all times.

5.22.4 Subordinates

Graph Controller has no subordinates.

5.22.5 Dependencies

The following things must be available in order for the graph controller to function.

- 5.17. Unit
- 5.1. Building
- 5.23. Unit Controller
- 5.7. Graph

5.22.6 Interfaces

Caller/Callee	Header
5.37. Menu View	`void AddBuilding(Building source, Building newBuilding)`
5.37. Menu View	`Graph AddBaseBuilding(Building newBaseBuilding)`
5.37. Menu View	`void RemoveBuilding(Building building)`
5.37. Menu View	`void SetWeight(Building building, int weight)`
5.29. Victor Turner	`void CalculateWeights()`
5.23. Unit Controller	`void MoveUnits(int number, Tile from, Tile to)`

5.22.7 Resources

5.23. Unit Controller The Graph Controller will call the Unit Controller with orders about changes in unit positioning.

5.22.8 References

There are no references available for the Graph Controller.

5.22.9 Processing

```
public void RemoveBuilding(Building building)
{
    GetGraph(building).Remove(building);
}

public void AddBuilding(Building source, Building newBuilding)
{
    GetComponent(source).Add(newBuilding);
}

public Graph AddBaseBuilding(Building newBaseBuilding)
{
    Graph newGraph = new Graph(newBaseBuilding);
```

```
        return newGraph;
    }

    public void SetWeight(Building building, int weight)
    {
        GetGraph(building).SetWeight(building, weight);
    }

    public void CalculateWeights()
    {
        for each Graph:
            get the total number of units in the graph;
            for each building in the graph:
                check that the ratio between the units in that building and its
                weight is proportional to the total number of units.

        move units accordingly
    }
```

5.22.10 Data

Data	Structure	Description
components	`Graph[]`	A list of all the Graph components.

5.23 Unit Controller

5.23.1 Type

The Unit Controller is a **module** in the **controller** part of the MVC.

5.23.2 Purpose

The purpose of the module is to control each individual unit so that they appear to be alive. The unit moves towards a target, and it is this that the unit controller will realize.

5.23.3 Function

```
public Unit[] GetUnits();
public BaseBuilding GetGraph();

public MoveUnits(Collection<Unit> units, int amount, Tile from, Tile to);

private void Update(Unit unit, int deltaT);
private void SetTarget(Unit unit, int tileX, int tileX);

private void Kill(Unit unit);
```

5.23.4 Subordinates

This component has no subordinates.

5.23.5 Dependencies

This component requires Unit and Graph.

- 5.17. Unit
- 5.7. Graph

5.23.6 Interfaces

Caller/Callee	Header
Graph	`public Building[] GetBuildings();`
Graph	`public int GetWeight(Building);`
Building	`public Unit[] GetUnits();`
Building	`public void AddUnits(Unit[]);`
Unit	`public void Update();`
Graph Controller	`public MoveUnits(Collection<Unit> units, int amount, Tile from, Tile to);`

5.23.7 Resources

This component does not require any resources.

5.23.8 References

This component does not require any references.

5.23.9 Processing

```
public void UnitController()
{
    // Reallocate units over the graph
    Get weights for each building in the graph
    Calculate the amount of units each node should have
    For each node n :
        If n has too few units, put it in the 'need' list
        For all units u in n:
            Put u in the 'free' list
            Remove u from n

    For all units u in 'free':
        Get the closest node n in the 'need'
        Set target of u to n
```

```
        // Move all units
        timeToRun = amount of time to run the update routine
        timeToStop = systemTime + timeToRun;
        update(timeToStop)
    }

    public void update(timeToStop)
    {
        while(systemTime < timeToStop)
            for each unit u in the graph
                if the unit is dead
                    remove from graph
                u.update(systemTime);
                if the unit has attacked a building
                    check whether or not the unit dies
    }
```

5.23.10 Data

Data	Structure	Description
graph	`Graph`	The graph this controller operates in

5.24 Unit Accountant

5.24.1 Type

The Unit Accountant is a **class** in the **controller** part of the MVC.

5.24.2 Purpose

The purpose of the Unit Accountant is to insert new units into its graph as they are created by the buildings.

5.24.3 Function

The unit accountant is called by turner.

```
private void ProduceUnits(); // Get all Base buildings and have them produce units.
```

5.24.4 Subordinates

This component has no subordinates.

5.24.5 Dependencies

- 5.18. World
- 5.12. Player
- 5.1. Building
- 5.4. Base Building

5.24.6 Interfaces

Caller/Callee	Header
Building	`public void addUnits(Unit[]);`

5.24.7 Resources

This component does not have any resources.

5.24.8 References

This component does not have any references.

5.24.9 Processing

```
function void ProduceUnits()
{
    for all Base Building b in bases
        for all graphs g in p
            for all buildings b in g.GetBuildings()
                if b is a Base Building
                    b.AddUnits(b.Production())
                    world.GetTile(b.x,b.y).AddUnits(//Units to produce)
}
```

5.24.10 Data

Data	Structure	Description
world	`BaseBuilding[]`	The world

5.25 Initializer

5.25.1 Type

This component is a **class** belonging to the **controller** part of MVC.

5.25.2 Purpose

The game has to be initialized to be able to run. The purpose of this component is to configure the graphics device and appropriate environmental variables of the OS to enable the game to run, as well as to launch the software itself.

5.25.3 Function

The initializer is the main entry point of program (the main static class) as well as the `constructor`, the `override void Initialize` and the `override void LoadContent` methods of the main game thread.

```
// Main entry point
static void Main(string[] args)

// Constructor
public Game()

// Initializers
protected override void Initialize()
protected override void LoadContent()
```

5.25.4 Subordinates

- 5.41. Main
- 5.26. Game Initializer

5.25.5 Dependencies

This component has no dependencies.

5.25.6 Interfaces

Calls all controller initializers.

5.25.7 Resources

This component has no resource requirements.

5.25.8 References

This component has no references.

5.25.9 Processing

This component only initializes the software, and does not do any complicated processing of its own.

5.25.10 Data

This thread will initialize all the data components present in the Main game thread class, it will not have any data fields of its own.

5.26 Game Initializer

5.26.1 Type

A method which is run one time in the beginning of a created game session, creating all objects needed in a 5.18. World .

5.26.2 Purpose

Will first call 5.33. World Generator and then it will add players to the world and place the required buildings and units to start a new game.

5.26.3 Function

`private bool Populate();` Creates necessary objects and adds them to game world.

5.26.4 Subordinates

- 5.33. World Generator

5.26.5 Dependencies

This method will be the first one run at a started game.

5.26.6 Interfaces

Caller/Callee	Header
5.25. Initializer	`private bool Populate();`

5.26.7 Resources

This component has no resources.

5.26.8 References

SRD – Appendix C: "Game concept summary"[1] might provide a basic idea of what needs to be created here.

5.26.9 Processing

```
private bool Populate()
{
    CreateGameObjects();

    if(success)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

5.26.10 Data

Data	Structure	Description
No data is stored in this component.		

5.27 Loader

5.27.1 Type

The Loader component is a **class** belonging to the **controller** part of MVC.

5.27.2 Purpose

The purpose of the Loader is to fulfill the requirements SR1.6 regarding the function to load a previous saved game.

5.27.3 Function

The function of this component is to load a previous saved game from a file on the harddrive. It will set the current state of the world to the loaded state.

5.27.4 Subordinates

There are no subordinates to this component.

5.27.5 Dependencies

The following things must be available in order for the loader to function.

- 5.18. World
- 5.12. Player

5.27.6 Interfaces

Caller/Callee	Header
Load View	`LoadWorld(World loadedWorld)`

5.27.7 Resources

Saved file A saved game shall exist on the harddrive.

5.27.8 References

5.27.9 Processing

```
public void LoadWorld(SaveGameData data)
{
    Load the game models according to data;
}
```

5.27.10 Data

Data	Structure	Description
No data is stored in this component.		

5.28 Localizer

5.28.1 Type

This component is a **class** belonging to the **controller** part of MVC.

5.28.2 Purpose

The purpose of the Localizer is to load the translations into the Language model, it will also set the current language for the application.

5.28.3 Function

The localizer will populate the translations in the Language model and then provide an interface for switching language.

```
// Keeps track of languages
private Language languageModel;

// Loads all strings into the language model
private void LoadTranslations();

// Changes the language
public void SetLanguage(Language.Type language);
```

5.28.4 Subordinates

There are no subordinates to this component.

5.28.5 Dependencies

The Localizer depends on the existance of the Language module.

5.28.6 Interfaces

Since the application will only be supporting one language at first, the only one interfacing against this module is the initializer which will set the language supported by the application.

Caller/Callee	Header
5.25. Initializer	`public void SetLanguage(Language.Type language);`

5.28.7 Resources

This component requires no resources, except for perhaps a language file.

5.28.8 References

There are no references for this component.

5.28.9 Processing

The Localizer will either...

1. ... have a collection of hardcoded strings to populate the Language model.
2. ... load the strings from a file and populate the Language model.

What implementation will be used depends on the amount of time and ambition put into this module.

5.28.10 Data

Data	Structure	Description
No data is stored in this component.		

5.29 Victor Turner

5.29.1 Type

The Victor Turner is a **module** and belongs to the **controller** part of the MVC.

5.29.2 Purpose

The purpose of the Victor Turner is to see which player's turn it is and whether or not any one players victory conditions are met. The Victor Turner passes on control to various other controllers.

The Victor Turner fulfills requirements SR1.4 and SR1.15.

5.29.3 Function

Upon its creation it allows players to act once each in a sequence and checks if any player has fulfilled its victory condition. Once all players have acted, the Victor Turner will initiate such a state that all game entities will be allowed to act upon the directives given during the players' turns.

```
public void Run();

private boolean HasWon(Player player);

private void PlayerAct(Player player);
private void WorldAct();
```

5.29.4 Subordinates

This component has no subordinates.

5.29.5 Dependencies

- 5.18. World
- 5.12. Player
- 5.7. Graph
- 5.34. World Controller
- 5.23. Unit Controller
- 5.22. Graph Controller

5.29.6 Interfaces

Caller/Callee	Header
Game Initializer	`public void VictorTurner();`
World Controller	`public WorldController(Player);`
Graph Controller	`public CalculateWeights();`
AI Player	`public void AIPlayer();`
Unit Controller	`public void UnitController();`
World	`public void RemovePlayer(Player player);`
Player	`public Graph[] GetGraphs();`

5.29.7 Resources

This component uses no resources.

5.29.8 References

This component needs no references.

5.29.9 Processing

This component does not perform any processing. It only lets players take turns and allows them to act. Once all players have finished moving, the world is allowed to act upon the players'

instructions.

```
void Run()
{
    while(!finished)
    {
        foreach (Player p in players)
        {
            if(HasLost(p))
            {
                world.RemovePlayer();
            }
            if(HasWon(p))
            {
                finished = true;
                break;
            }
            WorldController(p);
        }
        UnitController();
        GraphController.CalculateWeights();
    }

}

bool HasLost(Player p)
{
    if(p.GetGraphs() == 0)
    {
        return true;
    }
    else
    {
```

```
            return false;
        }
    }

    bool HasWon(Player p)
    {
        if(world.GetPlayers().Length()==1)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    void PlayerAct(Player p)
    {
        PlayerController(p);
    }
```

5.29.10 Data

Data	Structure	Description
players	`Player[]`	The players in the world.
world	`World`	A model of the world.
finished	`boolean`	Whether or not the game has ended.

5.30 Menu Controller

5.30.1 Type

This component is a **class** belonging to the **controller** part of MVC.

5.30.2 Purpose

The menu controller governs all menu logic in the game. It implements SR1.9.

5.30.3 Function

This component controls the different menus of the game. It uses templates for each menu type (Main menu template, Options template and the Radial menu template [2]) as a base and then loads menu data from the 5.10. Menu Model . It also listens to the 5.32. Tobii Controller and changes the data model accordingly, as well as sends display data to the 5.37. Menu View .

5.30.4 Subordinates

This component has no subcomponents.

5.30.5 Dependencies

- 5.10. Menu Model
- 5.37. Menu View

5.30.6 Interfaces

The component gets input from the 5.32. Tobii Controller in the form of an enumeration reference to a region. It reads the 5.10. Menu Model at the appropriate place, and generates a menu from the gathered information (which template is being used, what buttons belong to

[2]The menu used for building and order selection, see SRD figure 11

this instance and if anything has been clicked) and writes the new result to the "current" menu instance.

5.30.7 Resources

This component has no resource requirements.

5.30.8 References

This component has no references.

5.30.9 Processing

```
class MenuController
{

    private static MenuModel mm;

    public static void Initiate(MenuModel mm)
    {
        this.mm = mm
    }



    public static void LoadMenu(Menu m);
    {
        TobiiController.UnloadMenu();
        mm.Push(m);
        TobiiController.LoadMenu(m);
    }

    public static void UnloadMenu()
    {
```

```
        TobiiController.UnloadMenu();
        mm.Pop();
        TobiiController.LoadMenu(mm.Peek());
    }

    public static GUIRegion GetInput()
    {
        return TobiiController.GetActiveRegion();
    }

    public static void ClearMenu()
    {
        mm.Clear();
        TobiiController.UnloadMenu();
    }
}
```

5.30.10 Data

Data	Structure	Description
Menu Model	`MenuModel`	An instance of MenuModel.

5.31 Saver

5.31.1 Type

The Saver component is a **class** belonging to the **controller** part of MVC.

5.31.2 Purpose

The purpose of the Saver component is to fulfill the requirement SR1.12 regarding the function to save a game to be loaded later.

5.31.3 Function

The function of this component is to save a current game to a file on the harddrive. It will write the current state of the world to the save file.

5.31.4 Subordinates

There are no subordinates to this component.

5.31.5 Dependencies

The following things must be available in order for the Saver to function.

- 5.18. World
- 5.34. World Controller

5.31.6 Interfaces

Caller/Callee	Header
World Controller	`SaveWorld(string saveFilename, World theWorld)`

5.31.7 Resources

There are no resources needed for the Saver to function.

5.31.8 References

There are no required references for the Saver.

5.31.9 Processing

```
public void SaveWorld(string saveFilename, World theWorld)
{
    File saveFile = File.Open(saveFilename)

    saveFile.Write(theWorld)
}
```

5.31.10 Data

Data	Structure	Description
No data is stored in this component.		

5.32 Tobii Controller

5.32.1 Type

The Tobii Controller component is an **interface class**.

5.32.2 Purpose

This components only purpose is to provide an interface for eye tracking data and use it to interact with the GUI Region component. It realizes SR5.1.

5.32.3 Function

The Tobii Controller component provides functions to simplify access to Tobii eye tracking data as much as possible in order to make it convenient for us to implement the GUI Region component to our game.

5.32.4 Subordinates

This component has no subordinates.

5.32.5 Dependencies

- 5.8. GUI Region
- 5.34. World Controller

5.32.6 Interfaces

Caller/Callee	Header
World Controller	GUIRegion GetRegion()

5.32.7 Resources

The Tobii Controller will depend on the external interface Tobii Eye Control Suite.

5.32.8 References

This component will require an understanding of the Tobii SDK.

5.32.9 Processing

This component will process requests for eye-tracking information or otherwise generate events when the users gaze enters a certain region.

5.32.10 Data

Data	Structure	Description
No data is stored in this component.		

5.33 World Generator

5.33.1 Type

The World Generator component is a **class** belonging to the **controller** part of MVC.

5.33.2 Purpose

The purpose of the World Generator is to fulfill the requirements SR1.7 and SR1.14 regarding the world map in the game.

5.33.3 Function

The function of this component is to initiate the game world. The grid map will be randomly generated and to provide a method to play same map again the random seed used will be accessible. It can either be called with a specified seed or a random one.

5.33.4 Subordinates

There are no subordinates to this component.

5.33.5 Dependencies

The following things must be available in order for the World Generator to function.

- 5.18. World
- 5.16. Tile

5.33.6 Interfaces

Caller/Callee	Header
5.26. Game Initializer	`public void GenerateWorld(long mapSeed);`

5.33.7 Resources

- A random number generator, which is included in C#.

5.33.8 References

This component has no references.

5.33.9 Processsing

The component generates a grid of tiles to be used in the game. Here follows some suggestions on the algorithm to implement the functions.

```
void GenerateWorld(long mapSeed){

    Tile[][] theMap := GenerateMapFromSeed(mapSeed)
    World.SetMap(theMap)
}



World GenerateMapFromSeed(long mapSeed){
    Tile[][] returnMap =
    Tile[Random.Next(mapSeed)%MAXIMUM + MINIMUM]
    [[Random.Next(mapSeed)%MAXIMUM] + MINIMUM]
    //Create the grid with a size specified by the random number,
    //to limit the map size the MAXIMUM and MINIMUM is a specified number
    for each tile in map{
        int randomNumber = Random.Next(mapSeed)

        switch(randomNumber){
            case 1:
                tile = Tile(TerrainType.one)
            case 2:
                tile = Tile(TerrainType.two)
```

```
            ...
            case n:
                tile = Tile(TerrainType.n)
        }
    }
}
```

5.33.10 Data

Data	Structure	Description
mapSeed	`long`	The random generated seed to use
world	`World`	The world to modify.
map	`Tile[][]`	The map generated.

5.34 World Controller

5.34.1 Type

The World Controller is a **module** and belongs to the **controller** part of the MVC

5.34.2 Purpose

The purpose of this component is to control the entire world. It is part of the realization of SR1.7.

5.34.3 Function

This component will continously ask the Tobii controller for input from the user, or more specifically, which GUI Region the user is looking at or has activated, if any. After recieving this information, the world controller issues tasks to the relevant controllers, possibly itself, to modify their respective models in some way.

This controller may decide that it is itself responsible for processing the event, it will do so if the activated GUI Region is one that wants to scroll the viewport, change the view point, or one that is related to the modification of Tile components.

5.34.4 Subordinates

This component has no subordinates.

5.34.5 Dependencies

- 5.32. Tobii Controller
- 5.20. Building Controller
- 5.30. Menu Controller
- 5.29. Victor Turner
- 5.8. GUI Region

5.34.6 Interfaces

Caller/Callee	Header
5.29. Victor Turner	`Start(Player player)`

5.34.7 Resources

No external resources are required for this component.

5.34.8 References

No references are required for this component.

5.34.9 Processing

This component will upon recieving information from the Tobii controller about which GUI Region has been activated, either pass control over to another controler component, which will then have access to said GUI Regions `onActivate`-function, or handle the event by itself.

```
public void Start(Player player)
{
    switch(TobiiControler.GetRegion().type)
    {
        case x:
            pass control to controller x
        break;
        .
        .
        .
    }
}
```

5.34.10 Data

Data	Structure	Description
No data is stored in this component.		

5.35 AI View

5.35.1 Type

The AI View component is a **class** belonging to the **view** part of MVC.

5.35.2 Purpose

The purpose of the AI View is to feed information about the game world to the AI Player class. This is according to SR1.1, SR1.4, SR1.15 and SR1.19.

5.35.3 Function

This component is a view, and will therefore subscribe to events in one parts of the model

`Tile[][] GetVisibleMap()`
Returns a Tile-representation of the world map, containing information that the AI Player is allowed to see.

5.35.4 Subordinates

The AI View has no subordinates.

5.35.5 Dependencies

The following components must be available for the AI View to function:

- 5.18. World

5.35.6 Interfaces

Caller/Callee	Header
5.19. AI Player	`Unit[] GetMyUnits()`
5.19. AI Player	`Building[] GetMyBuildings()`
5.19. AI Player	`Tile[][] GetVisibleMap()`

5.35.7 Data

Data	Structure	Description

No data is stored in this component.

5.36 Graphic Rendering

5.36.1 Type

This component is a **class** and is our graphics interface to the XNA render-engine.

5.36.2 Purpose

The purpose of the graphic rendering class is to render the current screen utilizing the model classes involved.

5.36.3 Function

The Graphic renderer is the only class called upon in the main game thread's Draw() call and draws everything onscreen.

```
public void DrawScreen(SpriteBatch spriteBatch);
```

5.36.4 Subordinates

This component has no subcomponents.

5.36.5 Dependencies

- 5.14. Sprite Texture Map which contains all textures to be rendered.
- 5.39. World View which contains the world tile information.
- All drawable game units which contains their position as well as their texture.
- 5.37. Menu View which contains the GUI information with menus and their positions.

5.36.6 Interfaces

Caller/Callee	Header
Main.Draw()	`public void DrawScreen(SpriteBatch);`

5.36.7 Resources

5.14. Sprite Texture Map contains all textures to be rendered.

5.18. World contains the world tile information.

All drawable game units contain their position as well as their texture.

5.36.8 References

This component has no references.

5.36.9 Processing

The component first decides which units are to be rendered by creating a bounding rectangle with the dimensions and position of the screen, checking each unit to see if their sprite intersects with the rectangle. It then gets that units position and texture and renders it to the screen with the Z-ordering decided by its type. It also renders the UI and any other screens present in the game, such as the menus.

5.36.10 Data

Data	Structure	Description
No data is stored in this component.		

5.37 Menu View

5.37.1 Type

The Menu View component is a **module** belonging to the **view** part of MVC.

5.37.2 Purpose

As described in the software requirement SR3.4 the menu should enable the player to change options and issue various commands to the game. The purpose of the Menu View component is to generate the information needed for 5.36. Graphic Rendering to render the different menus in the game.

5.37.3 Function

The Menu View keeps track of the active menus, and gives 5.36. Graphic Rendering everything it needs to render the menus correctly withthe `GetDrawables()`-function.

Any changes in MenuManager will immediately be reflected in the Menu View as it follows the MenuEvent in 5.11. Menu Manager .

```
public DrawData GetDrawables();
```

5.37.4 Subordinates

This module has no subordinates.

5.37.5 Dependencies

- 5.11. Menu Manager
- 5.10. Menu Model

5.37.6 Interfaces

Caller/Callee	Header
5.36. Graphic Rendering	`public DrawData GetDrawables();`

5.37.7 Resources

This module has no need for external resources.

5.37.8 References

This model does not require any references.

5.37.9 Processing

```
public DrawData GetDrawables()
```

The GetDrawables function will return the all drawable parts needed by the 5.36. Graphic Rendering module.

5.37.10 Data

Data	Structure	Description
No data is stored in this component.		

5.38 Music Player

5.38.1 Type

This component is a **class** belonging to the **view** part of MVC.

5.38.2 Purpose

As described in the software requirement SR1.18 the game will have a functioning music player.

5.38.3 Function

Implements functionality of the MediaPlayer class in XNA, which allows the game to play music. Additional methods allows the programmer to enable users to mute the music, create playlists, change the volume and other similar functions. The functionality that this game will initially implement is the ability to mute the game (SR1.18) and possibly to change the volume.

```
// Functionality
public void ChangeSong(Globals.Songs song);
public void ToggleMute();

// Accessors get/set
public float Volume;
```

5.38.4 Subordinates

This component has no subcomponents.

5.38.5 Dependencies

This component has no dependencies.

5.38.6 Interfaces

The music player is controlled by the player from the 5.6. Game Options view.

Caller/Callee	Header
World Controller	`public void ChangeSong(Globals.Songs song);`
Game option	`public void ToggleMute();`
Game option	`public void Volume(value);`
Game option?	`public float Volume();`

5.38.7 Resources

5.6. Game Options is used for volume control. 5.13. Sounds is used to find an appropriate song handle.

5.38.8 References

This component has no references.

5.38.9 Processing

When the ChangeSong method is called, the component will access the 5.13. Sounds component to retrieve the handle for whatever song is to be played by use of the Globals.Songs enumeration.

```
public void ChangeSong(Globals.Songs song)
{
    Sounds.LoadSound(song).Play();
}
```

The MediaPlayer class that is used by the music player has a boolean state that determines if it is muted or not, this will be manipulated by the ToggleMute method.

A C# get/set accessor “Volume” is available to manipulate the MediaPlayer volume variable. It sets the volume by referencing the value found in the 5.6. Game Options component.

5.38.10 Data

Data	Structure	Description
No data is stored in this component.		

5.39 World View

5.39.1 Type

The World View component is a **component** belonging to the **View** part of MVC.

5.39.2 Purpose

The purpose of the world view is to provide the necessary data to render the game screen as described in the SRD 3.3. It also stores the information of the game state available to the player.

5.39.3 Function

The world view will store the parts of the world model which is available to the player. It also provides 5.36. Graphic Rendering with the DrawData needed to render a representation of the game world to the player.

```
// Properties
public Tile[][] GetCurrentMap();
public DrawData GetDrawables();
public Rectangle GetScreen();

public void UpdateScreen();
public void UpdateMapMatrix();
```

5.39.4 Subordinates

The World View has no subordinates.

5.39.5 Dependencies

- 5.18. World

- 5.16. Tile

5.39.6 Interfaces

Caller/Callee	Header
Menu Controller	`public Tile[][] GetCurrentMap();`
Graphic Rendering	`public DrawData GetDrawables();`
Menu Controller	`public Rectangle GetScreen();`
Menu View	`public void UpdateScreen();`
Menu View	`public void UpdateMapMatrix();`

5.39.7 Resources

This module has no need for external resources.

5.39.8 References

This model does not require any references.

5.39.9 Processing

```
// Properties
public Tile[][] GetCurrentMap();
public DrawData GetDrawables();
public Rectangle GetScreen();

public void UpdateScreen();
public void UpdateMapMatrix();
```

The get functions will be used to access data, these will be called by 5.36. Graphic Rendering and 5.34. World Controller . The update functions will be called by World View to update the the game world seen by the player.

5.39.10 Data

Data	Structure	Description
currentMap	`Tile[][]`	The map as seen by the player.
currentScreenPosition	`Rectangle`	The current screen bounding rectangle.

5.40 Debugger

5.40.1 Type

This component is a **ulility module**.

5.40.2 Purpose

The component is used as a tool for debugging the program, and contains useful methods to display internal states as well as different kinds of stress test code. Implements SR2.1, SR2.2, SR11.2, SR11.3, SR12.1, 12.4.

5.40.3 Function

The component will provide methods for debugging and testing parts of the program not reachable by conventional unit tests in the form of logging utilities and perhaps a console for developers.

5.40.4 Subordinates

This component has no subordinates.

5.40.5 Dependencies

This component is a utility class, and as such has no dependencies.

5.40.6 Interfaces

Since no directly testable methods will be defined before the implementation phase, they cannot be listed here.

5.40.7 Resources

This component has no resources.

5.40.8 References

This component has no references.

5.40.9 Processing

This module will process data from various sources and output them to different targets, such as a file, a console or standard out.

5.40.10 Data

Data	Structure	Description
No data is stored in this component.		

5.41 Main

5.41.1 Type

This component is a **static class**.

5.41.2 Purpose

Governs the main game loop and acts as the primary conduit between the application and XNA.

5.41.3 Function

Implements the primary Update() and Draw() methods which call upon all other main loops.

5.41.4 Subordinates

The Main component does not have any subordinates.

5.41.5 Dependencies

This component has no dependencies.

5.41.6 Interfaces

The main game thread calls upon all of the controllers' update methods, the sound engine and the Graphics renderer.

5.41.7 Resources

This component has no resources.

5.41.8 Processing

The two active threads are the Update:

```
override public void Update(GameTime gameTime)
{
    Update all control components();
    Update the sound engine();
    base.Update();
}

override public void Draw(GameTime gameTime)
{
    Call the graphics renderer();
}
```

5.41.9 Data

Data	Structure	Description
graphics	`GraphicsDeviceManager`	Macro for the graphicsdevicemanager in the game.
device	`GraphicsDevice`	The device associated with the game.
spriteBatch	`SpriteBatch`	A tool for quick consecutive rendering of sprites in XNA.
tobiiController	`TobiiController`	The class which we get input from
mouse/lastMouse	`MouseState`	two variables used for debugging the tobii hardware related methods.
keyboard/lastKeyboard	`KeyboardState`	two variables used for debugging
AllComponents	`Component`	All the components used in the game (one variable for each component)

This page intentionally left blank.

6 Feasability and Resource Estimates

6.1 Prioritization and feasibility study

Models

Component	Estimate	Group member	Priority
5.1. Building	6 hours	John Forsberg	Economy
5.2. Aggressive Building	2 hours	Viktor Eklund	Standard
5.3. Barrier Building	2 hours	Viktor Eklund	Standard
5.4. Base Building	2 hours	Viktor Eklund	Economy
5.5. Resource Building	2 hours	Viktor Eklund	Standard
5.6. Game Options	24 hours	Carl-Oscar Erneholm	Standard
5.7. Graph	6 hours	Martin Nycander	Economy
5.8. GUI Region	6 hours	Mattias Mikkola	Economy
5.9. Language	3 hours	Joel Ahlgren	Standard
5.10. Menu Model	48 hours	Carl-Oscar Erneholm	Economy
5.11. Menu Manager	3 hours	Carl-Oscar Erneholm	Economy
5.12. Player	6 hours	John Forsberg	Economy
5.13. Sounds	8 hours	Mattias Mikkola	Deluxe
5.15. Terrain Type	3 hours	Joel Ahlgren	Standard
5.14. Sprite Texture Map	6 hours	John Forsberg	Economy
5.16. Tile	6 hours	Joel Ahlgren	Economy
5.17. Unit	8 hours	Joel Ahlgren	Standard
5.18. World	6 hours	Marco Ahumada Juntunen	Economy

Controllers

Component	Estimate	Group member	Priority
5.19. AI Player	24 hours	Lukas Mattsson	Standard
5.20. Building Controller	12 hours	Marco Ahumada Juntunen	Economy
5.21. Configurator	4 hours	Mattias Mikkola	Standard
5.22. Graph Controller	10 hours	Martin Nycander	Economy
5.23. Unit Controller	24 hours	Marco Ahumada Juntunen	Economy

continued on next page

continued from previous page

Component	Estimate	Group member	Priority
5.24. Unit Accountant	12 hours	Joel Ahlgren	Economy
5.25. Initializer	1 hour	Fredrik Lindh	Economy
5.26. Game Initializer	6 hours	Fredrik Lindh	Economy
5.27. Loader	10 hours	Fredrik Lindh	Deluxe
5.28. Localizer	6 hours	Joel Ahlgren	Standard
5.29. Victor Turner	6 hours	John Forsberg	Economy
5.30. Menu Controller	6 hours	Carl-Oscar Erneholm	Economy
5.31. Saver	10 hours	Fredrik Lindh	Deluxe
5.32. Tobii Controller	24 hours	Mattias Mikkola, Viktor Eklund	Economy
5.33. World Generator	6 hours	John Forsberg	Economy
5.34. World Controller	8 hours	Marco Ahumada Juntunen	Economy

Views

Component	Estimate	Group member	Priority
5.35. AI View	5 hours	Lukas Mattsson	Standard
5.37. Menu View	6 hours	Carl-Oscar Erneholm	Economy
5.38. Music Player	8 hours	Fredrik Lindh	Deluxe
5.39. World View	24 hours	Marco Ahumada Juntunen	Economy

Other tasks

Component	Estimate	Group member	Priority
Produce sprite files	48 hours	Carl-Oscar Erneholm, Viktor Eklund	Economy
Produce sound files	48 hours	Marco Ahumada Juntunen, Viktor Eklund	Deluxe
5.41. Main	24 hours	Fredrik Lindh	Economy
5.36. Graphic Rendering	16 hours	Fredrik Lindh	Economy
5.40. Debugger	8 hours	Martin Nycander	Standard

6.2 Risks

The biggest risk this project faces is proving unable to provide a satisfactory Economy release by deadline. We have about four half-time weeks to complete the project and there is very little space for errors.

We are going to tackle this risk by carefully reviewing the schedule and allocation of project tasks continuously through-out the implementation phase. Since we have no real experience of projects by this size, we can not contain the risks at this stage.

6.3 Schedule

The schedule of this project is divided into three "sprints". These are periods of time with their own deadlines and sets of goals. The sprints can be observed in the following table.

Member	**Sprint 1**	**Sprint 2**	**Sprint 3**
Joel	5.15. Terrain Type	5.9. Language	5.28. Localizer
	5.16. Tile	5.24. Unit Accountant	
	5.17. Unit		
Marco	5.34. World Controller	5.20. Building Controller	Produce sound files
	5.18. World	5.23. Unit Controller	
	5.39. World View		
Carl-Oscar	5.6. Game Options	5.37. Menu View	Produce sprite files
	5.10. Menu Model	5.11. Menu Manager	
	5.30. Menu Controller		
Viktor	5.32. Tobii Controller	5.2. Aggressive Building	Produce sprite files
		5.3. Barrier Building	Produce sound files
		5.4. Base Building	
		5.5. Resource Building	
John	5.33. World Generator	5.12. Player	5.29. Victor Turner
		5.14. Sprite Texture Map	
Fredrik	5.25. Initializer	5.38. Music Player	5.27. Loader
	5.36. Graphic Rendering	5.26. Game Initializer	5.31. Saver
	5.41. Main		
Lukas	5.19. AI Player	5.19. AI Player	5.19. AI Player
		5.35. AI View	
Mattias	5.32. Tobii Controller	5.21. Configurator	5.13. Sounds

continued on next page

continued from previous page

Member	**Sprint 1**	**Sprint 2**	**Sprint 3**
	5.8. GUI Region		
Martin	5.40. Debugger	5.22. Graph Controller	
	5.7. Graph		

The deadlines of the individual sprints will be scheduled according to the current state of the projects members calendars.

7 Traceability Matrix

Component	SR
5.1. Building	SR1.3 Buildings
5.2. Aggressive Building	SR1.3 Buildings
5.3. Barrier Building	SR1.3 Buildings
5.4. Base Building	SR1.3 Buildings
5.5. Resource Building	SR1.3 Buildings, SR1.10 Resources
5.6. Game Options	SR1.8 Options
5.7. Graph	SR1.2 Autonomous Units
5.8. GUI Region	SR5.1 Eye-tracking
5.9. Language	N/A
5.10. Menu Model	SR1.13 Shutdown, SR3.1 Eye Resting Area, SR3.2 Help in Menu, SR3.3 Credit List, SR3.4 Menu, SR8.1 In-game help
5.11. Menu Manager	SR3.4 Menu
5.12. Player	SR1.16 Unit Upgrades
5.13. Sounds	SR1.18 Music Player
5.15. Terrain Type	SR1.14 Terrain Type
5.14. Sprite Texture Map	SR1.3 Buildings, SR1.2 Autonomous Units
5.16. Tile	SR1.7 Map, SR1.5 Fog of war
5.17. Unit	SR1.2 Autonomous Units
5.18. World	SR1.1 Opponent A.I., SR 1.2 Autonomous Units, SR1.3 Buildings, SR1.7 Map, SR1.10 Resources
5.19. AI Player	SR1.1 Opponent A.I., SR1.4 Conquest Victory, SR1.15 Turns, SR1.19 Combat Requirement
5.20. Building Controller	SR1.3 Buildings
5.21. Configurator	N/A
5.22. Graph Controller	SR1.2 Autonomous Units
5.23. Unit Controller	SR 1.2 Autonomous Units
5.24. Unit Accountant	SR 1.2 Autonomous Units
5.25. Initializer	SR4.1 Microsoft .NET, SR4.2 Tobii Control Suite, SR5.2 Operating System, SR5.3 Platform

continued on next page

continued from previous page

Component	**SR**
5.26. Game Initializer	N/A
5.27. Loader	SR1.6 Load
5.28. Localizer	N/A
5.29. Victor Turner	SR1.4 Conquest Victory, SR1.15 Turns
5.30. Menu Controller	SR1.9 Pause, SR1.11 Resume, SR3.4 Menu
5.31. Saver	SR1.12 Save
5.32. Tobii Controller	SR5.1 Eye-tracking
5.33. World Generator	SR1.7 Map, SR1.14 Terrain Type
5.34. World Controller	SR1.7 Map, SR1.14 Terrain Type
5.35. AI View	SR1.1 Opponent A.I., SR1.4 Conquest Victory, SR1.15 Turns, SR1.19 Combat Requirement
5.36. Graphic Rendering	N/A
5.37. Menu View	SR1.9 Pause, SR1.11 Resume, SR3.4 Menu
5.38. Music Player	SR1.18 Music player
5.39. World View	SR1.7 Map, SR1.14 Terrain Type
5.40. Debugger	SR2.1 RAM
5.40. Debugger	SR2.2 GPU
5.40. Debugger	SR11.2 Performance monitoring
5.40. Debugger	SR11.3 Stress Test
5.40. Debugger	SR12.1 Frames per second
5.40. Debugger	SR12.4 Uptime
5.41. Main	N/A

8 List of Components

A Meeting Notes

A.1 2010-03-09

Möte i D34 $10:35 \rightarrow 11:50$

A.1.1 Mötets högtidliga öppnande

Närvarande

- Martin Nycander
- Carl-Oscar Erneholm
- John Forsberg
- Mattias Mikkola
- Viktor Eklund
- Lukas Mattson
- Marco Ahumada Juntunen
- Joel Ahlgren

Frånvarande

- Fredrik Lindh

A.1.2 Kommunikation

Funkar bra

A.1.3 Grupper

CAM Vi har inga frågor till Tobii, vi har inte hört något från dom.

A.1.4 Förra mötet

SRD-todo Gruppen lyckades klämma ut tillräckligt mycket jobb de sista dagarna innan inlämningen.

SRD-presentation Gick väldigt bra enligt dom som är där.

A.1.5 ADD

Genomgång

- 1. Introduction – Liknande tidigare
- 2. System overview – Context, system design
- 3. System Context – Diagrams, interfaces
  - n. External interface definitions – Tobii, XNA, C12
- 4. System design – UML, designtekniker?
  - 1. Design method
  - 2. Decomposition description – Diagrams, top-level
- 5. Component design – Template
  - n. Component identifier
    - n.1 Type
    - n.2 Purpose (SRD)
    - n.3 Function (parameters, return type, logic)
    - n.4 Subordinates (subcomponents)
    - n.5 Dependencies (pre-conditions)
    - n.6 Interface (data to-from)
    - n.7 Resources (XNA, C12, etc?)
    - n.8 References (dokumentation)
    - n.9 Processing (inre dataflöde)

    - n.10 Data (data structures)

- 6. Feasability & Resource estimate – Priorities, future project tasks, task flowchart, effort for each task, table, risk
- 7. Matrix

A.1.6 Övriga

Folk som inte har något att göra gör följande:

- Project handbook
- ADD-slides
- PSS-05 ADD

A.1.7 Marcos referensmetod

Marco går igenom hur vi ska göra definitioner i LaTeXnumera: \defn{}{}.

A.2 2010-03-16

Möte i 1635 $10:40 \rightarrow 12:00$

A.2.1 Mötets högtidliga öppnande

Närvarande

- Martin Nycander
- John Forsberg
- Viktor Eklund
- Lukas Mattson
- Joel Ahlgren
- Fredrik Lindh
- Mattias Mikkola
- Marco Ahumada Juntunen

Frånvarande

- Carl-Oscar Erneholm

A.2.2 Allmänt/Kommunikation

Funderingar över juridiska dokument/kontrakt uppkommer, inget bestäms. Kommunikationen verkar bra.

A.2.3 Grupper

CAM Inget att rapportera.

A.2.4 ADD

Påläsningsgruppen Viss aktivitet, PSS-05-standarden visar sig tung att läsa.

Introduction (Mattias) Inget har hänt.

System Overview (Fredrik) Lite text skriven, beror mycket på två kommande delarna.

System Context (Marco) Inget har hänt.

External/Internal Definitions (Joel) Inget har hänt.

System Design (Martin, John) Början till datamodel finns.

A.2.5 Förra mötet

Marco's referens-H4X Mattias sätter sig och knåpar ihop det som behövs i LaTeX-mallen.

A.2.6 ADD-kött!

Arbetsmöten planeras in på fredag, söndag samt måndag.

A.2.7 MVC-modell

Controller	**Model**	**View**
Main (Fredrik)	World (Lukas)	Graphic-rendering (Fredrik)
Localizer (Martin)	Player (Lukas)	Media player (Fredrik)
Configurator (Joel)	Tile (Viktor)	Music player (Fredrik)
unitBrain (Marco)	Unit (Marco)	Abstract view (CO)
Turner (Marco)	Building (Martin)	MenuView < AbstractView (CO)
Unit accountant (Marco)	Building type (Martin)	PromptView < MenuView (CO)
Building constructor (Martin)	Terrain type (Viktor)	GameView < AbstractView (CO)
Grapher (Lukas)	Game option (Joel)	HelpView < AbstractView (CO)
World loader (Lukas)	UI region (Viktor)	
Tobii input translator (Viktor)	Texture (Fredrik)	
AI Player (Lukas)	Graph (Marco)	

A.3 2010-03-23

Möte i 1635 $10:40 \rightarrow 11:05$

A.3.1 Mötets högtidliga öppnande

Närvarande

- Martin Nycander
- John Forsberg
- Viktor Eklund
- Lukas Mattson
- Joel Ahlgren
- Marco Ahumada Juntunen

Frånvarande

- Mattias Mikkola
- Fredrik Lindh
- Carl-Oscar Erneholm

A.3.2 Kommunikation

Inga åsikter

A.3.3 Rollbesättning

A.3.4 Grupper

CAM Inget att rapportera.

ADD Gruppen har gjort en kraftansträngning under natten, ADD'n är uppe i över 130 sidor nu.

A.3.5 Spelnamn

4 mot 2 anser att "Re:CELLection – CELLdomly boring" (som röstades fram som bästa förslag 2010-02-09) är en dugligt namn.

A.3.6 Redovisning av ADD

Händer senare, skjuts upp.

A.3.7 ADD todo

- Matrisen behöver uppdateras
- Schemat för arbetsfasen behöver uppdateras
- Unit Accountant behövs
- Help View behöver avslutas
- Dagens mötesprotokoll ska in i appendix