1. **TP3 Update**

❖ **Added Graphics (Sources added to the code)**
- ➢ **Sources**
- ➢ Cannon & Cannonball: https://www.pixilart.com/art/bullet-bill-and-cannon-super-mario-bros-d28470dd1f9d0e8
- ➢ Plant: https://tenor.com/view/piranha-plant-super-smash-bros-super-mario-piranha-piranhas-gif-13386962
- ➢ Grid & Pipe & Gumba: https://www.deviantart.com/zedic0n/art/8-bit-Mario-Sprites-945075341
- ➢ Fire: https://tenor.com/view/fire-bar-mario-mario-obsticle-get-real-real-gif-21175776
- ➢ Heart(life): https://www.deviantart.com/joshuat1306/art/Super-Mario-Heart-2D-786384022
- ➢ Background: https://www.freepik.com/free-photos-vectors/mario-background
- ➢ Castle: https://www.youtube.com/watch?v=yOHaO-Bn90Q
- ➢ Font: https://www.fontbolt.com/font/super-mario-maker-font/
- ➢ Mario: https://www.pixilart.com/draw/mario-sprite-sheet-c8cb88f1ea7ee2d

❖ Added title page / instructions / win screen / lose screen

❖ Fire: rotates the fire sprites
❖ Mario: Different actions for moving and jumping
❖ Plant: Plant moving up and down
❖ Gumba: image moving horizontally
❖ Castle: Added image at the end of the map + added map size parameters
❖ Cannon: Added image for both cannon & cannonballs

❖ Enhanced overall code style + added comments

2. **TP2 Update**

❖ Added three enemies
- ➢ Gumba (moves horizontally)
- ➢ Plant (randomly summons a plant)
- ➢ Cannon (shoots cannonballs)

❖ Added unable to place message for any invalid actions during customization

3. TP1 Update

- Added Game Modes
  - Start Screen
    - Added instructions
  - Customization Screen
    - Moves to customization screen by pressing the space bar.
  - Play Screen
    - If the ‘play’ button is clicked from the customization screen
  - Win
    - If the character reaches the Castle
  - Lose
    - If the character runs out of lives (app.mario.life == 0)
- Castle Module (At the end of the map, mostly graphic focused)
- Basic Mario Motion / Features
  - Mario jump (gravity)
  - Interaction / Collision detection with blocks
  - Adding number of life remaining
  - Blinking while invincible
    - Mario does not take in damage for three seconds after taking damage
- Fire Block
  - Rotating lines, inflicts damages when touching mario
  - Collision detection
- Buttons for customization
  - 2 Blocks: Normal, Fire
  - 3 Enemies: To be added
  - Delete Button (Deletes any placed features
  - Play Button: finishes customization, moves on to the actual game.

4. **Project Description**:

Super Mario Maker (Temporary): My goal is to provide an entertaining, presentable game to the users that would be similar to Super Mario Maker, a Nintendo game. The game would consist of creator and play mode, where the user will start with the creator mode, creating the map for themselves, and then actually play inside the game as they transition to play mode. I am looking forward to a game that is smooth in terms of computational transitions and also an entertaining one with unique enemy features.

5. **Similar projects**:

<table>
<tr>
<td> 

Similar Project #1: Super Mario Maker by Nintendo</td>
<td> 

Similar Project #2: Super Pirate Maker by Clear Code</td>
</tr>
<tr>
<td>My initial inspiration for the project comes from the game Super Mario Maker by Nintendo. While the game itself has additional features, such as multiplayer mode and map sharing, I will be focusing primarily on the creator aspect of the game.

Also, in addition to what is already available in the game, I will implement additional features such as unique enemies and a difficulty calculator for the map.</td>
<td>Another similar project is Super Pirate Maker, a simplified version of Super Mario Maker using PyGame with multiple enemy features. The game focuses primarily on the graphics and sound, where there is only a single type of enemy and some types of landscape. However, my focus would be primarily on the interaction with a multitude of enemies, where there is not only one type but unique ones that users can place wherever they want to, each having a unique interaction with the user.</td>
</tr>
</table>

6. **Structural Plan**:

- ➔ Main File (Main.py)
  - ◆ OnAppStart
  - ◆ reDrawAll
  - ◆ onStep
  - ◆ onKeyHold
  - ◆ onMousePress
  - ◆ onKeyPress
- ➔ Classes
  - ◆ Grid
    - Click
    - Unclick (for deleting)
    - Draw
  - ◆ Button
    - Buttonpress
    - Draw
  - ◆ Fire
    - Add fire (rotating fire line)
    - Damage (determine if there's any damage inflicted)
    - Draw
  - ◆ Mario
    - Jump
    - Check_collision (with block)
    - Update
      - Update current status / check collision
    - Move left / right
    - Draw
    - Scroll screen
  - ◆ Castle
    - Draw
  - ◆ Instructions
    - Adds instructions
  - ◆ Gumba
    - Click, unclick, move, draw, damage
  - ◆ Plant
    - Click, unclick, move, draw, damage
  - ◆ Cannon
    - Click, unclick, move, draw, damage, summon cannonball
  - ◆ CannonBall
    - Damage, disappear, addcannonball
- ➔ Other functions
  - ◆ Scroll_screen: function to scroll/move the screen as the character moves
- ➔ Files:
  - ◆ Assets Folder: contains graphics, sound, and other media elements
  - ◆ (Settings.py): if needed, separate file for configurations and initial game settings

Algorithmic Plan:
Reference: [Side scroll examples from cmu 15-112 animations notes]

Algorithmic plan for screen scroll implementation: implementing smooth and responsive rendering of the game for the user can be challenging, where the game must continuously update and redraw the visible portion of the map to match the player's position, requiring precise calculations and synchronization. Below is a rough plan of how the screen scroll would be implemented in my game:

1. Determining screen boundaries
    a. Define the size of the game window (using canvas in CMU_graphics)
    b. Calculate the visible portion of the map within these boundaries based on player position
    c. Implement multiple algorithms that will shift the sprites' position inside the screen as it tracks the player's position
2. Updating Screen Display
    a. Monitor the player's position and compare it to the defined screen boundaries
    b. Trigger screen updates when the player approaches or reaches the screen edges
    c. Adjust rendering to display the relevant portion of the map and objects within the visible screen area
3. Scrolling Mechanism
    a. Determine the scroll speed and direction based on player movement
    b. Smoothly transition the screen view, avoiding abrupt movements for flickering

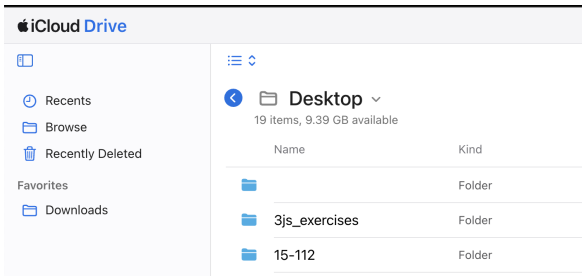
7. **Timeline Plan**

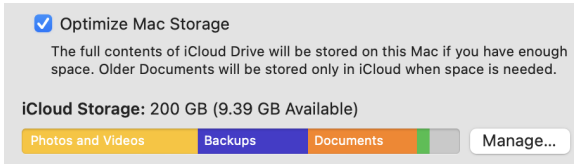
Week 1. TP0 - TP1: 11/20 - 11/26
1. Set up the basic project structure in Python
2. Develop the foundations for the map editor functionality (without the graphics)
3. Create a primary user interface for map creation, including several map/enemy features
4. Impleme the transition between creator and play modes.
    a. For play mode, also add the screen scroll feature

Week 2. TP1 - TP2: 11/27 - 12/1
1. Focus on enemy implementation: create a base Enemy class and complete one or two simple enemy types.
2. Integrate enemies into the map creation and gameplay screens.
3. Develop basic enemy behaviors like movement and interaction with the player.

Week 3. TP2 - TP3: 12/2 - 12/5
1. Refine and debug functionalities
2. Polish user interactions and gameplay mechanics
3. Add graphics and sound effects (preferably before)
4. Document the implemented functionalities and provide comments within the code
5. Polish remaining code.

8. **Version Control Plan**:





- My current backup plan for my project is an **iCloud backup**, where all of my files will be stored on the 15-112 in my desktop, constantly uploaded to iCloud whenever I am connected to the internet. I have already completed several projects using this method, where my projects were properly stored even when my computer / Macbook was malfunctioning.

9. **Module List**:

I am currently not planning on using any additional modules. While I considered using PyGame, the usage of PyGame would over-simplify and task and, therefore, is not adequate for use in this term project.