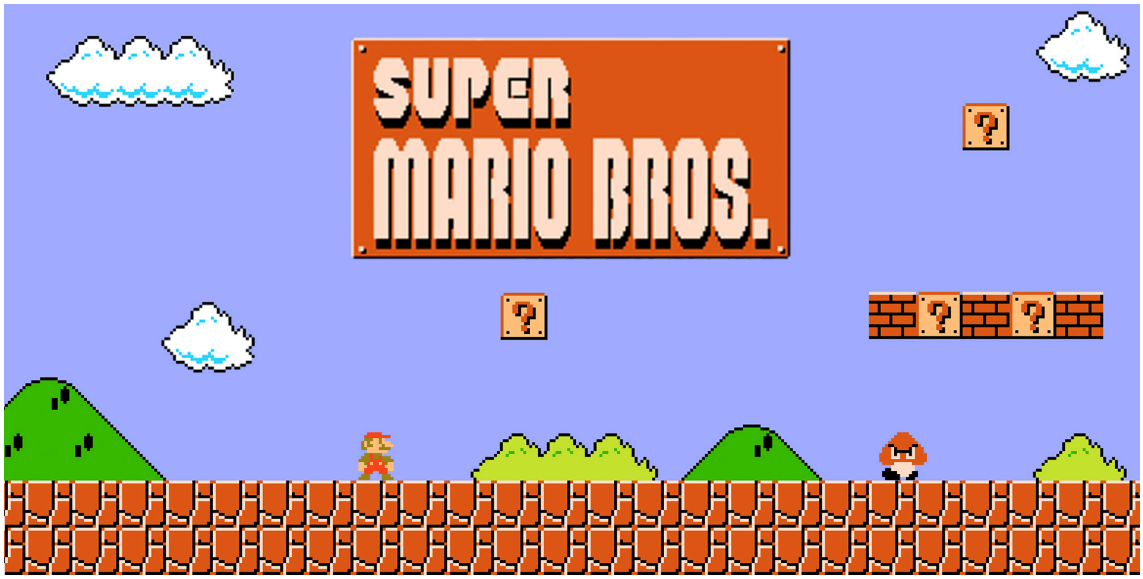
Super Mario Game

1) Some Super Mario games

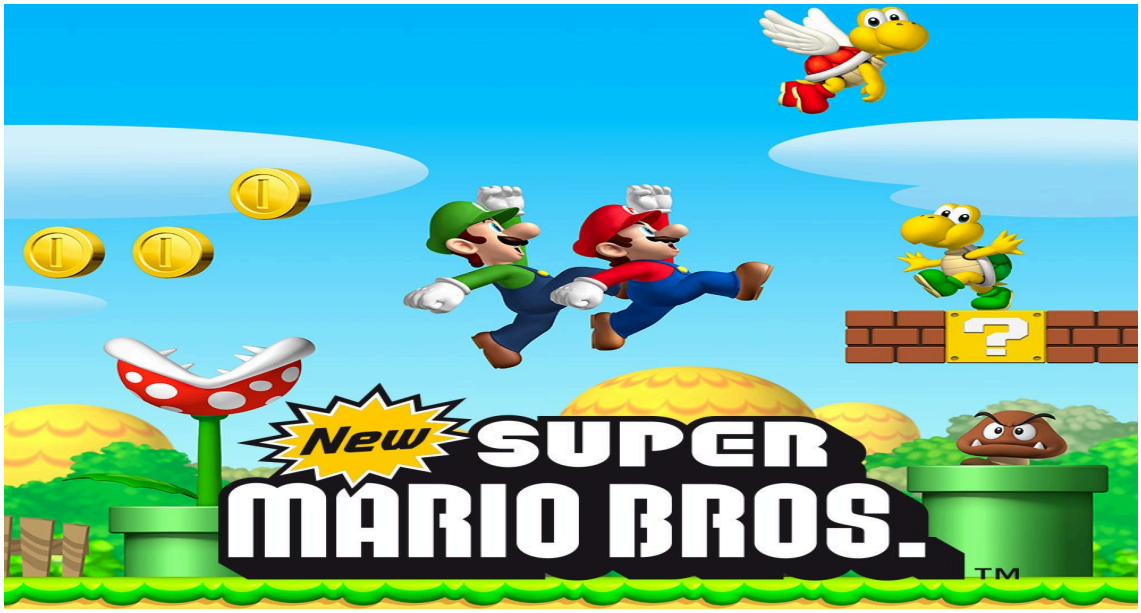
a) Super Mario Bros. (1985) (Wiki)

Reference: Super Mario Bros. (1985) Full Walkthrough NES Gameplay [Nostalgia]



b) New Super Mario Bros (Wiki)

Reference: New Super Mario Bros. DS HD - Full Game 100% Walkthrough



2) Project description

a) Objectives

In this project, you will develop a 2D Mario-style game using C++ with an emphasis on Object-Oriented Programming (OOP) principles. You are required to implement inheritance, encapsulation, polymorphism, and abstraction. Additionally, you must incorporate at least 5 design patterns (Factory, Singleton, Observer, ...) to create a modular, extensible, and well-structured game.

b) Requirements

Inheritance and Polymorphism:

1. **Example for Character System**:
    - A base class `Character` that defines the common attributes and methods (e.g., position, movement, jump).
    - Derive specific character classes, such as `Mario`, `Luigi`, `Enemy`, etc., from the base `Character` class.
    - Each character class should have unique behaviors or attributes (e.g., Mario can grow with a power-up, enemies can have different movement patterns).
2. **Example for Items and Power-ups**:
    - A base class `Item` with common methods like `activate()` and `collect()`.
    - Derive item classes like `Mushroom`, `Coin`, `FireFlower`, etc., from `Item`, each with different effects on characters.
    - Ensure each item interacts with the player character differently, affecting attributes like size, speed, or abilities.
3. **Example for polymorphism:** use polymorphism to handle different types of characters and items in a unified way (e.g., an array or vector of `Character*` to hold Mario, Luigi...).

Game Design Patterns:

- **Factory Pattern**: Use the Factory design pattern to instantiate characters and items dynamically. For example, depending on the level, different items or enemies should be created without directly specifying the classes.
- **Singleton Pattern**: Implement the Singleton pattern for the game manager or sound controller to ensure there is only one instance controlling the game state or sound effects.
- **Others..**.

Levels:

- The game must have 3 levels with increasing difficulty. Students can build 3 levels the same as the sample above.

User Input and Interaction:

- Capture user inputs (keyboard or controller) to control Mario's movements like walking, jumping, and interacting with objects.
- Implement collision detection to allow Mario to interact with enemies, items, and level boundaries.

Environment, Graphics and Sound:

- 2D/3D graphics game
- Implement basic sound effects for actions like jumping, collecting items, or defeating enemies.
- Game Engine:
    i. **SFML:** https://www.sfml-dev.org/learn.php
    ii. **SDL:** https://www.libsdl.org/
    iii. **raylib:** https://www.raylib.com/games.html
    iv. **Box2d**: https://github.com/erincatto/box2d

Game State Management:

- Use OOP principles to manage game states, such as starting the game, pausing, and ending the game.
- Store and update player progress, like score and remaining lives.
- Save/Load Game: Provide functionality to save and load game progress using file handling in C++.

Advanced features:

1. **AI for Enemies**:
    - Implement basic AI for enemy characters (e.g., Goombas or Koopas) that move automatically and interact with Mario based on proximity.
2. **Multiple Player Characters:**
    - Allow the player to switch between Mario, Luigi, or other characters, each with different abilities (e.g., Luigi can jump higher but runs slower).
    - Add a character selection screen at the beginning of the game or between levels.

Bonus features:

- Allow players to create and design their own levels, saving them as files that can be loaded and played later. This would require students to think about serialization and file handling.

c) Deliverables

1. **Source Code**: Well-documented ***C++ source code*** implementing the game.
2. **Design Documentation**:

- Class diagrams.
- Sequence diagrams explaining the use of design patterns (***optional***).
- Description of how OOP principles, design patterns are applied in the game.
- A demo video.

d) Evaluation Criteria

- Correct and efficient implementation of OOP concepts (inheritance, polymorphism, encapsulation).
- Effective use of 5 design patterns.
- Code readability, modularity, and adherence to C++ coding standards.
- Functionality and playability of the game.
- Creativity and originality in game mechanics and design.

3) Disclaimer

This game is a non-commercial, educational project created solely for learning and teaching purposes. All intellectual property rights of the original game and its assets belong to their respective owners. This project does not aim to infringe on any copyrights and is not intended for distribution or sale. No financial profit is being made from this project. If you are the copyright holder and have concerns about this use, please contact us to address them promptly.

Rubric for Super Mario Bros. Final Project

Functionality (65 points)

- Player Inputs, Movement and Collision (20 points)
- Enemy Behavior (10 points)
- Power-Ups and Items (10 points)
- 3 Level Completion (15 points)
- Sounds (10 points): effect, background

Design and Implementation (35 points)

- Object-Oriented Design (10 points)
- **Check 5 design patterns (25 points)**

Additional Requirements (15 points)

- AI (5 points)
- Multiple players (5 points)
- 3D Game (5 points)

Google Sheet

https://docs.google.com/spreadsheets/d/11P1AgDmA_io1BZS3azFMiLdnxrxHj0iNQif_vrhir5o/edit?usp=sharing