Final Report for Chalmersforce

Mystery Inc. (Group 3)

Jonathan Carbol	carbolj
Jennifer Krogh	kroghj
Emma Pettersson	emmp
Adam Rohdell	rohdell
Antonia Welzel	welzel

Object-Oriented Programming Project [TDA367]

Information Technology
Chalmers Institute of Technology
Gothenburg, Sweden
2019-10-25
Version 1.0

Abstract

This document is a compilation of several other reports:

- a **Requirements & Analysis Document**, detailing the planning stages of the project
- a **System Design Document** with information about the project's architecture and design
- a **Peer Review** of another group's project

Contents

1 Requirements & Analysis Document

1.1 Introduction

The aim of the project is to create a 2D Run-and-gun platform game called 'Chalmersforce'. In the game, there is a Player, who has to defeat enemies in order to get the highest score possible.

1.1.1 General characteristics and functionality

When you start a new game, you can choose a name, a character for your player and the difficulty of your game. The game will start of with relatively few enemies, but the number will increase as the game goes on. The enemies in the game have different difficulties, which also affects the number of points a player receives when defeating them. The harder they are to defeat, the more points the player gets. The points are added to a score which is visible on the screen during the whole gameplay, giving the player the opportunity to see how close they are to beating their previous score.

The player can find different types of food in the game, which can both increase or decrease the player's HP, strength or defence. There are no time constraints, but the length of the gameplay is controlled by the number of lives the player has left, which can decrease by for example eating a bad fruit or taking damage from enemies.

After the player has been defeated the score is saved on a local high score list. This can be accessed from the start page which also showcases the 10 current highest scores.

1.1.2 Definitions, acronyms, and abbreviations

HP

Health Points, the player's life. When the points are down to 0, the player is defeated. In the game, it is represented in the form of hearts in the bottom center of the screen, which are red and "full" when the player has the highest possible number of health points and slowly lose colour when a player's health decreases, for example during a fight.

Boost

A boost comes in form of a hat or a weapon. The items boost either the player's strength, defence or its health points depending on what player character the user has chosen in the beginning of the game.

Food

Food can be found at random in the game and will give the player some form of increase or decrease in its HP, defence or strength.

Hat

A Hat gives the player a Boost. Each player character is assigned their own Hat, all with

different benefits.

Weapon

A weapon is an item a player can use to fight enemies in the game. A player is given a weapon at the beginning of the game.

GUI

'GUI' stands for Graphical User Interface

1.2 Requirements

1.2.1 Finished User Stories

User Story
Story ID: 0
Story name: Application start
Description
As a user I want to have a start page, so that I am able to access the different options provided in the game.
Confirmation
☑ There is a start screen visible when the application is opened
☑ Can I access the different options in the game from the start page, like the actual game or a page for the high scores?

User Story
Story ID: 1
Story name: Game Area
Description
As a user I want to see the area where my player can move around, so that I have the opportunity to explore the game.
Confirmation
☑ Platforms on which the player can jump and move around
☑ Background and other aesthetics which imply a playable game area

User Story
Story ID: 2
Story name: Player in the Game
Description
As a user I want to see a character in the game, which I can control, so that I can play the game.
Confirmation
☑ Does a character appear on the screen?
☑ Is it clear that the character is for the player to use?

User Story
Story ID: 3
Story name: Movement of player
Description
As a user I want to be able to move sideways, so I can proceed and explore the game.
Confirmation
☑ Is the character in the game moving right when I use the right arrow key?
☑ Is the character in the game moving left when I use the left arrow key?

User Story
Story ID: 4
Story name: Player's ability to jump
Description
As a user I want to be able to jump while playing the game, in order to access otherwise remote locations.
Confirmation
☑ Can I see the character preforming a jumping motion if I pressed the up arrow key?

User Story
Story ID: 5
Story name: Enemies
Description
As a user I want enemies in the game, so I feel like there's a goal when playing.
Confirmation
☑ Are there other characters, besides the player's character apparent on the screen?
☑ Does the character act unfriendly, for example, do they attack the player?

User Story
Story ID: 6
Story name: Fight enemies
Description
As a user I want to be able to fight my enemies, in order to feel like I have a chance of winning.
Confirmation
☑ Do I have a weapon or other means to use against enemies?
☑ Does the enemy take damage when I use my weapon against them?

User Story
Story ID: 7
Story name: Reward for defeat
Description
As a user I want to be rewarded for defeating my enemies, in order to have a purpose of doing so.
Confirmation
☑ Do I gain points from defeating an enemy?
☑ Are the points collected in a total score?

User Story
Story ID: 8
Story name: Visible score
Description
As a user I want my score to be visible throughout the whole game, to know how close I am to beating my previous highscores.
Confirmation
☑ Is the score visible in the game screen?
☑ Does the score update when I advance in the game?

User Story
Story ID: 9
Story name: Visible HP
Description
As a user I want to see how many lives I have left, to know how much longer I can play the game.
Confirmation
☑ Is the HP visible on the gameplay screen?
☑ Does the HP update if I take damage?

User Story
Story ID: 10
Story name: Items in the game
Description
As a user I want items in the game which can give my character different boosts, so that there is variety and I have more fun playing the game.
Confirmation
☑ Can I see items?
☑ Can I interact with the items?
☐ Is there a visible difference when I have obtained an item?

User Story
Story ID: 11
Story name: Aesthetics in the game area
Description
As a user, I want the levels to have a background and other aesthetics to make my experience more fulfilling.
Confirmation
☑ Is there a background in the game which matches the overall theme of the current game area?
☑ Do platforms match the overall theme of the current area and the background?
☑ Are there animations which make the game feel more immersive?

User Story
Story ID: 12
Story name: High score
Description
As a user I want to be able to look at my high scores, to reflect on my past achievements.
Confirmation
☑ Is the high score from each game saved?
☑ Is there a page with the high scores which I can access?
☑ Are there instructions which guide me to the high score page?

User Story
Story ID: 13
Story name: Pause the game
Description
As a user I want to be able to pause the game, so that I can take a short break from the game.
Confirmation
☑ Is there a button with which I can pause the game?
☑ Does the game continue were I left it when I resume the game?

User Story
Story ID: 14
Story name: Physics
Description
As a user, I want world-like physics applied to the game, such as gravity and collision, in order to make the world feel more natural while moving around.
Confirmation
☑ Is gravity applied to the player and enemies?
☑ Do neither players nor enemies fall through the platforms?

1.2.2 Incomplete User Stories

User Story
Story ID: 0
Story name: Multiplayer Mode
Description
As a user, I want to be able to play the game with multiple players, so that we can have fun together at the same time.
Confirmation
☐ Is there the option to choose a multiplayer mode?
☐ The number of registered players is visible on the screen

User Story
Story ID: 1
Story name: Save interrupted game
Description
As a user I want to be able to save interrupted games, so that I can return and continue playing the same game at another time and not get frustrated at my lost efforts.
Confirmation
☐ Can I leave the game knowing the current game play is saved?
☐ Can I access my interrupted games from the main menu?
☐ Does the saved game continue from the same place I left the game at?

User Story
Story ID: 2
Story name: Player Character
Description
As a user I want to be able to choose a character I identify with, which I can use in the game, in order to make the game immersive.
Confirmation
☑ Can I choose my character among different ones?
☐ Is the character customisable?

User Story
Story ID: 3
Story name: Difficulty Levels
Description
As a user, I want the game to have different difficulty levels, so that I can test my skills and stay interested in the game for longer time.
Confirmation
☑ Are there options provided to the player which lets them choose what difficulty they want to play at?
☑ Is there a visible difference between the different levels, in term of aesthetics and gameplay?

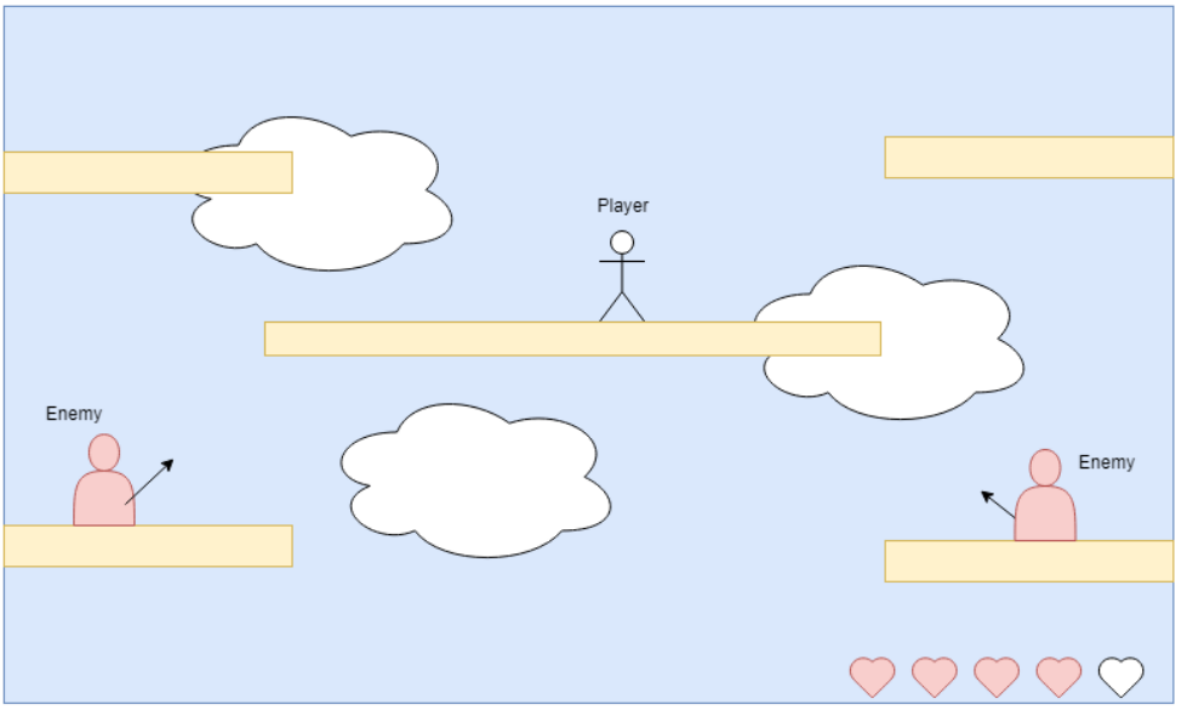
1.2.3 Definition of Done

For a User Story to be defined as **done**, the following should be true for the code:

- It should have been looked over and refactored if necessary
- It should be well documented with JavaDoc, and additional comments if necessary
  - Public classes should have a description, a list of @authors, and the current @version
  - Public methods should have a description, a list with explanations for @parameters, and @return values
- All public methods should be well tested using JUnit

1.2.4 User interface

The image below illustrates the first sketch of the user interface, when the overall look of the game had been decided on. It features a player, platforms and hearts in the bottom right of the screen, which indicate a player's health (HP). Background and platforms are supposed to match in appearance and in the overall theme of the current level to give the player a more immersive experience.



This later turned into a more detailed image of the game, also consistent with the Java 2D graphics, which is also the first background you see when you start playing the game, as shown below.

The title screen is the first page the user sees when they open the application. This screen has different options available for the user to select: view the high scores, play the game, or exit the application.

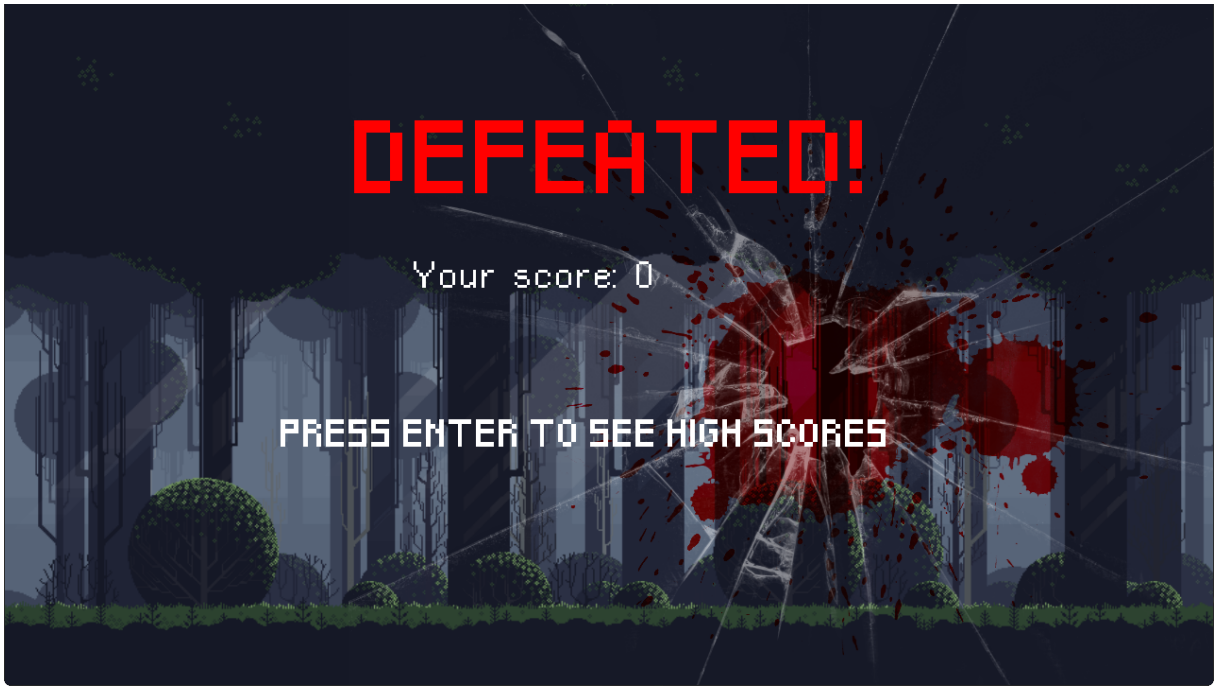
The screen for the high scores in the game is accessed from the title screen (or the defeat screen) and consists of a table of the highest scores from the game. The user interface fits into the overall theme of the game. It also gives the user the option to go back to the title screen.

When you press the play button, the player is first introduced to a help-page which displays the controls used in the game. From there the player can continue on to the selection screen.

The Selection screen consists of 3 different views. The player is prompted for their name, then asked to choose a character and a difficulty level. After this, the game begins.

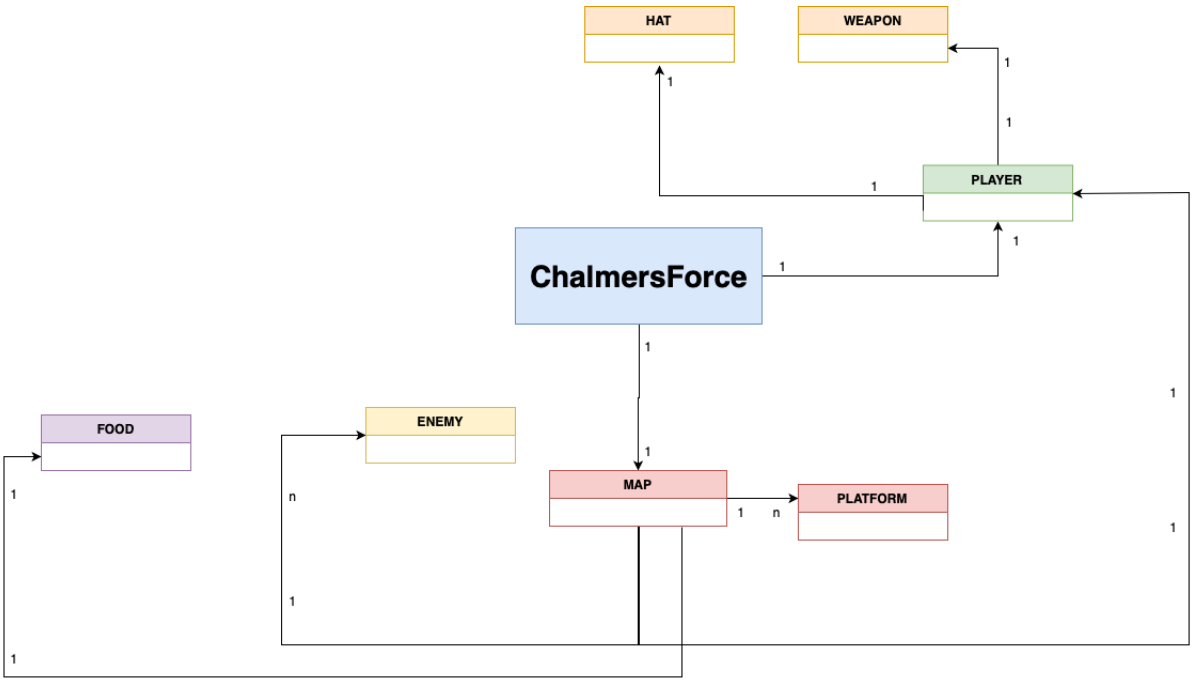
The pause screen is accessed from the game by clicking the pause button (P). The game screen

will be out of focus, so that the different options presented to the user are clearly visible such as return to the game or exit.



When the player is defeated their achieved score will be displayed on a new screen. From there the players only option is to navigate to the high score list, to see if they have managed to get on it. Lastly the player will bring themselves back to the menu page where it is possible to start a new game again.

1.3 Domain model



The figure above shows the domain model of this application. It represents the overall game model, which was designed in the beginning of the design process and further developed as the project progressed.

1.3.1 Class responsibilities

ChalmersForce

The overall representation of the game.

Player

The Player class contains the attributes of the player.

Enemy

This class is responsible for the enemies in the game.

Food

A class that represents the food found in the game.

Weapon

The class that represents a player's weapon.

Hat

The class that represents a player's hat.

Map

This class represents the game world in which the player moves around.

Platform

The Platform class handles the platforms displayed in the game, on which the player can move around.

2 System Design Document

2.1 Introduction

Chalmersforce is a computer based 2D platformer. The purpose of the game is to defeat as many enemies as possible before the player's life runs out. The defeated enemies are converted into a score depending on their difficulty. The player collects the score, giving them the opportunity to land a spot on the high score page.

2.1.1 Definitions, acronyms, and abbreviations

HP

Health Points, the player's life. When these go down to 0 the player is defeated. In the game, it is represented in the form of hearts in the bottom center of the screen, which are red when they are ”full” and slowly lose colour when a player's health decreases, for example during a fight.

Food

Food can be found at random in the game and will give the player some form of benefit or disadvantage depending on the food.

Hat

A Hat gives the player a Boost. Each player character is assigned their own Hat, all with different benefits.

Weapon

A weapon is an item a player can use to fight enemies in the game. A player is given a weapon at the beginning of the game.

GUI

'GUI' stands for Graphical User Interface

2.2 System architecture

The application was developed in Java 8. It uses an external game engine, called **LITIengine**, which is responsible for most of the graphic components in the application, as well as some game physics.

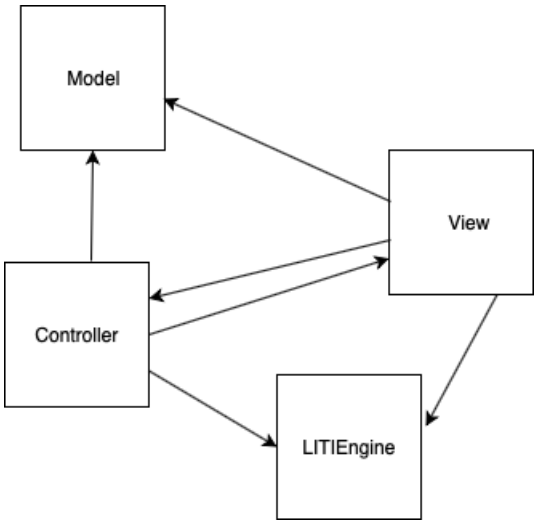
The program uses a **Model-View-Controller** architecture. The game application's controller contains multiple classes that extend or are in some other form dependent on classes in LITIengine. The application's view is also dependent on GUI-related components from LITIengine, in order to provide the user with multiple user interfaces and screen controls.

The model represents the basic game logic of the application, which is later instantiated in the controller.

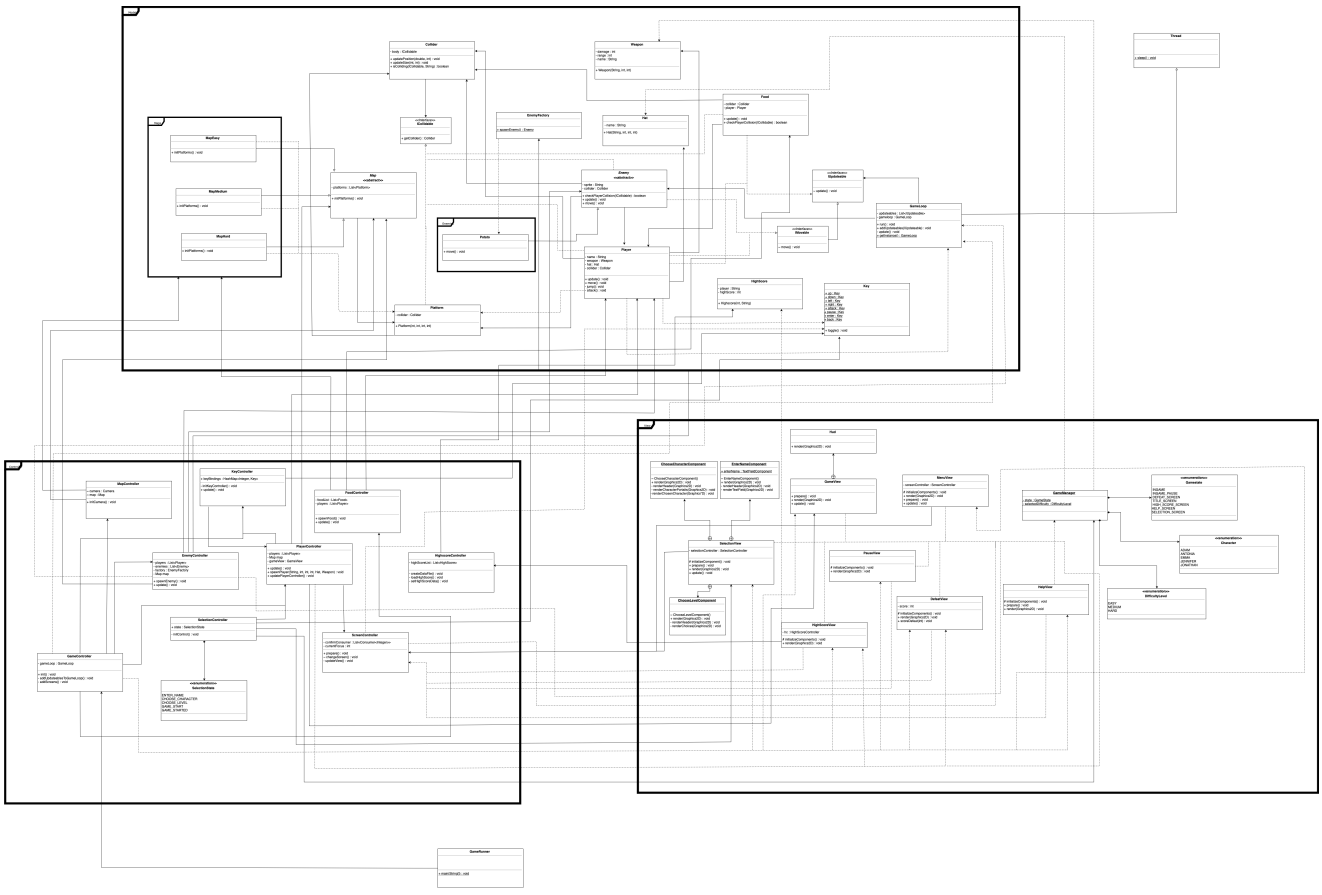
The application can be stopped in two ways, either by pausing it during gameplay, or by exiting it. If the game is paused, the player can return to the game and continue where they left off, or return to the menu. If they return to the menu, the current gameplay is lost. Likewise if the application is exited at any given time. The only thing that will be saved after the application has been closed is the high scores, which will only be saved when the player has navigated to the high score view after achieving the score.

2.3 System design

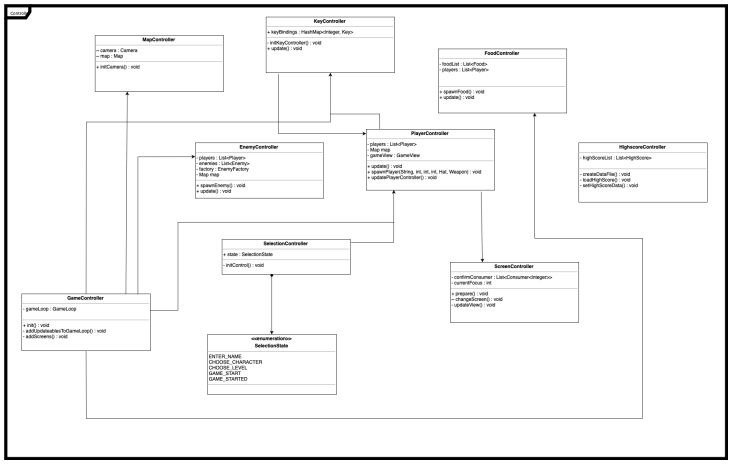
2.3.1 Design model



The figure above is a model of the application's overall composition. It also illustrates the MVC-architecture used as the basis for its design. LITIengine is the external game engine used in the program, on which both the controller and the view are dependent on.
The image below is the application's whole UML diagram. Only relevant variables and methods are included in the diagram.



2.3.2 Controller



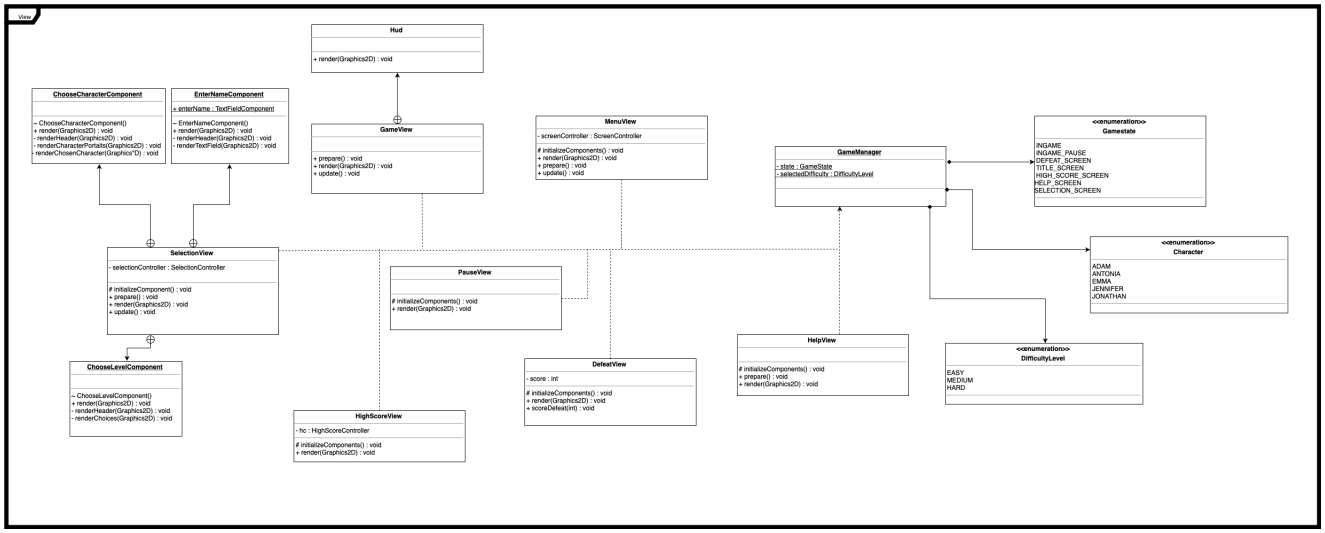
Most of the controller classes instantiate a class from the model, which also enables to use it together with LITIengine's game engine. Hence, many of the controller classes are dependent on said external game engine. For instance, the Player- and EnemyController are subclasses to one of LITIengine's entity classes, that lets the controller assign a sprite to players and enemies, which is a feature made available by the external game engine.

The ScreenController is responsible for the different screens in the application, and for showing the right View class depending on the user's input. The GameController initializes the overall application functionality in order for the main method (in the class "GameRunner") to be able to start the game.

The application offers the possibility to select one of five different characters as well as different difficulty levels in the game, which is handled by the SelectionController.

High scores are saved in an external file. This file is created during the first run of the game and updated before the high score page is accessed by the user. The reading and writing of the file is controlled by the HighScoreController. This class also makes sure the list is always sorted in descending order before it is used by the HighscoreView class or written anew to the file. Along with that it also holds a public method used to add high scores to the list of top scores, as long as the new score is higher than any of the already present ones. The high score list has a limit of 10 items resulting in elements being replaced if they have the lowest value when compared with the other. This method is called at the end of a game.

2.3.3 View

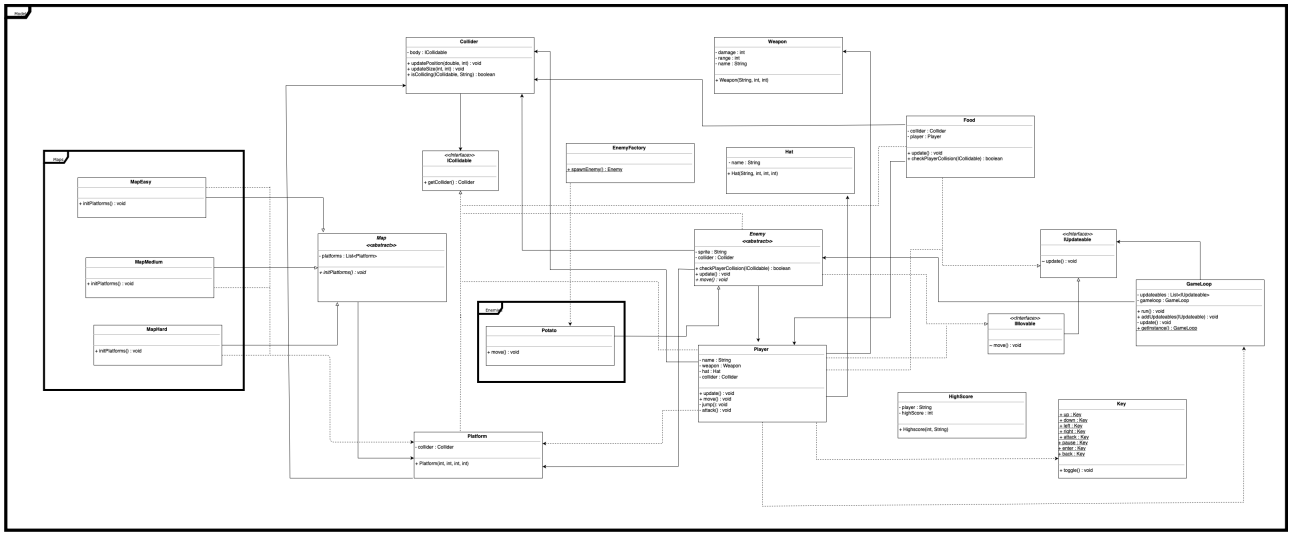


When the game is started the MenuView is displayed. This class contains the graphics visible on the screen as well as the methods for the different buttons available. This view is controlled by the ScreenController class which handles the key input. When the user presses enter on a selected button, the corresponding method is called by the ScreenController and the new view is presented.

The three buttons available on the menu screen are ”High score”, ”Play” and ”Exit”. The last one naturally exits the game and the application is shut down. If the user chooses the high score button, the HighScoreView is displayed. This exhibits the current top scores in the game and the names of the player achieving them. If the user presses enter, they return to the MenuView.

If the user chooses the button ”Play”, the ScreenController will set the screen to the HelpView class. This view is static and contains information about how to play the game. The class holds nothing more than its graphics and a method to display the next screen the user is presented with; the SelectionView. This view displays the different choices the player has, like entering their name, choosing a character to play as, and deciding the difficulty level. After that, the user end up in the GameView, which displays the game.

2.3.4 Model



The application's model handles the overall game logic. The classes represent the main actors in the game, like Player and Enemy.

Most of the classes both implement the interface ICollidable and have an association dependency with the class Collider, which are responsible for handling collisions in the game. The interface IUpdateable and the GameLoop class are the central parts of the application's game loop.

Maps and enemies have their behaviour specified in an abstract class by the same name and are instantiated using the more concrete implementations, which are the classes in the modules also with the names ”Maps” and ”Enemies”.

Domain model and Design model
The model ended up being very similar to the domain model created in the beginning of the application's development and while the domain model has undergone minor changes, the overall game logic has remained the same.

2.3.5 Design Patterns used in the application

MVC Architechture

The application is based around an MVC-model (Model-View-Controller). This gives the application the needed structure to minimize dependencies and facilitate the process of creating useful and appropriate dependencies.

Module Pattern

The Module Pattern is also used in the program, which is shown in the basic game structure.

Factory Pattern

This pattern is implemented at multiple points in the application and used to decrease dependencies, as well as give the program more structure.

Singleton Pattern

The GameLoop class in the application's model implements the Singleton Pattern, so that only one game loop is instantiated.

State Pattern

The State Pattern is used in several places, for instance to keep track of all the different game states. They are held in an enum called ”GameState”. There is also a State for the Player, to keep track of if they are dead or not.

2.4 Persistent data management

2.4.1 High Score Data

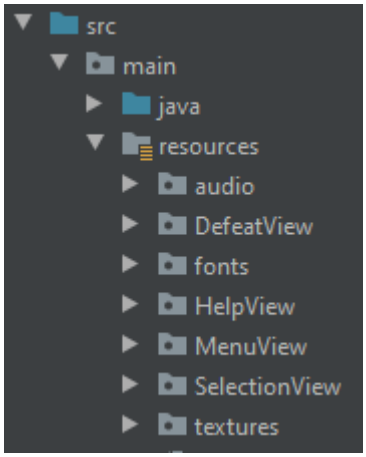
The high scores are stored in a file called ”HighscoreData”. The scores are formatted in a way that makes it easy to both extract and insert data: *PlayerName:Score*

The code for this can be found in the ”HighScoreController” class.

2.4.2 Resources

Other resources, such as audio and graphics for the game, are simply stored in a directory called ”Resources”.

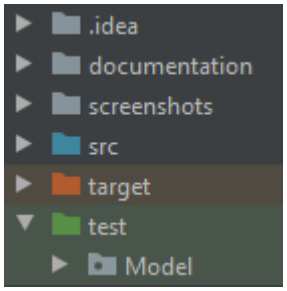
The graphics for each of the views are stored in directories named after the respective view (see image).



2.5 Quality

2.5.1 Testing

To test the project, JUnit was used. The tests can be found in the ”test” directory. Only the application's model was tested, focusing on its more relevant methods, where smaller methods, that acted for example similar to a setter, were not tested.



The project used Travis for continuous integration.

2.5.2 Known Issues

- There are currently issues with displaying the ”Player” and ”Enemy” sprites on screen.
- There are issues with the collision system not checking collision properly.
- The pause button only works sometimes. The issue is that sometimes the GameState does not change but the method is visibly called since the screen changes.
- When accessing the selection screen again after either finishing a game, or backing out of a game via the pause menu, it does not work.

2.5.3 Analytics

N/A. We did not run any analytical tools.

2.5.4 Access control and security

N/A. The application does not have any kind of access control or security.

3 Peer Review

The following peer review was done for the project **Upside Down & Dead**.

3.1 Do the design and implementation follow design principles?

The design follows the Single Responsibility Principle. Classes are generally well-defined with their own area of responsibility. The Open-Closed Principle is followed in multiple parts of the implementation, however there are some exceptions that go against the principle in the overall implementation.

3.1.1 Does the project use a consistent coding style?

The coding style is, for the most part, consistent, and it follows Google's Java Style Guide. The biggest inconsistency can be found in the way that blank spaces are handled. Sometimes, there is a blank space after the class declaration; sometimes, there is not. Occasionally, blank spaces are omitted when they should not be. Other times, there are suddenly twice as many as usual.

3.1.2 Is the code reusable?

A lot of the Model in this project depends on itself with abstract classes and methods. This does not allow for much external reusability of the code. With that said, the code does reuse a lot of itself in these abstractions. Inheritance of abstract classes allow for reusability of the code further down in the inheritance tree, such as the Player and NPC.

3.1.3 Is it easy to maintain?

To be maintainable, one of the things the code should be is well documented and easy to read. In the case of Upside Down & Dead, this is true. (see the section *Is the code documented?* for more information). The code should also be well tested, so that if a change is made that causes an error, it can immediately get caught. (see the section *Is the code well tested?* for more information).

3.1.4 Can we easily add / remove functionality?

Many abstract classes in the Model allow for easy addition of new functionality. This comes with some issues as the classes extend from each other as in:

$$NPC \leftarrow Charakter \leftarrow GameObject \leftarrow HitboxOwner,\ TextObservable$$

The class "Charakter" is redundant as it could simply be an interface since its only purpose is to change the name of said "Charakter". This interferes with the design principle Composition over inheritance. Another issue with the "GameObject" class is that it has a "GameObjectType"

that determines what GameObject it is. This breaks the Open-Closed principle as they have to add new values to the ”GameObjectType” everytime they add a new GameObject. This means that they are checking what ”GameObjectType” a GameObject has instead of comparing classes or type-casting.

3.1.5 Are design patterns used?

Yes. The Observer Pattern is used quite often and in a proper and effective way. According to the System Design Document, the Singleton Pattern will also be implemented in the future.

3.2 Is the code documented?

The code is quite well documented, especially in the Model and View module. Most of the necessary methods are documented through JavaDoc. However when looking through the Controller package, a lot of methods are undocumented. Whilst some of these are methods automatically generated, they are still modified and could be better documented. So in general, the code is well documented with a few exceptions.

3.3 Are proper names used?

Overall, class, method and variable names make the code relatively self-explanatory. Some interface names do not follow the typical naming conventions, e.g. ”HitBoxOwner”, as well as the class ”Scissors”. In the Controller module, some of the classes’ names are confusing and make the class seem misplaced, for example ”Updater”.

3.4 Is the design modular? Are there any unnecessary dependencies?

The project design follows the standard MVC-design with modules for Model, View and Controllers. There was an attempt, to make the design even more modular by moving different classes in the Model into packages. This makes the design easier to understand, but some classes which clearly belong to a package have not been moved, for example the ”Item” class to the ”items” package or ”Level” to the ”levels” package. However the design is quite modular as the Model package could be moved to a different View and Controller and it could still run. The code also avoids some dependencies by for example creating a new Point class instead of using the one that already exists in Java standard libraries.

3.5 Is the code well tested?

The code has a good amount of tests but it is still missing some test-classes. When running the currently created test-classes the code has 99% coverage for the tested classes. Test-methods in the test-classes have proper name-usage and explain what the test’s intention is. Note that one of the tests failed while running them.

3.6 Does the code use proper abstractions?

The code has a total of seven abstract classes in the Model alone. Beyond that, there are also two abstract classes in each of the View and Controller module. Abstract classes have to have at least one abstract method, if methods exist. ”Charakter” is one abstract class which lacks any abstract method, but has other methods, and therefore cannot be classified as an abstract class.

3.7 Are there any security problems, are there any performance issues?

From a security point of view, there are a few classes which have a lot of methods public that could be made private or package-private. With some of these methods being public, the user could for example skip some levels by accessing the public switchToNextLevelView(int level) method in MainController. This is a general theme throughout the code, where some methods and attributes could be changed to private or package-private. In general the program runs smoothly and no performance issues can be noted. There were some problems early on with the screen size not fitting on all screens, as the size was set to a specific x and y value.

3.8 Can the design or code be improved? Are there better solutions?

Code, in any project, can always be improved. In the answers to the questions above, we explain what we feel that the code should be improved on. A better solution to the abstract class inheritance tree is leaning towards the principle ”Composition over inheritance” and to reconsider the option of using an enum to determine which GameObject is which. The comparison of GameObjects can be simplified using the concrete implementation of the GameObject or type-casting.