System design document for Upside Down & Dead

Amanda Dehlén, Linnea Johansson,
Johannes Kvernes, Anna Nilsson, Samuel Widén

2019
version

1 Introduction

Upside Down & Dead is a puzzle game presented as an application for desktop users. The game starts with the user finding out that they are dead and the world is upside down - hence the name. The user's goal is to flip the world back to normal through puzzles. In this document the design of the code behind the game will be discussed and presented.

1.1 Definitions, acronyms, and abbreviations

NPC - Non player character. A character that players does not control, but can often interact with.
Gamer - A person who plays video games.
Arrow keys - The keys on the keyboard with arrows that are usually used to move things around on the screen.

2 System architecture

If in the future it is decided that functionality for saving the progress of the game should be implemented, this should use some kind of database or similar. However, this has not yet been done.

3 System design

Design patterns (will be expanded on):
Our application implements several instances of the Observer Pattern. These instances are: Controller which observes KeyEvents, Updater which observes Controller, TextObserver which observes TextObservable
The application also has a MainController, which performs tasks like displaying the application, which view to display, and which controller to update. The MainController is an instance of the Singleton pattern (WIP, not implemented in the peer review version).

4 Persistent data management

Not applicable

5 Quality

We test the code we write using JUnit, in a test package you can find in "src".

- Known issues:
  - Switching to the next level results in index out of bounds, should be fixed by implementing a factory for Levels.
  - Circular dependency between MainController and other implemented controllers (StartMenuController, NameInputController and InGameController), will be solved by using state pattern.
  - Switching levels is done in PuzzleInteractionUpdater, should be done in Door.
  - New InGameView subclasses have to be created for every level, which is not extensible. Can be solved by making InGameView more generic.
  - New Level subclasses have to be created for every level. Can be fixed by implementing a class to read levels from files (such as .json files).
- Analytical tools used for testing quality of code:
  - We tried to get STAN (stan4j) to work, but had no success.
  - PMD has been used to eliminate some bad practices

5.1 Access control and security

Not applicable

6 References

Not applicable