Example SDD.
http://www.cse.chalmers.se/edu/year/2017/course/tda367/lectures/MonopolySDD.pdf
This below is our SDD draft ↓

System design document for the ¬*Expected Magic* project

Version: 1.0

Date:

Authors: Rasmus Tomasson, Arvid Runvik, Robert Agrell, Sofia Larborn

1. Introduction

1.1 Design Goals
Goals:

- no sound delay or visual delay during gameplay
- Extendable, modular design, overall good code structure
- Testable

1.2 Definitions, acronyms and abbreviations
Just copy definitions from RAD?

- libGDX
- Ashley
- Unexpected Magic/¬Expected Magic
- .uxm
-

2 System Architecture
The application implements the Entity-Component System pattern.
smth about libGDX framework---

It runs on a single desktop computer. (say something about multiple external keyboards for input?)

The application consists of two top level packages; "model", unaware of libGDX, "gameEngine", using model to run the game with libGDX and "services", -----smth about it------. The "model" package contains classes defining how a game round should run and for counting score, and a "song" package responsible for song handling. The "gameEngine" package holds all libGDX-related code that drives the game. (For more detail, see "3.n The gameEngine package".) "services" contains/holds/handles/whatever---------. (For more detail, see "3.n The "services" Package".)

packages (the arrows show dependencies):
{image}

2.1 General observations
.uxm files?
other notable things?

2.n Composition Over Inheritance
When it comes to organizing the game objects, inheritance is a straightforward way to let an object use and override the behavior of a base class [jv-inherit]. In Java, only single inheritance is permitted [jv-mult-inh]. This eliminates the risk of ending up with the multiple inheritance problem sometimes referred to as "fork-join inheritance" or "Diamond of Death", where a class is derived from two or more classes that share a common ancestor. [comm-anc-dil] However, only being able to inherit from one class can give rise to other problems.

In the case of Unexpected Magic, the main game objects are the notes, and plans for further development include different kinds of notes as well as other on-screen entities like player representations and enemies. For example; the Note object, Player object and Enemy object could all be derived from a class game object. What then, if a hostile note was to be added? Would it inherit from note object or enemy object?
Since organizing game objects in inheritance hierarchies is proven to be a tedious work likely to end up in limiting rigidity and conflicts, the Unexpected Magic project instead follows the principle of "composition over inheritance". This principle is applied through the use of an Entity-Component system called Ashley.

2.n Entity-Component System
The traditional way of developing games has been to represent entities with objects, which inherit from appropriate superclasses. A common design pattern for use with this is MVC (Model-View-Controller). Therefore, this method was initially considered when planning the implementation. However, relatively quickly, it was noted that it was a disadvantage to have separate modules for view and control as well as to structure the parts in Inheritance Hierarchies, as this system restricted flexibility and made the code difficult to understand and organize.

The MVC pattern is about splitting the application into a Model that manages the problem domain, a View that presents the model for the user and a Controller that communicates with Model and View. A strength of MVC is that the structure enables painless modification of views without affecting the model. [mvcp] However, this flexibility does not give significant benefit to the implementation of Unexpected Magic, as the problem domain's abstract representation and the presented concrete model in this case are very closely related. The kind of separation provided by MVC was considered unnecessary and likely to become tedious due to the fact that the user interface is very similar to the model and that the game is run according to the sequence pattern Game Loop [gameloop] in which the game logic processes the game's different entities in each iteration.

In order to solve this, the principle of Composition Over Inheritance was tried out, which proved more suitable for the Unexpected Magic application. This principle is applied in Unexpected Magic by structuring the application's various objects according to and managing them using the software architecture pattern Entity-Component System. This is done using Ashley, a Java Entity System supported by libGDX. [ash-gith]

[mvcp]
https://developer.apple.com/library/content/documentation/General/Conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html
[gameloop] http://gameprogrammingpatterns.com/game-loop.html
[comm-anc-dil] https://people.cs.kuleuven.be/~eddy.truyen/PUBLICATIONS/DAW2004.pdf 2 The common ancestor dilemma
[jv-mult-inh] https://docs.oracle.com/javase/tutorial/java/IandI/multipleinheritance.html
[jv-inherit] https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html
[ash-gith] https://github.com/libgdx/ashley

http://www.gamearchitect.net/Articles/GameObjects1.html

3 Subsystems decomposition

3.n The "gameEngine" package

The "gameEngine" package contains the following packages:

· components
· managers
· scenes
· screens
· systems

3.n.n components

Contains component classes that implements the libGDX Ashley interface "Component". The sole purpose of a component is to hold a single kind of data, for example a PositionComponent could hold a position, which might be implemented as two float values for x and y. Instances of components can be grouped together to form an "Entity". The components used in the Unexpected Magic game are as follows:

· PositionComponent - Floats for x and y position.
· VelocityComponent - Ints for x and y position.
· NoteComponent - A single instance of the Unexpected Magic "Note" class.
· SpriteComponent - A drawable libGDX "Sprite", an image that can be drawn on the screen.

3.n.n managers

Contains classes that manage different ....<somethings>.... The managers used in the Unexpected Magic game are as follows:

· EntityManager - Handles entity creation and destruction of entities.
· EntityFactory - Used by EntityManager. Constructs and returns entities.
· RoundManager - Handles a game round. Manages a Round, an EntityManager and a Ticker.
· SoundManager - Handles playing notes using the javax.sound.midi synthesizer.
· NoteThread - Plays notes. Used for playing notes in a voice that doesn't have a player assigned.
· KeyboardInputManager - Handles input from the keyboard.

3.n.n scenes

Contains classes that define scenes that can be rendered. The scenes used in the Unexpected Magic game are as follows:

· HUD - Defines the heads-up display that overlays the game visuals with labels.
· PianoRoll - Defines the area showing the piano roll with the falling notes.

3.n.n screens

Contains the screen classes controlling and drawing the different screens. All Unexpected Magic screens are derived from the libGDX "ScreenAdapter" class which implements the libGDX interface "Screen". The screens used in the Unexpected Magic game are as follows:

· AbstractScreen - Screen superclass, contains shared functionality, for example camera, viewport, resizing, etc.
· MainMenuScreen - The main menu screen, presenting the user with options such as playing, changing preferences, etc.
· NewgameScreen - Screen that is shown when the user choses to start a new game from main menu. Presents options for selecting song, players, etc.
· GameScreen - Screen that handles the in-game activity. On this screen the piano roll and heads-up scenes are drawn.
· OptionsScreen - Screen that shows options that the user can change. Can be reached from main menu.

3.n.n systems
Contains classes that contain the logic for processing game entities. The system classes are derived the libGDX Ashley "EntitySystem" class. The systems used in the Unexpected Magic game are as follows:

· MovementSystem - uses the PositionComponent and VelocityComponent of an entity. Calculates the new position using its VelocityComponent and sets the PositionComponent to the new position.

3.n The "services" Package
Handles file reading. Is in Unexpected Magic used for reading .uxm files. The service classes used in Unexpected Magic are as follows:
· FileReader - ……..<what does it do>

4 Persistent data management
Unexpected Magic does not currently have much persistent storage, but it does read files of a format called UXM. These files contain the title, rhythm and notes of a song, in a format that can easily be parsed by the game while also being easy for humans to read and write.
We also intend to add a persistent high-score list.

5 Access control and security
wut is dis??????

6 References
--