SP-Project

Map editor, specifications

Révision	Date	Auteur	Description
A1	16/06/2022	Jbristhuille	First draft

General purpose **3**

System menu **4**

Tiles collection **5**

Layers management **6**

Editing modes 7

Indicator 7

Keys binding **8**

Aids and indicators 8

Publication **9**

Publication server 9

Notation 10

Map search 10

Exportation 10

Storage format 11

Map generator **12**

General purpose

Users can access a dedicated edition view.It will consist of several important parts:

- **System menu**: Allows access to the functions of saving, validation, etc. (see chapter *System menu*).
- **Collection**: Composed of several categories, it will allow the selection of tiles (see chapter *Collection of tiles*).
- **Edition view**: In the center of the screen allows the positioning, movement and deletion of the different tiles.
- **Layers indicator**: Indicates the layer being edited (see chapter *Managing layers*).
- **Commands indicator**: Always displayed, displays the different commands available with the icon of the key to press. (see chapter *Keys binding*).

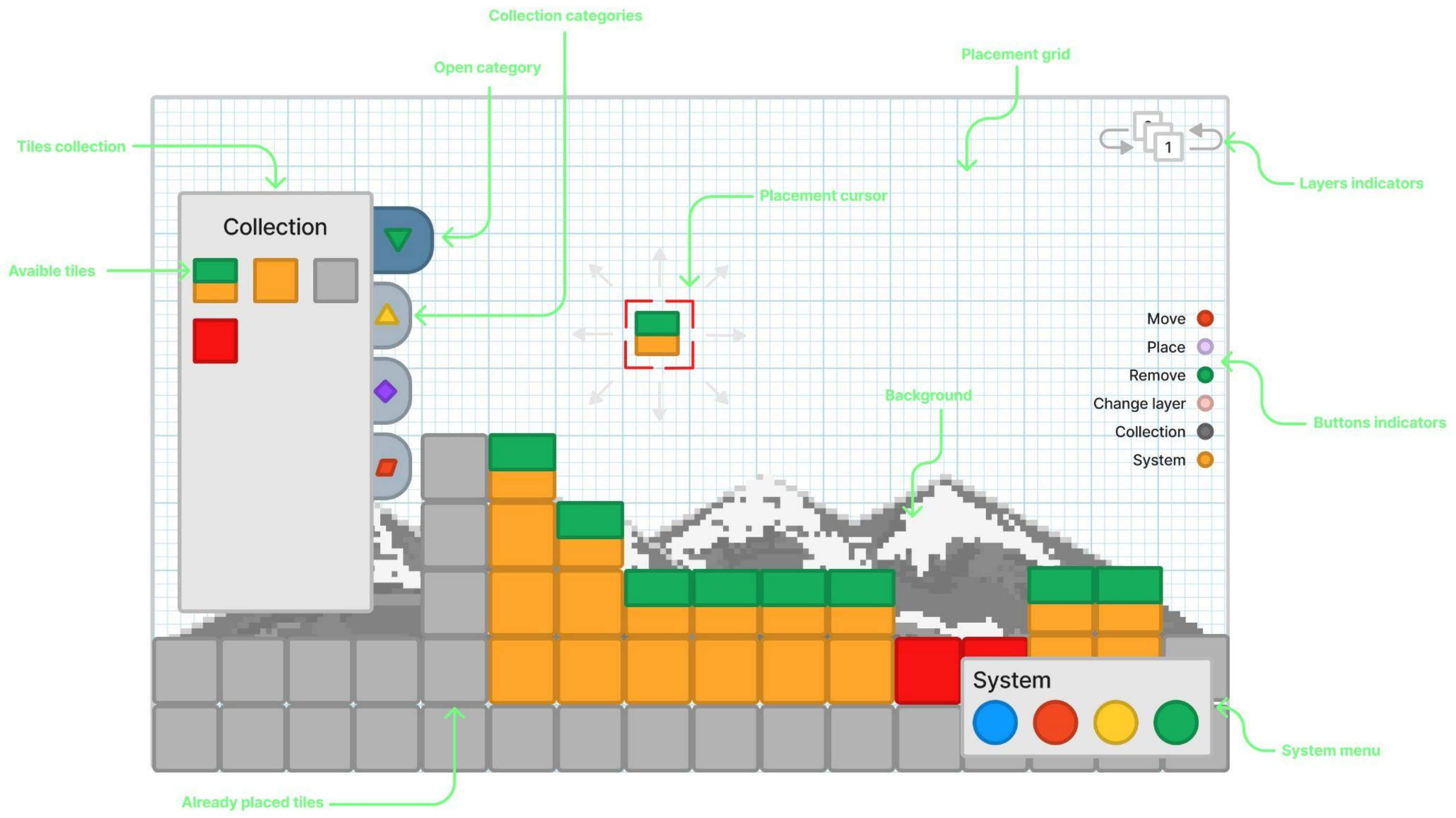


Figure 1*: Block diagram of the map editor*

System menu

This menu contains all the important actions of the editor:

- **Save**: Allows you to save changes locally:
    - ***Name***: Le nom donné à la carte et visible par les autres joueurs une fois publié.
    - ***Description***: The name given to the map and visible to other players once published.
    - ***Overview***: Map preview image.
    - ***Tiles***: Concrete data of the map, describes all the objects arranged on their respective layer.
    - ***Tags***: Keywords to make the map easier to find once published.
- **Validate**: Compulsory validation of the map, before being able to publish the map it must be completed to validate its feasibility (see *Publication* chapter).
- **Export**: Allows you to export the backup file to a local directory (see *Export* chapter).
- **Publish**: Allows you to publish the validated map (see the *Publication* chapter).

Tiles collection

The tile collection lists all the tiles and elements available to create a map. The different blocks will be sorted into categories to facilitate navigation and selection (ex: decorations, floor, wall, obstacles, etc…).

When a tile is selected, the "*Collection*" menu will close, and will re-open when the dedicated key is pressed (see chapter *Keys binding*).

Navigation in the collection menu will be done with the same keys as the editor via a cursor moving from tile to tile (see chapter *Keys binding*).

The accessible tiles will be those present in the game engine and used by the map generator, so this menu must load and display the different tiles by itself (see chapter *Map generator*).

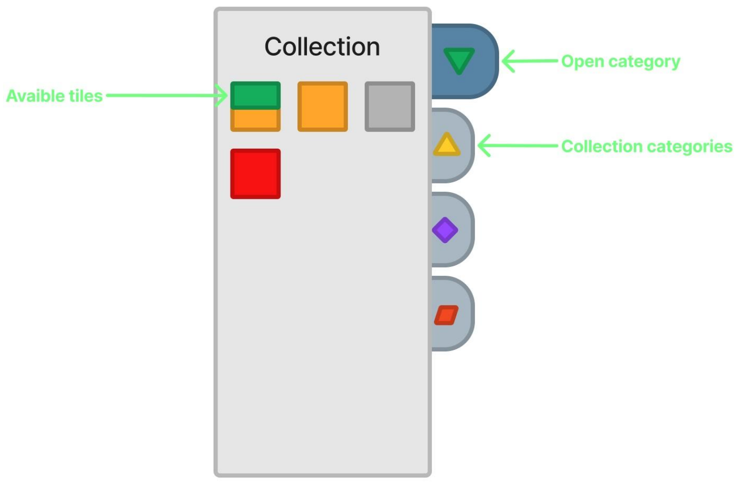


***Figure 2**: Diagram tile collection*

Layers management

Layers allow you to place elements forward or backward relative to the character as well as manage parallax effects.

The layer manager will have two types of layers:

- **Parallax layers**: At the count of three, each parallax layer has a different scroll speed:
  - ***-1***: Background layer, slow scrolling
  - ***0***: Calque sur lequel le personnage évolue, défilement normal.
  - ***1***: Layer on which the character is moving, normal scrolling.
- **Secondary layers**: Chaque **Parallax layers** is then composed of three secondary layer, or layers, allowing to place elements more or less in front (without changing the scrolling speed):
  - ***A***: Frontmost layer. Items placed on this layer overlay items on layers **B** and **C**.
  - ***B***: Intermediate layer. For layer **0**, corresponds to the plane on which the character moves (the latter will pass in front of the elements of this layer)
  - ***C***: Backmost layer. Elements placed on this layer will always pass behind those on higher layers.

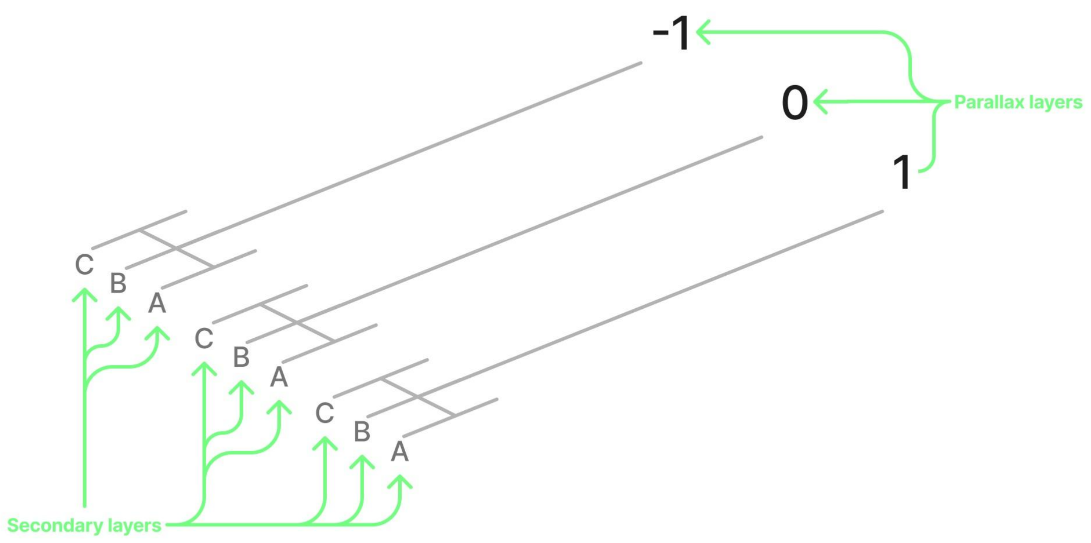


Figure 3: *Schematic diagram of how layers work*

Editing modes

To facilitate access to the editor, it will have two modes:

- **Simple**: Parallax management is locked, leaving only the management of secondary layers of layer **0** (**A**, **B**, **C**).
- **Advanced**: Contains all the functions of the editor.

Initially, only the **simple** mode will be made available to players. The **advanced** mode will be reserved for the development of the cards internally while perfecting the operation and ergonomics.

Indicator

On the map editor, the layer indicator lets the user know at a glance which layer is being edited.

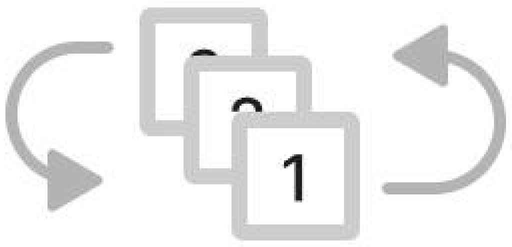


Figure 4: *Layer indicator diagram (simple mode)*

Keys binding

Below are all the keys that can be used in the editor with their uses:

- **Left stick/D-Pad**: Navigation in the menus (system and collection) or cursor movement in the editor.
- **Right stick:** Camera movement in the editor.
- **A**: Validate selection in collection or place tile.
- **B**: Cancel selection in collection or delete tile.
- **X**: Lance l'essai de la carte.
- **Y**: Start the map test.
- **RT**: Go to the previous layer.
- **RB**: Go to the next layer
- **Start**: Open or close menu system
- **Select**: Open collection menu

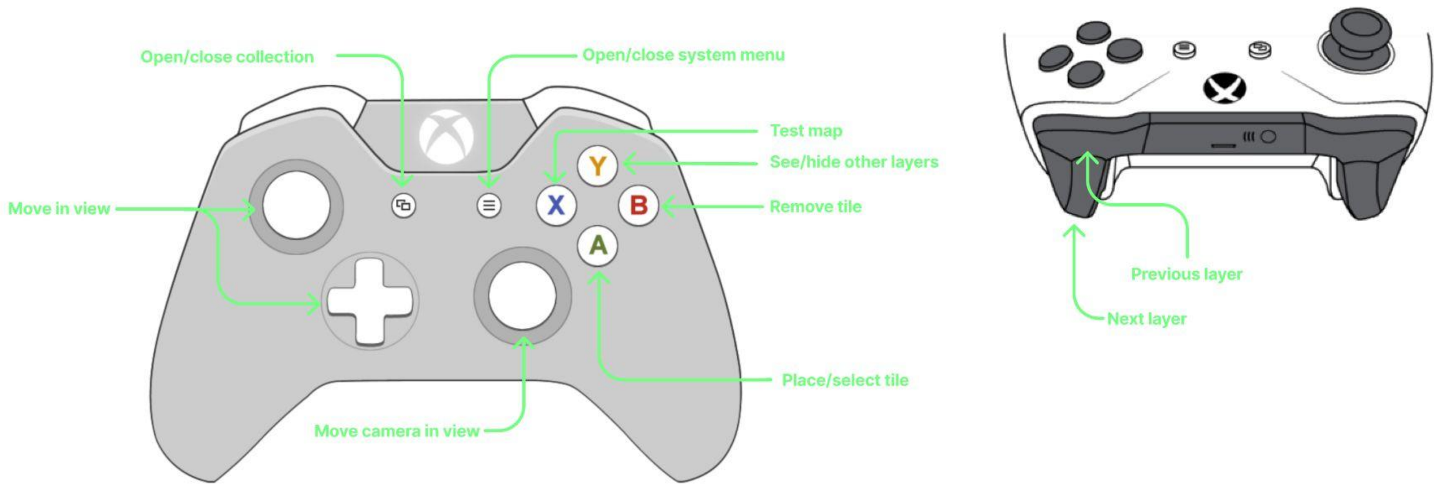


***Figure 5**: Attribution des touches*

Aids and indicators

On the editor, on the right, will be indicated the different commands available with the icon of the associated key.

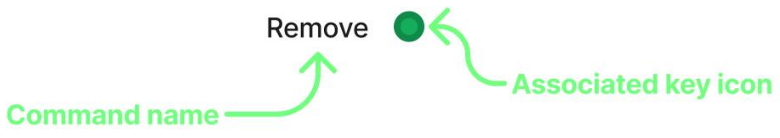


***Figure 6**: Command indicator diagram*

Publication

Publishing the map will allow the creator to broadcast their creation to other players.
For this, several steps are necessary for the creator before publishing:

- **Validation**: Validation is the most important step, it verifies that the card is playable. For this the creator must have positioned a start, an arrival and be able to finish the map.
- **Meta-data**: Once validated, the creator will have to fill in the various information to identify the map (name, description, tags, difficulty, thumbnail, etc.).
- **Publication**: When all the mandatory steps are validated, the map will be published and accessible to the rest of the players.

Publication server

In order to recover and share the different cards of the community, it will be necessary to provide an external server where the different cards will be sent and recovered.

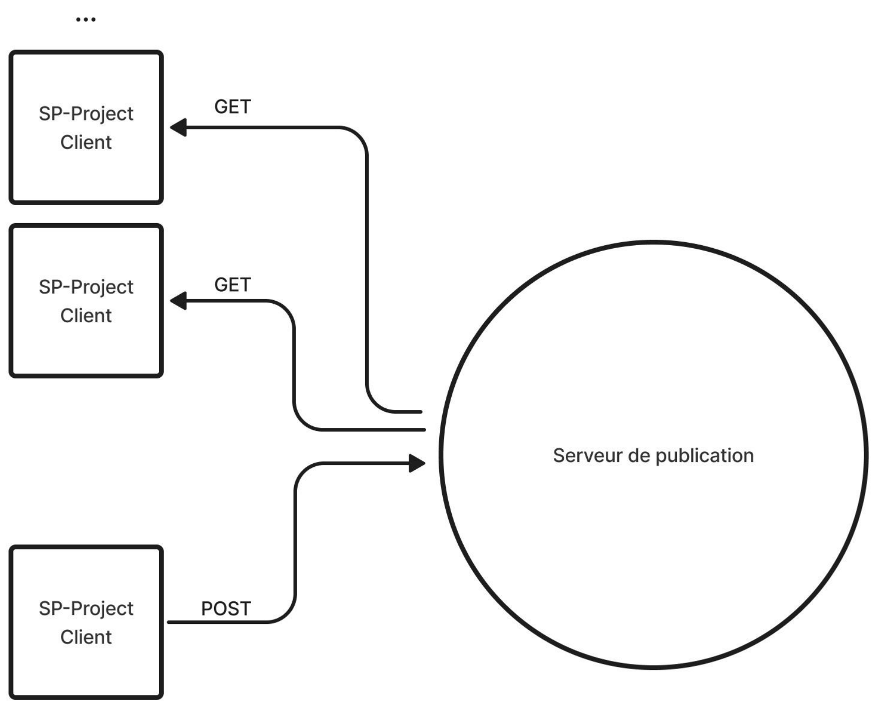


***Figure 7**: Publication server diagram*

Notation

Players, after having tested a map, will be able to rate the map via a star system according to their appreciation.

The average rating will be displayed on the card page and will be used to sort cards by rating.

Map search

Players will have access to a menu allowing them to search for cards according to different criteria:

- Name
- Tags
- Grades
- Date

This menu will display the list of maps that match the search criteria, allowing players to easily find a map that matches their requirements.

Exportation

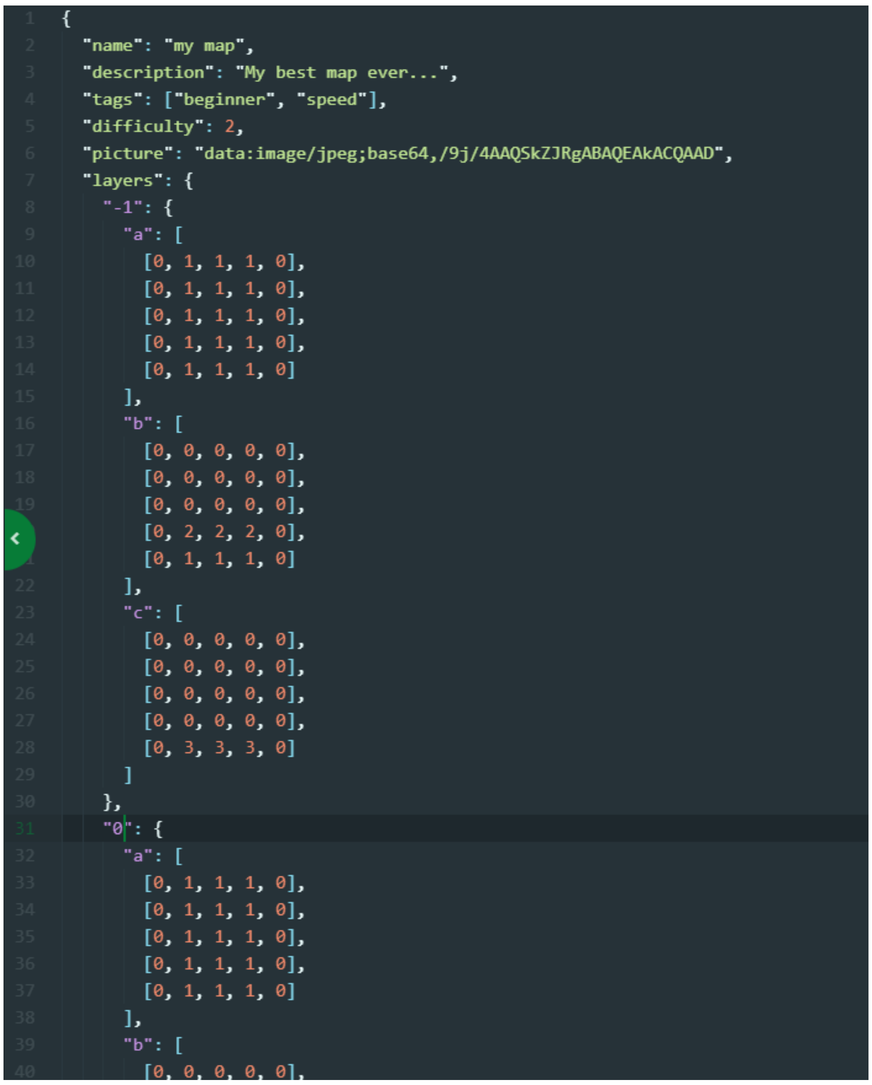
The export must allow map creators to be able to output the files allowing the generation of the map in order, possibly, to edit this file on an external tool or to make a backup.

Storage format

The backup file must describe all the layers and their contents as well as retain all the information relating to the map.

This file should include the following fields:

- **name**: *(String)* Map name, set by creator.
- **description**: *(String)* Map description, set by the creator.
- **tags**: *(Array[string])* List of tags, set by the creator
- **difficulty**: *(Number)* Difficulty, the higher the number, the more difficult the map is considered.
- **picture**: *(String)* The image presenting the map, in base 64.
- **layers**: *(Object)* Object describing layers and their contents.
    - ***x***: *(Object)* -1, 0 or 1, Parallax layer
        - ***n***: *(Array[array])* a, b or c, secondary layers, table of tables, each entry representing a line, itself containing the representation of the tile in the form of a number (or hexadecimal code)



```json
{
  "name": "my map",
  "description": "My best map ever...",
  "tags": ["beginner", "speed"],
  "difficulty": 2,
  "picture": "data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAkACQAAD",
  "layers": {
    "-1": {
      "a": [
        [0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0]
      ],
      "b": [
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 2, 2, 2, 0],
        [0, 1, 1, 1, 0]
      ],
      "c": [
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 3, 3, 3, 0]
      ]
    },
    "0": {
      "a": [
        [0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0]
      ],
      "b": [
        [0, 0, 0, 0, 0],
```

Figure 8*: Example storage file in json*

Map generator

The map generator will aim to load the map from the backup file, it must be able to read and interpret the information contained in this file.

The tile codes read in the “*layers*” descriptor (see chapter *Storage format*), will have a correspondence with the tiles actually recorded in the game engine (tiles map). Correspondence management will be managed by this generator.