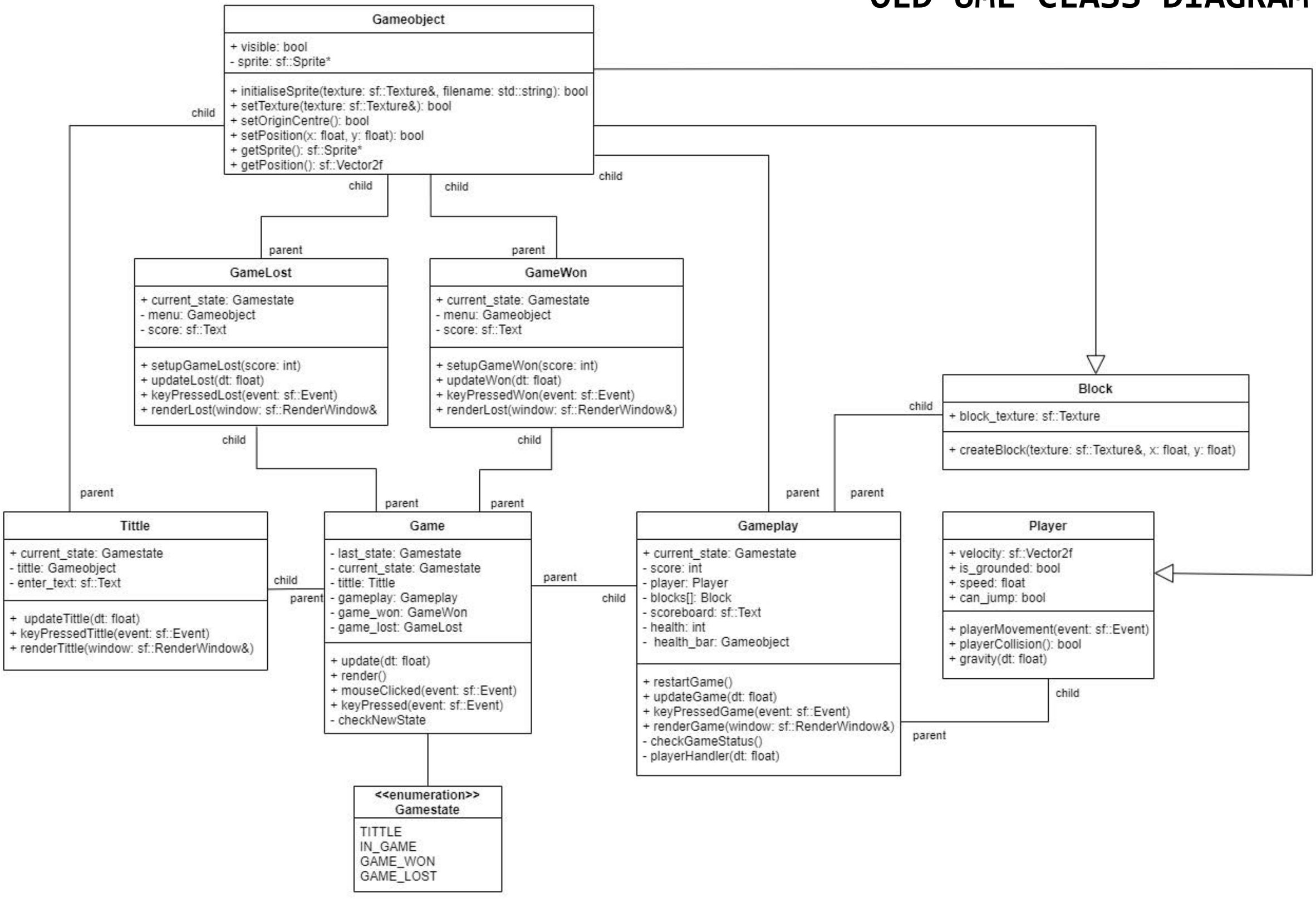
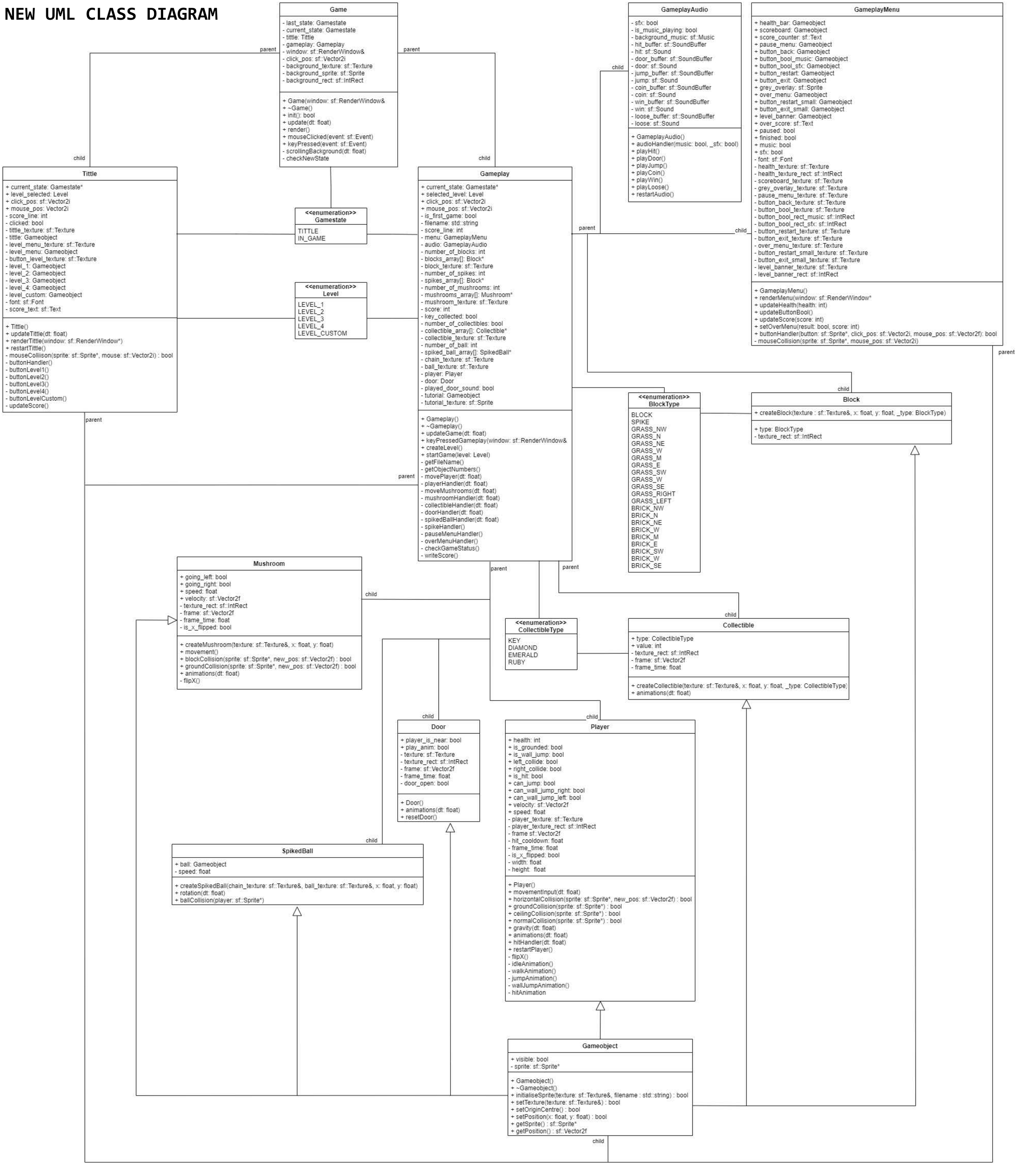
OLD UML CLASS DIAGRAM



NEW UML CLASS DIAGRAM



UML Class Diagram Differences/Improvements

The old UML Class Diagram is a lot simpler than the new one since I was not sure if I have time to add more features into the game. After I finished the basic player movement and level, I added more and more features such as different block types, rotating spiked balls, audio, etc.

The old UML Class Diagram also features the GameWon and GameLost Gamestates. However, after implementing them I decided it looks better to show the game over screens over the completed level.

Player Movement Pseudocode

Function for the terrain collision of the player

```
bool collision(Sprite block_sprite, Vector2f new_pos)
{
  Vector2f player_min = new_pos
  Vector2f player_max = (new_pos.x + player width,
                         new_pos.y + player height)

  Vector2f block_min = block sprite position
  Vector2f block_max = (block_min.x + block width,
                        block_max.y + block height)

  if (player_min.x < block_max.x and
      player_max.x > sprite_min.x and
      player_min.y < block_max.y and
      player_max.y > sprite_min.y):
        return true
  else:
        return false
}
```

Function for the movement input of the player

```
void movementInput(float deltaTime)
{
  if (Keyboard A Key is pressed):
        if (velocity.x > -1):
            velocity.x -= 5 * deltaTime
        else:
            velocity.x = -1

  else if (Keyboard D Key is pressed):
        if (velocity.x < 1):
            velocity.x += 5* deltaTime
        else :
            velocity.x = 1
  else:
      velocity.x = 0

  if (Keyboard SPACE Key is pressed and player is on the ground):
      velocity.y = 4
}
```

Function for gravity affecting the player

```
bool gravity(float deltaTime)
{
  if (player is not on ground):
      if (velocity.y > -9.81):
          velocity.y -= 9.81 * deltaTime
      else:
          velocity.y = -9.81
}
```

Function that handles player movement

```
void playerHandler(float deltaTime)
{
  gravity(deltaTime)
  movementInput(deltaTime)

  Vector2f old_pos = current player position
  Vector2f new_pos = (old_pos.x + (velocity.x * speed * deltaTime),
                      Old_pos.y + (velocity.y * speed * deltaTime))

  Bool collision_found = false

  for each Block in blocks_array:
      if (collision_found is false):
          if (collision(block sprite, new_pos)):
              collision_found = true

  if (collision_found is false):
      move player to new_pos
}
```

The final code for the player movement ended up much more complicated than this, I had to divide the collision into 3 different functions (horizontal, ground and ceiling collisions) to facilitate the basic player movement, wall sliding and wall jumping. However, most of the code uses the same logic as the pseudocode above.