Contents

[1. Project overview. 2](#_Toc69330364)

[1.1. Executive Summary. 2](#_Toc69330365)

[1.2. Project Goals 2](#_Toc69330366)

[1.3. Development Methodology. 3](#_Toc69330367)

[2. Project Analysis. 3](#_Toc69330368)

[2.1. Colony Ruler requirement analysis. 3](#_Toc69330369)

[2.1.1. Functional requirements 3](#_Toc69330370)

[2.1.2. Non-functional requirements 5](#_Toc69330371)

[2.1.3. External tools requirements. 5](#_Toc69330372)

[2.2. Requirements Modelling. 5](#_Toc69330373)

[3. Project Design and Implementation. 6](#_Toc69330374)

[3.1. Architecture Design. 6](#_Toc69330375)

[3.2. Detailed Design. 8](#_Toc69330376)

[3.2.1. Class Diagram. 8](#_Toc69330377)

[6. References. 10](#_Toc69330378)

[References 10](#_Toc69330379)

# Project overview.

## Executive Summary.

The colonisation ship crashed upon landing on an alien planet. Not everyone survived, and all tools for survival and repairs were destroyed. As a result, the game begins in the stone age, making use of old-fashioned and crude tools, basically starting from scratch.

The main goal is to create a high-level civilization and be able to send another ship to another star or planet. The player is the manager and has access to a window which controls resources, consumption and production.

In each stage you can make new tools which unlocks new ages with new challenges. The faster you expand your base/factory and unlock new ages, the better. After obtaining new technology or unlocking new ages and some other events, new story lines or story line continuations is triggered. The player can also expand their territory by fighting against local planet inhabitants.

As a manger, the player should find the balance between resource consumption and production within the colony. To progress in the game, the player needs to develop new technology and solve all the problems that come with it.

The game is web-based and for playing, making or restoring saves player should login on it or use non-authorized guest account, which available for everyone.

## Project Goals

The goal of this project is producing a web-based game “Colony ruler” and web-site for playing on it with the following features:

1. A managing game about space colonization.
2. A web-site for playing on it.
3. Design, develop and test the game on each phase.
4. Fix all the bugs and do benchmarking for mobile projects.
5. Publish the demo version in the market.

## Development Methodology.

The team has not enough knowledge about the project and the project could have as bugs as non-fixed duration of each phase. It means, requirements could be changed at any time during the project. The next requirement for methodology – it should be iterative, because the full game development process has been split into small phases.

Agile framework methodology fits in both requirements and could be used by the team. However, there is a lot of implementation of this framework. One of them, that will be used by the team is DSDM – Dynamic System Deployment Methodology, further development of RAD methodology. This methodology mostly focused on time and final goals rather than current requirements.

# Project Analysis.

Game requirement could be divided into: development requirements, game designer requirements, 2d artist requirements. However, for the development team the most important is only development requirements.

## Colony Ruler requirement analysis.

## Functional requirements

The game will have the next functionality:

1. Resources
   1. Natural resources, like stones, ores. +
   2. Growing resources, like trees, fish, animals +
   3. Cutting trees increasing the territory, new tree rising cutting the territory.
2. Colony. Main icon for showing the population and colony needs. +
3. Production icon:
   1. Main icon. Should show the production and should be unique.+
   2. Buttons plus/minus for add/delete workers into/from production +
   3. Third button should have the following functionality: upgrade storage space for production/repairing. Learn science subject then it consumes enough resources for it. Demolish buildings +
   4. Product should consume resources, if it needed +.
4. Arrows:
   1. Arrows should start from resource and finish on production.+
   2. Arrows should have the next colors:+
      1. White, if there are no workers on production.+
      2. Yellow, if it is a tool.+
      3. Green, if there are some workers on production and enough resources for production.+
      4. Red, if there are not enough resources for production.+
   3. Arrows should have two modes:
      1. show a full tree of production, started from primary resources and finished on selected icon.+
      2. show a short tree of production, started from resources and finished on selected icon +
5. Storage.
   1. Any production should be strict by some space. +
   2. Small items can increase their storage by small storages like basket or jar.+
   3. Big items needed warehouses.+
   4. Buildings needed territory.+
   5. Territory should not be infinitive +
6. Population/happy
   1. At start, player should have only 10 workers. +
   2. Each production can hire/fire workers. +
   3. Some items can increase happiness. +
   4. If all happiness is more than 75%, a new worker come at the end of the year.
   5. If all happiness below 25% a worker could leave.
   6. Once a year population should be increased:+
      1. on 10%, if there is enough happiness +
      2. on 1 if population below 10. +
7. Science.
   1. Science should consume some resources then it has workers. +
   2. If it consumes all the required resources, third button become available. +
   3. After pressing third button, new production/researches should be opened, this one icon should be deleted. +
8. Tools.
   1. Tools can increase production. +
   2. Production should get tools as much as workers on it and return if some workers leave. +
   3. Tools can be damaged or destroyed during the using. +
   4. Damaged tools should be moved to “repairing tool” production. +
   5. After repairing, they should be returned in the stock. +
9. Buildings
   1. Houses: using for living, can hold happy items +
   2. Warehouses – increasing large items storage +
10. Fields. Can be used for domestic animals and growing food.
    1. Grass field – resource for farm field and animals. Can be cleaned +
    2. Farm field – resource for any planted field. +
    3. Planted field – after some time food can be harvested and became empty. Became grass field after one month.
11. Traps
    1. Placed Trap – two days later became as unverified
    2. Unverified Trap can be: empty(80%), damaged(5%), with dead animal(14%), with alive animal(1%). All traps after validation should be placed in the tools.
12. Domestic animals
13. Tooltips.
    1. Should be shown for any new production type:
       1. Human+
       2. Resource
       3. Production +
       4. Farming
       5. Hunting
       6. Traps
       7. Animals
    2. Can be disabled +
    3. Saving/loading settings +
    4. Outline +
14. Color legend
    1. Color legend for each arrow type.+
    2. Hide/Unhide panel. +
       1. Save/load conditions
15. Extended view
    1. Should be shown for each production, except resources.
    2. Tools should be shown as icons.
       1. Tooltips for icons
       2. Number of used tools
    3. Clear recipe of each production type
16. Log. Should be added for every critical issue:
    1. not enough products,
    2. hungry,
    3. starving,
    4. science finished.
    5. New people arrive.
17. Main menu.
    1. New game: start new game +
    2. Save button: save on the hard drive. No more 10. +
    3. Load button: load from the hard drive. +
    4. Settings button: user settings, including localization. +

## External tools requirements.

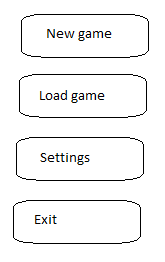
1. Excel.
2. Game should be able to load data from xml files.
3. Unity pre-build event should unload all data from excel to xml files.

# Project Design and Implementation.

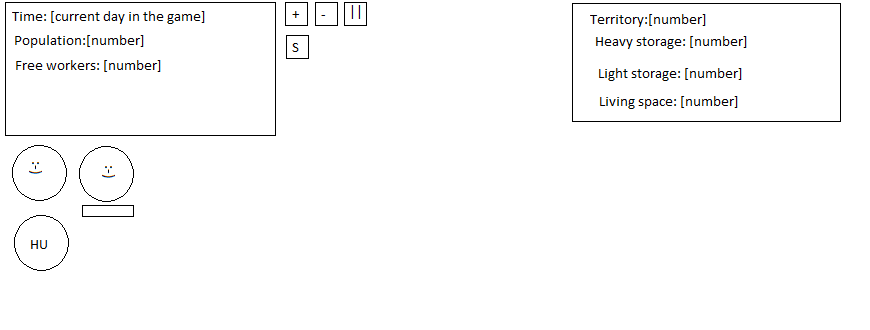
## Architecture Design.

It is a detailed design of the system. There provided concept-view pages, menus and game in two modes: System Interface design as fast-drown concept and User Interface design as mockups.

1. Game “Colony Ruler” – main menu



2. Game “Colony Ruler” – main screen UI.



There “+”, “-” means increase/ decrease time speed, “||” means game pause. S – settings. First from the left corner “;-)” means indicator of how people happy in the colony. The second is happy boost, which will be started after every research or during celebrations. “HU” means “hungry” – indicates that people in the colony is starving and will die soon.

3. Icon design

The game uses the maximum abstract layer and there everything is icons.

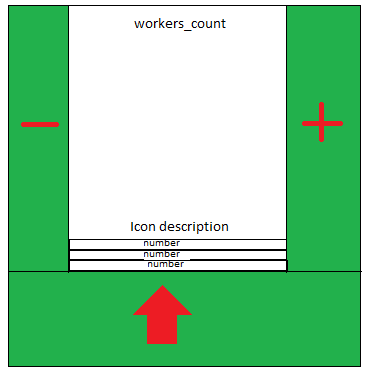
In the icon below “+”, “-” means increase, decrease number of workers on the current productions. Both buttons could be disabled/hidden, depends on icon condition and production type.

Arrow up means “special action”, which depends on the current production and could be “upgrade”, “complete research”, “demolish building”. This button symbol should be specified by production type.

Workers\_count means how many workers has on the current production.

Icon description could be the same as icon name, if current language is English and should be localized into current language.

The next three numbers also is scales and means: “current count”, “productivity” – how many items was produced minus how many items was consumed. The last one used only for tools/buildings and means how many broken items here.



5. Save/Load screen

The difference between both screens is only the name of one button – “save” or “load”



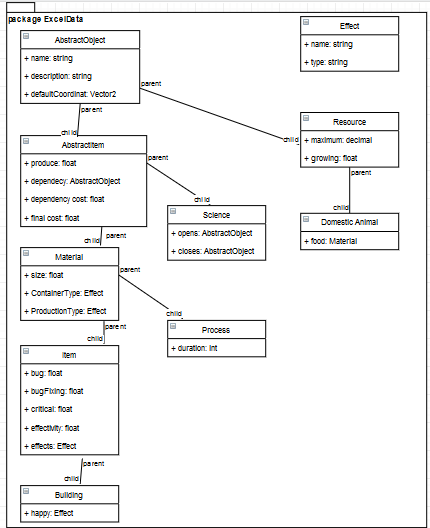
## Detailed Design.

### Class Diagram.

In this document, all classes have been drawn only with important field names and without any methods due to the size of the classes.

1. Excel export classes

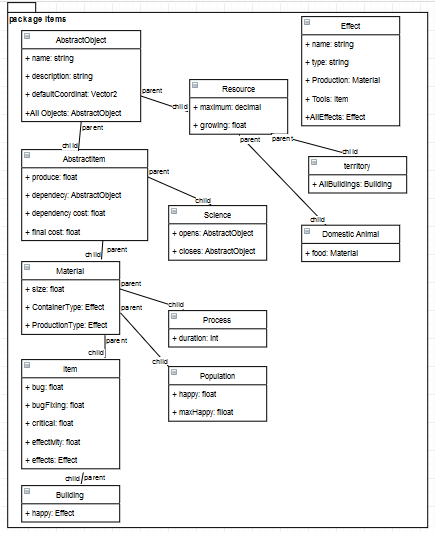
These classes used only for getting data from Excel and does not has any methods. Also, they cannot be changed because they was auto generated from xml schema file.



1. Items in the game.

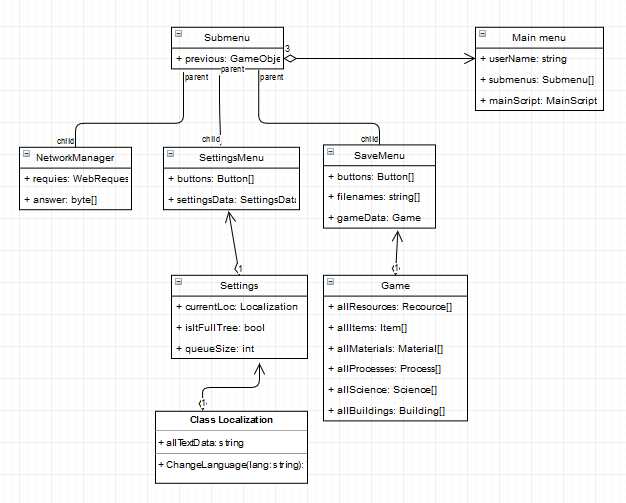
These classes has the next structure: all classes has realized static methods “Load” for loading Excel data and “Parse”. Children classes calls parents method “Parse”. So, only the current class knows how to parse themselves. The full call list for Building looks: Building.Load(string filename) -> Building.Parse(Excel obj) ->Item.Parse(Excel obj) -> Material.Parse(Excel obj) -> AbstractItem.Parse(Excel obj) -> AbstractObject.Parse(Excel obj).

The same thing happens with Copy(AbstractIbject obj) but it was realized throw virtual method Copy.



3. Menus

Menus has the next logic: one main menu, which connected with mainScript and holds all submenus. NetworkManager shows only on while answer does not received. Settings holds Localization object(only one on the game), which contain all localized text data and Save menu(also played Load role) can save/load all game data and implement it.



# References.

# References

D'Aoust, K. (2014). *Unity Game Development.* BIRMINGHAM - MUMBAI: PACKT.

Lammers, K. (2014). *Unity Shaders and Effects Cookbook.* BIRMINGHAM - MUMBAI: PACKT.