Testing and Inspection Report

Prepared by

Ronny Recinos, Jeremy Robles, Shin Imai, and Jon-Michael Hoang

For use in CS 440 at the

University of Illinois - Chicago

Fall 2019

Table of Contents

List of Tables

I Project Description

1 Project Overview

Working, Loving, Programming!! is a programming puzzle game with a romantic twist, set in an office environment with the main character (MC) being the player. The MC is known to be an incredible programmer with vast knowledge in Computer Science, and naturally co-workers seek out the MC for programming help. The game involves a “romancing” aspect where for each problem that the MC helps a co-worker with, they increasingly like the MC.

2 Project Domain

The domain of this project will cover the programming quiz segment(s) of the game, character dialogue and choices, visiting different locations, saving/loading and user progression.

3 Relationship to Other Documents

The project has several relationships with other documents, such as, ProgPuzzle requirements document, ProgPuzzle design document, and ProgPuzzle project description document.

4 Naming Conventions and Definitions

4a Definitions of Key Terms

Affection: A measure of the relationship between the main character (user) and a character within the game (same as affinity).

Affinity: A measure of the relationship between the main character (user) and a character within the game (same as affection).

Credibility: Measure of quantifying the player’s accuracy in providing the correct answer to programming puzzles

Main Character: In a video game, this is the person whose point of view the user sees within the storyline. This is also generally the person that the story of the game depends on/revolves around.

MC: See *Main Character*.

Menu: An interface that the user interacts with within the game to access and use features. “Menu” is not used to refer to a list of dishes available in a restaurant.

Non-Playable Character: A character within the game that cannot be controlled directly by the user.

NPC: See *Non-Playable Character*.

Sprite: A graphic which adds to the visual aspect of the game. “Sprite” is not used to refer to fairies, elves, and the beverage.

Visual Novel: A game that features an interactive text-based story with narrative-based literature and interactivity aided by visuals most often using anime-style art.

VN: See *Visual Novel*.

4b UML and Other Notation Used in This Document

This document generally follows the UML standards as described by Fowler in [1]. Any exceptions are noted where used.

4c Data Dictionary for Any Included Models

Data Structure(s):

```
public class DialogText {

	private String text;

	private static int id = 0;

	// setters and getters here…

}
```

The above data structure is a base class for text within the game.

```
public class Choice extends DialogText {
```

```
        private String text, filepath;

        private int index;

        // @Override setters and getters…from DialogText

        // In addition to adding additional setters and getters

        // for those that were not defined in DialogText

}
```

The above Choice data structure contains the necessary items for loading the choices the user has to pick to advance the story of the game. *text*, as it implies, provides the text for the choice, where *index* allows for the organization of elements within the game in respect to the story, and *filepath* specifies additional resources that the Choice class will load.

```
public class Dialog {

        public String title, charName, charImg;

        public static int id = 0;

        public ArrayList<DialogText> textList;

}
```

The above Dialog data structure contains necessary items for the narrative of the story. *charName* and *charImg* provides the name and image of the respective character speaking given a *textList* that is loaded in from XML files within the *raw/* directory of the code. *title* and *id* provides a means of organizing the dialog in respect to the narrative.

```
public abstract class Puzzle {

        int stringID_title, //String of puzzle title

        stringID_description, //String of puzzle description

        stringID_code,  //The code represented as a string

        stringID_hint,  //Hints provided by the character

        stringID_explanation,   //The explanation of solution
```

```
      initialCredibility, //The initial value (difficulty) of puzzle

      currentCredibility, //The current reward value of puzzle (will
initially be equal initialCredibility)

      puzzleID;

      boolean isComplete;

      String title, //String of puzzle title

            description, //String of puzzle description

            hint,  //Hints provided by the character

            code,  //The code represented as a string

            explanation;   //The explanation of solution

}
```

Puzzle: The Puzzle data structure is for the encapsulation of puzzles located in the strings.xml resource. Puzzle has two subclasses: Puzzle_FillInTheCode (fill-in-the-blank puzzles), and Puzzle_MultiChoice (multiple choice puzzles). Each subclass contains data members which represent possible solutions to the puzzle.

```
public class Puzzle_Catalog {

      int credibility;

      int pythonCredibility, cppCredibility, assemblyCredibility,
sqlCredibility, javaCredibility, cCredibility;

      ArrayList<Puzzle> puzzleArrayList;

      int puzzlesFound, puzzlesSolved;

}
```

Puzzle_Catalog: This data structure is for the encapsulation of save data including credibility ratings for each NPC character, and puzzles solved/found.

Scores that are within the game:

Results for the equations listed in this section will always result in a non-negative value.

Points awarded per puzzle = **50 - (Number of times wrong * 5)** such that the minimum points awarded is **10**.

Affection/Affinity = Points awarded per puzzle * Number of available puzzles.

II Testing

5 Items to be Tested

- Puzzles
    - Fill in the code
    - Multichoice
- Dialog
    - Character Dialog
    - Character Choices
- Menu Screen
    - Office
    - Park
    - Cafe
    - Bar
    - Clipboard
- Clipboard
    - Credibility Rating
    - Puzzles Solved
    - Individual Rating
    - Puzzles Found
    - Puzzle Replayability

6 Test Specifications

ID# - Name T1 - Activity Transitions

Description: Test if menu buttons will start the intended activity

Items covered by this test:

1. Menu: Office, Park, Cafe, Bar

2. Clipboard
3. Dialog Choice
4. Title screen

Requirements addressed by this test: N/A

Environmental needs: The application is running on an emulated phone with Intellij.

Intercase Dependencies: N/A

Test Procedures:

1. Start application
2. Activate every item that involves a transition
3. Record results

Input Specification:

- Pressing the button of Office, Park, Cafe, Bar, or Clipboard in the Menu screen
- Pressing the button of a dialog choice that will transition to a puzzle activity
- Replaying (launching) the puzzle activity from the clipboard activity
- Pressing on the title screen

Output Specifications: After pressing the icon then it must transition to the next activity.

Pass/Fail Criteria:

Pass: Successfully transitioned to the next activity

Fail: Crashes the application, transitions and crashes, does not transition at all.

ID# - Name T2 - Character Dialog

Description: Test character dialog parsing and processing

Items covered by this test:

- Character dialog
- Character choices

Requirements addressed by this test: While not addressed in project description

Environmental needs: Android studio, Android emulator, dialog-formatted XML files

Intercase Dependencies: T1

Test Procedures: Automate the process:

For each starter dialog xml file:

- Encapsulate the dialog via dialog objects
- Automate clicks on the emulator to progress dialog
- Click on choices if necessary to progress dialog

Input Specification: Character dialog XML files which will be formatted with appropriate tags

Output Specifications: An ArrayList<Dialog> containing instances of Dialog which have been partitioned to represent each scene from the dialog XML file

Pass/Fail Criteria:

The test passes when: For every click on the TextView frame id “dialog_text”, the displayed text on dialog_text should change to one of the following:

If the next tag in the XML file is:

1. <Text> => display the next line of text
2. <Dialog> => change the “character” ImageView to display image in <CharacterImg> tag, change the text in “dialog_name” to text in <name> tag, change the text in “dialog_text” to be the text of the closest <Text> tag
3. <Choices> => Make the “choice1” and “choice2” Buttons visible, and update text with text in XML <Text> tag

If the behavior does not match the aforementioned specifications, the test fails.

ID# - Name T3 - Answer grading

Description: Test if answer provided by user is correct for the puzzle

Items covered by this test:

Puzzles:

- Fill in the code

- correct output
- spot the error

Requirements addressed by this test: None of the functional requirements address the puzzles, but the test is partially addressed by the quality requirements in the "Interact with level" use-case.

Environmental needs: Android studio, Android emulator, Activities and classes related to the puzzles {Puzzle_FillInTheCode, Puzzle_MultiChoice, Puzzle, PuzzleActivity}

Intercase Dependencies: T1

Test Procedures: Automate the process:

For each puzzle starting with the <string-array> in Strings.xml:

- Load the puzzle in emulator through the puzzle activity
- Input an incorrect answer
- Input a correct answer: an answer that is equivalent to a possible solution string (in a Puzzle_FillInTheCode object) or an answer that matches the SolutionIndex of a Puzzle_MultiChoice object.

Input Specification: Strings.xml with puzzles, Choice

Output Specifications: A log containing all puzzles tested, and answers that were tried for each puzzle.

Pass/Fail Criteria: The test passes if

- All tests from strings.xml is encapsulated by an appropriate Puzzle object
- All inputs that are incorrect deduct credibility points and create toast message
- All inputs inputs that are correct add credibility and create toast message

If any of these conditions are not met, or the test execution exits abnormally/prematurely, the test fails

ID# - Name T4 - Save Progress

Description: Save completed puzzles, and progress of character dialogs

Items covered by this test: Clipboard

- Credibility Rating
- Puzzles Solved
- Individual Rating
- Puzzles Found
- Puzzle Replayability

Requirements addressed by this test: Functional Requirement: #6, #7

Environmental needs: Android studio, Android emulator

Intercase Dependencies: T2

Test Procedures:

- Delete save files
- Complete an arbitrary set of puzzles
- Record details about Clipboard, (credibility, puzzles solved, & all other “items covered by this test”)
- Reboot the app
- Check that the data in the Clipboard has been loaded back in the same state as it was before rebooting

Input Specification: The data about the clipboard that is to be written in a save file, (Credibility Rating, Puzzles Solved, Individual Rating, Puzzles Found, Puzzle Replayability)

Output Specifications:

An instance of the PuzzleCatalog class with loaded data

Pass/Fail Criteria: The test passes if save file is correctly encapsulated by the PuzzleCatalog class. If credibility {for each NPC, overall}, Puzzles solved, individual rating, puzzles found, and puzzle replayability are successfully loaded as recorded in the save file, then the test passes. If any of the aforementioned parameters are not successfully loaded in the PuzzleCatalog object, then the test fails.

7 Test Results

ID# - Name T1 - Activity Transitions

Date(s) of Execution: N/A

Staff conducting tests: Ronny

Expected Results: Each item that involved transitions to another activity would lead to their respective activity.

Actual Results: Each of the items did successfully transition to the activity each were assigned to.

Test Status: PASS

ID# - Name T3 - Answer grading

Date(s) of Execution: 11/27/2019

Staff conducting tests: Shin

Expected Results: A log containing all puzzles tested, and answers that were tried for each puzzle.

Actual Results: The test ran to the end and did not end prematurely. All correct and incorrect answers were handled correctly.

Test Status: PASS

ID# - Name T2 - Answer grading

Date(s) of Execution: 11/13/2019

Staff conducting tests: Jeremy

Expected Results: The dialog XML files located in the raw resources directory get encapsulated by an ArrayList of Dialog objects. The Dialog objects get displayed through Views.

Actual Results: The resulting ArrayList was able to encapsulate the lines from the dialog XML files. All transitions in the dialog (text to text, text to next scene, text to choice) are working correctly through clicking the dialog_text TextView.

Test Status: Passed

ID# - Name T4 - Save Progress

Date(s) of Execution: 11/21/19

Staff conducting tests: Jon-Michael

Expected Results: For every reboot, the save file should be correctly encapsulated by the clipboard and dialog

Actual Results: After a hard reset, loading works for the first reboot, but the compiler fails after every subsequent reboot

Test Status: FAIL - Clipboard and dialog progress is successfully loaded in first reboot, but fails for each subsequent reboot and load

8 Regression Testing

There is no need for regression testing until further development through the coming years.

III Inspection

9 Items to be Inspected

- Puzzles
    - Fill in the code
    - Multichoice
- Dialog
    - Character Dialog
    - Character Choices
- Menu Screen
    - Office
    - Park
    - Cafe
    - Bar
    - Clipboard
- Clipboard
    - Credibility Rating
    - Puzzles Solved
    - Individual Rating
    - Puzzles Found
    - Puzzle Replayability

10 Inspection Procedures

Inspection - **ID: Inspect-1** Features: PuzzleActivity			
FILE HEADER: Are the following items included and consistent?	**yes**	**no**	**comments**
Author and current maintainer identity	x		
Cross-reference to design entity	x		
Overview of package structure, if the class is the principal of a package	x		
FILE FOOTER: Does it include the following items?	**yes**	**no**	**comments**
Revision log to minimum of 1 year or at least to most recent point release, whichever is longer		x	

IMPORT SECTION: Are the following requirements satisfied?	**yes**	**no**	**comments**
Brief comment on each import with the exception of standard set: java.io.*, java.until.*		x	
Each imported package corresponds to a dependence in the design documentation		x	
CLASS DECLARATION: Are the following requirements satisfied?	**yes**	**no**	**comments**
The visibility marker matches the design document	x		
The constructor is explicit (if the class is not static)	x		
The visibility of the class is consistent with the design document	x		
CLASS DECLARATION JAVADOC: Does the Javadoc header include:	**yes**	**no**	**comments**
One sentence summary of class functionality	x		
Guaranteed invariants (for data structure classes)	x		
Usage instructions	x		
CLASS: Are names compliant with the following rules?	**yes**	**no**	**comments**
Class or interface: CapitalizedWithEachInternalWordCapitlized	x		
Exception: ClassNameEndsWithException	x		
Constants (final): ALL_CAPS_WITH_UNDERSCORED	x		
Field name: capsAfterFirstWord. Name must be meaningful outside of context	x		

Inspection - **ID: Inspect-2** Features: MapActivity			
FILE HEADER: Are the following items included and consistent?	**yes**	**no**	**comments**
Author and current maintainer identity	x		

Cross-reference to design entity	x		
Overview of package structure, if the class is the principal of a package	x		
FILE FOOTER: Does it include the following items?	**yes**	**no**	**comments**
Revision log to minimum of 1 year or at least to most recent point release, whichever is longer		x	
IMPORT SECTION: Are the following requirements satisfied?	**yes**	**no**	**comments**
Brief comment on each import with the exception of standard set: java.io.*, java.until.*		x	
Each imported package corresponds to a dependence in the design documentation		x	
CLASS DECLARATION: Are the following requirements satisfied?	**yes**	**no**	**comments**
The visibility marker matches the design document	x		
The constructor is explicit (if the class is not static)	x		
The visibility of the class is consistent with the design document	x		
CLASS DECLARATION JAVADOC: Does the Javadoc header include:	**yes**	**no**	**comments**
One sentence summary of class functionality	x		
Guaranteed invariants (for data structure classes)	x		
Usage instructions	x		
CLASS: Are names compliant with the following rules?	**yes**	**no**	**comments**
Class or interface: CapitalizedWithEachInternalWordCapitlized	x		
Exception: ClassNameEndsWithException	x		
Constants (final): ALL_CAPS_WITH_UNDERSCORED	x		
Field name: capsAfterFirstWord. Name must be meaningful outside of context	x		

Inspection - **ID: Inspect-3** Features: Character puzzles			
FILE HEADER: Are the following items included and consistent?	**yes**	**no**	**comments**
Author and current maintainer identity	x		
Cross-reference to design entity	x		
Overview of package structure, if the class is the principal of a package	x		
FILE FOOTER: Does it include the following items?	**yes**	**no**	**comments**
Revision log to minimum of 1 year or at least to most recent point release, whichever is longer		x	
IMPORT SECTION: Are the following requirements satisfied?	**yes**	**no**	**comments**
Brief comment on each import with the exception of standard set: java.io.*, java.until.*		x	
Each imported package corresponds to a dependence in the design documentation		x	
CLASS DECLARATION: Are the following requirements satisfied?	**yes**	**no**	**comments**
The visibility marker matches the design document	x		
The constructor is explicit (if the class is not static)	x		
The visibility of the class is consistent with the design document	x		
CLASS DECLARATION JAVADOC: Does the Javadoc header include:	**yes**	**no**	**comments**
One sentence summary of class functionality	x		
Guaranteed invariants (for data structure classes)	x		
Usage instructions	x		
CLASS: Are names compliant with the following rules?	**yes**	**no**	**comments**

Class or interface: CapitalizedWithEachInternalWordCapitlized	x		
Exception: ClassNameEndsWithException	x		
Constants (final): ALL_CAPS_WITH_UNDERSCORED	x		
Field name: capsAfterFirstWord. Name must be meaningful outside of context	x		

Inspection - **ID: Inspect-4** Features: Character Dialogue			
FILE HEADER: Are the following items included and consistent?	**yes**	**no**	**comments**
Author and current maintainer identity	x		
Cross-reference to design entity	x		
Overview of package structure, if the class is the principal of a package	x		
FILE FOOTER: Does it include the following items?	**yes**	**no**	**comments**
Revision log to minimum of 1 year or at least to most recent point release, whichever is longer		x	
IMPORT SECTION: Are the following requirements satisfied?	**yes**	**no**	**comments**
Brief comment on each import with the exception of standard set: java.io.*, java.until.*		x	
Each imported package corresponds to a dependence in the design documentation		x	
CLASS DECLARATION: Are the following requirements satisfied?	**yes**	**no**	**comments**
The visibility marker matches the design document	x		
The constructor is explicit (if the class is not static)	x		
The visibility of the class is consistent with the design document	x		

CLASS DECLARATION JAVADOC: Does the Javadoc header include:	**yes**	**no**	**comments**
One sentence summary of class functionality	x		
Guaranteed invariants (for data structure classes)	x		
Usage instructions	x		
CLASS: Are names compliant with the following rules?	**yes**	**no**	**comments**
Class or interface: CapitalizedWithEachInternalWordCapitlized	x		
Exception: ClassNameEndsWithException	x		
Constants (final): ALL_CAPS_WITH_UNDERSCORED	x		
Field name: capsAfterFirstWord. Name must be meaningful outside of context	x		

11 Inspection Results

Inspect-1

- **What was inspected?** PuzzleActivity Class
- **Inspectors:** Ronny, Jon-Michael, Shin
- **Time and date: 12:00 P.M. 11/04/19**
- **Results:** Further inspection may be required in the future when new puzzles are added

Inspect-2

- **What was inspected?** MapActivity Class
- **Inspectors:** Ronny, Jeremy, Shin
- **Time and date: 1:20 P.M. 11/06/19**
- **Results:** Further inspection may be required in the future when new locations are added

Inspect-3

- **What was inspected?** ListOfCharacterPuzzles XML
- **Inspectors:** Shin, Jeremy, Jon-Michael
- **Time and date: 2:00 P.M. 11/08/19**
- **Results:** Further inspection may be required in the future when new coding puzzles are added

Inspect-4

- **What was inspected?** shin_MainActivity Class (Dialog Class)

- **Inspectors:** Ronny, Jon-Michael, Jeremy
- **Time and date: 12:30 P.M. 11/11/19**
- **Results:** Further inspection may be required in the future when new dialog options are added

IV Recommendations and Conclusions

- Puzzles
    - Fill in the code **(Passed)**
    - Multichoice **(Passed)**
- Dialog
    - Character Dialog **(Passed)**
    - Character Choices **(Passed)**
- Menu Screen
    - Office **(Passed)**
    - Park **(Passed)**
    - Cafe **(Passed)**
    - Bar **(Passed)**
    - Clipboard **(Passed)**
- Clipboard
    - Credibility Rating **(Passed)**
    - Puzzles Solved **(Passed)**
    - Individual Rating **(Passed)**
    - Puzzles Found **(Passed)**
    - Puzzle Replayability **(Passed)**

All items that were to be tested have passed with flying colors.

V Project Issues

12 Open Issues

- Continuation of using voice actors for each character. Involving more people into the project will result in having a budget which is nonexistent currently.
- Porting the game to other platforms will introduce complications which will require funding.
- Puzzles not having a working compiler will likely hinder the education of users.
- The use of copyrighted artwork will have to be addressed before the final product

13 Waiting Room

1. Additional characters.
    a. Java
    b. C++
    c. Scala
2. Additional visitable locations.
    a. Company Breakroom
    b. Company Lounge
    c. Company Gym
3. Expanded character arcs for all launch characters
    a. SQL
    b. Assembly
    c. Python
    d. C
4. Implementation of a quick load/save system
5. Additional puzzles for each character and implement more puzzle modes

14 Ideas for Solutions

1. Adding a day and night cycle which many other visual novel games implement as a feature
2. Switching to an actual visual novel engine to have it do most of the groundwork and building off of that
3. Implementation of several load and save slots
4. Connecting the application to the google games service so the user can create a game account using their google account
5. Fast forward button that will skip character dialogue and head straight to doing the puzzles

15 Project Retrospective

As the developers of the ProgPuzzle project we have had numerous issues throughout the development of the project, such as, figuring out what kind of programming language we all wanted to code in and also what kind of IDE we would use to write the code in. We resulted in developing an Android application for this project which worked out mostly well, but there were some issues in developing in Android, such as, there was a large skill gap between members in the group where most were very well versed in it and others had to learn quickly to be able to contribute to the coding project. So in the future it would be better to develop in an environment that everyone would be very familiar with so there wouldn't be any complications and hardships when learning in a new environment.

As for the coding part of the project, there were also issues of correctly connecting everyone's independent code and also working on code segments that were being edited by multiple people which resulted in merge conflicts on GitHub. Learning how to resolve merge conflicts on GitHub would be a better alternative than manually having to merge locally.

Also, as for group synergy it benefited us that we all knew each other before taking the class which resulted in better communication among the group members that helped with the project immensely.

VI Glossary

Affection: A measure of the relationship between the main character (user) and a character within the game (same as affinity).

Affinity: A measure of the relationship between the main character (user) and a character within the game (same as affection).

Main Character: In a video game, this is the person whose point of view the user sees within the storyline. This is also generally the person that the story of the game depends on/revolves around.

MC: See *Main Character*.

Menu: An interface that the user interacts with within the game to access and use features. "Menu" is not used to refer to a list of dishes available in a restaurant.

Non-Playable Character: A character within the game that cannot be controlled directly by the user.

NPC: See *Non-Playable Character*.

Sprite: A graphic which adds to the visual aspect of the game. "Sprite" is not used to refer to fairies, elves, and the beverage.

Visual Novel: A game that features an interactive text-based story with narrative-based literature and interactivity aided by visuals most often using anime-style art.

VN: See *Visual Novel*.

XML: E**X**tensible **M**arkup **L**anguage is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

ArrayList: is a part of collection framework and is present in java.util package.

VII References / Bibliography

[1] Martin Fowler, UML Distilled, Third Edition, Boston: Pearson Education, 2004.

VIII Index