INM375
Games Development Process

Jacob Arbib
City, University of London, 2020-2021

KAE / Rhythm Adventure Game

Design Documentation

Contents

Introduction

Background

The **endless run** genre is still one of the most popular casual games genres, especially on mobile markets. Its very simple controls and objectives made it perfect for the pick-up and go formula of most players which cannot spend too much time on a game. Your objective is simply to have your character go as far as possible to get a high score by tapping and/or swiping your touch screen. Examples are *Temple Run*, *Subway Surfer*, *Jetpack Joyride* and **Flappy Bird**. The last one was a one-hit-wonder that immediately created a global phenomenon, even though the game remained on mobile stores for a very short while. In this game, you move to tap the screen to have your bird jump, while gravity pushes it down, and you try to avoid as most obstacles as possible to get a high score.

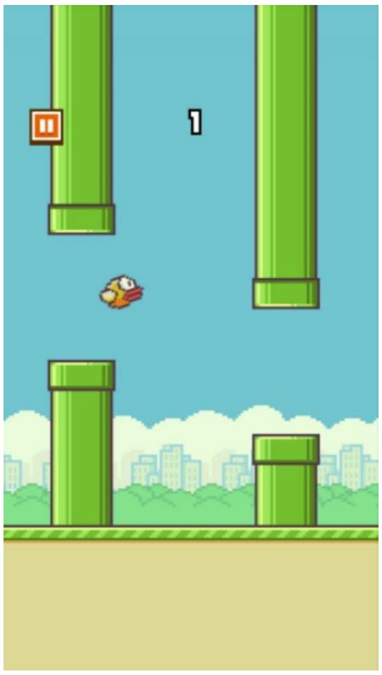


Figure 1: A screenshot of Flappy Bird

A completely unrelated genre is probably the **rhythm game**. Made famous by Japanese publisher *Konami* with its line of *BEMANI* arcade games, in a typical rhythm game you play along with a song and input the right commands timed with the music. Examples of classic rhythm games are *Dance Dance Revolution*, *Beatmania IIDX*, *Osu!*. The mobile market has had its fair share of famous rhythm games, like *Piano Tiles* and *Cytus*.

A fundamental feature of any good rhythm game is being *easy to learn, hard to master*, i.e. being pick-up and play in scope, but needing the commitment to become good at them. This idea, together with the hyper-casual gameplay of a game like Flappy Bird, is what we are exploiting for KAE.

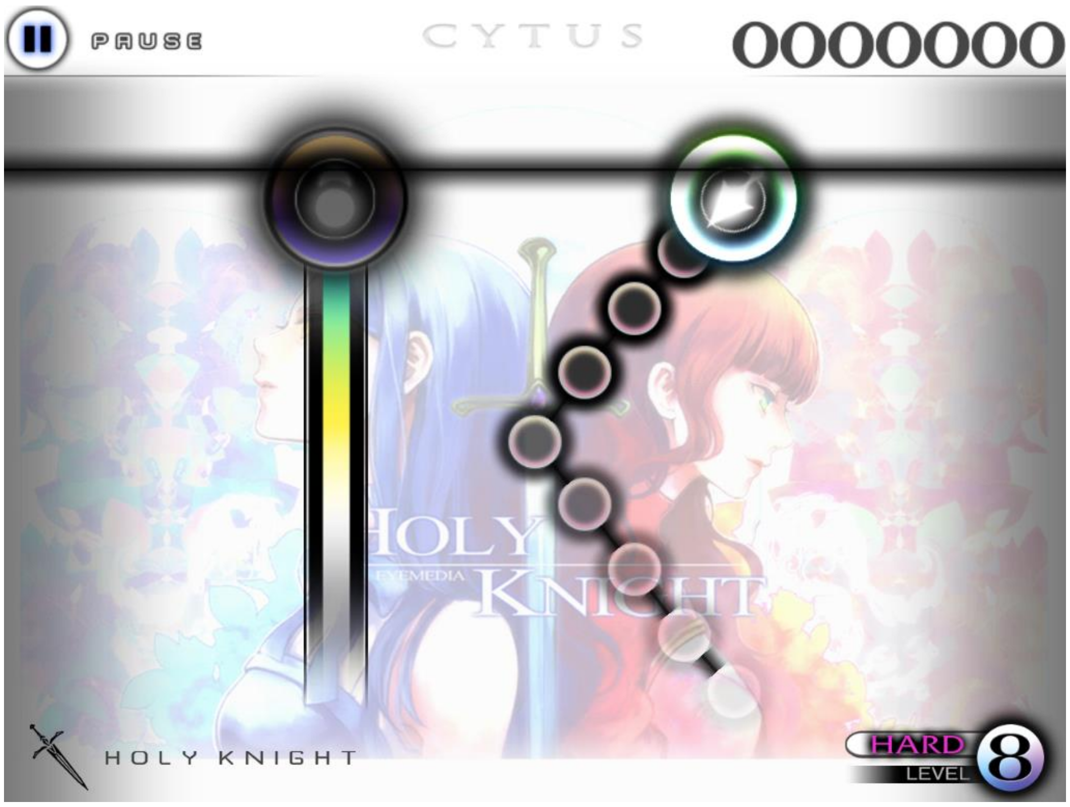


Figure 2: Gameplay of Cytus, on iPad

The idea

What if we could combine the casual-heavy style of an endless run game like *Flappy Bird* with the skill-based progression of a rhythm game?

KAE is a **rhythm-based casual adventure game**, based on this precise mixture. A game with Flappy Bird style physics and controls, but where each input needs to be timed with the currently playing song.

It will follow the story of KAE, a creature made of light separated from their family, which needs to return home. A level structure divided into chapters and boss fights will form the story frame of the game.

Target

The game is targeted at **young adults**; the simplicity of the story together with the immediateness of the gameplay will make it a suitable choice for anyone who has **spare time** during their day.

The mixture of different genres creates a unique product that will be suitable both to **hyper-casual games consumers** and more **hardcore rhythm games** fan, thanks to the harder difficulties.

Design implementation

Basics

The player will control KAE, as in *Flappy Bird*, Kae will continuously **fall** due to gravity, the player needs to **tap** the screen to have KAE gain height and avoid obstacles. The player needs to input the commands following the **music timing**, this has a series of consequences that makes it different from *Flappy Bird*:

- Not following the music will have KAE gain a **smaller amount of height or not gain height at all**, which will, in turn, have them go against **obstacles**.
- To avoid frustration, KAE will have **hit points**, therefore the game does not end after KAE hits the first obstacle.
- The obstacles become **visual flair** for the real objective of the game, which is to jump following the music as **accurately** as possible. A good rhythm game should be able to be played with your eyes closed.
- Obstacles can be of very different types, following the timing of the song (a series of close obstacles for a fast stream of notes could be an example of this). See image below.

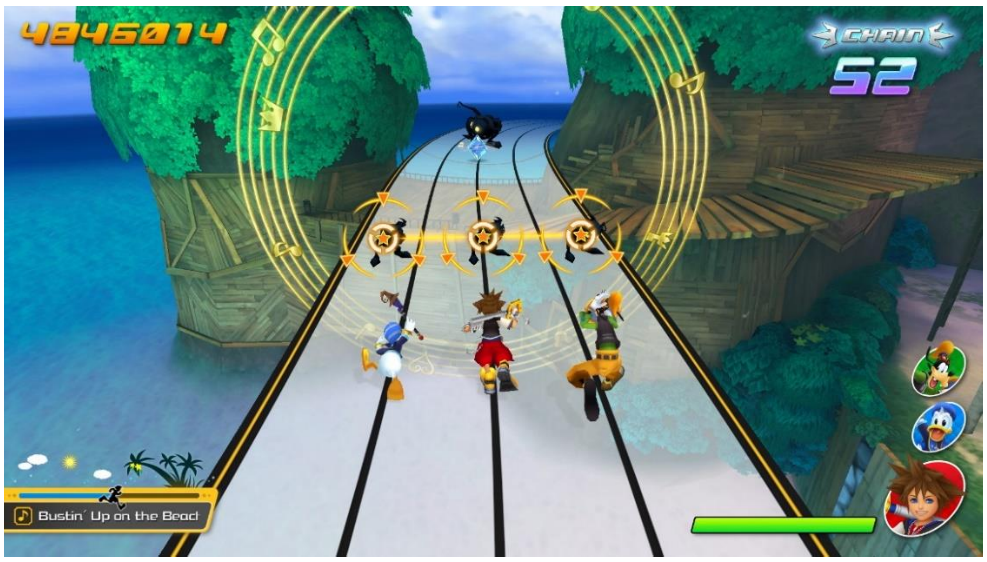


Figure 3: An example of timed obstacles in the rhythm game Kingdom Hearts Melodies of Memory

To have the player experience many different songs with a coherent story structure, the game will be divided into **levels**, where each level plays a **different song**. These levels are grouped into **chapters** with a **boss fight** level at the end of each chapter.

At the end of each level, a result screen shows the player score, based on accuracy, and other data like combo chain (i.e. how many consecutive inputs with a certain degree of accuracy). The player will be then assigned a mark based on the above values, from **E** to **S**. The high score will be included in the **global ranking** for each level.

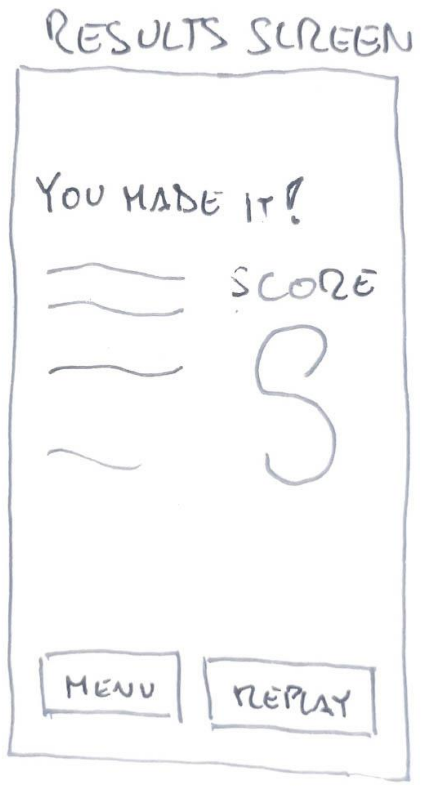


Figure 4: Results screen sketch

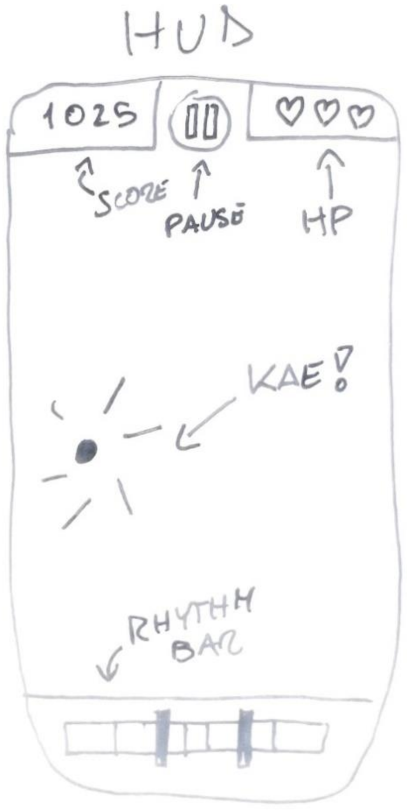


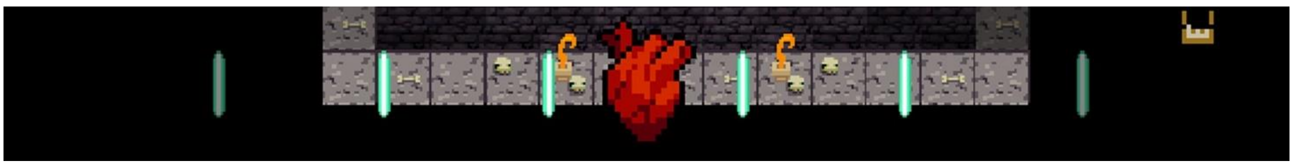
Figure 5: HUD mockup

A **rhythm bar** at the bottom of the screen will help the player touch the screen with the correct timing; an example of such a bar coming from the game *Crypt of the Necrodancer* can be seen below.

The story is conveyed via little scenes at the end of each chapter, where a textbox with visuals will explain what is happening.

In the following sections, each element of the game will be developed and looked into more extensively.

Figure 6: Rhythm bar from Crypt of the Necrodancer



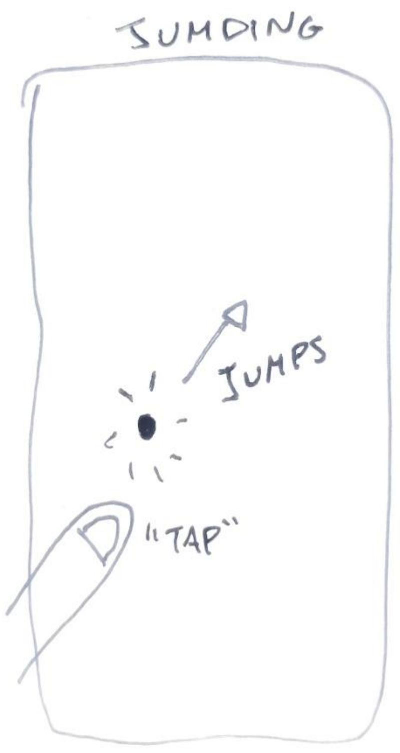


Figure 8: Jumping in KAE

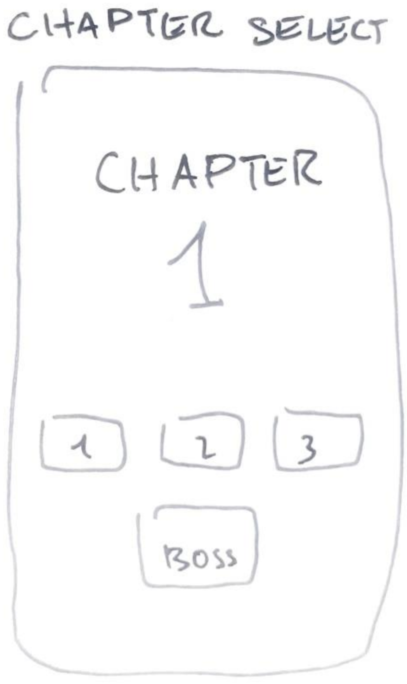


Figure 7: Chapter select screen mockup

Controls

The game is aimed toward mobile phones, the player touches the screen to have KAE jump. No other input is required.

Progression

The game will have a **chapter structure**, and it currently aims to have 3 chapters with 4 levels each (including boss fights), for a total of **12 levels**. The game goes therefore through three simple gameplay states: **normal levels**, **boss levels**, and **story interludes** between chapters.

Normal level

This will follow the basic gameplay feedback described previously. A song plays while the player taps through it, trying to get to the end of the song without losing all their hit points.

Boss level

In this level, the basic formula gets shaken up a bit. The boss will appear at the right part of the screen, and while a song is still playing and KAE just needs to avoid obstacles as normal, the boss can activate **abilities** that will make the level harder to play, as an example:

- Changing the **tempo** of the song
- Having the screen turn **black** for a short amount of time
- **Stopping** the game altogether and resuming it after a timer
- Having some **camera play** that will distract the player

The boss is defeated when the player reaches the **end of the level**.

Story interlude

At the end of each chapter a simple story interlude plays, these are simple screen with animated sprites and text boxes that will show one or more characters interacting between each other.

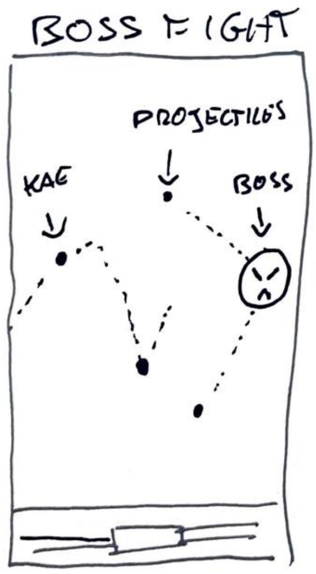


Figure 11: Boss fight sketch with additional obstacles

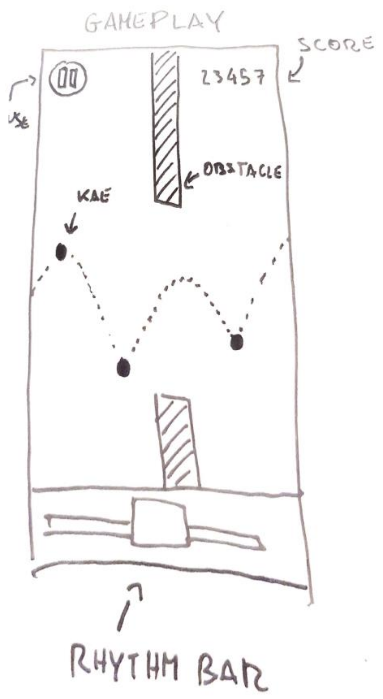
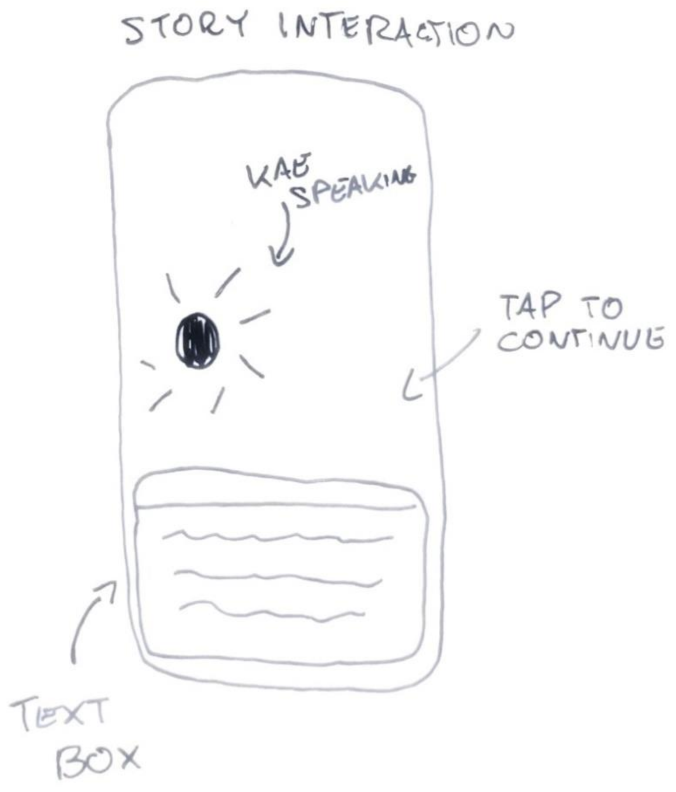


Figure 10: Normal level gameplay

Figure 9: Story interlude



Level structure or endless?

Despite taking inspiration from one of the most famous endless run games, KAE has a **level structure**. This approach has been chosen for the following two main reasons:

- Having levels means letting the player experience **many different songs and visuals**
- A level approach is more compatible with the *easy to learn, hard to master* approach, with the player being able to retry the song over and over while getting each time a little better

Levels will be relatively short, with each song being **1:30 to 2 minutes** in length; this is done to keep the game easy to pick-up and play during the day.

Difficulty and Accessibility

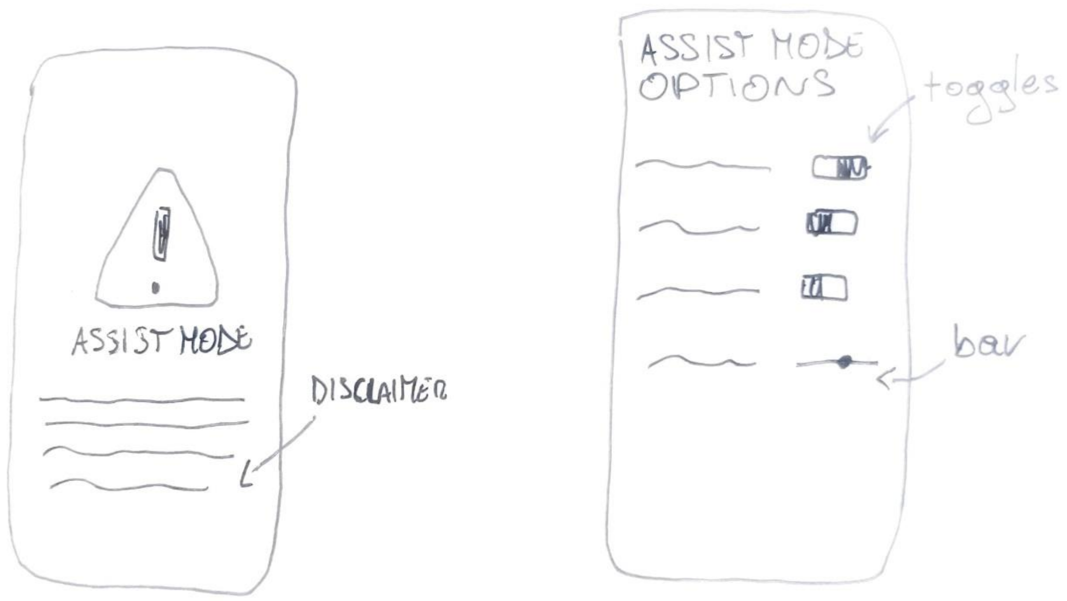
Difficulty and **accessibility** are fundamental topics in modern games design discourse. KAE proposes a mix of traditional and modern in its approach.

- For the traditional, each level has an **easy**, **normal** and **hard** difficulty mode, and each one of these can be played from the get-go, without the need to unlock them. Completion of any of these is enough to unlock the following level, thus leaving out the frustration of having to complete a level at a certain difficulty to progress.
- The modern approach follows the example of indie game **Celeste**; the game offers an assist mode, which will allow the player to tail the experience exactly how they want. Deactivating obstacles, auto-jump, changing the music tempo to make it slower to name a few options. To activate this option the player is first shown a disclaimer with the following text (subject to change):

  " With Assist Mode you can tailor up the experience exactly as you want, with lots of different options. Please note that this will inevitably impact the original design the developers intended for the game. Scores are deactivated when Assist mode is active. "

  This will ensure that as most people as possible**, including people with impairments**, can enjoy the game and reach the end.

Figure 12: Disclaimer and option screen



Music

The game is, of course, strongly focused on **music**. In each level, a particular song is played. On top of these, music tracks for the menu and the cutscene interactions are also present, for a total of **~15 original tracks**. These are made from scratch for the game by the development team, in particular by the music artist **SirFeffo**, and level tracks are studied to be **fun to play**.

An example of a song present in the game can be heard here (if reading from PDF).

Graphics

The graphical style will be **2.5D**; i.e. we will have **2D sprites** and elements moving in a **3D environment**. This has a series of advantages:

- The 3D environment can be generated with a mixture of ready-made assets from the Unity asset store (see *Platforms and technology*) and assets made from scratch by our 3D artist; this includes meshes, particle effects, and other potential resources.
- 2.5D creates an immersive and unique graphical style that will diversify the game from its hyper-casual roots, which more often than note are in 2D.

In particular, 3D elements will be used only for level backgrounds, and they will consist of (but not limited to):

- Trees
- Mountains
- Grass
- Skybox

Where 2D sprites will be used for:

- KAE, both in-game sprites and story cutscene sprites
- Other characters
- Obstacles
- Bosses
- UI elements for HUD and menus
- Rhythm bar

Platforms and technology

The game is intended as a mobile app for **iOS** and **Android**, emphasizing its pick-up and play nature. Endless run games have found their natural growth in the mobile market, and rhythm games have still a huge potential for expansion.

Due to the simplicity of the premise, and to cut on development costs and times, the game will be developed entirely using the **Unity game engine**. This has a series of advantages:

- Many ready-made systems can help in the implementation of the rhythm game portions and other components for which we lack experts (like UI programmers)
- Graphical assets from the asset store can be used in conjunction with ones made from scratch; this under the supervision of the game designer

Recently, AAA game publisher *Square Enix* released a *Kingdom Hearts* themed rhythm game completely made in Unity (see Figure 3).

Plot and art style

Plot

KAE comes from the word 帰る – *kaeru* – **which in Japanese means** ***to return, to go home.*** KAE is a **LUX** (*light* in Latin), little creatures made of pure light who live in the dimensional fractures of our reality. The evil **VOX** (*voice* in Latin), a distressed entity which manifests themselves only through noises and a creepy voice, during a terrible night, kidnaps KAE's family and all the LUX population. KAE needs to reach VOX and save their family and people.

The adventure will be a simple hero's journey through this weird dimensional world, where KAE go through different sceneries following VOX's presence. Bosses will be VOX's minions, which share with them the possibility of **manipulating sounds**, thus the music manipulation mechanics of boss fights.

KAE does not kill anyone during their adventure, and bosses will simply let KAE pass through after witnessing their courage and determination, up until reaching the final encounter with VOX. After defeating them, it is revealed that VOX wanted the light and clarity of the LUX to find the meaning of their existence. But after the confrontation with KAE, realizes that no such answers can be found from the LUX. In a bittersweet ending, VOX releases all the LUX and goes on a journey to **create** a meaning for themselves, a *raison d'être*, while the KAE is finally reunited with the LUX.

VOX represents the need for humans to find a meaning for their existence, where LUX represent the illusion of an easy answer to such an impossible question. Such an interpretation, although intended, **will not be delivered explicitly** to the player, which will be free of making their conclusions from the story – or simply enjoy it as a little frame to the gameplay.

Art style

The art style for the game will follow a **minimalistic** approach, and it will take inspiration from games like *Monument Valley* and *Ori and the Blind Forest*; dark and moody environments with forests and weird architectures with a stylised approach. Concept arts for the game will serve as visual inspirations for the direction of the game.

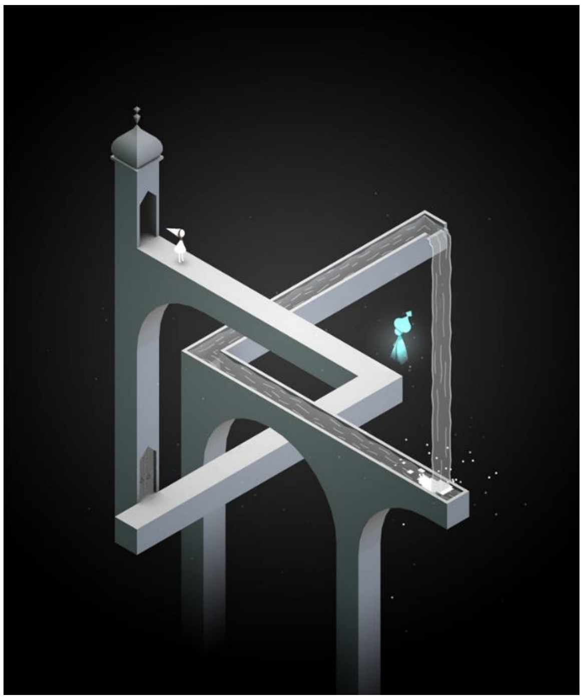


Figure 14: Monument Valley, notice the stylised graphcs.

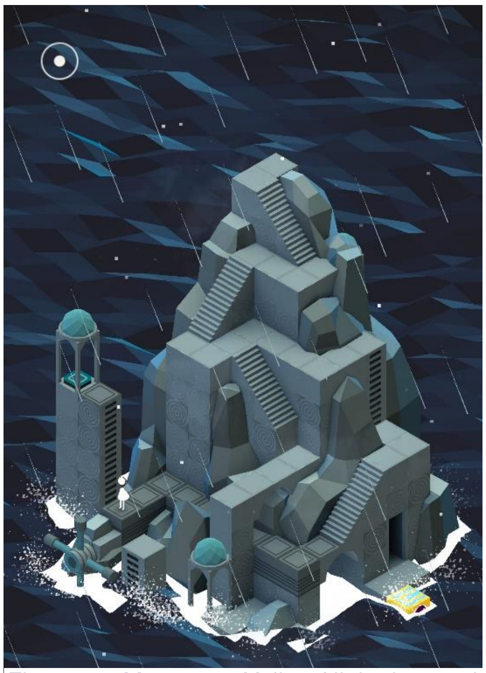


Figure 13: Monument Valley. Night time and rain give an example of the intended mood for the game.

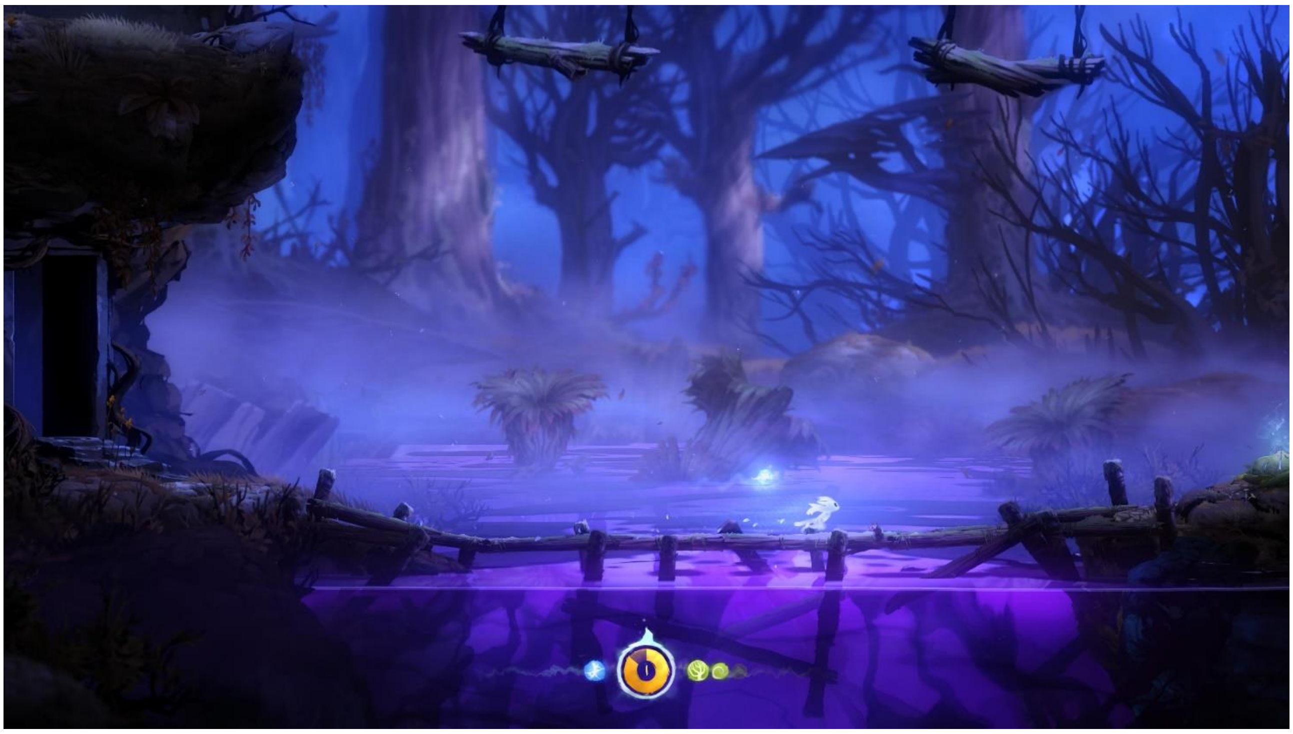


Figure 15: An example of a forest environment from Ori and the Blind Forest

Team, roadmap and budget

The team will be composed of:

- **Lead programmer/Game designer**: responsible for the creative direction of the game and the main implementation of gameplay mechanics.
- **Associate programmer**: responsible for helping to polish the code and gameplay mechanics and fixing bugs.
- **Music producer (PT)**: responsible for the creation of the music tracks that will be used in the game.
- **Concept artist/Illustrator (PT)**: 2D sprites, concept art for the game and promotional material on the website and stores.
- **3D artist/Environment artist/Tech artist**: they will create the 3D assets for the level backgrounds and be responsible for the integration of art assets in the game engine.

We expect an eight months development cycle, which is divided into the following milestones:

- **Milestone 1 (1 month)**: functional prototype with basic music gameplay mechanics (only lead programmer and music producer required).
- **Milestone 2 (3 months)**: closed beta for company feedback.
- **Milestone 3 (4 months)**: Full release, stretch goals will be implemented in this period if we will able to.

Given the milestones, this is the expected budget for the project:

	M1 – 1 month	M2 – 3 months	M3 – 4 months	**Total – 8 months**
Lead programmer	3,800Ł	11,400Ł	15,200Ł	30,400Ł
Associate progr.	0Ł	6,750Ł	9,000Ł	15,750Ł
Music prod. (PT)	1,060Ł	3,180Ł	4,240Ł	8,480Ł
2D artist (PT)	0Ł	4,500Ł	6,000Ł	10,500Ł
3D artist	0Ł	9,000Ł	12,000Ł	21,000Ł
Total	4,860Ł	34,830Ł	46,440Ł	**86,130Ł**

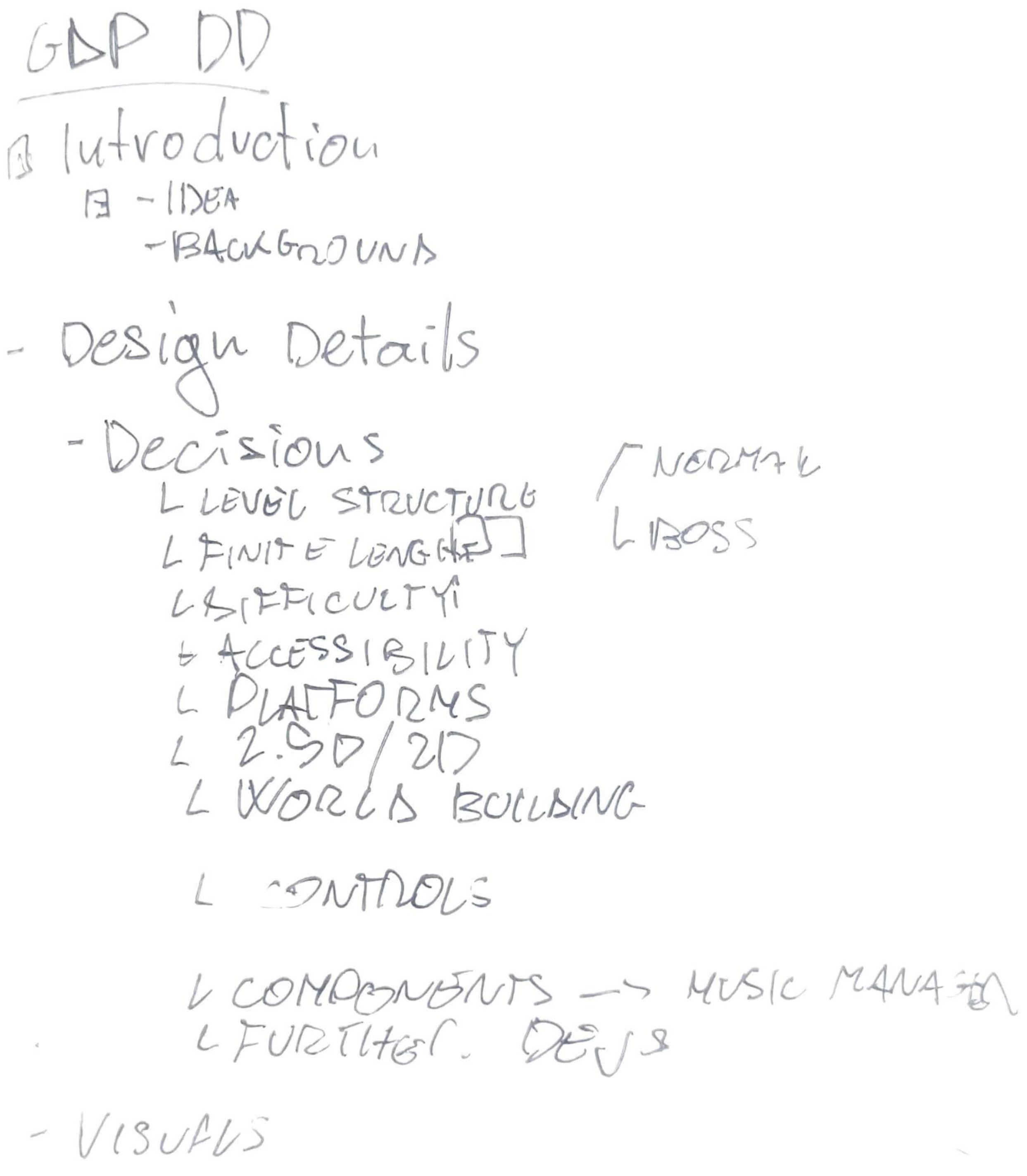
In particular, investors will be able to fund the project based on milestones so that we can offer various degrees of commitment. The budget includes a license for Unity Pro.

Further developments

Due to the chapter structure of the game and the story left for open interpretation, the game has a great potential for **free/paid DLC**. These could be made available as **additional chapters** that will expand the story of the original game, while at the same time providing new music level to play and enjoy.

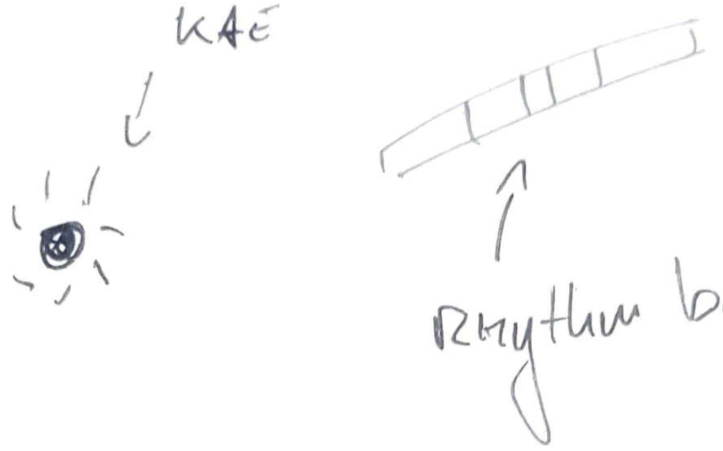
Furthermore, the game can be easily expanded with an actual **endless mode**, where a dynamic, and long, song which changes style every so often can be played. Obstacles will still be procedurally generated from the playing song, but after each song loop, the tempo will get faster, with additional challenges that resemble the one the player will experience in boss fights; like camera play, blank screen, etc.

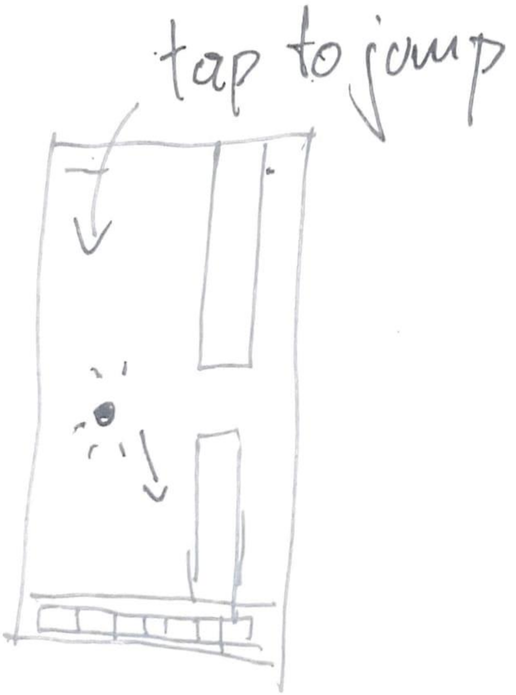
Designer sketches



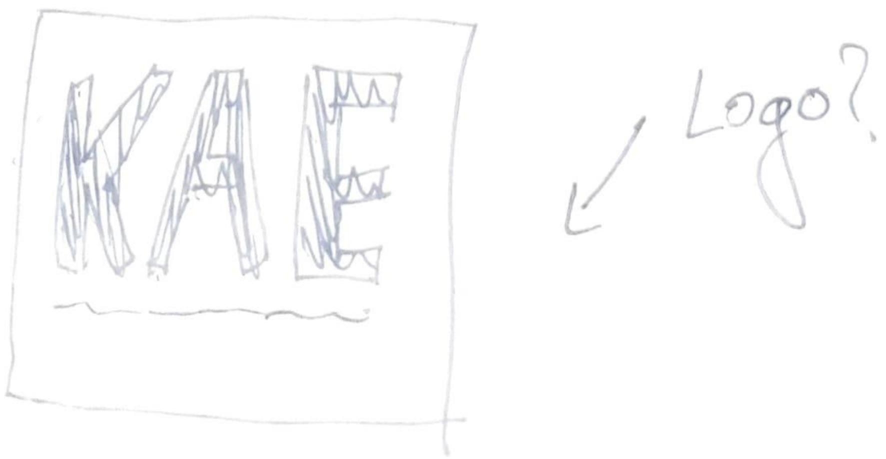
GDP DD

- Introduction
  - IDEA
  - BACKGROUND
- Design Details
  - Decisions
    - L LEVEL STRUCTURE ┌ NORMAL └ BOSS
    - L FINITE LENGTH
    - L DIFFICULTY
    - L ACCESSIBILITY
    - L PLATFORMS
    - L 2.5D/2D
    - L WORLD BUILDING
    - L CONTROLS
    - L COMPONENTS → MUSIC MANAGER
    - L FURTHER DEVS
- VISUALS


KAE
Rhythm bar


tap to jump

circles
osu-like
KAE!


KAE
Logo?

Background environment

↳ Moody?

↳ Dark forest
(ori-like)

Boss fights powers

- change tempo
- stop music
- BLANK SCREEN

$\downarrow$

CAMERA PLAY?
(LIKE DRAKENGARD 3)

References and image sources

Image sources

- Figure 1: https://it.wikipedia.org/wiki/Flappy_Bird
- Figure 2: https://cytus.fandom.com/wiki/Holy_Knight
- Figure 3: https://xboxsquad.fr/news/2020/06/kingdom-hearts-melody-of-memory-en-images/
- Figure 6: https://store.steampowered.com/app/247080/Crypt_of_the_NecroDancer/ (cropped)
- Figure 11: https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Fconstructg.com%2Fwp-content%2Fuploads%2FMonument-Valley-3.jpg&f=1&nofb=1
- Figure 12: https://www.androidworld.it/2014/05/19/monument-valley-recensione-dellintenso-puzzle-game-ustwo-229409/monument-valley-2/
- Figure 13: https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Fcrohasit.com%2Fwp-content%2Fuploads%2F2016%2F08%2FOri_2015_03_14_01_09_45_354.jpg&f=1&nofb=1

References

- **Flappy Bird**, dotGEARS Studios, 2013
- **Cytus**, Rayark, 2012-2019
- **Kingdom Hearts Melodies of Memory**, Square Enix, 2020
- **Crypt of the Necrodancer**, Brace Yourself Games, 2015-2018
- **Monument Valley**, Ustwo, 2015-2017
- **Ori and the Blind Forest**, Moon Studios, 2015
- **DryWetMIDI**, Maxim Dobroselsky: https://www.codeproject.com/Articles/1200014/DryWetMIDI-High-level-processing-of-MIDI-files
- **Music Syncing in Rhythm Games**, Yu Chao, published on GamaSutra on 03/16/17: https://www.gamasutra.com/blogs/YuChao/20170316/293814/Music_Syncing_in_Rhythm_Games.php
- **Music by SirFeffo**: https://www.youtube.com/user/GreenTheragon

Appendix: Song manager implementation sketch

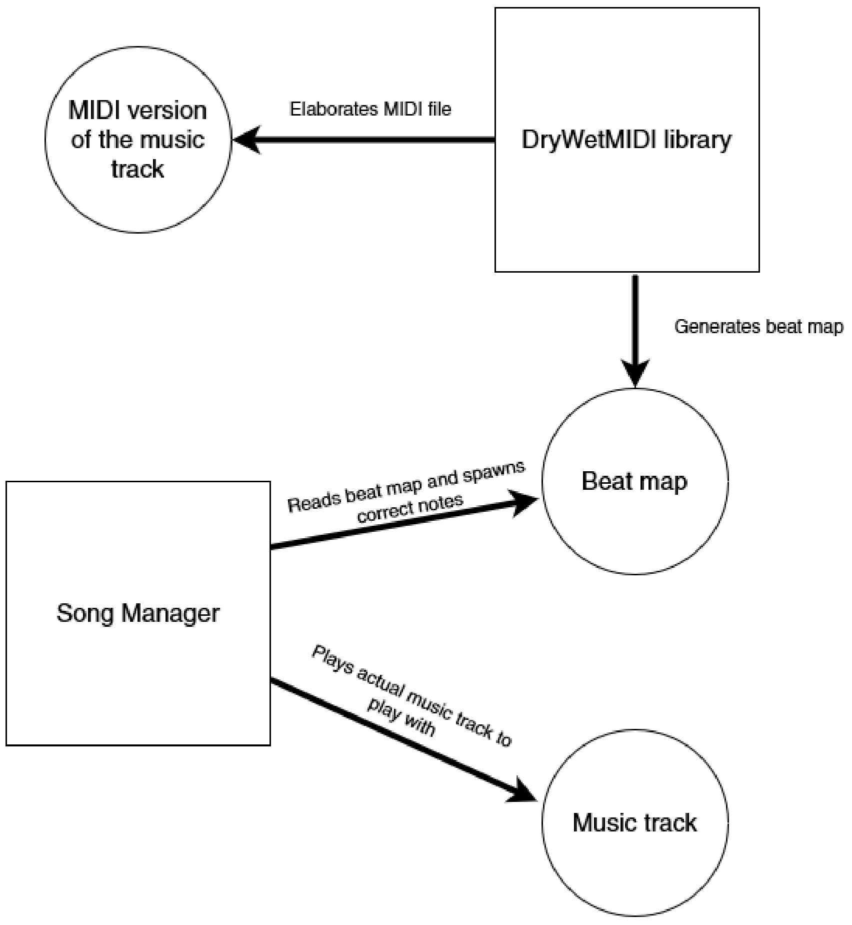

MIDI version of the music track
Elaborates MIDI file
DryWetMIDI library
Generates beat map
Beat map
Reads beat map and spawns correct notes
Song Manager
Plays actual music track to play with
Music track

Figure 16: A high-level diagram of the above process

Here will follow a sketch on how the most important part of any rhythm game will be implemented; the system for the generation of *beat maps*[1]. The following high-level explanation assumes a *C#+.NET* framework, as the one Unity scripts use.

A binary music file like *wav* or *mp3* does not contain info about their beats per se, therefore for each song a *beat map* needs to be generated. This is a process that can be done by hand, but a simpler process is to procedurally generate *beat maps* using a music file that does contain info about its notes; that is a **MIDI file.**

Our music producer creates MIDI versions of the songs that are played in each level and deletes from the MIDI version all notes considered **too hard to play**, creating a MIDI file with notes that will translate perfectly into a *beat map*.

This MIDI file is then first elaborated using the excellent **DryWetMIDI** .NET library by Maxim Dobroselsky, which will create an *array* of *floats* corresponding to the absolute position in seconds of each note we want to play in the song.

[1] *Beat maps* are to be considered as the series of note the player will actually play, by inputting their command with the correct timing.

From this array, we can then proceed to create our *beat map*. The following procedure has been inspired by the digression on rhythm game systems originally written by Yu Chao on **GamaSutra**.

- We first compute how many seconds are in each beat for the song (*SPB*), given the *BPM* for the song:

$$\mathrm{SPB} = 60 \, / \, BPM$$

- We then compute the position in beats of each note in our array by dividing each value by *SPB*. This is our actual *beat map.*

Now we can *play* a song:

- While reproducing the song, we update our position in beats by dividing the current time position in the song by *SPB*.
- Using an index variable, we keep track of the next beat to play by comparing the value in our *beat map* at that particular index with our current position in beats.

Depending on the timing of our input, we can create an **accuracy** system by comparing the position in the song with the next beat to play.

If we then want to animate the notes correctly, we linearly interpolate each animation value by the difference of the note position in the song and the current song position in beats.