The making of Doors.
A 2D game in SDL.

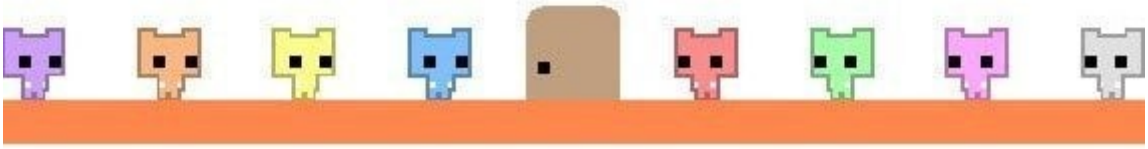
By OHMAD Abdessamade

Summary:

- **Introduction.**
- **Development and Show case.**
- **Faced problems.**

Introduction:

As a project in our C++ class we were tasked to make a 2D game of type puzzle Platformer similar to PICO PARK, so I made this game using SDL, and following the Object Oriented paradigm of Programming.



SDL: Simple DirectMedia Library, is a cross-platform development library designed to provide low level access.

In order to make this game I had to make everything from scratch, including all the classes which the game uses, but it was all worth it.

Development & Show case:

The first step I took in development was to make a repository in ***GitHub*** as to render the workflow smoother
After that it was initializing a ***git*** local Repo and connecting it to the remote branch.

I made a ***Makefile*** to facilitate compiling, and save time.

-The first commit was the Game class, which Initiates **SDL** and the window, runs the main Game loop, a static Renderer.
The Renderer's role is to render all game objects into the screen, we made it static to use it in all scenes.

-The Game loop runs in every 60FPS which we capped for control, each iteration of the game loop is a frame, the time between each frame is called *deltaTime*.

-During each frame we must update and render our game objects, and handle events set by the keyboard or mouse.

Scenes:

In order to have levels and menus, I had to make scenes, so I made A scene class which takes care of the background image, and two sub classes: ***MenuScene***, and ***LevelScene***.

The Main menu scene:

Each scene is running in its own loop, all inside the main game loop, so if we click on *Start*, the current scene stops, and the Levels menu scene takes off.

The Levels menu scene:

As we can see both scenes use buttons to control the menus.

All buttons are of type class *MenuItem*, which inherit from the class *GameObject*.

The selected scenes handle the events separately. For example, when a mouse hovers over the button, it changes color.

This is done through checking for the mouse position and comparing it to the position of every menu item, then if the latter if clicked, the appropriate event takes place.

The menu Items use **SDL_ttf.h** which is an SDL library to handle fonts, while **SDL_image.h** handles all images.

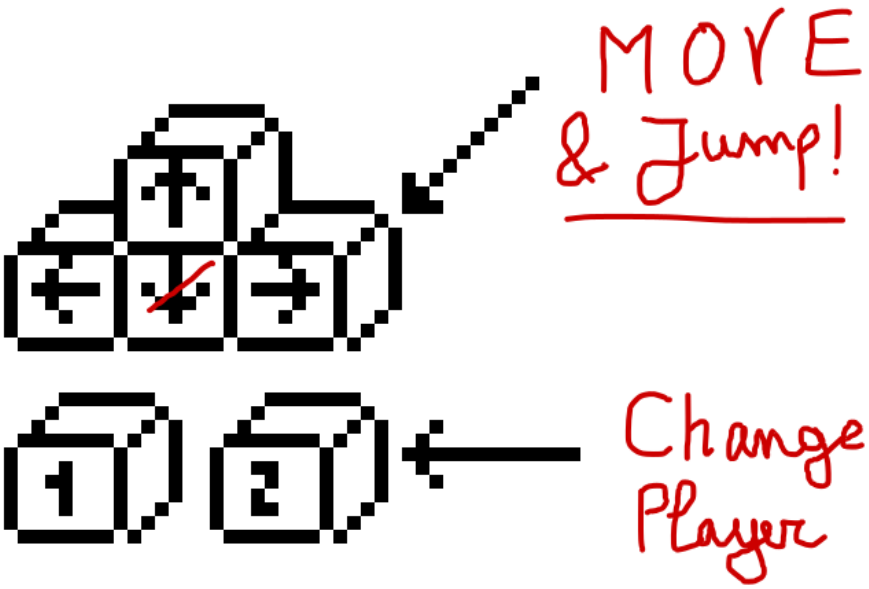
The Levels scenes:

The level scenes use the ***LevelScene*** class, which handles all the different complex events that take place in each level (level 1, 2, 3).

Each level we have is an instance of the *LevelScene* class, we create all the game objects we want and we pass them to the scene, with **setter** methods.
The whole class system offers a good abstraction layer, which makes the development smoother.

The game Idea is to control two players in turn to solve the puzzle and pass through the portals (doors).
In each level the key must be collected, in order to use the portal.

Controls:



The game also accepts W,A,D (or Z,Q,D azerty) for movement.
Let’s take a look at the first level:

Level 1:

As we can see, there are multiple game objects, from the platforms to the fire balls.

-There are two buttons, the back button is clickable, the other one shows us which player is selected (Blue or red).

-The players have their own Class which also inherits from GameObject, The Player Class has a more complex update method along with most methods, which override the parent class's methods.

-In the development the first step was to make the players move(respond to the arrow keys).

-I then made them stay inside the game screen with bounds checking which is the first form of collision I use.

-Added basic AABB(Axis Aligned Bounding Box) collision, which is the simplest and fastest algorithm of any collision type.

-Next was to add gravity and the jumping mechanics.

None of these were easy tasks an they each took a lot of time.

-Next was to add the key, spikes, fireBalls, and portal.

-Added collision Detection to check if a player got the key (or touched a spike).

-I made the two characters (blue and red) collide with each, so that the player can use that collision to jump to higher grounds.

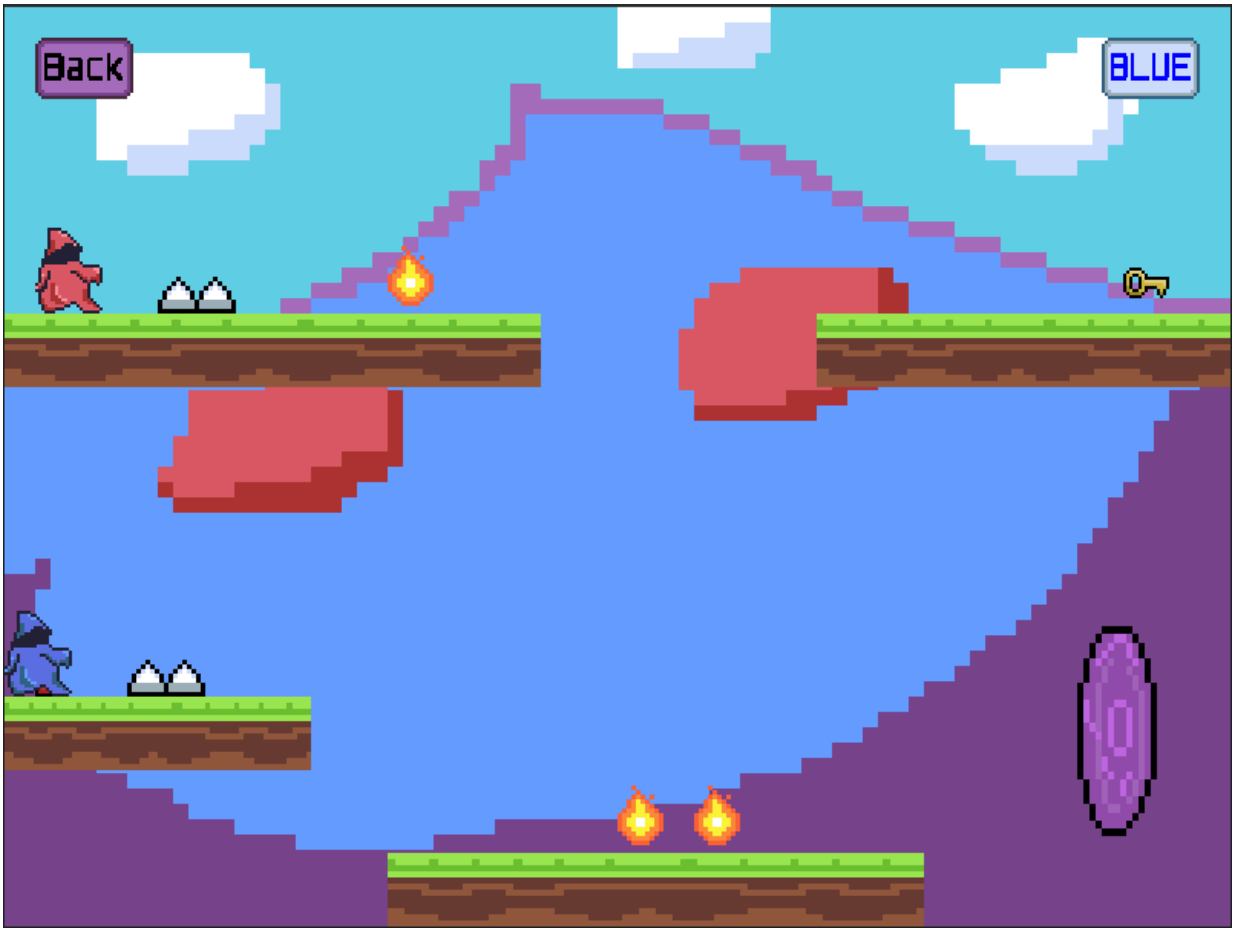
→As a matter of fact, that's the key to winning the first level.

Level 2:

In this Level we add two new objects; The button, and the FireWall. Which combined make for a new interesting mechanic.

-In order to get the key, the button must be pressed by either player. once that event takes place, the Fire wall is no longer rendered.

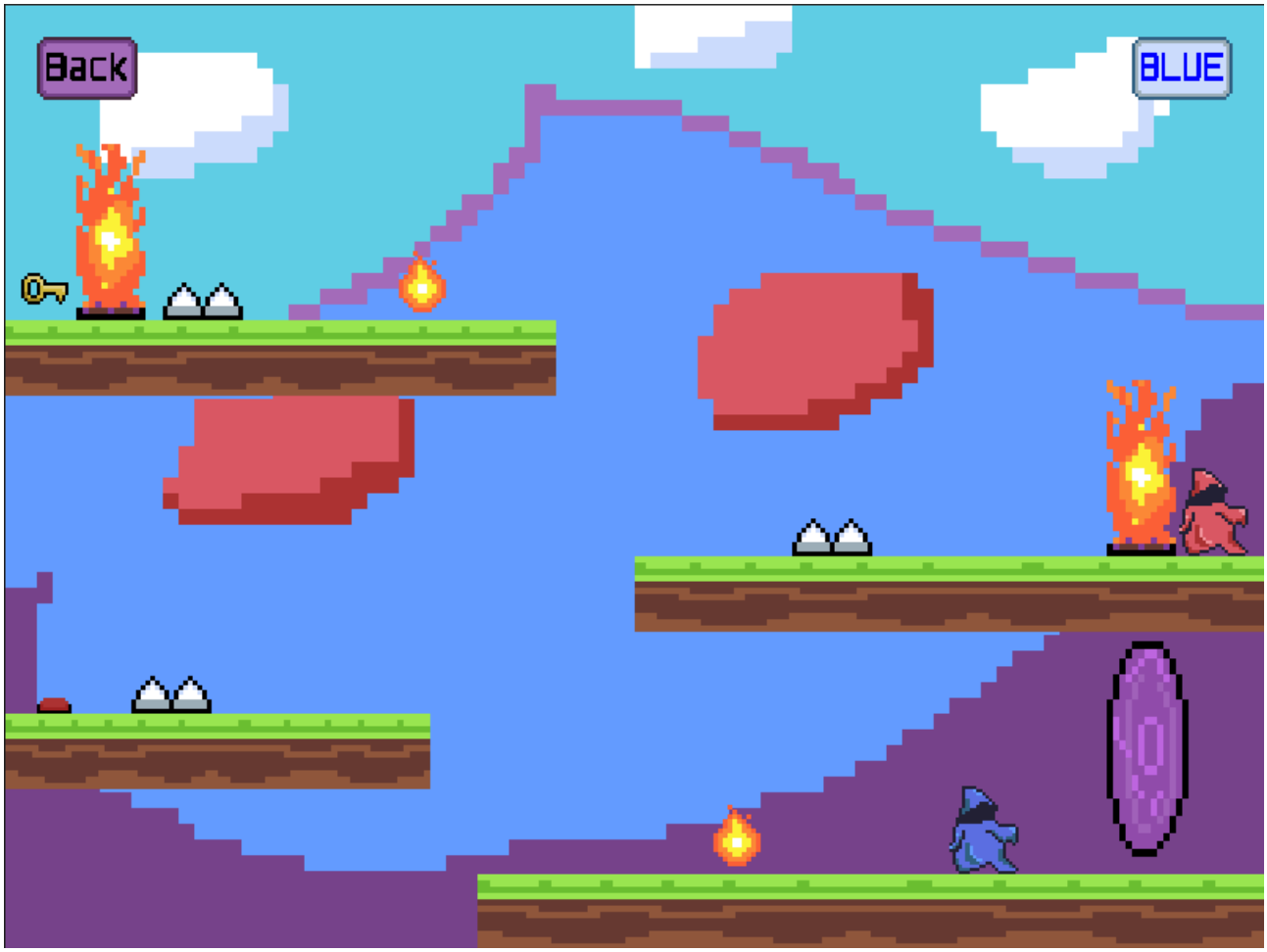
-Using the blue character, the player must keep the button pressed, while the red character gets the key safely. Once the key is obtained, we get a notification showing us that we have the key and thus can now pass through the door.



As we can see the Blue player is making the FireWall disabled, by pressing the button.

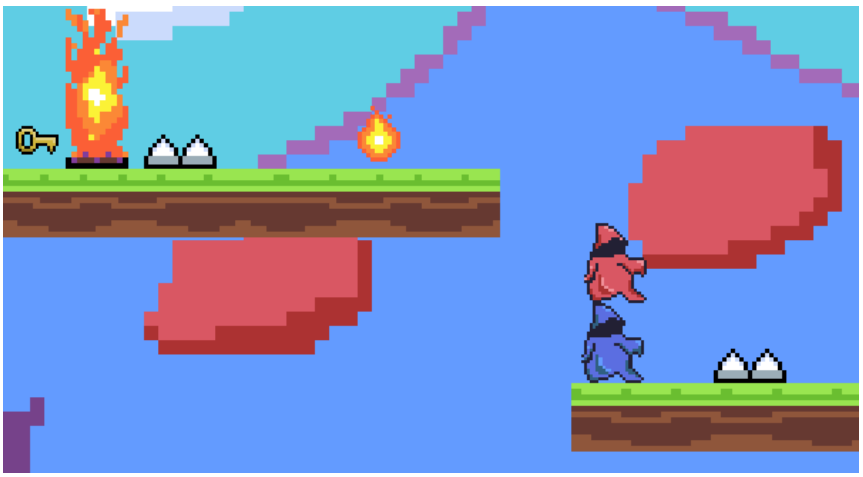
Level 3:

The third and final level is one which fuses both mechanics of having to use both characters to jump and firewalls.



The Red Character is Imprisoned !

In this level the Blue character starts off alone and must rescues Red. Once the players uses Blue to press the button and free Red, the jump to the key is too long, so both characters must be used.



-After that the button must be held again in order to free the key.

-In case of loss or success the LevelScene shows a notification.

After a few seconds (counter) The scene is either reset of left.
In case of Defeat the next Scene is the current level, and in case of Victory, the next scene is the level menu.

-The class system:

Scene -----> MenuScene, LevelScene.
GameObject -----> MenuItem, Player.

Scene class handles general issues, such as the loading of the texture sheet (static variable), the background Image, and the running Boolean.

GameObject class has two **SDL_FRect** variables which are used to draw the texture on.

*MenuScene: Has a vector of MenuItems, handles all the different choices.
*LevelScene: Vector of players, Vector of MenuItems, and of different Game Objects. (platforms, spikes, etc.)

*Player: the player class has a variety of methods to handle collision, gravity, and movement.

*Menu Item: has a new texture and a temporary surface to initiate the text.

Faced Problems:

I encountered a lot of set backs during the development of the game, a lot of it had to do with the fact that I'm new to SDL, and that every thing must be programmed from scratch, so I will ignore those.

-Rough movement:

The first Faced problem was that the player's movement was choppy and rough, I thought it had something to do with deltaTime.

```
deltaTime = (float)(SDL_GetTicks() - frameStart)/1000;
```

After outputting its value I noticed that indeed it is correct (0,0166 for 60 FPS).

So having eliminated deltaTime as the source of the problem, I noticed that the player uses SDL_Rect from the GameObject class.

Which is a structure that's composed of 4 integer values.

```
SDL_Rect = {int x, int y, int width, int height }
```

I look in the SDL Wiki and found an SDL_FRect which replaces the integer values for floating ones.

While this was indeed the solution I still had choppy movement because I handled the events and updated the new position in the same function. HandleEvents().
I updated the player's position in a separate function which made sure it was updated every frame. => solution.

-Collision:

There were many challenges concerning this.
I used AABB collision for all the methods, and it works.

Collision was hard to implement because the function must know where the player's heading. In order to put it back.
I also had a problem with different sizes (for example checking the key)

so I added a collisionDetection methode and a FullCollision Method.
-The first one is used to check for collision with all items, and returns true when the condition is met.
-FullCollision is the one that makes the character collide and not pass through elements, from any side.

-Jumping and Gravity:

There were many faced problems to achieve gravity and to jump. The main one was to make the player jump and not change direction in the air.

-I added a new collision method, VerticalCollision. Which checks whether the player has hit the ground.

-I made the player's x velocity equal to 0 when Airborne but that couldn't work as it made the player jump vertically only.

-The solution was to make the player move horizontally only when it's not Airborne, and to save it's velocity before jumping: beforeJumpVelocity, which is used for x movement in the air only.

-The next problem was that the player would glide after hitting the ground (since changing the x value was not possible), I made the player stop their x velocity after impact. BeforeJumpVelocity is set to zero.

-One texture sheet:

At first I was loading different images for each texture, but I then moved them all to a texture sheet (static texture in the Scene class) to optimize performance and lower memory usage by the GPU.

Ps: all the art was done by me using Aseprite.