Tempo Blast

Game Design Document

Author: Dominik Kowalczyk

Table of Contents

Concept

Idea

Game genre: Rhythm game

The game is inspired by the launchpad instrument, which is a grid of unlabeled buttons playing different sound samples. The goal is to imitate and simplify the experience of playing songs using one.

Example real life performance on YouTube: Link

Another vital aspect of the game is a system that lets the user play his favorite songs, and automatically generates the needed data for it.

Features

Prototype 1

Obviously implementing game features is a very time consuming process, here is a list of features that will be included in the first playable prototype:

- Playing songs
  - 3x3 grid of buttons
  - Visual indicator of good performance (high note streak)
  - Timer
- Player progress
  - 3-star rating system
  - ratings remembered for each song
- Song creation
  - custom algorithm that has to be manually executed by the developer

Coming next

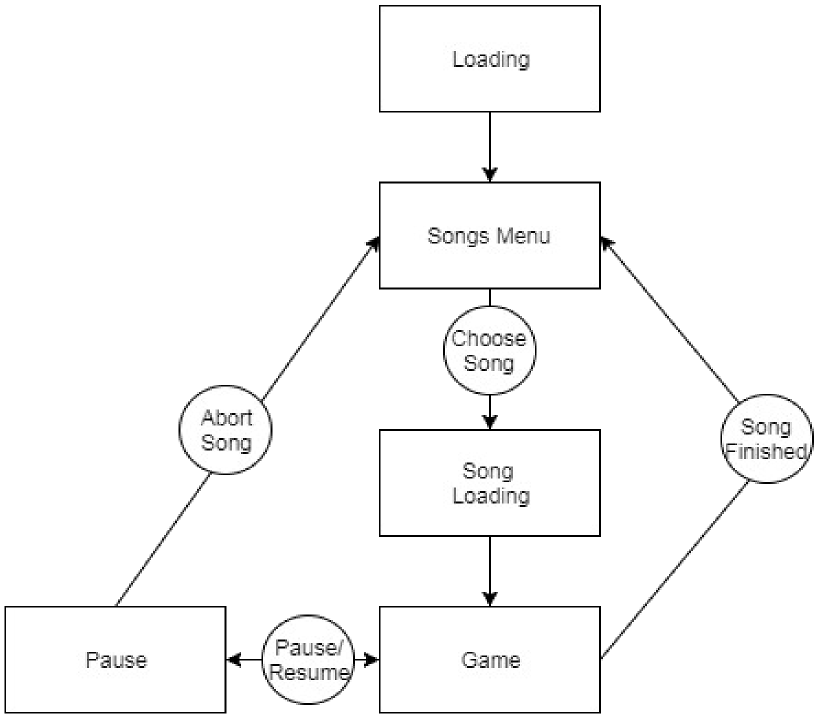
- Difficulty levels
- Automated user-friendly custom song creation
- Tutorial
- Graphical polish

Implementation

Technology choice

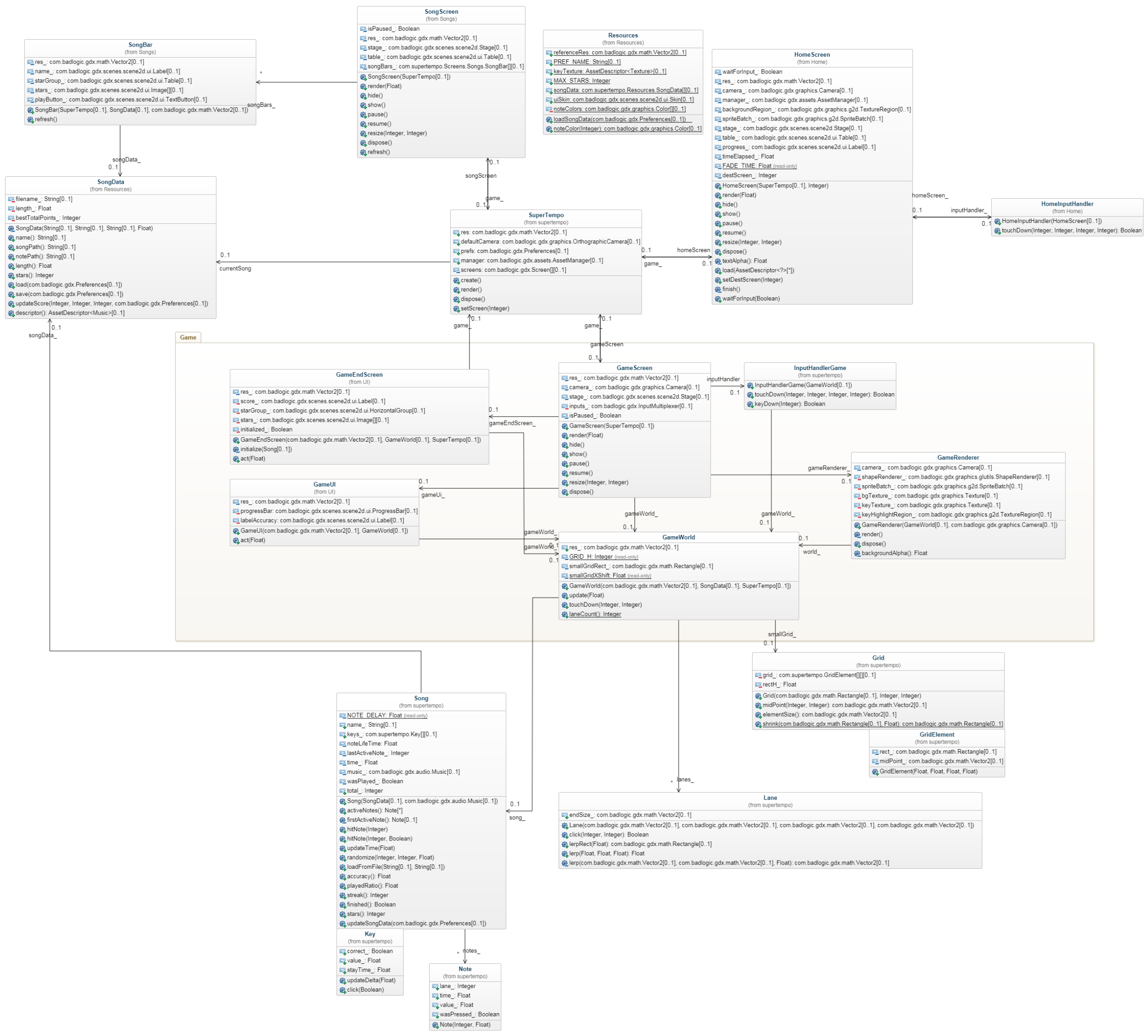
The target platforms are: Android, iOS, Desktop. The game is written in Java using libGDX library. The song processing system is prototyped using Matlab and further on will be reimplemented in Java and integrated into the game.

Screen Flow

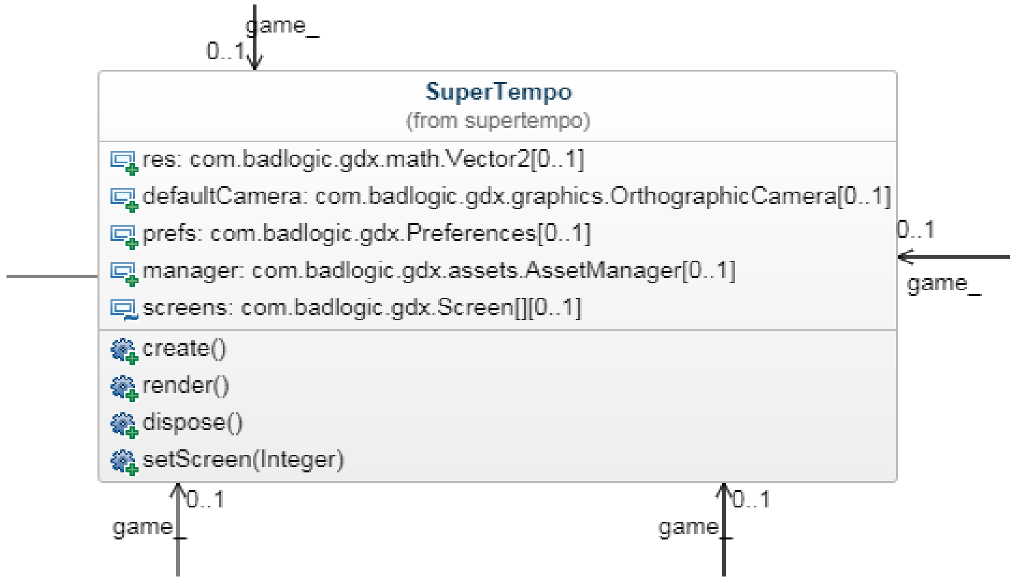


Code architecture

Presented below is a full class diagram for an overview look. Its individual parts will be discussed in more detail further on.



Main class

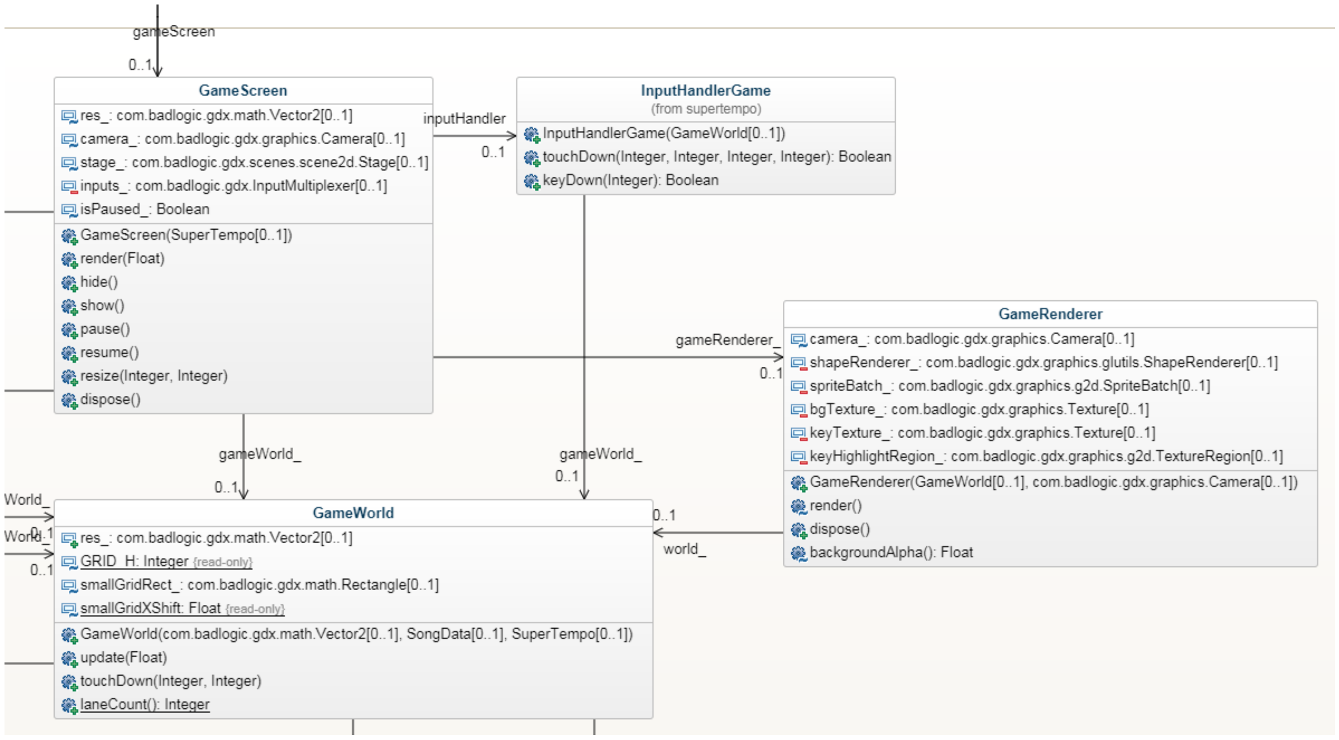

game_
0..1
SuperTempo
(from supertempo)
res: com.badlogic.gdx.math.Vector2[0..1]
defaultCamera: com.badlogic.gdx.graphics.OrthographicCamera[0..1]
prefs: com.badlogic.gdx.Preferences[0..1]
manager: com.badlogic.gdx.assets.AssetManager[0..1]
screens: com.badlogic.gdx.Screen[][0..1]
create()
render()
dispose()
setScreen(Integer)
0..1
game_
0..1
game_
0..1
game_

The main class extends the libGDX Game class and is kept simple. It manages switching between different screens.

For now it also contains some general global game data like resolution, camera, persistent data and assets, it might be refactored into some globally accessible class later so that accessing these resources does not require having a reference to the main class instance.

UPDATE the main class now utilizes the Singleton pattern, the pictures illustrate the previous version though.

Game Screen


gameScreen
0..1
GameScreen
res_: com.badlogic.gdx.math.Vector2[0..1]
camera_: com.badlogic.gdx.graphics.Camera[0..1]
stage_: com.badlogic.gdx.scenes.scene2d.Stage[0..1]
inputs_: com.badlogic.gdx.InputMultiplexer[0..1]
isPaused_: Boolean
GameScreen(SuperTempo[0..1])
render(Float)
hide()
show()
pause()
resume()
resize(Integer, Integer)
dispose()
inputHandler
0..1
InputHandlerGame
(from supertempo)
InputHandlerGame(GameWorld[0..1])
touchDown(Integer, Integer, Integer, Integer): Boolean
keyDown(Integer): Boolean
gameRenderer_
0..1
GameRenderer
camera_: com.badlogic.gdx.graphics.Camera[0..1]
shapeRenderer_: com.badlogic.gdx.graphics.glutils.ShapeRenderer[0..1]
spriteBatch_: com.badlogic.gdx.graphics.g2d.SpriteBatch[0..1]
bgTexture_: com.badlogic.gdx.graphics.Texture[0..1]
keyTexture_: com.badlogic.gdx.graphics.Texture[0..1]
keyHighlightRegion_: com.badlogic.gdx.graphics.g2d.TextureRegion[0..1]
GameRenderer(GameWorld[0..1], com.badlogic.gdx.graphics.Camera[0..1])
render()
dispose()
backgroundAlpha(): Float
gameWorld_
0..1
gameWorld_
0..1
0..1
world_
World_
World
0..1
0..1
GameWorld
res_: com.badlogic.gdx.math.Vector2[0..1]
GRID_H: Integer {read-only}
smallGridRect_: com.badlogic.gdx.math.Rectangle[0..1]
smallGridXShift: Float {read-only}
GameWorld(com.badlogic.gdx.math.Vector2[0..1], SongData[0..1], SuperTempo[0..1])
update(Float)
touchDown(Integer, Integer)
laneCount(): Integer

As all screens in the game it implements the libGDX Screen interface. The GameScreen is the most complex of screens because it is where the gameplay happens. That's why it is split into separate independent components:

- Game World
- Game Renderer
- Game Input Handler

Game World is a class representing the actual game logic and data, it is agnostic of other classes and used as an information source by them, it's structure will be discussed later.

Game Renderer is a class that uses Game World information to display the game on the screen, or more specifically, it renders the core gameplay elements like notes and keys, the UI uses Scene2D and is moved to another class.

Game Input Handler handles gameplay related input. Currently it allows hitting notes using touch controls on mobile devices or a mouse, or playing with a numpad using a keyboard.