RoomX

Project Report

09/02/2016

Bo Li
Snehal Patil
Manasa Nagaraju
Juntao Zhang

Prepared for
SCU COEN 275—OOADP
Instructor: Leyna Cotran, Ph.D.
Summer 2016

Table of Contents

1. Requirements

1.1 Background

Escape Room is a adventure game in which players are locked in a room and must work together to find keys within a set time limit to finally escape from the locked rooms. In this game, players start out in a room which is locked. They must explore the room to find hidden clues and puzzles. Frequently, they will come across a box that requires a passcode to open, or a door that needs a key. The passcode and the key can be discovered by solving either puzzles or by finding clues. With enough searching, they will be able to successfully figure out codes, open all locked containers, root out hidden objects and passageways, and work their way into either the next room or to freedom.

1.2 Objective

In this project, a single-player video game version of Escape Room is implemented. Users will be placed in a locked room and they need to find the key (4-digit passcode) to open the door and leave the room in a limited time. Users can see an overview of the room and they can click on items in the room to get hints about the passcode. For example, they can click on a vase to find a piece of paper with some word puzzle. By solving the puzzle they might get a digit which is included in the passcode to open the digit lock of the door. Users are able save the items they found to check it anytime they want. There is also a timer system to count down and display the remaining time and a hint message system to help user when they are stuck in the game.

1.3 Use Cases

(→ : Next use case to go when triggered; “if” statement describe the workflow based on users’ behavior)

- Case A: Start the game from main page
    - The main page is displayed
    - If user clicks “Instructions” → a introduction window is popped up
    - If user selects level and clicks start → Case B executes
    - If user clicks Exit → App window is closed
- Case B: Play the game
    - The game panel is displayed
    - Background music starts to play
    - Timer starts to count down
    - If user clicks pauses → Case H
    - If user clicks stop → Case G
    - If user clicks on a hidden item → Case C
    - If user clicks on lock icon → Case D
    - If user clicks on hint icon → Case E

    - If user clicks on the right sidebar → Case F
    - If time is over → Case I
- Case C: Check an item
    - A dialog pops up and user reads it
    - All buttons will be disabled before user closes the dialog
    - User confirms and the clues will be visible on the right sidebar → Case B
    - If time is over → Case I
- Case D: Click on lock icon and try to open it
    - A dialog pops up and user tries passcode
    - All buttons will be disabled before user closes the dialog
    - User tries password
        - If password is correct → Case I
        - If password is wrong, display "your password is wrong" → Case D
    - User clicks on return → Case B executes
    - If time is over → Case I
- Case E: Check hint message
    - A dialog pops up and user reads hint
    - All buttons will be disabled before user closes the dialog
    - User clicks on return → Case B
    - If time is over → Case I
- Case F : Check the found clues on the right sidebar
    - A dialog pops up and user reads hint
    - All buttons will be disabled before user closes the dialog
    - User clicks one icon → Case C
- Case G: Stop the game
    - Timer stops
    - All in-game buttons is disabled before resumed
    - If user clicks on Quit → Case I
    - If user clicks on Continue → Case J
- Case H: Pause the game
    - Timer stops
    - User clicks on the continue icon→ Case B
- Case I: Game stops
    - If the lock is open, game panel switches to the congratulation window
    - If the lock is not open, game panel switches to the failure window
- Case J: Resume the Game
    - Timer resumes
    - All buttons will be enabled

- Automatically → Case B

1.4 Non-Functional Requirements

1.4.1 Short Game Time

The game time should not be too long since this game aims to help people to take a break and have fun.

1.4.2 Simple Gameplay

The game should be simple to use. There should be a simple and yet detailed introduction about the game interface. Also, the game interface should be simple and the logic of the game should be easy to understand.

1.4.3 Cartoon theme

Based on our interviews with potential users of this software, cartoon is one of the most popular theme for the game interface as they would like to see in the game since cartoon theme make the users feel more free-of-pressure.

1.4.4 Relaxing background music

In contrary to the tense pace as of its original real-life form, our Escape Room is more like a relaxing game to help players free from their regular life/work pressure. Therefore, the background music should be relaxing rather than thrilling.

1.4.5 Comprehensible clues

The clues should be easy to understand and comprehend but not direct, for the purpose to make the game full of fun but not too hard.

2. Design

2.1 Use Case Diagram

2.1.1 Win/Fail

The first use case diagram show the overall game process when the user plays the game.

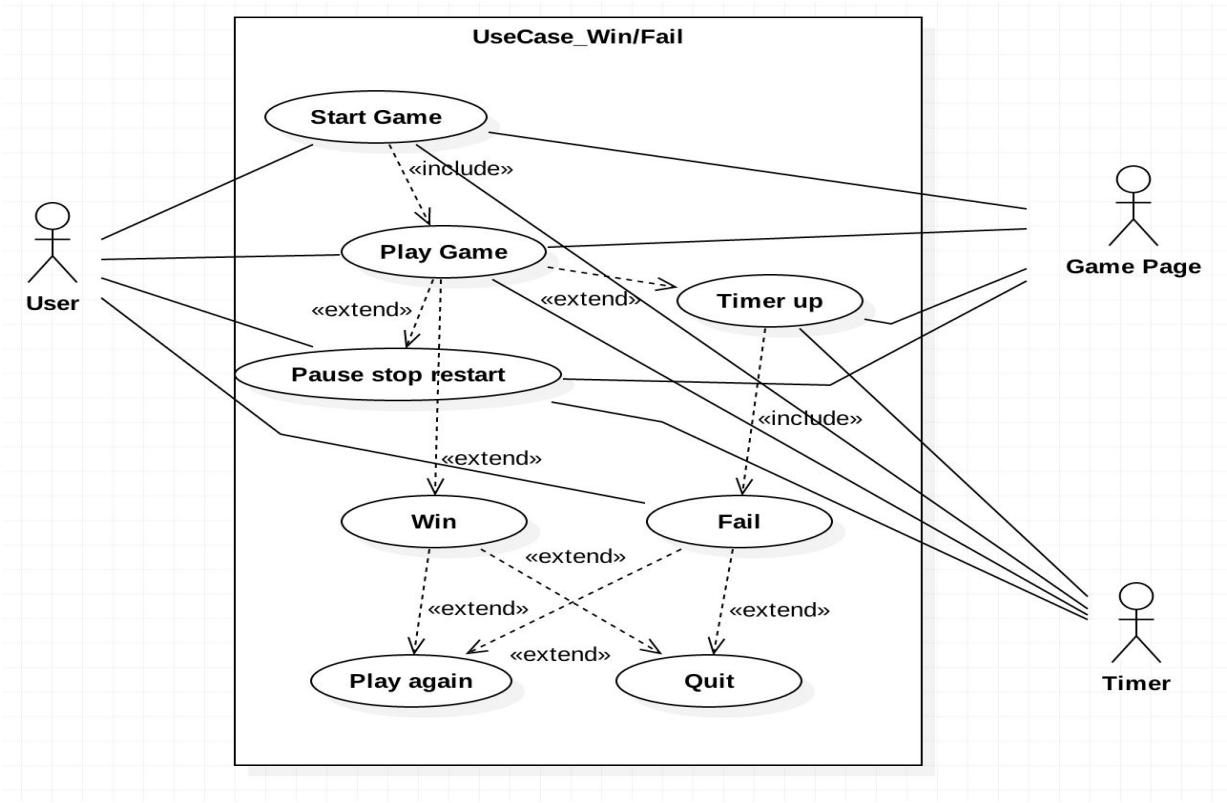


Fig. 1

2.1.2 Start/Leave

The Start/Leave diagram show how the system should support a quick start/leave functionality for the convenience of users.

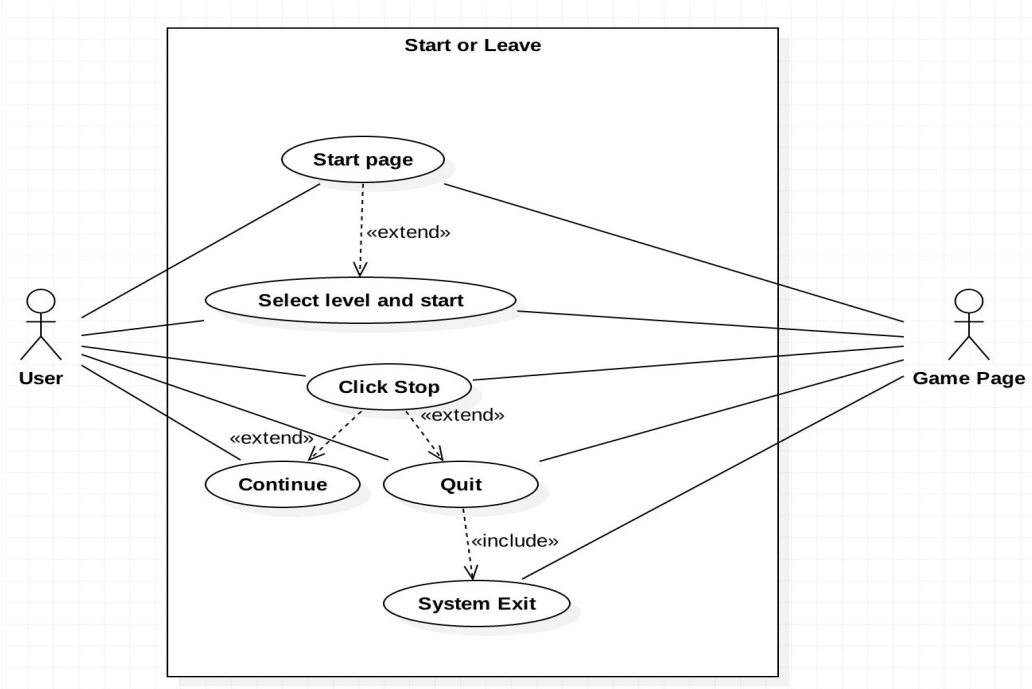


Fig. 2

2.1.3 In-game Experience

The third use case diagram shows how the user will interact with different elements on the game panel and make a progress in the game.

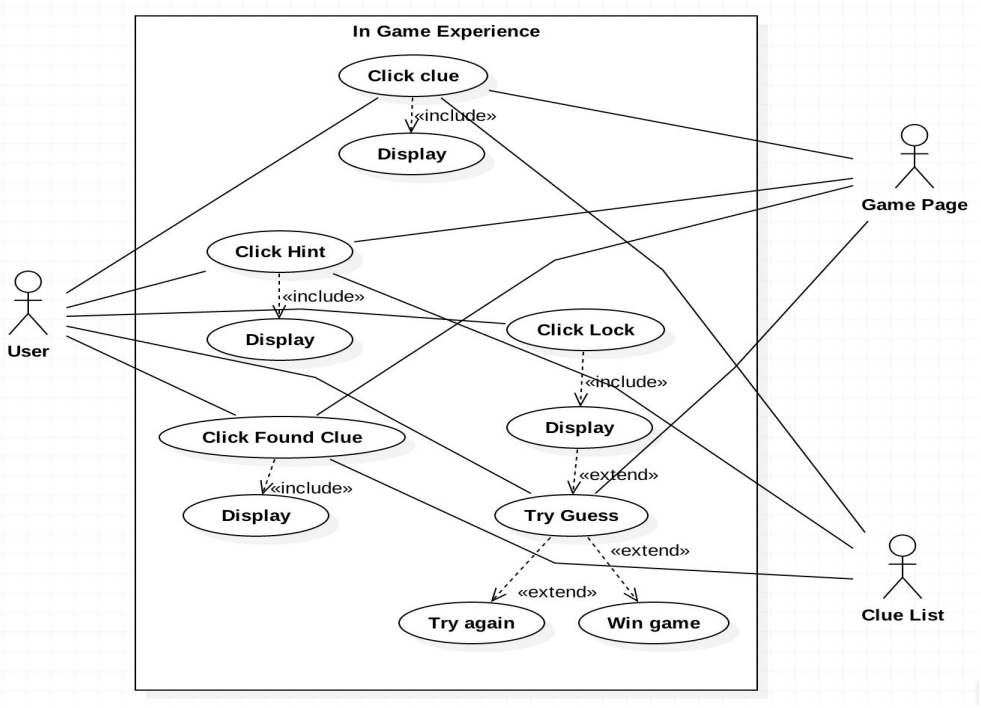


Fig. 3

2.2 Sequence Diagram

2.2.1 Timer Related

Since the timer plays a key role in the game, the first sequence diagram is used to describe the process of the game with the timer. When a player starts the game, the process class will create a window and launch the entry panel. When the user select a level and start the game, a game panel will be created which will create a level class with all the information about this level and be initialized based on this level class. A timer will be created and initialized according to the game time given in the level class. Once the game panel is launched, the timer starts to count down. If the user click on either the pause or the stop button, the timer will be stopped and destroyed after the current left time has been saved to the game panel. When the user resumes the game, a new timer will be created, initialized based on the left time and starts. When the timer counts down to 0, the timer will notify the game panel. The game panel will take control from here, destroy everything including the current timer and request to switch to the failure panel.

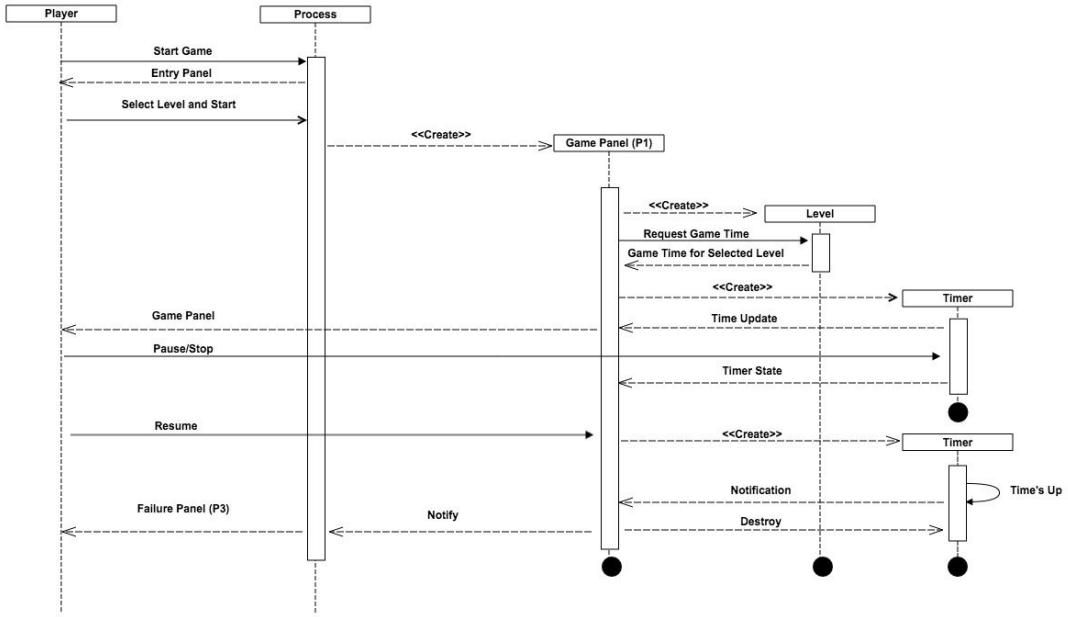


Fig. 4

2.2.2 Game Experience

This sequence diagram describes how each button of the game panel will work as the user clicks on them. When the user clicks on the clue/found clue/hint button, the game panel will determine which function in ActionPerformed should be called and display a new pop-up window based on the information retrieved from the list of clues saved in the game pangel class. The user can click on the close button to return back to the game panel. When the user clicks on the lock icon, a pop-up window will be displayed. The user can input a guess and it will be compared with the correct answer, if it is wrong, the game panel will be notified and a wrong answer notification will be displayed on the lock pop-up window. If the user succeeds, the game panel will close all everything and request to switch to the success panel.

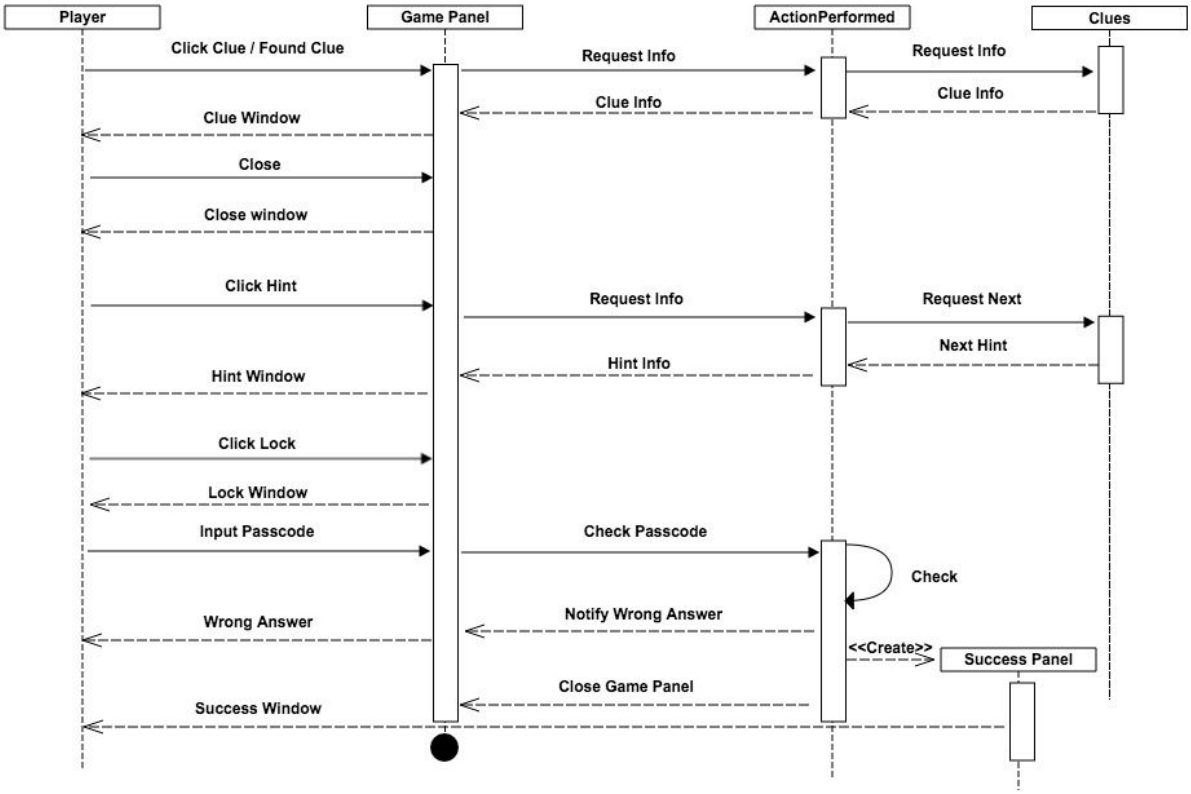


Fig. 5

2.3 UML Class Diagram

2.3.1 Implementation Class diagram

Below is our main class diagram. P0, P1, P2 and P3 stands for the entry panel, the game panel, the success panel and the failure panel. Frame is where these panel will be launched. PanelController will receive the request from each panel and talk with Frame in order to help each panel to switch to others. The Process class is the main process of the game. The RemindTask1 is used to define the behavior of the timer.

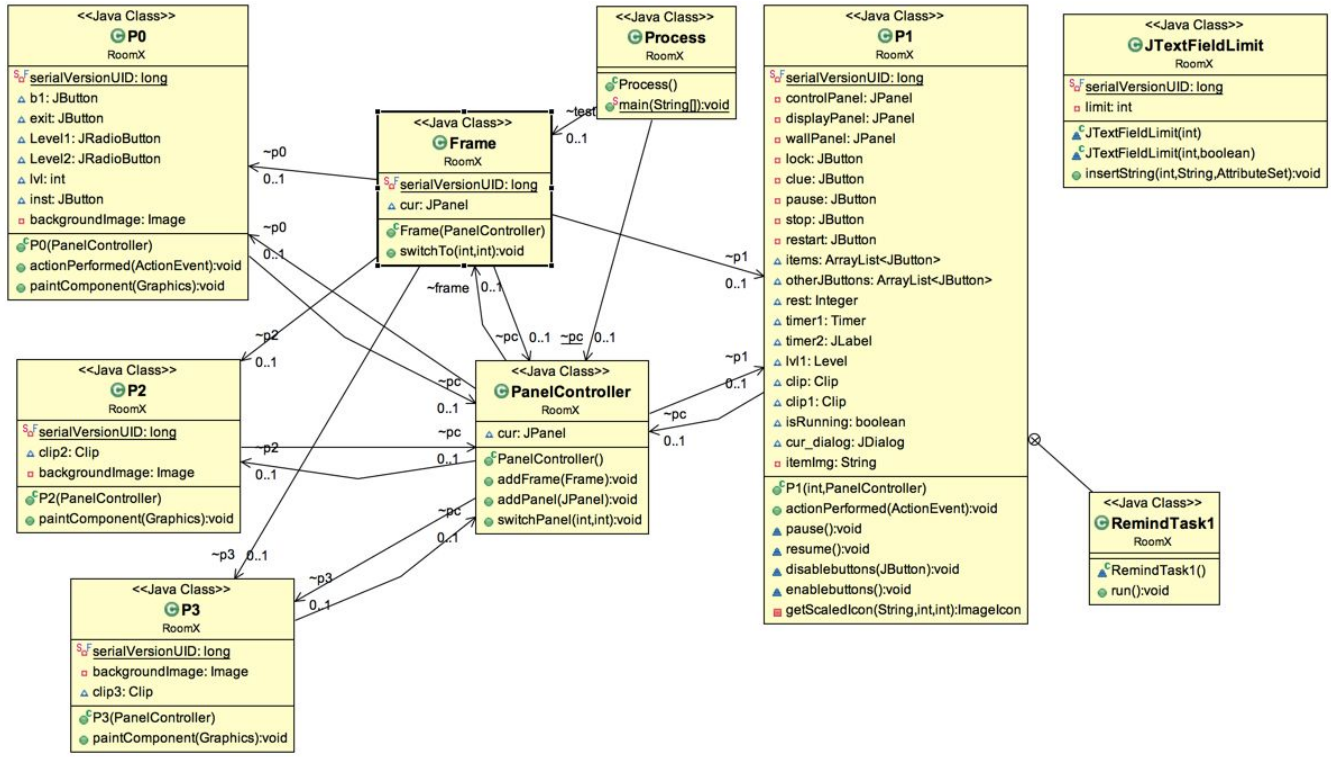


Fig. 6

2.3.2 Game Information

As we mentioned above, for each level, there is a level object which includes all the information about this level, including a list of clues. A basic class clue is defined and includes all the common attributes about a clue such as position information and corresponding hint. Based on it, we have an class EmptyClue extending it, which is a generalization of clues containing some content. Currently, we extend EmptyClue to two type of subclasses. The first one is WordClue which is used for the word puzzle. The second one is PictureClue which provide a picture to the user as clue.

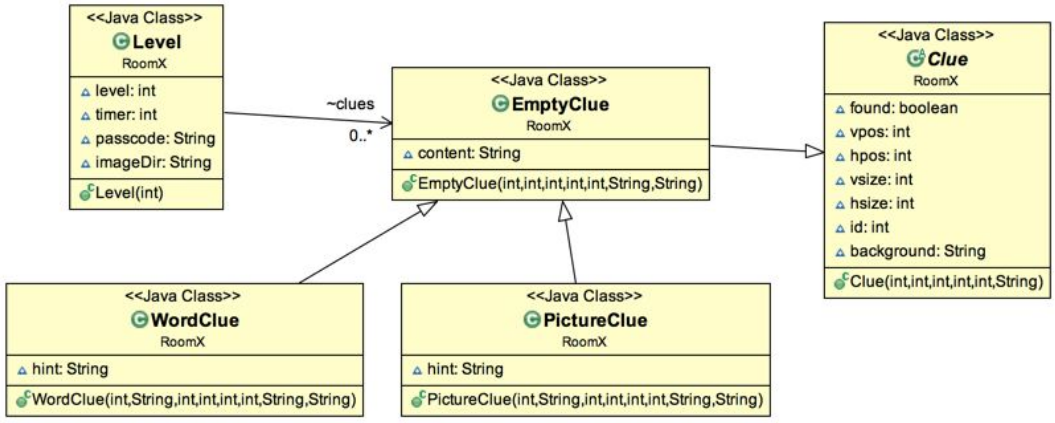


Fig. 7

2.3.3 Game Panel

In Fig. 8, the framework of the game panel (P1) class is shown. When P1 is created, a level object and two button lists will be created and initialized, which will be used later to control the behavior of all the buttons. A timer is created based on the information from the level class. Then based on the information from the level object, three panel will be created and deployed on different part of the game panel and each button within them will be created and deployed on the panels. Finally, all the clickable items including the game process control (pause, stop, resume), gameplay(hidden clues, found clues, hint, lock) will be linked to an action list, which is used to trigger different event based on the user's behavior.

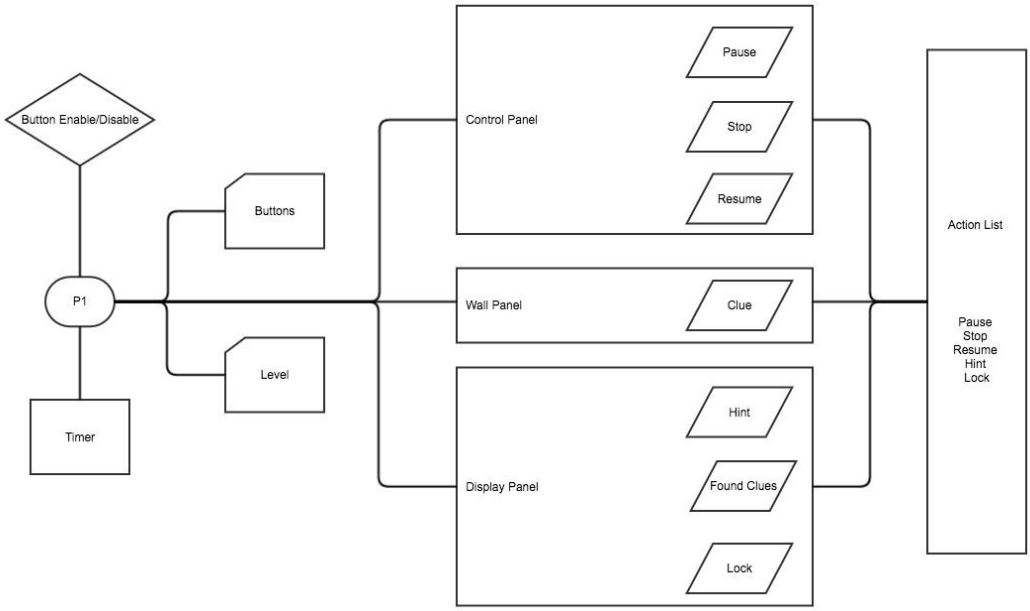


Fig. 8

3. Implementations

3.1 Language

Java.

3.2 Platform

Eclipse, version 4.5.2.

3.3 API

The Java AWT and Swing API is used for building GUI and adding graphics functionalities. The Java Sound API is used to implement the sound effect of each button and the controlling of the background music for each page. The Java package Util is used for the implementation of timer.

4. Testing

4.1 Traceability

Traceability Matrix is a table that links the requirements throughout the validation process. It tracks the completeness of requirements/roles relationship in system development.The requirements are each listed in a row of the matrix and the columns of the matrix are used to identify where each requirement has been addressed in Use case, Class Diagram, Sequence Diagram, Implementation and Testing.

User Requirements	UC	CD	SD	Code	Testing
Start a single player game	2	P0	1	Process(), P0(),PanelController.switchPanel (),P1()	9
Quit the game	2	P0	1	P0()	8
System play background music	3	P1	2	P1()	2
Music on clicking Clue	3	P1	2	P1()	2
Stop the current game and move back to main screen	2	P1	1	P1()	3
Pause the game[stop timer]	1	P1	1	P1.pause()	1
Play game[Restart timer]	1	P1	1	P1.play()	9
Try different lock code	3	P1	2	P1()	4
Save solved clues	3	P1	2	P1()	6
Read saved clues	3	P1	2	P1()	6
Find hints of hidden objects	3	P1	2	P1()	7
Try correct lock and win the game	1	P1,P2	2	P1(),PanelController.switchPanel(),P2()	4
Try incorrect lock and continue	3	P1	2	P1()	4
Timer expires and lose game	1	P1,P3	1	RemindTask1(),PanelController.switchPanel(),P3()	10,5
Leave the lock window	3	P1	2	P1()	4

Fig. 9

4.2 Test Cases

Manual Testing was used to test the functionality and requirements of Escape Room game development. The table below lists the requirements for testing the functionality of the game, each with a test case ID and columns are used to identify where these requirements are tested in use cases, class functionality and non functionality requirements.

TC ID	TC Name	UC1	UC2	UC3	CF1	CF2	CF3	CF4	NF1	NF2
1	Pause/Resume					X			X	
2	Background music	X	X	X		X	X		X	X
3	Try Stop Window		X			X			X	
4	Try Open Lock	X		X		X	X	X	X	
5	Fail the game	X				X	X		X	
6	Click/Revisit clue			X		X				
7	Click hint			X		X				X
8	Quit game	X	X		X	X	X		X	
9	Restart Game	X					X			
10	Fail with open window	X				X	X	X	X	
11	Check instruction				X					X

UC1: Win/Fail
UC2: Start/Leave
UC3: Game Experience
CF1: Main Panel
CF2: Game Panel
CF3: Win/Fail Panel
CF4: Lock
NF1: Music
NF2: UI Experience

Fig. 10

4.3 User Experience

4.3.1 Objective

Since the product is an interactive game, it must be user-oriented. Therefore, user experience and their feedback is critical for us to improve both the functional and nonfunctional parts of our product.

4.3.2 Objects

We invite some friends and our family members to try the game and provide their feedback about the game.

4.3.3 Findings

From the feedback of most potential users we have the following findings. Users would like the game to be of relaxing theme which will help refresh their minds and release their life/work pressure, rather than the original tense pace. Most of the users prefer the cartoon style for the game from many potential themes for the game provided by us. Also, they would like the game to be simple and they think one room including every clues hidden in it is good enough. Lastly, they would like to see some misleading clues which may not provide useful information or contain no information to make the game a little bit more difficult but not too hard.

5. Discussion

5.1 Approach to OO

5.1.1 OO Complexity

Hierarchical Structure:

We have applied hierarchical structure to the Clue Functionality. We have decomposed the problem of generating a single clue into different sub categories based on their type:

1. Picture Clue

2. Word Clue

Both of their implementation will inherit from a superclass EmptyClue which extends an abstract class Clue. Clue is the superclass of all clues. EmptyClue is one type of Clue, which contains some content to be displayed to user.

Relative Primitives:

disableButtons() and enableButtons() are primitive components of the system. These functionality can be extended to disable/enable buttons individually.

Separation of Concerns:

Our system is decomposed into separate components: Panel Controller, Process [controls level and game info], Actions [handles user actions in the GUI Panels].Each of these parts can be studied in relative isolation.

Common Patterns:

getScaledIcon() to scale image is reused for creating picture icons for JButtons.

Stable Intermediate Forms:

Since our system was updated very frequently as we developed our system in the Agile way, we needed to make sure our system work anytime and could be extended with more functionality, therefore, we implemented many stable intermediate form for each class of our system such that if these forms could be used to add more functionality in a more convenient way and if something went wrong we could easily revert the operation and go back to a most recent stable version. We have done version control by using Github.

WordClue and PictureClue is identified as the stable intermediate forms. These intermediate forms can be evolved to build a clue system consisting of separate clue class for different types of word clues and picture clues.

5.1.2 OO Principles

Abstraction :

Our system is divided into individual cohesive components and each sub system focuses only on the essential behavior of the object and hence separates the object's essential behavior from implementation details.For example, the panel controller has method to addPanel(),

addFrame().These functionality is used whenever a panel switch is required and thus provides a crisp boundary between how it is implemented and its usage.

Abstraction can also be seen in our Clue generation design. An abstract class Clue is designed and depending on the type of clue that needs to be displayed,appropriate functionality is invoked.

Inheritance :

Inheritance principle is used for Clue system design. We have used the extends[“IS-A”] functionality for the Clue class

Picture Clue “IS - A” EmptyClue

Word Clue “IS - A” EmptyClue

EmptyClue “IS -A” Clue

Encapsulation :

Information about game level that should be hidden from unauthorized access are

1.Timer, lock code, image directory

2.Panel information

These data is hidden in our system by storing these information as private members of the level class and Frame class.

Modularity :

The game system is decomposed into individual components: Panel Controller, P0, P1, P2, P3, Frame, Process. Each components can be independently created and used and they are loosely coupled to each other.

5.1.3 5Cs

In this part, we give some examples to show how we address the 5Cs of class design.

Cohesion

Class P1 is functionally cohesive. It speaks only about the functionality of game panel and how to control the individual components within the game panel. Likewise P0, P2, P3 is functionally cohesive as each class only deals with functionalities within respective panel. As for other functions such as panel switch, we have implemented an observer pattern to separate this functionality from the implementation of each panel.

Completeness

Each Class is complete and includes all the necessary functionalities. For example, to our best, the game panel (P1) has been designed to support all game panel related functionalities within it.

Consistency

We have named each functions and each attributes in the game panel according to their roles in the implementation of the class.

Convenience

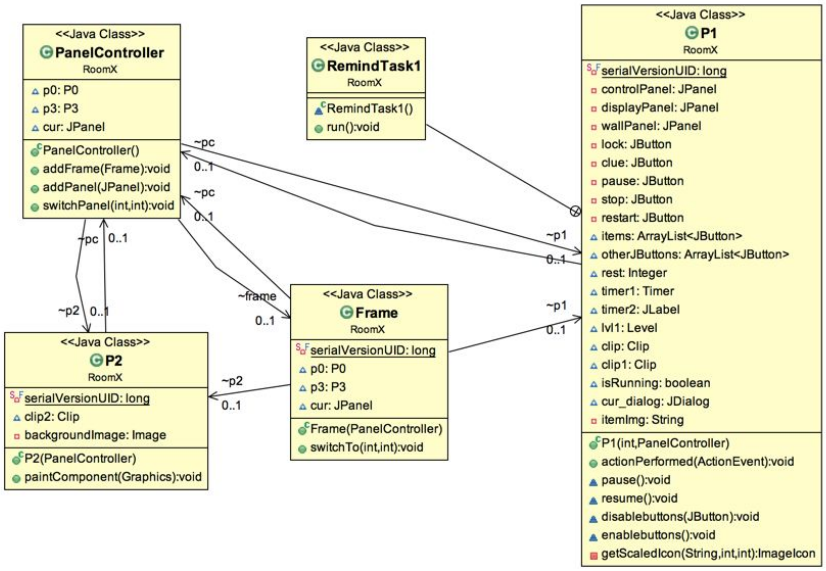
There is no complex function in the design of the game panel, each function will carry out some simple task. The only big function is the actionPerformed function in P1, which describes the behavior of each button when the user click on it and uses a switch statement to control which one will be called at runtime based on the user's choice.

Clarity

All classes play a clear role in the design of the application. Process is our main game process. P1, P2, P3, P4 works as different game panel, which will be launched on Frame and switched by putting a request to PanelController. Level provides all information about one level. EmptyClue extends the abstract class Clue and serves as a super clue type which contains some content to display. Then WordClue and PictureClue extends it as different sub clue types.

5.1.4 Design Pattern

We have implemented the observer pattern in a reverse way (multiple publisher and one listener) for switching between panel on the main window (JFrame). Each panel will put a request to the PanelController which serves as an exchange, and PanelController will talk directly to the Frame (an object which extends JFrame) of the game to do the switching work. We have all the 4 panels as multiple publishers, and the Frame object as the only one listener.



5.2 Improvement

Later we would like to decomposition the game panel for the future use. For example, the control panel with pause/stop/resume buttons on it may be able to be reused for another game for future, if we could separate its implementation from the game panel. The reason why we did not implement it here that way is that the decomposition of the game panel will lead to a more complex communication which costs more effort and time than encapsulating everything inside the implementation of the game panel.

5.3 New Features

5.3.1 Sound Effect Clue

In the future, we would like to add more type of clues. One of the most potential types is a sound-effect clue. User may hear some dialog or some music and get some clue by analyzing the sound they hear.

5.3.2 Player Ranking

Currently, the game does not save any user information. In the future, user information could be saved and used to support the implementation of a top list of all players. Players can check their records and see their ranking in the player ranking list.