System design document for Dance Fans

TDA367

Hedy Pettersson, Jakob Persson, Joar Granström, David Salmo, Johan Berg, Isabelle Ermeryd Tankred

Oct 8 2021

Version 2

1 Introduction

Dance fans is a strategy game where two players play as dancers who perform dance moves to get the most dance fans on the dancefloor. An object oriented programming approach was taken to implement the game, as is described in more detail in this document.

1.1 Definitions, acronyms, and abbreviations

GUI = Graphical User Interface.
MVC = Model-View-Controller. A design pattern which is commonly used if you develop applications with a GUI. It separates data and logic (model) from the view and controller.
LibGDX = A library for developing games in Java.
TexturePacker = A tool for creating sprite sheets.

Game specific definitions
Player = a user of the program and their corresponding representation in-game.
Main dancer = The dancer the user controls to make dance moves in order to gain dance fans and win the game.
Dance fan = One of the dance fans, which join the dance floor inspired by the Main Dancers dance moves. The player with the most dance fans at the end of the game wins.
Dancer = A dance fan or a main dancer
Dance floor = The world in which a dancer can be located and move around within, in the game.
Dance move = A pattern and a set number of steps a player can move across the dance floor. The pattern tells us how the dance floor will change after the dance move is performed. The dance move inspires dance fans to join in on the player's dance team. These dance moves are represented in the game as cards in a deck of cards (a main dancer's repertoire of dance moves).
Dance pattern = The pattern part of the dance move.
Card = Representation of dance moves in the game, to help the player choose strategy.
Preview = A special view of what the dance floor would look like if the player performed the dance move of the currently selected card at the location the selection marker is currently located on the dance floor.

2 System architecture

We use the MVC pattern.

Parts in the model:

DanceFloor contains a list of tiles and the order of them. It is also responsible for changing the details of the specific tiles.

TIles contains information about what is currently standing on the tile, and can tell us what the name of that thing is.

CardDeck contains multiple lists of card:s, and in what order they are in. The different lists are the currently opened cards, the used cards, the unused cards, and a copy of the original list which was used to create the deck in the first place. It can shuffle card:s in a list, and also handles moving the cards between the different lists.

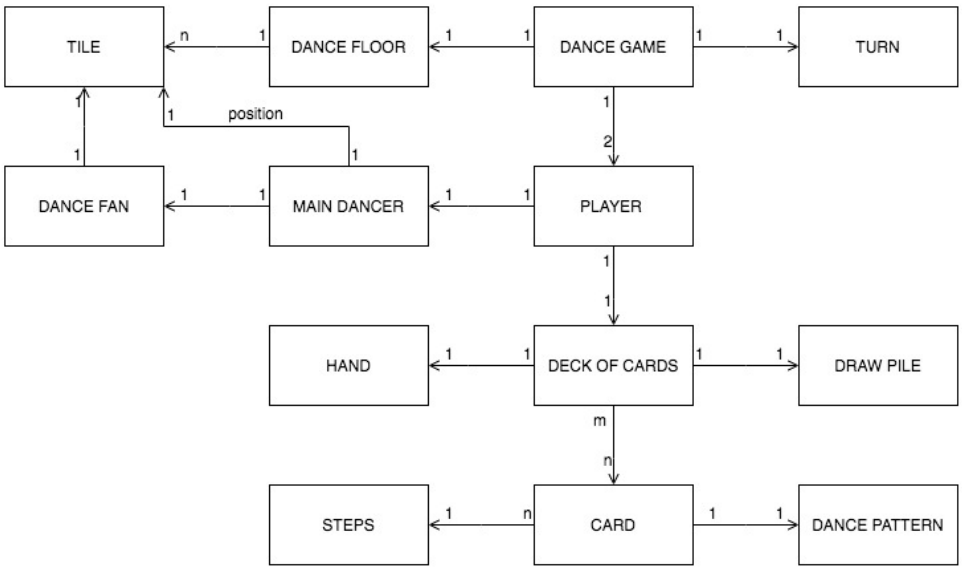
Card contains a matrix which is used to represent the dance pattern, an int for the amount of steps and an image representation of the card.

Main dancer contains a String for its name and two ints for its positions, one for which tile it is currently on, and one for showing the preview.

Dance fan has a String for its name and an int for its position.

Player has a CardDeck, a main dancer, and a dance fan.

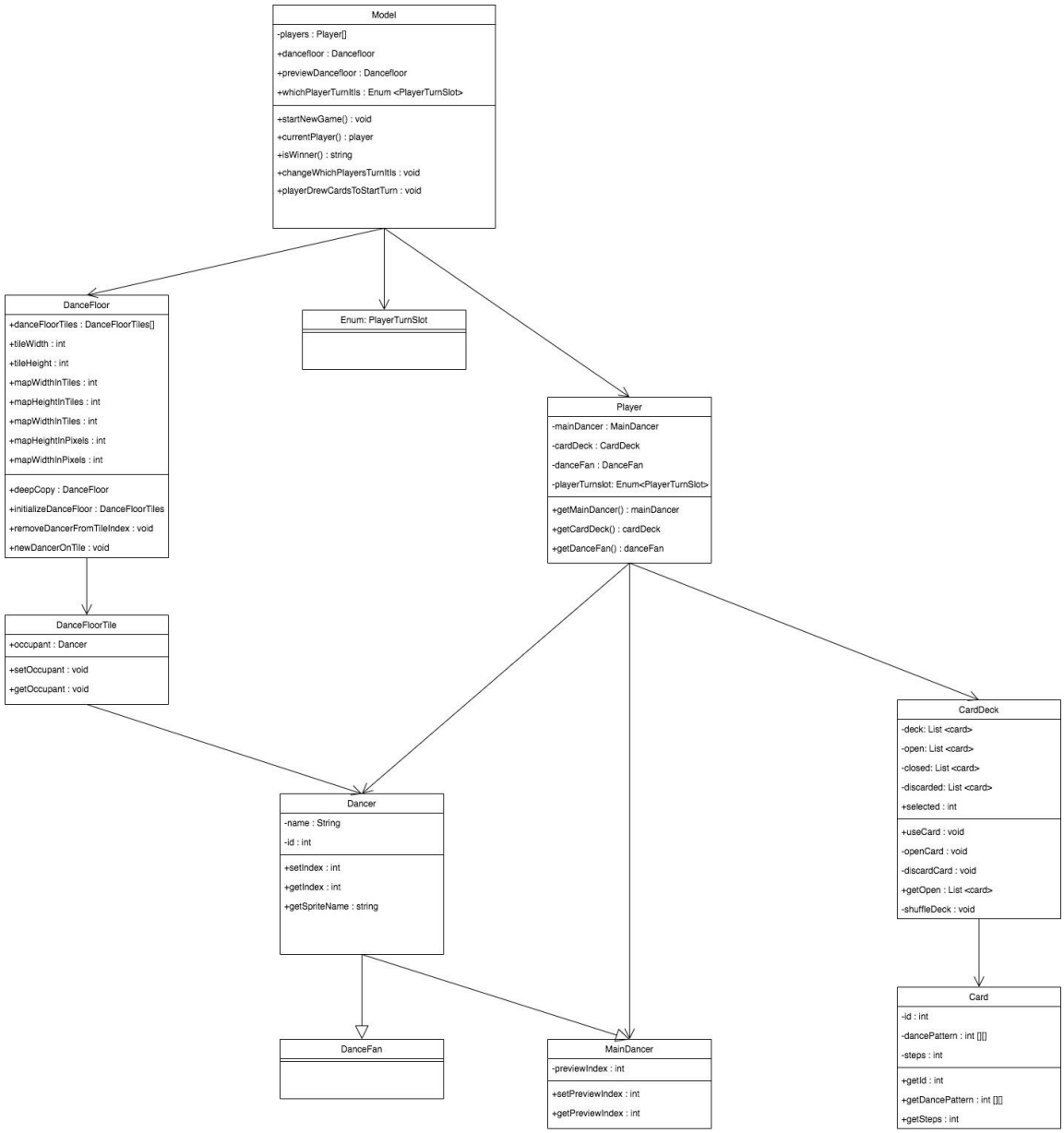
3 System design



We have one package, the model. The model does not access the view or the controller.

The view accesses the model and uses information to display the right information like which players turn it is, which if someone has won yet and through the dancefloor the characters and the dancefloor itself is drawn.
The controller mostly calls on methods in the model. We would like to change to using listeners for some parts.

No design patterns have been implemented yet, but we plan to use the listener pattern for the key inputs.



4 Persistent data management

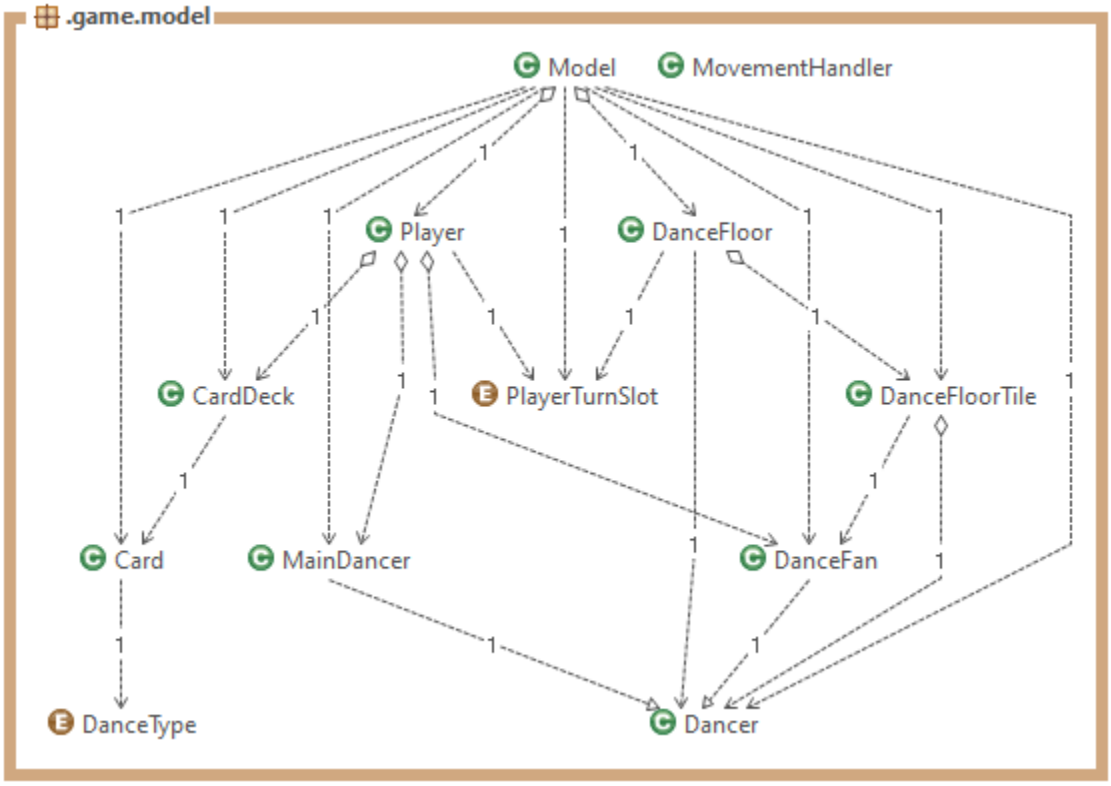
The game does not need the players to have profiles or similar data stored.

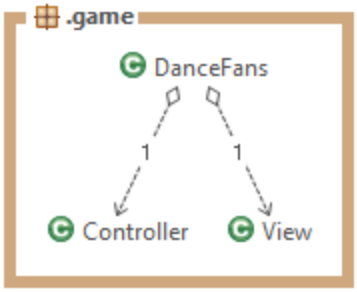
Icons and images have been saved in sprite sheets using TexturePacker. These are paired up with text files which describe the location of each image and using atlases this makes it easier to store many images, and will hopefully make it easier to add many new icons and images.

5 Quality

So far the code has mostly been tested by looking at what is displayed. There are some JUnit tests, but more are needed.

Running a dependency analysis with Stan4j returns the following graph.





Current issues:
Game does not end.

Some analytical tools have been used (STAN), but we should use more.

Header comments have not been written, and should be added as well.

5.1 Access control and security

Not Applicable

6 References

LibGDX
https://github.com/rafaskb/awesome-libgdx#readme

TexturePacker
https://www.codeandweb.com/texturepacker

STAN
http://stan4j.com/

List all references to external tools, platforms, libraries, papers, etc. The purpose is that the reader can find additional information quickly and use this to understand how your application works.