¬Expected Magic

Requirements and Analysis Document

Version 1.0

Agrell Robert, Larborn Sofia, Runvik Arvid, Tomasson Rasmus

2017

Software Engineering
Chalmers University of Technology
Sweden

Contents

1 Introduction **2**
1.1 Definitions, Acronyms and Abbreviations 2

2 Requirements **3**
2.1 Graphical User Interface 3
2.2 Functional requirements 3
2.3 Non-Functional Requirements 4
2.3.1 Usability 4
2.3.2 Performance 4
2.3.3 Supportability 4
2.3.4 Implementation 4
2.3.5 Packaging and Installation 4
2.3.6 Testability 4

3 Use Cases **5**
3.1 Use Case Diagram 5
3.2 Use Case Listing 5
3.2.1 Set Up New Game 5
3.2.2 Set Options 6
3.2.3 Exit 6
3.2.4 Select Song 7
3.2.5 Start Game 7
3.2.6 Set Players 8
3.2.7 Strike Note 8
3.2.8 View Results 9
3.2.9 Pause Game 9
3.2.10 Resume Game 10

4 Domain Model **11**
4.1 Class Responsibilities 11

1 Introduction

Music has always been an important element in our society and in recent years games like Guitar Hero and Rock Band have flourished [1]. Our vision is to create a similar cooperative entertaining experience where players can have lots of fun on a casual level.

Unlike Guitar Hero and Rock Band whose playability depends on whether or not the user wants to spend lots of money on gimmicky plastic guitars and other accessories, Unexpected Magic will incorporate the fun and entertaining cooperative aspect of a music party game while at the same time keeping the price at a reasonable level, being no charge at all.

Much like the educational music teaching application Synthesia [2], our game will mimic the way notes fall down towards the bottom of the screen, while adding the element of a game by letting the user interact directly with the falling notes when they reach a certain point and thus gain score based on accuracy. The screen will be divided into 12 segments, representing an octave, and when playing cooperative mode every user will be responsible for his/her own set of notes while all the players simultaneously play the same song.

In short the user sees a falling note, they push a key on the keyboard and the application responds by playing the corresponding note. If the played note was correct the user will receive points, or if the played note was wrong the user will lose points.

The application will be solely developed as an offline desktop application for Windows, Linux and Mac. With our described game mechanics hand in hand with charming pixel graphics we think this application will be an entertaining musical experience for anyone interested in music and/or games.

1.1 Definitions, Acronyms and Abbreviations

- ¬*Expected Magic* (Unexpected Magic) - The application name.
- *Voice* - Derived from the musical term, each player plays one of the song's voices.
- *Note* - an object containing information about a note.
- *Pianoroll* - derived from the name of the music storage medium. Just as a piano roll was originally a long strip of paper with notes marked on it, the pianoroll in the ¬Expected Magic game can be described as a "strip" of coordinates with note game entities placed on it, and is displayed during gameplay as the camera moves over it.
- `.uxm` - a file format created specifically for the ¬*Expected Magic* project, for defining a song in terms of musical information as well as metadata.
- *libGDX* - An external library for game development.

2 Requirements

2.1 Graphical User Interface

The user interface will consist of a middle area called the *pianoroll* where all the notes appear and fall towards the bottom, and a bottom area where the player interacts with the falling notes. The bottom area will also display information about the participating players. There will be a top area showing song title and beats per minute (BPM) for the current song. The graphical interface of the final product differs a bit from the initial sketch, as seen below.

Figure 1: The initial sketch of the in-game graphical user interface

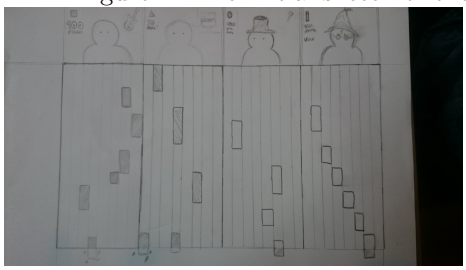
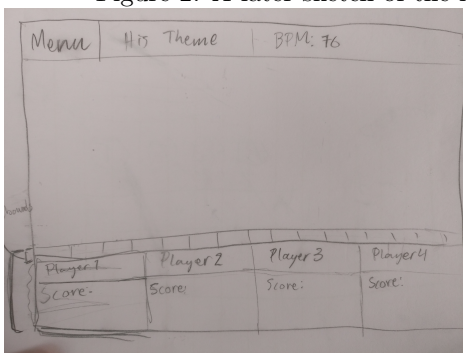


Figure 2: A later sketch of the in-game graphical user interface



In the final design there are no avatar images for the in-game players, and instead of each player having their own octave area, the screen is divided into an octave of 12 notes so that all the players share the same area.

2.2 Functional requirements

- Options - view and set options.
- Set up a new game round - pick song, pick number of players, name players.
- Play a song/round - play notes, receive score.

2.3 Non-Functional Requirements

2.3.1 Usability

¬*Expected Magic* will attempt to be conventional in its game design and graphical user interface design. The gameplay mimicking previously created music games will be intuitive for experienced players. This will be achieved with likeness towards actual instruments, especially piano.

2.3.2 Performance

The goal for ¬*Expected Magic* is that the application should be able to play a song without any sound delay or visual delay to allow the player to enter a state of *flow*, thus keeping the experience immersive and focused on the musical aspect of gameplay.

2.3.3 Supportability

¬*Expected Magic* will as previously mentioned run on Windows, Linux and Mac OS. Possibly support for multiple keyboards or similar controls will be implemented.

2.3.4 Implementation

The application will be created using the Java environment. The external library libGDX will be used for aiding with functionality on the aspect of game logic. Pixel art in the game will be created using the open-source GNU Image Manipulation Program (GIMP). MIDI sounds will be handled by the built-in sound library in Java.

2.3.5 Packaging and Installation

The application will be packaged in a zip-archive containing all the needed resources such as images and `.uxm`-files as well as the application itself as a JAR-file.

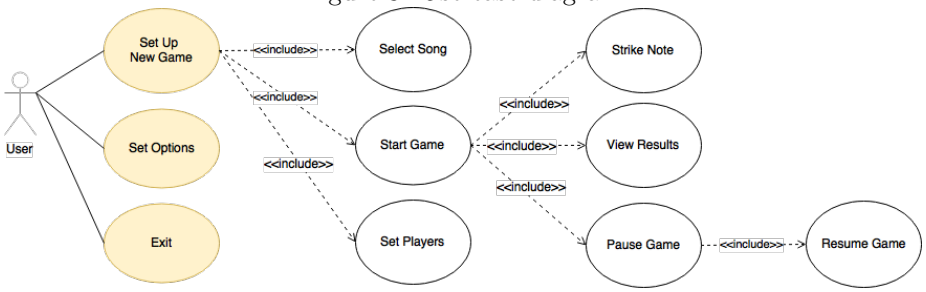
2.3.6 Testability

There should be automated test verifying all use cases unrelated to external libraries.

3 Use Cases

3.1 Use Case Diagram

Figure 3: Use case diagram



Upon starting the application, the user is presented with the choices of either setting up a new game, setting options, or exiting the application.
When setting up a new game, the user can select a song to play, set the players, and start the game.
After the game is started, the player(s) can strike notes (using the keyboard), pause the game, and view results.
If a player pauses the game, the game can be resumed.
See use case listing below for further details about each use case.

3.2 Use Case Listing

3.2.1 Set Up New Game

Summary: Allows the player to set up a new game round.
Priority: High
Extends:
Includes: Select Song, Start Game, Set Players
Participators: User

Set Up New Game		
	Actor	**System**
1	Click the "NEW GAME" button	
2		Show the "New Game" screen; Present the user with choices (see "Select Song", "Set Players" and "Start Game"

3.2.2 Set Options

Summary: Allows the player to change settings.
Priority: Low
Extends:
Includes:
Participators: User

Set Options		
	Actor	**System**
1	Click the "OPTIONS" button	
2		Show the "Options" screen
3	View/edit options	
4	Press "MAIN MENU" button	
5		Apply options; Return to main menu

3.2.3 Exit

Summary: Allows the player to exit the application.
Priority: High
Extends:
Includes:
Participators: User

Exit		
	Actor	**System**
1	Click the "EXIT" button	
2		Close the application

3.2.4 Select Song

Summary: Allows the player to select which song will be played.
Priority: High
Extends:
Includes:
Participators: User

Select Song		
	Actor	**System**
1	Click the drop down menu	
2		Display list of available songs
3	Click song title	
4		Set selected song; Collapse list

3.2.5 Start Game

Summary: Allows the player to start the game.
Priority: High
Extends:
Includes: Strike Note, Pause Game, View Results
Participators: User

Start Game		
	Actor	**System**
1	Click the "PLAY" button	
2		Start the game and show the "In Game" screen
3		(See "Strike Note", "View Results" and "Pause Game")

3.2.6 Set Players

Summary: Allows the user to set the number of players and their names.
Priority: High
Extends:
Includes:
Participators: User

Set Players		
	Actor	**System**
1	Select number of players	
2		Set selected; If 1 or more players, also display editable default player names
1	Possibly edit player names	
2		If names were edited, set player names

3.2.7 Strike Note

Summary: Allows a player to play a note.
Priority: High
Extends:
Includes:
Participators: Players

Strike Note		
	Actor	**System**
1	Press a note key	
2		Start playing corresponding sound; Start increasing/decreasing score
1	Release the key	
2		Stop playing the sound; Stop increasing/decreasing score

3.2.8 View Results

Summary: Allows the players to view their scores.
Priority: Medium
Extends:
Includes:
Participators: Players

View Results		
	Actor	**System**
1		Finish song
2		Show the "Score" screen
3	View the results	

3.2.9 Pause Game

Summary: Allows a player to pause the game.
Priority: Low
Extends:
Includes: Resume Game
Participators: Players

Pause Game		
	Actor	**System**
1	Press the *pause* key	
2		If game is running, pause game

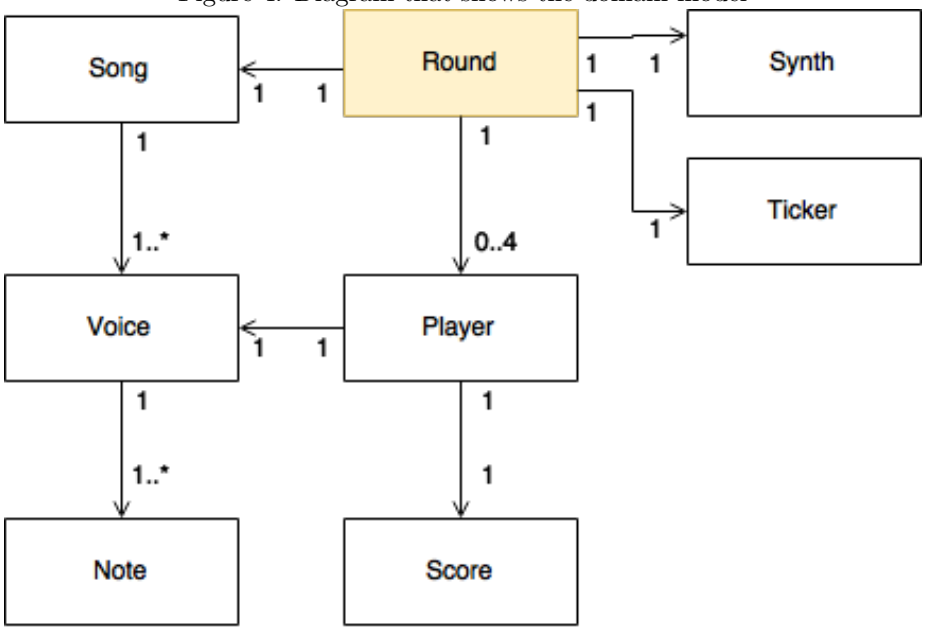
3.2.10 Resume Game

Summary: Allows a player to resume the game
Priority: Low
Extends:
Includes:
Participators: Players

Resume Game		
	Actor	**System**
1	Press the *pause* key	
2		If game is paused, un-pause game

4 Domain Model

Figure 4: Diagram that shows the domain model



4.1 Class Responsibilities

- `Player` - An in-game player. Has a name, a score and a voice.
- `Song` - A song with metadata, made up of voices.
- `Voice` - A voice made up of notes.
- `Note` - A note that has a pitch and a note value.
- `Score` - A score sheet that holds information about *score* and *streak*.

References

[1] A. Webster, "Roots of rhythm: a brief history of the music game genre." [Online]. Retrived from: `https://arstechnica.com/gaming/2009/03/ne-music-game-feature/`, April 2009.

[2] "Synthesia." [Online]. Retrived from: `http://www.synthesiagame.com/`, May 2017.