|  |
| --- |
| CPSC 2720 FALL 2015 |
| Project Advent |
| Blinding Eclipse Group |
|  |
|  |
|  |
|  |
| **Juan Monterrosa**  **Amrit Grewal**  **Frank(Guoqiang) Zhao** |
| **Nov 8th 2015** |

Contents

[Revision History 3](#_Toc432455197)

[Introduction 4](#_Toc432455198)

[Project Management 4](#_Toc432455199)

[Team Organization 4](#_Toc432455200)

[Risk Management 4](#_Toc432455201)

[Meetings & Communication: 4](#_Toc432455202)

[Development standards: 5](#_Toc432455203)

[Software Design 5](#_Toc432455204)

[*Design* 5](#_Toc432455205)

[Design Rationale 6](#_Toc432455206)

[Solid + Dry Principles 6](#_Toc432455207)

[Appendices 7](#_Toc432455208)

# Revision History

|  |  |  |
| --- | --- | --- |
| **Revision Date** | **Changes Made** | **Name** |
| Oct 6th, 2015 | Added Introduction | Chris Wharf |
| Oct 6th, 2015 | Added Team organization and Risk Management sections | Juan Monterrosa |
| Oct 6th, 2015 | Added Meeting and development standard sections | Amrit Grewal |
| Oct. 8, 2015 | Added SOLID+DRY principles description | Juan Monterrosa |
| Oct 12, 2015 | Added Class diagram description | Amrit Grewal |
| Oct 28th, 2015 | Edited class diagram discriptions | Juan Monterrosa |
| Nov 1st, 2015 | Edited Risk Management | Amrit Grewal |
| Nov 3rd, 2015 | Modified some classes. Removed multiple characters and kept main character class | Amrt Grewal |
| Nov 6th, 2015 | Added Pattern Design Information | Frank Ghao |

# Introduction

The document that follows is a design plan for the Advent Project. This project is a text based adventure game set in a fantasy world. The game is set in the land of Torvald. The brave adventurer that the player will control travels through the land, and encountering various characters along the way proceeds to the castle to face his ultimate test. The player moves through 4 different environments: Forest, Village, Cave and Castle. Player will capture the sacred gems of the world. The player will have health that will be increased or decreased during the game. Player can carry items such as potion, super potion and bombs. When the player consumes potion then the health will be increased. Enemies can be attacked by fire, ice, quake and master sword in addition to the basic attack. Player will lose health during the fights.

We used C++ to code this project. It is compiled using g++11 compiler. Many design patterns have been used such as strategy pattern.

The design sections we will cover in this document include our teams organization, the risk management procedures we will follow and the general software design for the project.

# Project Management

## Team Organization

Our team will have four distinct roles. We will each take responsibility for our respective obligations and will be held accountable for completing our part or the project.

***Juan Monterrosa*** – Team Lead

* Responsible for implementing the environments
* Implement Player class
* Collaborate with Design Lead and Development lead to ensure coding standards are met
* Ensure work is divided properly among the team
* Responsible for the git merging and ensuring the project is ready to be submitted on master branch

***Amrit Grewal*** - Documentation and Design Lead

* Responsible for updating and maintaining the design document and ensuring the project meets the assignment requirements
* Design and write the Manual of the Game
* Implement code for Character classes and anything in relation to that
* Implement game saving mechanism

***Frank Zhao*** - Development Lead and Designer

* Responsible for maintaining the Argo UML up to date
* Implement Driver class and assist Juan with coding the environments
* Implement Land of Torvald class
* Implement Exceptions handling

***Christopher Wharf*** - Quality Assurance Lead

* Member is withdrawn from the course
* Responsibilities have been adjusted among the team

## Risk Management

We understand that there can be unforeseeable issues that arise throughout the semester. Known constraints that may occur include illness and unanticipated life events and conflicting work/school schedules. In such case, we have designed plans to resolve any possible issues that may arise and to keep our team in order. If a team member is unavailable for a team meeting then they will let the rest of the other group members know with a valid reason for the absence. In such case, that person is responsible for catching up and contributing to the work that was done in his leave. Our group will focus on constant communication to keep a steady workflow and keep every week productive. By having a set weekly schedule, we will minimize these possible occurrences.

### If one group members drops out, we will divide the work equally among the rest of the group. If two or more group members drop out, we will still divide the work but touch base with our supervisor Dr. Anvik to keep him in the loop and discuss any impact that this situation may have on the final product he will receive. We will request him to consider either changing the requirements or assigning more staff.

If the team is over ambitious about the project, it will be team lead’s responsibility to control this. If team lead is not able to control the situation then it will be team members’ responsibility to bring this up in the meetings so we can change team lead.

We believe that in order for project team to work at its full potential everybody’s best expertise should be harnessed. In our weekly meetings, when dividing work, we will discuss who is best for what kind of tasks such as who is best at exception handling. We will let that person take the lead of that particular skill. If two people want the same task, we will vote on it as to who should get it. If that draws out, then we will draw secret ballot.

### Meetings & Communication:

* Two one hour weekly meetings
* Tuesday and Thursday 11:00AM - 12:00PM
* Additional meetings will be scheduled as needed
* Unexcused absence will require extra work to be completed assigned by the rest of the group
* Skype is an acceptable form of attendance
* Group messages via Hangouts to keep in communication with the entire group

### Development standards:

● Use C++11 standard with G++ compiler

● Document all classes using Doxygen annotations

# Software Design

Our software design includes 20 classes and implement Solid + DRY principle for class responsibility. The classes are also designed using strategy design pattern. We used ArgoUML software to design the class diagrams. In the next sections you will see description of all these classes as well as an explanation of the design principles we used.

## Design

1. **Player:** Player has a name, health that he/she has to maintain during the game, couple of abilities (i.e. actions such as talk, attack etc) and a few items that he/she can use during the game.
2. **Weapon:** An abstraction for different types of weapons a player can use. Classes such as Basicattack, Fireattk, Iceattk, Quake and MasterSword will extend this super class. A weapon will have a name and the actual methods to do the action will be implemented in the sub-classes.
3. **BasicAttk:** A specific implementation of action that provides methods to attack when the player attacks without any super powers.
4. **FireAttk:** A specific implementation of action that provides methods for player’s FireAttack. FireAttk will have much more damage than the basic attk.
5. **IceAttk:** This class will provide implementation of action by while providing methods that allow a player to attack using Ice that has a different damage.
6. **MasterSword:** A certain type of weapon that has a different damage level.
7. **Quake:** Quake is specific implementation of weapon that damages in it’s own way.
8. **Item:** Items will have a name. Player will carry these around throughout the game. They may be used during the interactions with the environments and other characters. An example of items can be potion that he uses to regain health.
9. **Potion:** This class will implement action by providing additional methods that allow player to eat the potion and regain his/her strength. Eating restores a player’s health.
10. **SuperPotion:** SuperPotion can be be carried as an item by the player hence item is a base class for it. It differs from Potion in the sense that it helps regain a lot more health.
11. **Bomb:** Player is able to carry bomb as one of the items with him. It extends Item class.
12. **LandOfTorvald:** Land of Torvalds is the world the game is being played in. It consists of different environments within which the player interacts.
13. **Environments:** This will be an abstraction of environments. This class will contain the name of each environment and a list of all the characters in that environment. Environments will contain a virtual method called setup that must be implements by any environment extending it.
14. **Cave:** Cave implements environments. It will provide implementation for setup method so that the environment is setup for caves.
15. **Forest:** Forest will extend environments by setting up the environment while overriding environment’s setup. Forest will have no. of trees. It will have a weather of rain or no rain.
16. **Castle:** Castle is also an extension of environments. Castle will have towers and will override environments’ setup.
17. **Village:** Village will also implement environments’ setup in its own way and hold its own characters.
18. **Character:** Characters will be interacting within different environments with the player. A character will have a name, a description and a dialog for some characters. This will be an abstraction of characters. The specific characters will implement this class.

## Design Rationale

### Solid + Dry Principles

Our design will follow the SOLID design principles.

***Single Responsibility →*** every class in our design has one single responsibility. Each action class does one function, each character does one thing, each environment generates its own environment

***Open/Closed →*** all the entities in our classes are open for extension such that if we want to add a new environment or action to our system we simply add it to the respective Super Class where it will inherit all the pure virtual functions needed. In our design we have abstracted what is common and encapsulated what is varied.

***Liskov Substitution →*** our design will incorporate inheritance throughout and will have classes that can be substitutable with its proper subtypes

***Interface Segregation →*** the system will have many actions and items and environments available but no one particular class in the system will be forced to use methods it doesn’t need. We have created it in such a way that we will be able to choose what functionality we want for every particular class

***Dependency Inversion →*** the program will be set up with abstraction in mind. Resulting in high and low level modules not depending on each other but both depending on the interfaces we’ve created such as environment, actions and characters.

***DRY →***  we will try to adapt the KIS Principle and during the implementation phase of the project we will create functions to minimize repeated code. This will be determined at the lower level of the system.

### Design Patterns used

Strategy Pattern

Strategy pattern is used in the implementation and design of the Weapon class, it is used to further abstract weapons into each attack so that for a given weapon, a different attack can be performed against another character aka. Enemy. This allows the definition of a family of algorithms, in this case would be the various attacks available in the game for the player.

Command Design Pattern

Command decouples the object that invokes the operation from the one that knows how to perform it. Since in our program we are using commands entered by the player. Our environments are implementing command design pattern to instantiate the correct type of weapons and classes based on the commands provided.

Mediator Pattern

Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently. We implement this pattern by letting environments control the interaction between the characters and the player. Environment here acts as the mediator.

# Appendices

1. **SOLID**: It stands for Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion. The purpose of this principle was that by using this a programmer will develop a system that more maintainable and flexible.
2. **DRY:** It stands for Don’t Repeat Yourself. This principle, as the name implies, focuses on one source of truth approach within a software system. Every single entity in a software architecture should only have one definition and an unambiguous representation that everybody understands, agrees upon and is used by everyone rather than creating their own version of it.