Project Hero

Project Manual

01001000
01000101
01010010
01001111

Team: Djordjevic Filip
Reichl Markus *(Project Manager)*
Tekin Abdurrahim
Wellner Florian

Supervision: Dolezal Dominik

January 17, 2017

Abstract

Project Hero is a **bullethell roguelike multiplayer RPG** providing a very unique storytelling and gameplay experience. Its main goal as a school project is to create a solid base for a video game ready for further development. The core should be highly abstract, stable and easy to extend.

This document explains how the Hero Project, including all features found in the *Functional Specification* document, is realised by the team.

Contents

1 Introduction **2**

2 Project Goals **2**
2.1 Targets . . . 2

3 Team **4**

4 Documentation Guidelines **5**
4.1 Project Documentation . . . 5
4.2 Work Documentation . . . 5
4.2.1 Development Documentation . . . 5

5 Preconditions **6**
5.1 General Preconditions . . . 6
5.1.1 GitHub . . . 6
5.2 Technical Preconditions . . . 6
5.2.1 Unity . . . 6

6 Requirements **7**
6.1 Functionality . . . 7
6.1.1 Artificial Intelligence . . . 7
6.1.2 Character . . . 7
6.1.3 Effects . . . 8
6.1.4 Resources . . . 8
6.1.5 Items . . . 8
6.1.6 Inventory . . . 8
6.1.7 Attributes . . . 9
6.1.8 Player . . . 9
6.2 Specifications . . . 9
6.2.1 Cross-Platform . . . 9
6.2.2 Extendable . . . 9
6.2.3 Multiplayer Ready . . . 9

7 Architecture **10**

8 Implementation **12**

1 Introduction

The Hero Project is a **bullethell** [I] **roguelike** [II] **multiplayer** [III] **RPG**[IV] providing a very unique storytelling and gameplay experience.

It is highly inspired by previous titles in the roguelike and roleplay game genres, the most noteworthy being Realm of the Mad God, Titan Souls and Enter the Gungeon.

While all mentioned games are using a topdown pixelart setting, the Hero Project is drawn in a very unique combination of high and low resolutions.

2 Project Goals

The main goal of Project Hero as a school project is to create a solid base for a video game ready for further development. This base includes all elements listed in the *Functional* and *Technical* Specifications sections found in the *Functional Specifications* document.

2.1 Targets

Before development can start the aim of the project has to be as clear as possible. Therefore the following targets and non-targets have been declared.

Targets

[I] A video game sub-genre where the screen is usually covered in bullets.

[II] A video game sub-genre which, based on the Roguebasin Interpretation, is defined by

- **Permanent Failure:** The player is encouraged to take responsibility for the risks he takes.
- **Procedural Environments:** Most of the game world is generated and provides complexity in resources and other elements of the game.
- **Resources:** The player can manage a limited amount of resources.

The Roguebasin, Berlin and Roguetemple Interpretation may give a more detailed explanation on this subject.

[III] A multiplayer game allows but does not require clients to play together. The Hero Project is not meant to be played online only!

[IV] A roleplay game is a game in which the player assumes the roles of characters in a fictional setting and takes responsibility for his acting either through literal acting.

A core game A prototype ready for further development including all Functional and Technical Requirements. Date of of submission is set on 17th of January 2017.

A prototype for presentation A simple setup including a player, an enemy and items including effects to declare an attack.

Non Targets

A full game The core game is not meant to provide a full game including graphics, gameplay and multiplayer. This goal is not reachable in time and may not be in budget.

A demo version The core game should demonstrate how the game works and how it can be extended. Not how the game will look like in production.

3 Team

Project Hero's core is developed by a static team of 4 members and 1 professor. All members are listed below followed by a short description including their role, responsibilities and contact information.

Djordjevic Filip Sound Designer
Filip is responsible for all types of music and sound throughout the game.

Email: fdjordjevic@student.tgm.ac.at

Reichl Markus Project Manager *(Product Owner)*
Markus is both project manager and product owner. On development he provides concept art and is responsible for the system's architecture.

Email: mreichl@student.tgm.ac.at

Tekin Abdurrahim Artist
Burak provides art for both concept and production and is responsible for the aesthetics.

Email: atekin@student.tgm.ac.at

Wellner Florian Software Developer
Florian is responsible for the software development process and helps out at pixelart.

Email: fwellner@student.tgm.ac.at

The teams professor **Dolezal Dominik** acts as a supervisor and provides knowledge in all fields. He also helps during development and in the project management process.

4 Documentation Guidelines

4.1 Project Documentation

Project Hero is based on the v-Model[I], which requires the team to provide at least 2 documents.

Functional Specification This document defines all features the product has to provide as a response to a *Requirement Specification* document using the results of a feasibility study.

Project Manual This document explains how the project, including all features found in the Functional Specification document, is realised by the team.

4.2 Work Documentation

Every person working on the project has to provide some documentation for every session including at least:

Current date:	`YY-MM-DD:` Title
Time spent:	START: `00:00`
	TIME SPENT: `0h 0m`
Description:	A short description of this session

4.2.1 Development Documentation

The team is using Git commits to backup their work documentation and record changes during development.

[I] The V-model represents the sequence of steps in a project life cycle development. It describes the activities to be performed and the results that have to be produced during product development. In software development, the V-model represents a development process that may be considered an extension of the waterfall model.

5 Preconditions

5.1 General Preconditions

5.1.1 GitHub

The whole development process of the project is managed using a private GitHub[I] repository. In this case GitHub is also used in the project management process utilising *Issues*[II] and *Milestones*[III]. To backup work documentations and provide more detailed information about the development process, git commits[IV] are used as the teams development documentation.

5.2 Technical Preconditions

As seen in the projects feasibility study there have to be at least some technical conditions to the project.

5.2.1 Unity

First of all the project is running on top of the Unity game engine using C# as its main scripting language. Game elements are therefore implemented as Unity's *Game Objects*[V] and actions are realised as *Mono Behaviours*[VI].

When used correctly this pattern helps a lot with decoupling although it requires the core game to provide more concrete entities. Another limitation is Unity's lacking support for generics, what makes using interfaces for behaviours a lot harder.

[I] GitHub is a web-based Git repository hosting service offering distributed version control and source code management.
[II] Issues are used to keep track of tasks, enhancements, and bugs for GitHub projects.
[III] A milestone acts like a container for issues. This is useful for associating issues with specific features or project phases.
[IV] A Git commit records changes to a repository.
[V] Base class for all entities in Unity.
[VI] Base class from which every Script Component derives.

6 Requirements

6.1 Functionality

6.1.1 Artificial Intelligence

Entities[I] similar to a character and controlled by the game can interact with their environment by using effects. The game itself will differentiate between two main types of AIs, NPCs[II] and Enemies, for most of the time. This does not imply that an AI is either a NPC or an enemy, they just differentiate through their behaviour.

NPCs NPCs may interact with the player through conversation or providing resources.

Enemies While NPCs are most likely to be friendly, enemies will use their effects to harm the player. They can also be used to provide items and other resources for the player when an enemy is killed.

Other AIs AIs different from NPCs and enemies can be objects provided by the environment which interact with the player by using effects.

6.1.2 Character

Characters are entities directly controlled by the client or the game. Every character relies on attributes and has the ability to move, level and use effects.

Movement The character's movement can be controlled by the game or the client using an input of his choice.

Levelling A character is able to gain levels which directly modify his attributes. Attributes can also influence AIs, mostly by modifying their attributes, spawn rates[III] or resources. Levels can be given by default or

[I] Every interacting object in the game is referred to as a (game) entity
[II] A non player character is a character which is not controlled by the player
[III] Entities created in a limited amount of time

increase on specific actions. Common ways for the player to improve his level include fighting enemies and completing quests[I].

6.1.3 Effects

Entities can be given effects allowing them to interact with their environment. They can be activated, enabled or interact permanently. Common effects include attribute manipulation, conversations and also attacks. Players should be able to control their effects using an input of their choice.

Attack A specific effect, which is also implemented in the core game is the attack. Attacks are used to reduce another entity's *life* attribute leading to its death. Attacks interact by firing projectiles.

Projectiles Projectiles are used by attacks to harm other entities. Short range attacks are implemented as a single projectile with a bigger hitbox[II].

6.1.4 Resources

Resources refer to all collectable objects in the game. They do not interact by themselves but can be interacted with through a character's inventory.

6.1.5 Items

Items are resources that can be given effects, which can only be activated by another entity. They should also be ready to be displayed to the player although this feature will not be implemented in the core game.

6.1.6 Inventory

An inventory keeps a limited amount of items and can hold base items, equipment and other resources.

[I] A search or pursuit made in order to find or obtain something. In this case a quest describes a task for the player to complete.

[II] An area in which one object interacts with another

6.1.7 Attributes

All entities using this functionality are influenced by 8 attributes called *health, mana, damage, defence, vitality, wisdom, speed* and *agility*. Their values are used as variables in the entities actions and can also be modified. The number of attributes can also be increased during development. Attributes in the core game have to be accessible but not yet interactable for an AI.

6.1.8 Player

A single client is able to control at least one character's actions and manage its resources.

6.2 Specifications

6.2.1 Cross-Platform

The game runs on multiple platforms including all major desktop operating systems and popular consoles using the cross-platform features of the Unity game engine.

6.2.2 Extendable

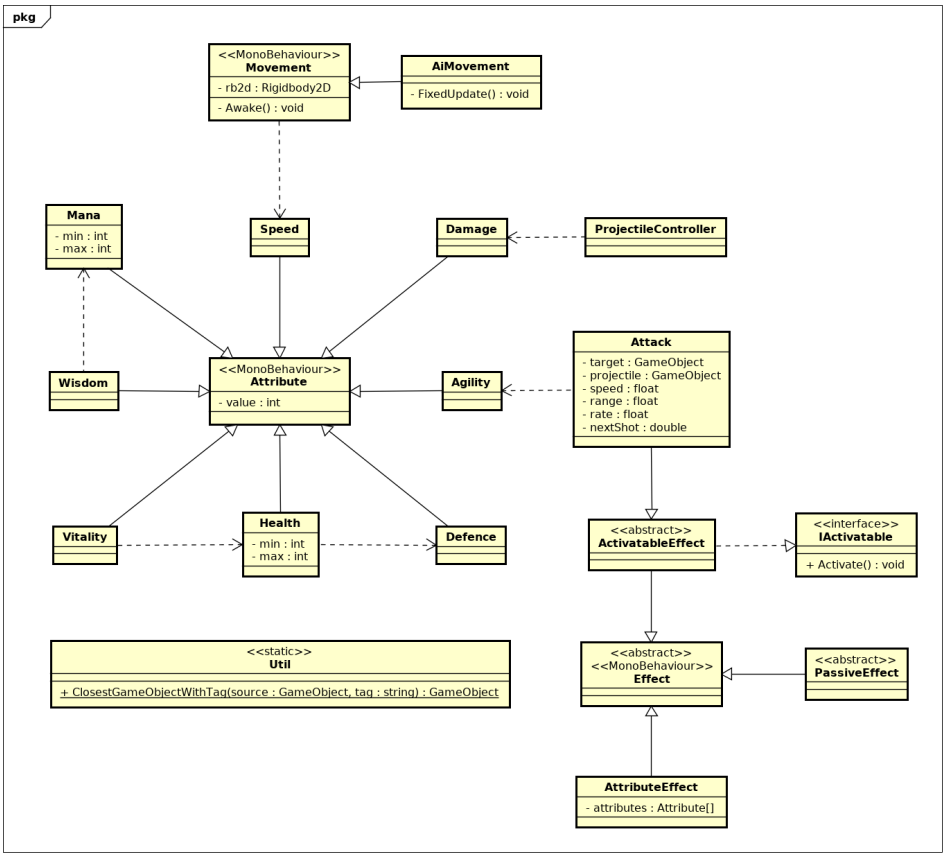
The core game will be highly extendable through abstraction, decoupling and maintainability.

6.2.3 Multiplayer Ready

Because the actual game is thought to be played on a server with multiple clients the core is already developed for this purpose. A real multiplayer is not necessary in the core game but definitely nice to have.

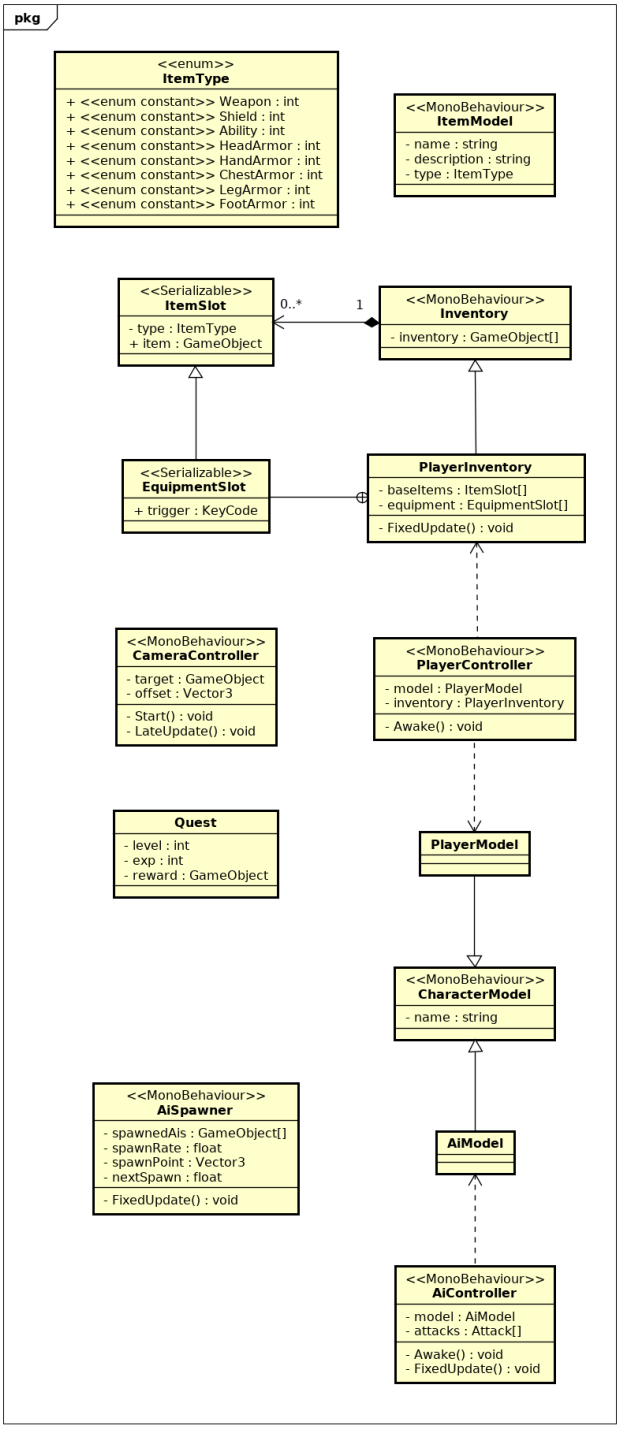
7 Architecture

The software architecture is implemented based on the following UML[I] Class diagrams[II].



I The Unified Modeling Language is a general-purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualise the design of a system.

II In software engineering, a class diagram in UML is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations, and the relationships among objects.


pkg
<<enum>>
ItemType
+ <<enum constant>> Weapon : int
+ <<enum constant>> Shield : int
+ <<enum constant>> Ability : int
+ <<enum constant>> HeadArmor : int
+ <<enum constant>> HandArmor : int
+ <<enum constant>> ChestArmor : int
+ <<enum constant>> LegArmor : int
+ <<enum constant>> FootArmor : int
<<MonoBehaviour>>
ItemModel
- name : string
- description : string
- type : ItemType
<<Serializable>>
ItemSlot
- type : ItemType
+ item : GameObject
0..*
1
<<MonoBehaviour>>
Inventory
- inventory : GameObject[]
<<Serializable>>
EquipmentSlot
+ trigger : KeyCode
PlayerInventory
- baseItems : ItemSlot[]
- equipment : EquipmentSlot[]
- FixedUpdate() : void
<<MonoBehaviour>>
CameraController
- target : GameObject
- offset : Vector3
- Start() : void
- LateUpdate() : void
<<MonoBehaviour>>
PlayerController
- model : PlayerModel
- inventory : PlayerInventory
- Awake() : void
Quest
- level : int
- exp : int
- reward : GameObject
PlayerModel
<<MonoBehaviour>>
CharacterModel
- name : string
<<MonoBehaviour>>
AiSpawner
- spawnedAis : GameObject[]
- spawnRate : float
- spawnPoint : Vector3
- nextSpawn : float
- FixedUpdate() : void
AiModel
<<MonoBehaviour>>
AiController
- model : AiModel
- attacks : Attack[]
- Awake() : void
- FixedUpdate() : void

8 Implementation

This section provides information about the development process of the Hero project.