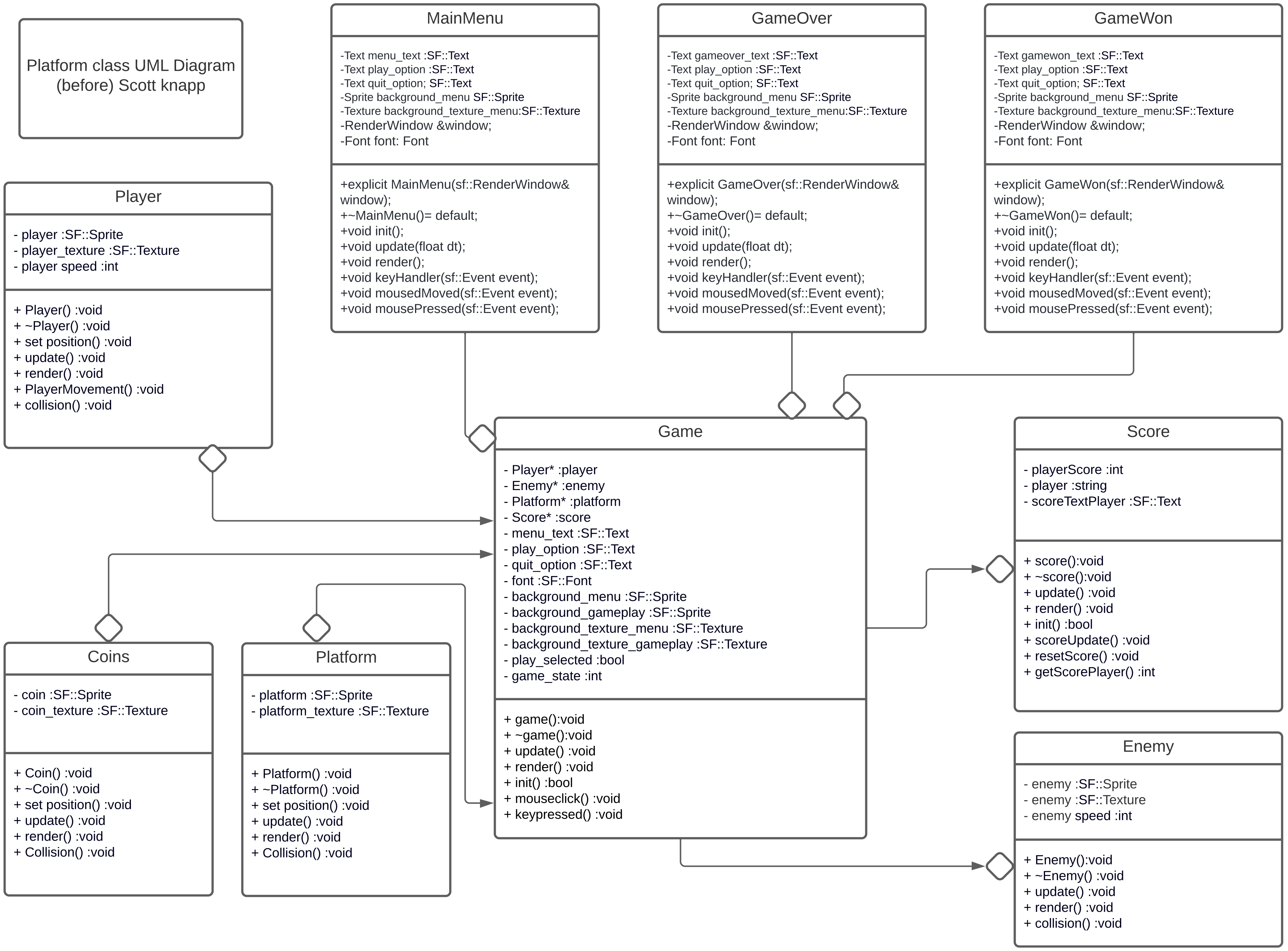
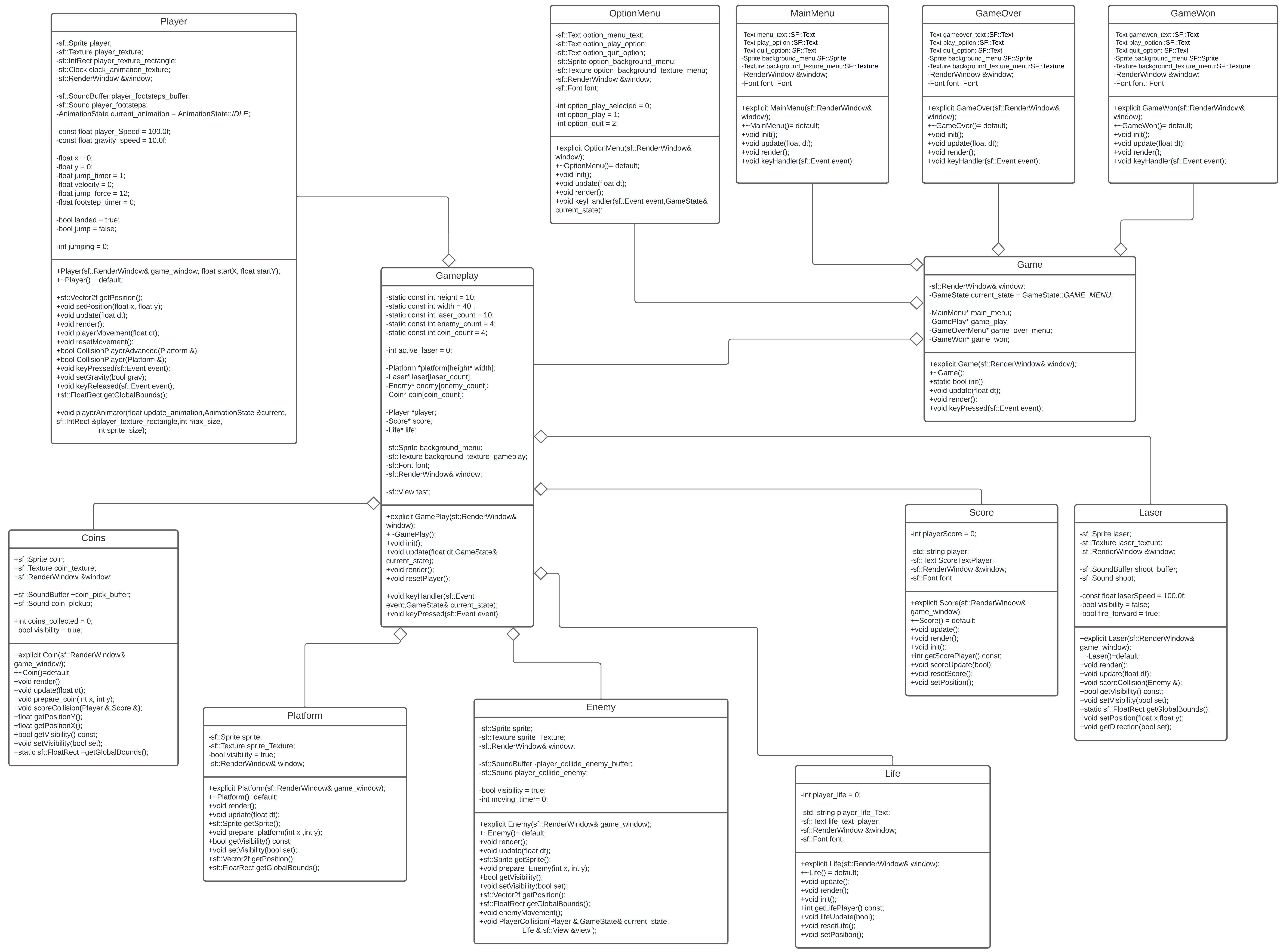
Platform class UML Diagram (before) Scott knapp



Movement pseudo code

```
Gravity = -1
Playerspeed = 5

If input key is D
     vector .x = 1

     (Player moves right on screen as x increases)

If input key is A
     vector .x = -1

     (Player moves left on screen as x decreases)

If  key released A or D
     vector .x = 0

     (Player stands still on screen as x is not changed)

If key pressed is Spacebar and islanded true

     Sprite set position .y += 5
     (Player jumps in the air increasing the y)
     Bool is_landed = false

If Key released is Spacebar
     vector. y = -1

     (Player falling)

Update;

Setposition.x += vector. X * playerSpeed * delta_time
Setposition.y += vector. Y * gravity * delta_time

If player sprite bottom intersects with top of platform sprite
     vector.y = 0

     Bool is_landed = true
     (Player does not jump not increasing the y)
```

Platform class UML Diagram
(After) Scott knapp



Player
```
-sf::Sprite player;
-sf::Texture player_texture;
-sf::IntRect player_texture_rectangle;
-sf::Clock clock_animation_texture;
-sf::RenderWindow &window;

-sf::SoundBuffer player_footsteps_buffer;
-sf::Sound player_footsteps;
-AnimationState current_animation = AnimationState::IDLE;

-const float player_Speed = 100.0f;
-const float gravity_speed = 10.0f;

-float x = 0;
-float y = 0;
-float jump_timer = 1;
-float velocity = 0;
-float jump_force = 12;
-float footstep_timer = 0;

-bool landed = true;
-bool jump = false;

-int jumping = 0;
```
```
+Player(sf::RenderWindow& game_window, float startX, float startY);
+~Player() = default;

+sf::Vector2f getPosition();
+void setPosition(float x, float y);
+void update(float dt);
+void render();
+void playerMovement(float dt);
+void resetMovement();
+bool CollisionPlayerAdvanced(Platform &);
+bool CollisionPlayer(Platform &);
+void keyPressed(sf::Event event);
+void setGravity(bool grav);
+void keyReleased(sf::Event event);
+sf::FloatRect getGlobalBounds();

+void playerAnimator(float update_animation,AnimationState &current,
sf::IntRect &player_texture_rectangle,int max_size,
                   int sprite_size);
```

OptionMenu
```
-sf::Text option_menu_text;
-sf::Text option_play_option;
-sf::Text option_quit_option;
-sf::Sprite option_background_menu;
-sf::Texture option_background_texture_menu;
-sf::RenderWindow &window;
-sf::Font font;

-int option_play_selected = 0;
-int option_play = 1;
-int option_quit = 2;
```
```
+explicit OptionMenu(sf::RenderWindow&
window);
+~OptionMenu()= default;
+void init();
+void update(float dt);
+void render();
+void keyHandler(sf::Event event,GameState&
current_state);
```

MainMenu
```
-Text menu_text :SF::Text
-Text play_option :SF::Text
-Text quit_option; SF::Text
-Sprite background_menu SF::Sprite
-Texture background_texture_menu:SF::Texture
-RenderWindow &window;
-Font font: Font
```
```
+explicit MainMenu(sf::RenderWindow&
window);
+~MainMenu()= default;
+void init();
+void update(float dt);
+void render();
+void keyHandler(sf::Event event);
```

GameOver
```
-Text gameover_text :SF::Text
-Text play_option :SF::Text
-Text quit_option; SF::Text
-Sprite background_menu SF::Sprite
-Texture background_texture_menu:SF::Texture
-RenderWindow &window;
-Font font: Font
```
```
+explicit GameOver(sf::RenderWindow&
window);
+~GameOver()= default;
+void init();
+void update(float dt);
+void render();
+void keyHandler(sf::Event event);
```

GameWon
```
-Text gamewon_text :SF::Text
-Text play_option :SF::Text
-Text quit_option; SF::Text
-Sprite background_menu SF::Sprite
-Texture background_texture_menu:SF::Texture
-RenderWindow &window;
-Font font: Font
```
```
+explicit GameWon(sf::RenderWindow&
window);
+~GameWon()= default;
+void init();
+void update(float dt);
+void render();
+void keyHandler(sf::Event event);
```

Game
```
-sf::RenderWindow& window;
-GameState current_state = GameState::GAME_MENU;

-MainMenu* main_menu;
-GamePlay* game_play;
-GameOverMenu* game_over_menu;
-GameWon* game_won;
```
```
+explicit Game(sf::RenderWindow& window);
+~Game();
+static bool init();
+void update(float dt);
+void render();
+void keyPressed(sf::Event event);
```

Gameplay
```
-static const int height = 10;
-static const int width = 40 ;
-static const int laser_count = 10;
-static const int enemy_count = 4;
-static const int coin_count = 4;

-int active_laser = 0;

-Platform *platform[height* width];
-Laser* laser[laser_count];
-Enemy* enemy[enemy_count];
-Coin* coin[coin_count];

-Player *player;
-Score* score;
-Life* life;

-sf::Sprite background_menu;
-sf::Texture background_texture_gameplay;
-sf::Font font;
-sf::RenderWindow& window;

-sf::View test;
```
```
+explicit GamePlay(sf::RenderWindow&
window);
+~GamePlay();
+void init();
+void update(float dt,GameState&
current_state);
+void render();
+void resetPlayer();

+void keyHandler(sf::Event
event,GameState& current_state);
+void keyPressed(sf::Event event);
```

Score
```
-int playerScore = 0;

-std::string player;
-sf::Text ScoreTextPlayer;
-sf::RenderWindow &window;
-sf::Font font
```
```
+explicit Score(sf::RenderWindow&
game_window);
+~Score() = default;
+void update();
+void render();
+void init();
+int getScorePlayer() const;
+void scoreUpdate(bool);
+void resetScore();
+void setPosition();
```

Laser
```
-sf::Sprite laser;
-sf::Texture laser_texture;
-sf::RenderWindow &window;

-sf::SoundBuffer shoot_buffer;
-sf::Sound shoot;

-const float laserSpeed = 100.0f;
-bool visibility = false;
-bool fire_forward = true;
```
```
+explicit Laser(sf::RenderWindow&
game_window);
+~Laser()=default;
+void render();
+void update(float dt);
+void scoreCollision(Enemy &);
+bool getVisibility() const;
+void setVisibility(bool set);
+static sf::FloatRect getGlobalBounds();
+void setPosition(float x,float y);
+void getDirection(bool set);
```

Coins
```
+sf::Sprite coin;
+sf::Texture coin_texture;
+sf::RenderWindow &window;

+sf::SoundBuffer +coin_pick_buffer;
+sf::Sound coin_pickup;

+int coins_collected = 0;
+bool visibility = true;
```
```
+explicit Coin(sf::RenderWindow&
game_window);
+~Coin()=default;
+void render();
+void update(float dt);
+void prepare_coin(int x, int y);
+void scoreCollision(Player &,Score &);
+float getPositionY();
+float getPositionX();
+bool getVisibility() const;
+void setVisibility(bool set);
+static sf::FloatRect +getGlobalBounds();
```

Platform
```
-sf::Sprite sprite;
-sf::Texture sprite_Texture;
-bool visibility = true;
-sf::RenderWindow& window;
```
```
+explicit Platform(sf::RenderWindow& game_window);
+~Platform()=default;
+void render();
+void update(float dt);
+sf::Sprite getSprite();
+void prepare_platform(int x ,int y);
+bool getVisibility() const;
+void setVisibility(bool set);
+sf::Vector2f getPosition();
+sf::FloatRect getGlobalBounds();
```

Enemy
```
-sf::Sprite sprite;
-sf::Texture sprite_Texture;
-sf::RenderWindow& window;

-sf::SoundBuffer -player_collide_enemy_buffer;
-sf::Sound player_collide_enemy;

-bool visibility = true;
-int moving_timer= 0;
```
```
+explicit Enemy(sf::RenderWindow& game_window);
+~Enemy()= default;
+void render();
+void update(float dt);
+sf::Sprite getSprite();
+void prepare_Enemy(int x, int y);
+bool getVisibility();
+void setVisibility(bool set);
+sf::Vector2f getPosition();
+sf::FloatRect getGlobalBounds();
+void enemyMovement();
+void PlayerCollision(Player &,GameState& current_state,
                 Life &,sf::View &view );
```

Life
```
-int player_life = 0;

-std::string player_life_Text;
-sf::Text life_text_player;
-sf::RenderWindow &window;
-sf::Font font;
```
```
+explicit Life(sf::RenderWindow& window);
+~Life() = default;
+void update();
+void render();
+void init();
+int getLifePlayer() const;
+void lifeUpdate(bool);
+void resetLife();
+void setPosition();
```

Difference between UML diagram before and after

The big difference between the UML diagrams is that I move a lot of stuff out of the game class and moved it to its own class gameplay. This made more sense to keep all elements of the gameplay attached to this class, this allowed me to turn the game class into a state machine and this allows for easier to read code.

Another big difference is the laser class I added, originally, I was not going to have a way to kill enemies but I thought this would be a cool gameplay mechanic and give the gameplay more variety the game is still completable without using the laser but still very fun to play with.

The game menu was also added much later as this was going to be on-screen during gameplay but felt too intrusive to the gameplay so was moved into its own menu class that's is shown before gameplay and I feel still gives the same level of functionality.

A lot of the sound and animation variables were also not added in the original class diagram as they were very much a stretch goal and did not want to pressure myself with them originally. Sounds and animations will be a standard in all my games going forward, hopefully

I also underestimated the amount of variables I would need for player movements such as isJumping, islanded and timers that went with them and the sound and animations.

Overall, most of my UML diagram was in place before with most of the features I wanted it was mostly tweaking them for better gameplay performance and easier to read code.