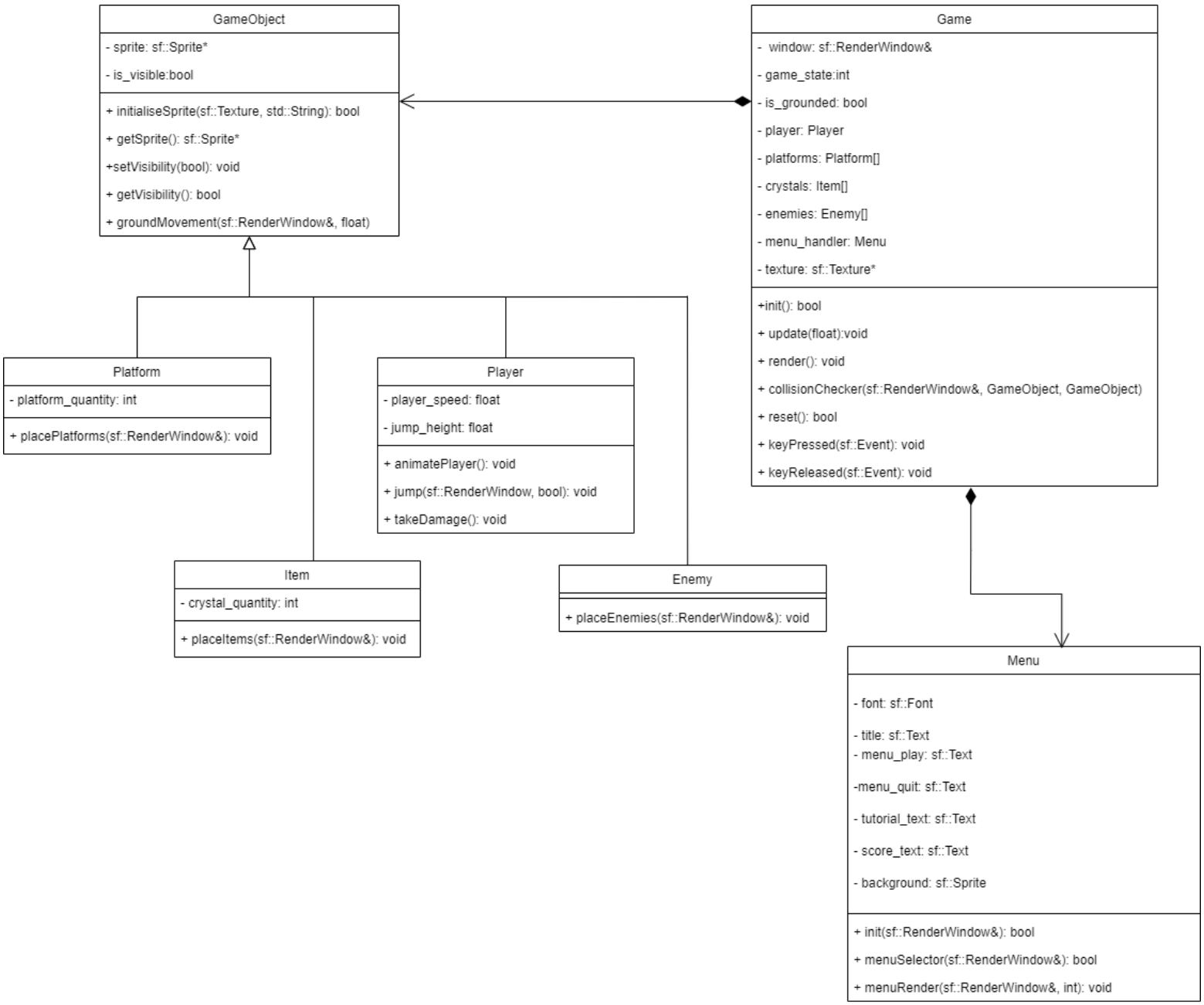
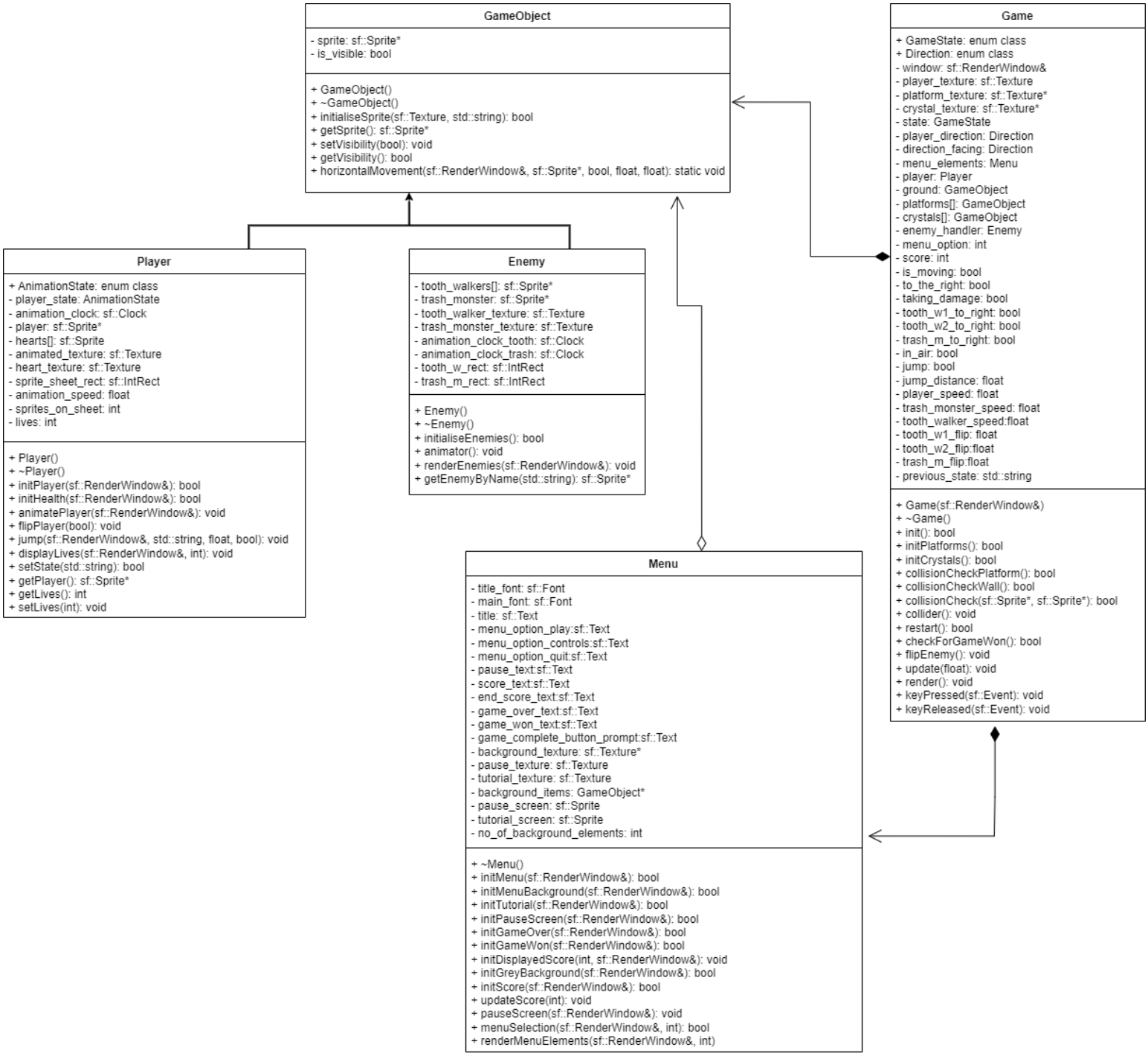

GameObject
- sprite: sf::Sprite*
- is_visible:bool
+ initialiseSprite(sf::Texture, std::String): bool
+ getSprite(): sf::Sprite*
+setVisibility(bool): void
+ getVisibility(): bool
+ groundMovement(sf::RenderWindow&, float)
Game
- window: sf::RenderWindow&
- game_state:int
- is_grounded: bool
- player: Player
- platforms: Platform[]
- crystals: Item[]
- enemies: Enemy[]
- menu_handler: Menu
- texture: sf::Texture*
+init(): bool
+ update(float):void
+ render(): void
+ collisionChecker(sf::RenderWindow&, GameObject, GameObject)
+ reset(): bool
+ keyPressed(sf::Event): void
+ keyReleased(sf::Event): void
Platform
- platform_quantity: int
+ placePlatforms(sf::RenderWindow&): void
Player
- player_speed: float
- jump_height: float
+ animatePlayer(): void
+ jump(sf::RenderWindow, bool): void
+ takeDamage(): void
Item
- crystal_quantity: int
+ placeItems(sf::RenderWindow&): void
Enemy
+ placeEnemies(sf::RenderWindow&): void
Menu
- font: sf::Font
- title: sf::Text
- menu_play: sf::Text
-menu_quit: sf::Text
- tutorial_text: sf::Text
- score_text: sf::Text
- background: sf::Sprite
+ init(sf::RenderWindow&): bool
+ menuSelector(sf::RenderWindow&): bool
+ menuRender(sf::RenderWindow&, int): void



Featured are two UML diagrams, the first being the initially planned one and the second being the final result. Initially there was intended to be platform and item classes that inherited from game object as well, but there would be no differences in the classes and so just using game objects for these worked fine and the need for the extra classes was unnecessary. Game object was not intended to be used in the menu class, but the background comes from different layers of sprites so to layer these up I made use of the game object class to save using multiple sprites. The player was initially meant to be a sprite from the game object class but I moved this to be stored in the class itself to make it easier to animate using functions in the class as opposed to repeatedly parsing in a sprite through parameters. Some menu items such as the pause screen and tutorial have been

changed to sprites instead of text, to save the need for a lot of different text items I just made backgrounds externally to display. Another change made was the enemies are handled directly in the enemy class similar to player, as opposed to making an array of enemies in the game class.

Pseudocode for player movement

```
IF player.idle
        player. move(0, jump height)
ELSE
        player.move(x distance, jump height)
```

above is the logic for a simple jump, where the x distance is negative or positive depending if the player is moving to the left or right this will be called in the key press when the player presses space, and will set a variable that jump is true to access this in the update. Idle will be a variable updated in the key press if the player is holding A or D

```
IF jump
        player.jump
        jump height += 1

        IF platform collision
                jump = false
                grounded = true
                reset jump height
ELSE
        IF platform collision
                player.move(0, fall height)
```

jump height begins with a negative because the reverse of negative to positive will reverse the direction and create an arced trajectory. if the player is not in the state of jumping then they are grounded and therefore it will check if there is a platform below them.

```
FOR i in range 0 to number of platforms
        IF player.y + player.height >= platform[i].y AND
           player.y + player.height <= platform[i].y + platform[i].height AND
           player.x >= platform[i].x AND
           player.x + player.width <= platform[i].x + platform[i].width
                RETURN true
```

above is a collision check between the player and the platforms and will return if the player is stood on the platform, it is used to detect if the player has hit the ground after jumping or if the player is about to fall after walking off the edge of the platform.