**TECNOLÓGICO DE MONTERREY**

TC2005B: Construcción de software y toma de decisiones


# Update exercise of a relational database in MySQL and UML diagrams

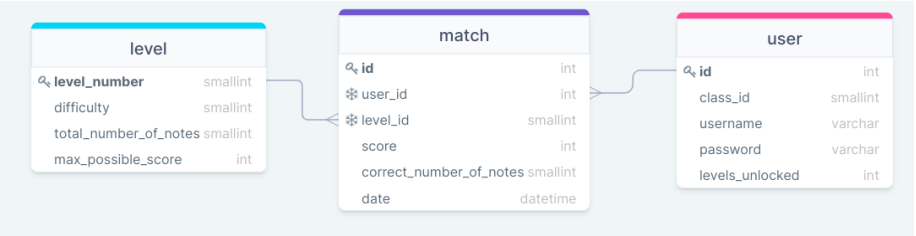
Pablo Rocha Ojeda - A01028638

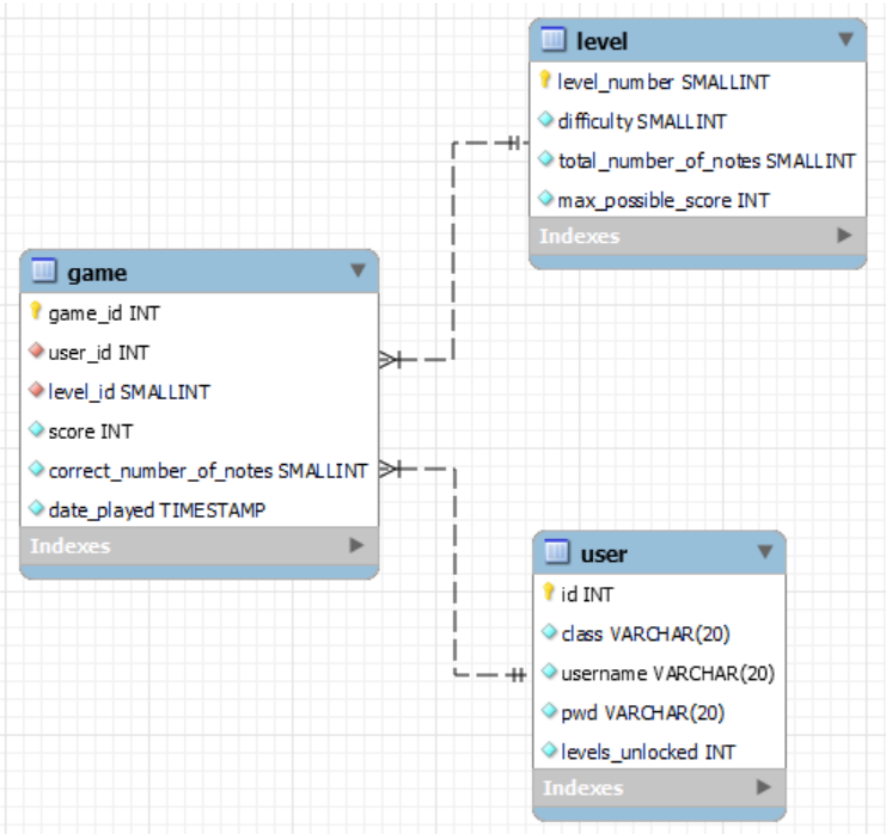
Luis Javier Karam - A01751941

Miguel Arriaga Velasco - A01028570

***Delivery Date*** - 01/04/2022

## Entity-relationship diagram that represents the project database




## Data schema in MySQL

SQL database creation and dummy data scripts can be found on this github address: https://github.com/PRocha0503/Construccion-de-software/tree/main/db-module/Data%20Base%20Structure/MySQL_scripts

## Two views, two triggers, and two stored procedures to your MySQL schema

SQL views, triggers and stored procedures scripts with code comments can be found on this github address:
https://github.com/PRocha0503/Construccion-de-software/tree/main/db-module/Data%20Base%20Structure/MySQL_scripts

# Normal form justification

To make sure that the tables are made correctly we will apply the normalization rules, up to the third normal way.

**1** ***First normal form***

To comply with the first normal form the tables must:

**Atomic**

⚛ All attributes are atomic, that is, they are indivisible. They also use the data types allowed by MySQL.

**Level**

In this table it can be seen that no element is divisible. The difficulty, total number of notes and maximum possible result are atomic.

**Match/Game**

No element of the table can be divided into more sections. "user_id" and "level_id" are foreign keys and cannot be separated. The score is a single atomic integer. The correct number of notes and dates cannot be divided any further either.

**User**

Looking at this table we can notice that none of its attributes could be divided. The username, password, and class are attributes that wouldn't make sense to separate any further. The unlocked level is a single number, which makes it as atomic as possible.

**Keys**

🗝 All tables have primary braces.

The primary key has no null attributes.

**Level**

Its primary key is "level_number".

It is composed of only 1 attribute and has restriction that it cannot be null and autoincremental.

**Match/Game**

Its primary key is "id".

It is composed of only 1 attribute and has restriction that it cannot be null and autoincremental.

**User**

Its primary key is "user_id".

It is composed of only 1 attribute and has restriction that it cannot be null and autoincremental.

**Columns**

❕ There can be no variation in the number of columns.

**Level**

All rows will have the columns of "level_number", "difficulty", "total_number_of_notes" and "max_possible_score".

**Match/Game**

All rows will have the columns of "id", "user_id", "level_id", "score", "correct_number_of_notes" and "date".

**User**

All rows will have the columns of "id", "class", "username", "password", and "levels_unlocked".

**Dependence**

🔐 Non-key fields must be identified by the key.

**Level**

The difficulty of a level, its total number of notes and maximum result depend on which level it is.

**Match/Game**

Which user plays, what level it is, the result, the correct number of notes and the date all depend on the "match" or "game".

**User**

The user's class, their username, password, and up to what level they have unlocked depends on the user.

**Redundancy**

🔁 There should be no groups of repeated values.

**Level**

There are the data "level_number", "difficulty", "total_number_of_notes", and "max_possible_score" which are not repeated and have an individual column each of them.

**Match/Game**

There are the data "id", "user_id", "level_id" which all (semantically and physically) identify different aspects of the database.

**User**

It contains data such as "class", "username", "password", "id" and "levels_unlocked" which represent different aspects.

***2 The Second Normal Form***

To be in the second form you must meet:

**Functional dependencies**

😞 There should be no partial functional dependencies. That is, all the values of the columns in a row must depend on the primary key.

**Level**

All the attributes of the table depend on the primary key "level_number", if the level number changes, the total number of notes and the highest possible score also change.

**Match/Game**

In this table, a primary key id is used, on which all the values of the columns depend: "user_id", "level_id", "score", "correct_number_of_notes" and "date".

**User**

All attributes of "user" are bound to the primary key id, where each unique "id" represents a combination of different "class", "username", "password", and "levels_unlocked".

**Keys**

🔑 The primary key must be formed of only 1 column that has an indivisible value.

**Level**

There is the primary key "level_number" which represents an integer that increases, that does not repeat and that cannot be divided since it only exists in one column. Additionally, it is not a composite key, that is, it is not made up of the values of the other columns in the table.

**Match/Game**

In "match" or "game" there is the primary key id which is an integer value that increases, is not repeated and only exists in a column. Additionally, it is not a composite key, that is, it is not made up of the values of the other columns in the table.

**User**

In "user" is the primary key id which is an integer value that increases, does not repeat and only exists in a column. Additionally, it is not a composite key, that is, it is not made up of the values of the other columns in the table.

***3 Third Normal Form***

To be in the third form you must meet:

💻 There should be no transitive dependencies between columns in a table. That is, the columns that are not part of the primary key must depend only on that key, never on another column.

**Level**

Both "difficulty", "total_number_of_notes" and "max_possible_score" depend solely on the primary key "level_number". Although it could be argued that the difficulty depends on the total number of notes, this is not entirely true, as there could be more difficult levels with fewer notes. It can also be observed that there is some kind of dependence between "max_possible_score" and "total_number_of_notes". However, there are cases where this does not fully apply. For example, if in the future it is decided to opt for different notes to have different score values, such as a long note that must be left pressed.

**Match/Game**

There are no transitive dependencies in the table. In the first instance you could believe that "score" depends on "correct_number_of_notes", this is not true since the score depends on the multiplier that the player has.

**User**

There are no transitive dependencies in the table. You might think that the "password" depends on the "username", however, two users can have the same password.

**Integrity restrictions**

**Level**

- level_number SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,

This means that level_number is always a small number (there will not be many levels), is not null and automatically self-incretes.

- difficulty SMALLINT NOT NULL,

It makes the difficulty a small number and cannot be null.

- total_number_of_notes SMALLINT NOT NULL,

Makes the total number of notes a small number

- max_possible_score INT NOT NULL,

It makes the maximum score a non-zero number.

- PRIMARY KEY (level_number)

It makes the level number unrepeatable, makes it indexable and referenceable by other tables.

**Match/Game**

- game_id INT UNSIGNED NOT NULL AUTO_INCREMENT,

The NOT NULL integrity constraint makes this attribute non-negotiable, that is, it always requires a value. Additionally, the restriction AUTO_INCREMENT causes the game_id attribute to automatically initialize incrementally.

- user_id INT UNSIGNED NOT NULL,

These restrictions ensure that the user_id is an unsigned integer and is not null.

- level_id SMALLINT UNSIGNED NOT NULL,

These restrictions ensure that the user_id is a small unsigned integer and is not null.

- score INT NOT NULL,

These restrictions ensure that the user_id is an unsigned integer and is not null.

- correct_number_of_notes SMALLINT NOT NULL,

These restrictions ensure that the user_id is a small unsigned integer and is not null.

- date_played TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP

These restrictions determine that the date on which it is played is of type TIMESTAMP and that by default it puts the date on which the data is added to the database, this also happens when the data is updated.

- PRIMARY KEY (game_id),

it is determined to game_id as the primary key of the table, in this way it can be referenced by other tables.

- FOREIGN KEY(user_id) REFERENCES user(id) ON DELETE RESTRICT ON UPDATE CASCADE,

user_id is determined as a secondary key and referenced from the user table. Additionally, it is updated in cascade in case its value is changed in another of the tables, nor can data be deleted from this column by the restriction of ON DELETE RESTRICT.

- FOREIGN KEY(level_id) REFERENCES level(level_number) ON DELETE RESTRICT ON UPDATE CASCADE

level_id is determined as a secondary key and referenced from the level table. Additionally, it is updated in cascade in case its value is changed in another of the tables, nor can data be deleted from this column by the restriction of ON DELETE RESTRICT.

**User**

- id INT UNSIGNED NOT NULL AUTO_INCREMENT,

It makes the id a number, not a null number that self-corrects itself.

- class VARCHAR(20) NOT NULL,

It is done by a word of maximum 20 characters.

- username VARCHAR(20) NOT NULL,

It is done by a word of maximum 20 characters.

- pwd VARCHAR(20) NOT NULL,

It is done by a word of maximum 20 characters.

- levels_unlocked INT NOT NULL,

It makes the levels you have unlocked a non-zero number.

- PRIMARY KEY (id),

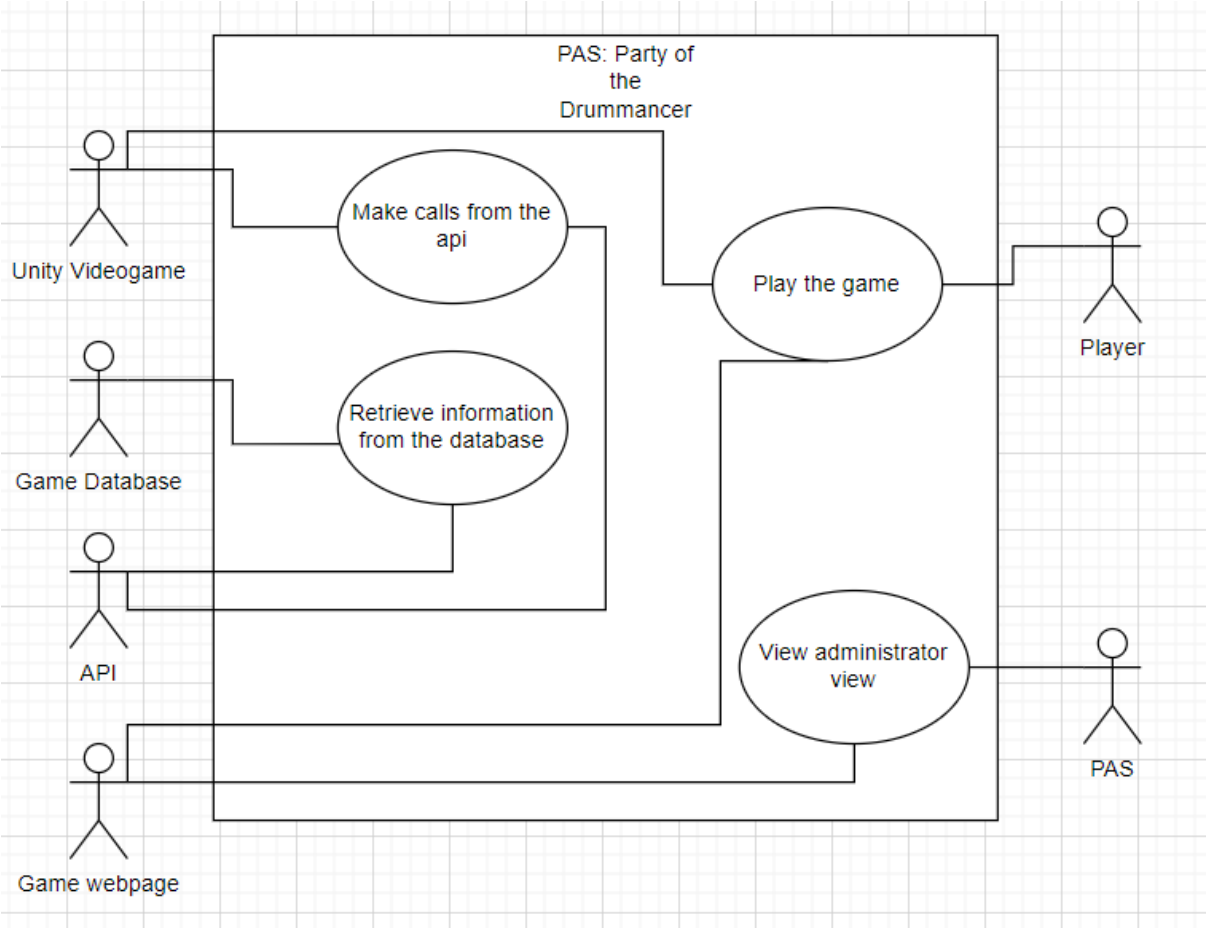
It makes the id unrepeatable, it makes it indexable and referenceable by other tables.

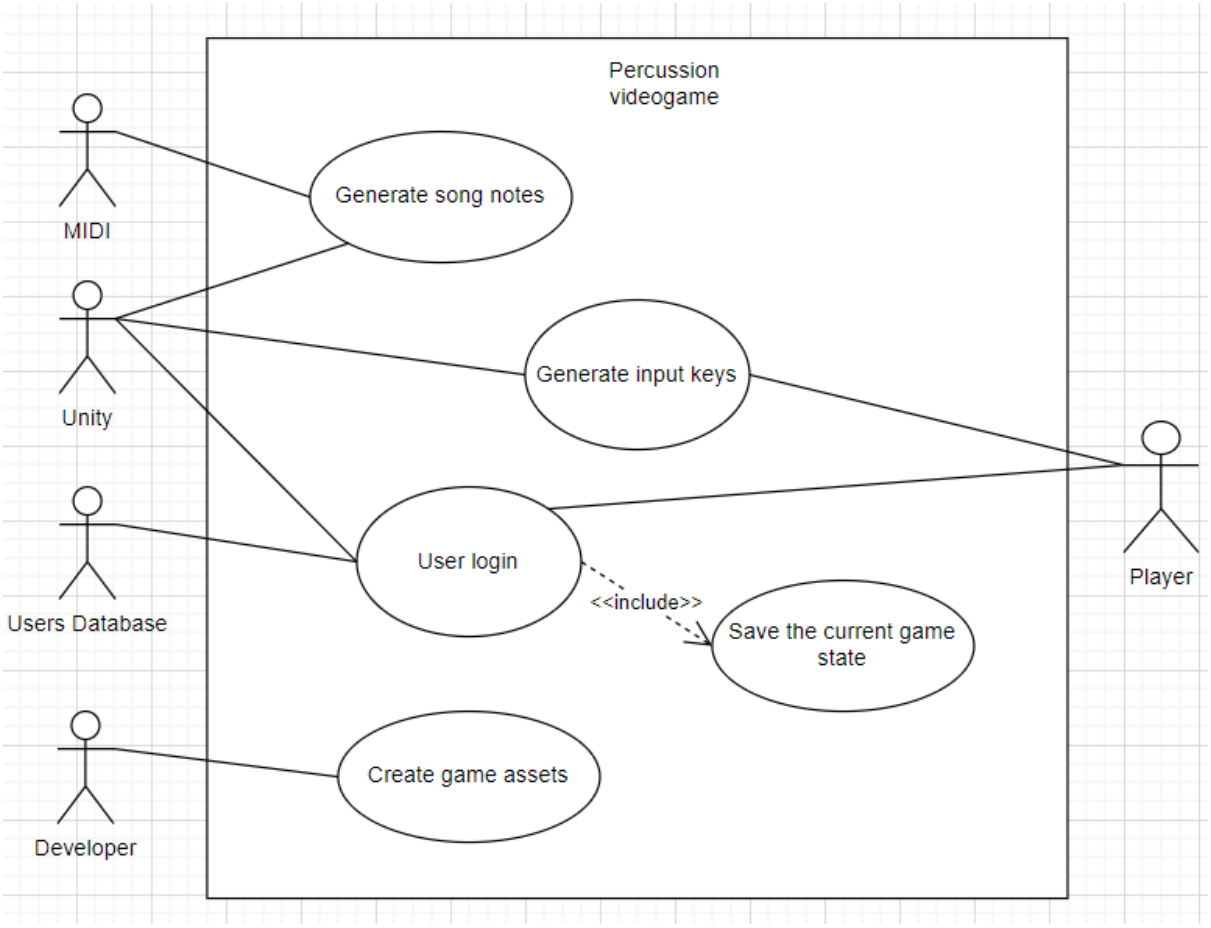
- INDEX idx_class (class)

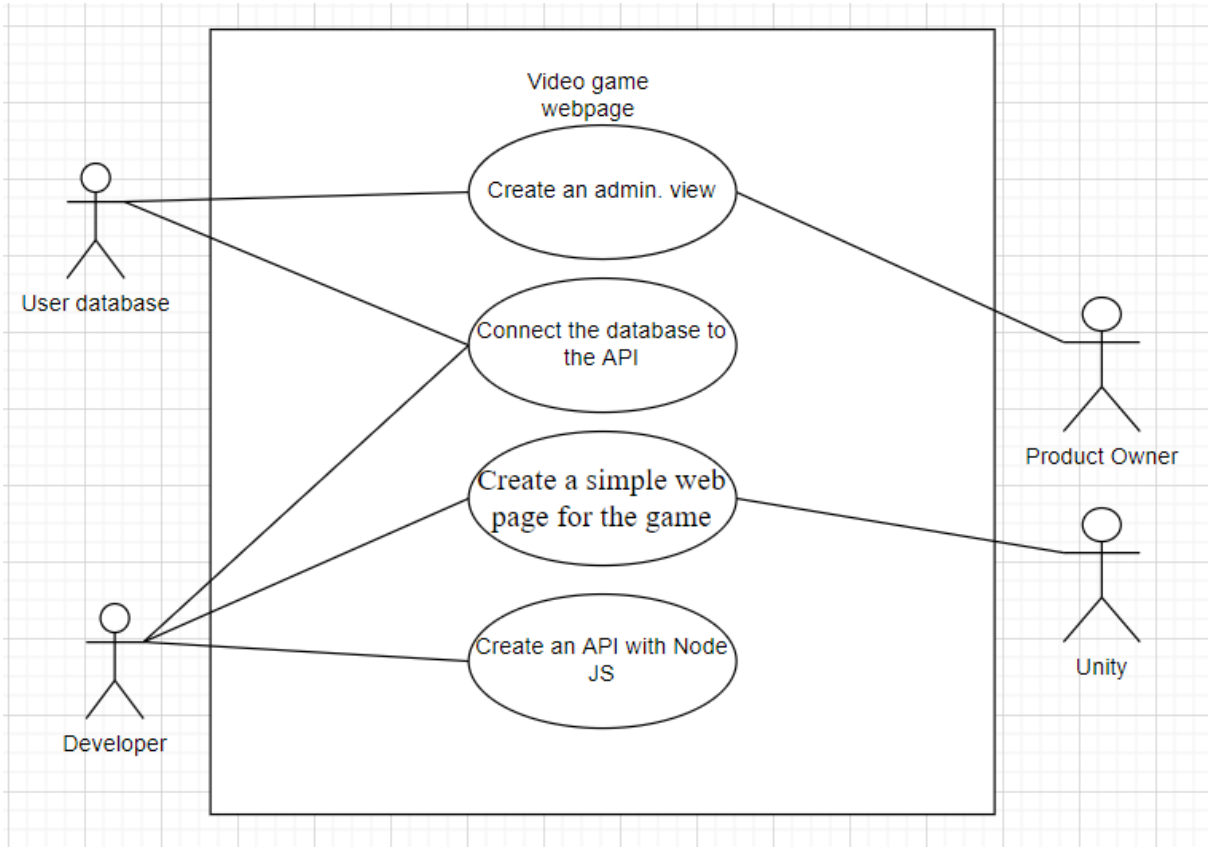
It makes easier access to the class in which the person is.

## Use case diagrams

Link:
https://drive.google.com/file/d/1fw1fh4GsFW3W6OVMM372a9XNNlQvyU_8/view?usp=sharing


PAS: Party of the Drummancer
Unity Videogame
Make calls from the api
Play the game
Player
Game Database
Retrieve information from the database
API
View administrator view
PAS
Game webpage


Percussion
videogame
MIDI
Generate song notes
Unity
Generate input keys
Player
User login
<<include>>
Save the current game
state
Users Database
Developer
Create game assets


Video game
webpage
User database
Create an admin. view
Connect the database to
the API
Product Owner
Create a simple web
page for the game
Unity
Developer
Create an API with Node
JS

## User stories

| User story #1 Instruments | |
|---|---|
| **As** organization<br>**I want** a video game about percussion instruments<br>**to be able to** educate about percussion. | |
| **Validation:**<br>● Show demo to SF.<br>● Check that the video game is useful for its members. | **Priority:** 10<br>**Estimate:** 300h |

| User story #2 Web page | |
|---|---|
| **As** user<br>**I want** a web page with the game<br>**to be able to** have easy access to the game | |
| **Validation:**<br>● Check that the game works correctly within the page<br>● Check that all the data is displayed correctly | **Priority:** 10<br>**Estimate:** 10h |

| User story #3 Collect Information | |
|---|---|
| **As** an organization<br>**I want** to collect information on the progress of the players<br>**to be able to** see their progress | |
| **Validation:**<br>● Session information must be collected<br>● The information must be useful | **Priority:** 8<br>**Estimate:** 12h |

| User story #4 Administrator view |
|---|
| **As an** administrator<br>**I want** to be able to view the information |

| | |
|---|---|
| **in order to** interpret it and adjust the database. | |
| **Validation:**<br>● Login as administrator.<br>● View the database.<br>● Verify that the data is correct. | **Priority:** 7<br>**Estimate:** 5h |

| | |
|---|---|
| User Story #5 Informational Game | |
| **As** an organization<br>**I want** to educate people about percussive instruments<br>**so I can** create awareness about percussive instruments | |
| **Validation:**<br>● Check content is correct<br>● Check content has real value | **Priority:** 8<br>**Estimate:** 1h |

| | |
|---|---|
| User Story #6 Ranking | |
| **As** organization<br>**I want** a video game suitable for all ages<br>**so that I can** inspire young and old alike to learn about drumming. | |
| **Validation:**<br>● Show a demo to the SF.<br>● Let students of all ages play it.<br>● Ask them for their opinion. | **Priority:** 2<br>**Estimate:** 1h |

| | |
|---|---|
| User Story #7 Three Levels | |
| **As** an organization<br>**I want** 3 levels<br>**in order to** have a progressively difficult game. | |
| **Validation:**<br>● There are 3 levels<br>● Levels are progressive | **Priority:** 8<br>**Estimate:** 18h |

| User story #8 Programming Languages | |
|---|---|
| **As a** team<br>**I want** to use Unity, Javascript, NodeJS and MySQL<br>**to be able to** distribute a quality game. | |
| **Validation:**<br>● Unity<br>● is used JS<br>● is used SQL is used | **Priority:** 4<br>**Estimate:** 80h |

| User story #9 Login | |
|---|---|
| **As a** user<br>**I want** to be able to create my login credentials<br>**to be able to** store my information | |
| **Validation:**<br>● Create a user<br>● Login with their credentials | **Priority :** 6<br>**Estimate:** 4h |

| User story #10 Constant art | |
|---|---|
| **As** an organization<br>**I want** the game to have a unified art style in<br>**order to** improve the gameplay of the game | |
| **Validation:**<br>● All levels and screens in the game have the same visual style | **Priority:** 1<br>**Estimate :** 12h |

| User story #11 Main platform pc | |
|---|---|
| **As** an organization<br>**I want** the game to run on pc<br>**so I can** distribute the game to as many people as possible | |
| **Validation:**<br>● Check that the game runs correctly on pc | **Priority:** 10<br>**Estimate :** 2h |

| User story #12 Single player | |
|---|---|
| **As** organization<br>**I want** a single player game<br>**so I can** measure everyone's progress individually. | |
| **Validation:**<br>● No more players can play | **Priority:** 7<br>**Estimate:** 0h |

| User Story #13 Recorded Progress | |
|---|---|
| **As** a player<br>**I want** to save the current state of my game<br>**so I can** see my progress. | |
| **Validation:**<br>● Start a new session.<br>● Exit game.<br>● Re-enter the game.<br>● Validate that the progress is saved. | **Priority:** 5<br>**Estimate:** 20h |

| User Story #14 Game Story | |
|---|---|
| **As** an organization<br>**I want** the game<br>**to be told in order to** have a consistent game | |
| **Validation:**<br>● The game story is told through itself | **Priority:** 1<br>**Estimate:** 5h |

| Story User Story # 15 No External Hardware | |
|---|---|
| **As** an organization<br>**I want** no external hardware to be needed<br>**to be able to** play it only with the computer | |
| **Validation:**<br>● The game can be played only with the computer | **Priority:** 1<br>**Estimate:** 0h |

| User Story # 16 Database | |
|---|---|
| **As** an organization<br>**I want** the information to be stored in a database in<br>**order to** have a record of the players and sessions. | |
| **Validation:**<br>● The information is stored securely in a database. | **Priority:** 10<br>**Estimate:** 12h |

## UML Use Cases

Cards for each use case:

| **Generate song notes** | |
|---|---|
| Description of the use case / Use case description detail | With the help of MIDI and UNITY the notes of the songs in the video game will be generated. |
| Related requirements / Related requirements | 1,2,4,5,6 |
| Objective in context / Goal in context | Generate the appropriate notes for each song. |
| Preconditions / Preconditions | A MIDI type file must have been generated and inserted into the project. |
| Successful end condition / Successful end condition | The notes of the song must be coordinated with the music. Likewise, their times and types must correspond to the score. |
| Failed end condition / Failed end condition | The notes are not generated or are generated incorrectly and not according to the MIDI file. |
| Primary actors | ● MIDI<br>● Unity |
| Secondary actors | Music audio |
| Trigger | Level starts |
| Main flow | 1. Player starts a level.<br>2. UNITY reads the MIDI file<br>3. UNITY generates the corresponding notes on screen. |
| Extensions / Extensions | 1. UNITY reads the file<br>2. UNITY generates the notes |

| | |
|---|---|
| | incorrectly. |

| **Generate key inputs** | |
|---|---|
| Description of the use case / Use case description detail | Creation of different inputs for different actions within the video game. |
| Related requirements / Related requirements | 3 |
| Objective in context / Goal in context | That the player is able to interact with the game by means of keys. If the player hits the right key at the right time, points are added. |
| Preconditions / Preconditions | ● The player must be in a level.<br>● There should be notes on the screen. |
| Successful end condition | If the player hits the right note at the right time the player should be awarded points. |
| Failed end condition / Failed end condition | The keys do not create any changes in the game. |
| Primary actors / Primary actors | ● Unity<br>● Player |
| Secondary actors / | Midi parser |
| Trigger | The player presses a key. |
| Main flow / Main flow | 1. Player is on a level<br>2. The score appears on the screen.<br>3. The note comes to the right place.<br>4. The player presses the key.<br>5. The corresponding note disappears.<br>6. Appropriate points are given. |
| Extensions / Extensions | 1. The player presses the key.<br>2. Nothing happens. |

| **User login** | |
|---|---|
| Use case description / Use case description detail | When the game starts, a screen appears to login and recover its previous state. If it is not registered, it is registered. |

| | |
|---|---|
| Related requirements / Related requirements | 13 |
| Objective in context / Goal in context | That the player can enter the game several times and have a record of his status. |
| Preconditions / Preconditions | The player must be registered. |
| Successful end condition / Successful end condition | Upon entering the game the player is able to enter their game data and enter. |
| Failed end condition | ● The player cannot enter the game<br>● No login required |
| Primary actors | ● User<br>● Database |
| Secondary actors | API |
| Trigger | The player starts the game |
| Main flow | 1. The player starts the game match.<br>2. Login screen appears.<br>3. Player enters his data.<br>4. If the data is correct, go to your profile. |
| Extensions / Extensions | 1. The player starts the game.<br>2. The game goes directly to the level screen or does not open. |

| **Save current game state** | |
|---|---|
| Use case description / Use case description detail | The game must be able to save the player's progress so that they continue from where they left off. |
| Related requirements / Related requirements | 14, 15 |
| Objective in context / Goal in context | That the player can save and observe their progress, a better user and educational experience. |
| Preconditions / Preconditions | ● Having<br>● made progress in the game |
| Successful end condition | The player recovers his progress and can continue from where he left off. |
| Failed end condition / Failed end condition | The player's progress is deleted and must |

| | |
|---|---|
| | start from the beginning. |
| Primary actors | ● Player<br>● Unity<br>● User database |
| Secondary actors | ● API |
| Trigger | ● The player starts the game with a previous game file. |
| Main flow / Main flow | 1. The player starts the game.<br>2. The player logs in into their profile.<br>3. The player can play from their previous progress. |
| Extensions / Extensions | 1. The player doesn't log in<br>2. The game starts from scratch. |

| **Asset creation** | |
|---|---|
| Use case description / Use case description detail | All the visual and audio art of the game must be created to have a pleasant experience. |
| Related requirements / Related requirements | 20,21,22,23,24,25,26 |
| Objective in context / Goal in context | Create a uniform art style around all levels. |
| Preconditions / Preconditions | Raised game story and theme |
| Successful end condition | The art is nice and consistent throughout the game. |
| Failed end condition | Characters, backgrounds, or other art are missing/Art has different styles. |
| Primary actors / Primary actors | Developer |
| Secondary actors / | User |
| Trigger / Trigger | The player starts the game |
| Main flow / Main flow | 1. A theme is made.<br>2. Characters are created.<br>3. Notes are created.<br>4. The background is created. |
| Extensions / Extensions | 1. Some characters are not created. |

| | |
|---|---|
| | 2. The style of play changes between levels. |

| **Administration view** | |
|---|---|
| Use case description / Use case description detail | An administration window is required for the product owner, organization or developer team. |
| Related requirements / Related requirements | 16 |
| Objective in context / Goal in context | That the trends of the database can be observed and interpreted to obtain information. |
| Preconditions / Preconditions | ● That the database is existing and populated.<br>● That the API can obtain information (GET) from the database.<br>● That the web page is built. |
| Successful end condition / Successful end condition | That the administrator can observe different aspects and information of the database. |
| Failed end condition / Failed end condition | That the database cannot be interpreted/observed. |
| Primary actors / Primary actors | ● User database<br>● Product owner |
| Secondary actors / | Developer |
| Trigger | The page is accessed as administrator. |
| Main flow / Main flow | 1. The page is accessed as an administrator.<br>2. Information is obtained from the database.<br>3. information is interpreted. |
| Extensions / Extensions | 1. The page is accessed as an administrator.<br>2. Information cannot be displayed from the database or it is incorrect. |

| **Connect the database to the API** |
|---|

| | |
|---|---|
| Description of the use case / Use case description detail | Connect the database to the API to be able to modify the tables with GET PUT, DELETE. |
| Related requirements / Related requirements | 10 |
| Goal in context / Goal in context | To be able to modify the database with API calls in order to save game information. |
| Preconditions / Preconditions | ● Database made and normalized.<br>● API created. |
| Successful end condition / Successful end condition | Several calls can be made to the API that affect the database and are capable of: fetching information, changing information, creating information and deleting information. |
| Failed end condition / Failed end condition | The API fails to affect the information in the databases. |
| Primary actors / Primary actors | ● Database<br>● Developer |
| Secondary actors / | Web page |
| Trigger | The player performs an action in the game that requires modifying the database. |
| Main flow / Main flow | 1. The player does an action that requires modifying the database or requires information from the database. Ex: login<br>2. The corresponding API call is made.<br>3. Changes are reflected in the game<br>4. Changes are reflected in the database. |
| Extensions / Extensions | 1. The player is not able to extract information from the database.<br>2. Some error appears in the API call. |

| **Create a simple web page with the game** | |
|---|---|
| Use case description / Use case description detail | The game must be inside a web page. |
| Related requirements / Related requirements | 17, 18, 19 |

| | |
|---|---|
| Objective in context / Goal in context | That the game can be played by as many players as possible, that it be easily accessible. |
| Preconditions / Preconditions | ● Create the videogame.<br>● Export the game.<br>● Have a domain/host for the page. |
| Successful end condition / Successful end condition | That the game can be played from a web page. |
| Failed end condition | That the game cannot be played from the web page |
| Primary actors | ● Developer<br>● Video game |
| Secondary actors | API<br>Database |
| Trigger | Export the game as WebGL |
| Main flow | 1. The video game.<br>2. The web page is created.<br>3. The video game is exported to the web page. |
| Extensions / Extensions | 1. The video game is not exported correctly to the web page.<br>2. Can't play the game. |

| **Create an API with NodeJS** | |
|---|---|
| Description of the use case / Use case description detail | A backend of the video game will be created that will serve as an intermediate between our database and frontend. |
| Related requirements / Related requirements | 10,11,12 |
| Objective in context / Goal in context | That our frontend is able to communicate with the database and perform the required actions. |
| Preconditions / Preconditions | Have a complete and normalized database. |
| Successful end condition / Successful end condition | A server is created with Node Js with several routes capable of modifying the database. |

| | |
|---|---|
| Failed end condition / Failed end condition | • The server is not created.<br>• Some route is not working properly. |
| Primary actors / Primary actors | Developer |
| Secondary actors / | Database |
| Trigger / Trigger | The database training is finished. |
| Main flow / Main flow | 1. A simple Node JS project is created.<br>2. Routes are made to modify users.<br>3. Routes are made to modify classes.<br>4. Routes are made to modify sessions. |
| Extensions / Extensions | 1. Routes fail. |

## Activity diagrams

The activity diagrams can be found in the following link: https://drive.google.com/file/d/1_nNwEuVKdUTzc8h8uI5qEL75arn3lyzg/view?usp=sharing

## SCRUM statistics

The product backlog, documentation of each sprint and SCRUM statistics can be found in the following document:

https://hill-limburger-3c6.notion.site/Documentation-Party-of-the-drumancer-0273914c292549298ddc3217197c1b4c