¬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

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, every player plays their voice of the song.
- *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 user interacts with the falling notes. The bottom area will also display information about the players. Lastly there will be a top area showing song title and beats per minute for the current song. The final product is a bit modified in comparison with the initial graphical sketch as seen below.

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

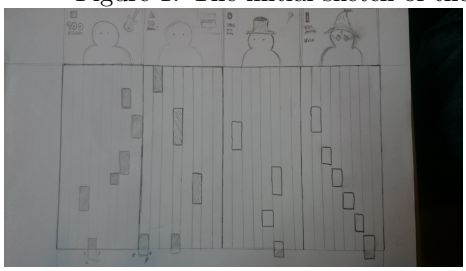
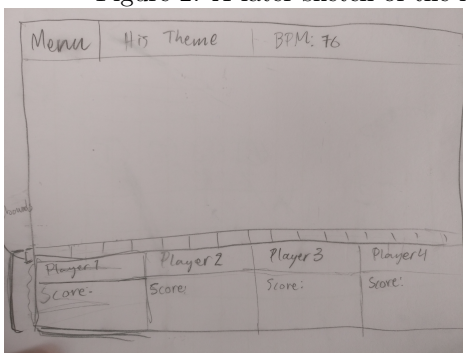


Figure 2: A later sketch of the in-game graphical user interface



The final design differs a little from the initial sketch. 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 where 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.

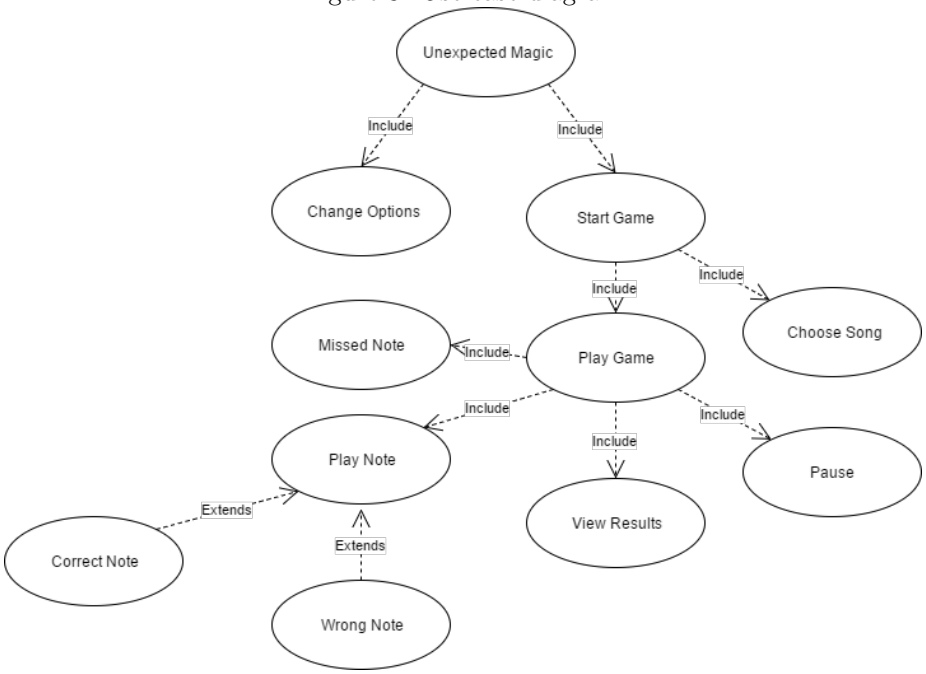
2.3.6 Testability

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



3.2 Use Case Listing

Start Game		
Step	**User**	**System**
1	Click *New Game* button	
2		Display *New Game* screen
3	Click *Play* button	
4		Start game

Choose Song		
Step	**User**	**System**
1	Click drop down menu	
2		Display list of available songs
3	Click song title	
4		Set selected; Collapse list

Play Game		
Step	**User**	**System**
1		Start game
2 Plays Note	See "Play Note"	
3 Misses Note	See "Miss Note"	
4 Pauses Game	See "Pause Game"	
5		Finish song
6	See "View Results"	

Play Note		
Step	**User**	**System**
1	Press key(s)	
2		Determine whether correct or not
3 Is Correct	See "Correct Note"	
4 Is not Correct	See "Wrong Note"	

Correct Note		
Step	**User**	**System**
1		Start playing note; Increase *streak*
2		Continuously award points
3	Release key	
4		Stop playing; Stop awarding points

Wrong Note		
Step	**User**	**System**
1		Start playing note; Continuously remove points
2	release key	
3		Stop playing; Stop awarding points

Miss Note		
Step	**User**	**System**
1		Display note(s)
2	Do nothing	
3		Determine that note was not played
4		Break *streak*

Pause Game		
Step	**User**	**System**
1	Press *pause* key	
2		Pause game
3	Press *pause* key	
4		Unpause game

View Results		
Step	**User**	**System**
1		Finish song
2		Show score screen
3	Press *Main Menu* button	
4		Return to Main Menu

Change Options		
Step	**User**	**System**
1	Press *Options* button	
2		Show *Options* screen
3	Edit options	
4	Press *Main Menu* button	
5		Apply options; Return to main menu

4 Domain Model

Figure 4: Diagram that shows the domain model

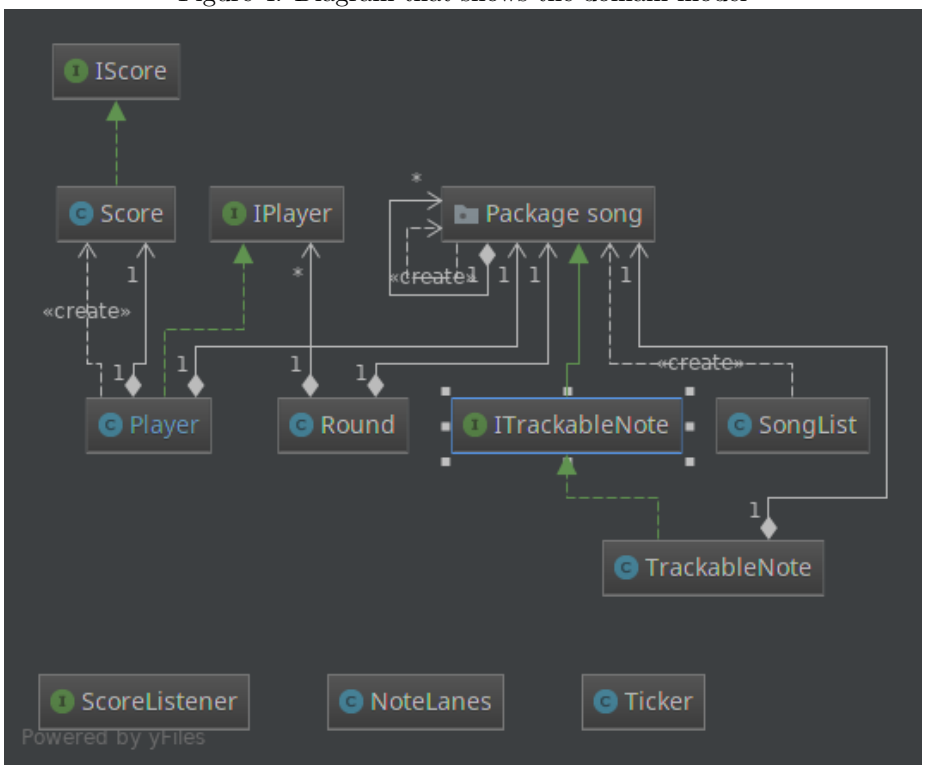
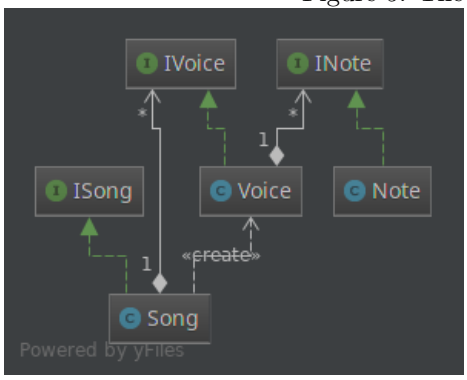


Figure 5: The song package



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/`, 2017.