Wstępny opis projektu

Bartłomiej Bielak

Gra RPG Demon Hunter

Klasa, **AnimationComponent**, jest odpowiedzialna za zarządzanie animacjami dla obiektów w grze. Oto opis jej składowych i metod:

Składowe:

- **sprite**: Referencja do obiektu **sf::Sprite**, który będzie animowany.
- **textureSheet**: Referencja do tekstury zawierającej klatki animacji.
- **animations**: Mapa przechowująca nazwy animacji i wskaźniki do obiektów **Animation**.
- **lastAnimation**: Wskaźnik do ostatnio odtwarzanej animacji.
- **priorityAnimation**: Wskaźnik do animacji o priorytetowej odtwarzanej animacji.

Metody:

- **Konstruktor**: Inicjalizuje **AnimationComponent** z podanym **sprite** i **texture_sheet**.
- **Destruktor**: Zwolnienie pamięci zaalokowanej dla obiektów **Animation**.
- **isDone(const std::string key)**: Zwraca informację, czy animacja o podanej nazwie **key** została zakończona.
- **addAnimation**: Dodaje nową animację o nazwie **key** do mapy **animations**.
- **play(const std::string key, const float& dtime, const bool priority = false)**: Odtwarza animację o nazwie **key**. **dtime** to czas delta, a **priority** określa, czy animacja ma pierwszeństwo nad aktualnie odtwarzaną.
- **play(const std::string key, const float& dtime, const float& modifier, const float& modifier_max, const bool priority = false)**: Odtwarza animację z modyfikatorem czasu. **modifier** to procentowy współczynnik modyfikacji czasu, **modifier_max** to maksymalny dopuszczalny współczynnik modyfikacji, **priority** określa, czy animacja ma pierwszeństwo nad aktualnie odtwarzaną.

Klasa wewnętrzna Animation:

- Klasa ta definiuje pojedynczą animację.
- Przechowuje informacje o klatce, czasie animacji itp.
- Posiada metody **isDone**, **play**, **reset**, które kontrolują stan animacji.

Klasa **HitboxComponent**, która reprezentuje hitbox (obszar kolizji) dla obiektu w grze. Oto opis klasy i jej metod:

Klasa HitboxComponent:

Prywatne składowe:

- **sprite**: Referencja do obiektu **sf::Sprite**, do którego przypisany jest hitbox.
- **hitbox**: **sf::RectangleShape** reprezentujący obszar hitbox.
- **nextPosition**: **sf::FloatRect** reprezentujący następną pozycję hitboxa.
- **offsetX, offsetY**: Przesunięcie hitboxa względem sprite'a w osiach x i y.

Publiczne metody:

- **Konstruktor**: Inicjalizuje **HitboxComponent** z podanym **sprite**, przesunięciem **offset_x** i **offset_y**, oraz szerokością i wysokością hitboxa.
- **Destruktor**: Zwolnienie pamięci.
- **getPosition()**: Zwraca aktualną pozycję hitboxa.
- **getGlobalBounds()**: Zwraca globalne granice hitboxa.
- **getNextPosition(const sf::Vector2f& velocity)**: Zwraca następną pozycję hitboxa na podstawie przekazanej prędkości **velocity**.
- **setPosition(const sf::Vector2f& position)**: Ustawia pozycję hitboxa na podstawie przekazanego wektora pozycji **position**.
- **setPosition(const float x, const float y)**: Przeciążona wersja metody **setPosition** dla przekazanych współrzędnych x i y.
- **intersects(const sf::FloatRect& frect)**: Sprawdza, czy hitbox przecina się z podanym obszarem prostokątnym **frect**.
- **update()**: Aktualizuje hitbox.
- **render(sf::RenderTarget& target)**: Renderuje hitbox na podanym obiekcie renderującym **target**

Klasa **MovementComponent**, która odpowiada za obsługę ruchu obiektów w grze. Oto opis klasy i jej metod:

Klasa MovementComponent:

Prywatne składowe:

- **sprite**: Referencja do obiektu **sf::Sprite**, który będzie poruszany.
- **maxVelocity**: Maksymalna prędkość poruszania się obiektu.
- **acceleration**: Wartość przyspieszenia obiektu.
- **deceleration**: Wartość spowolnienia obiektu.
- **velocity**: Wektor reprezentujący prędkość obiektu.

Publiczne metody:

- **Konstruktor**: Inicjalizuje **MovementComponent** z podanym **sprite**, **maxVelocity**, **acceleration** i **deceleration**.

- **Destruktor**: Zwolnienie pamięci.
- **getMaxVelocity()**: Zwraca maksymalną prędkość obiektu.
- **getVelocity()**: Zwraca aktualną prędkość obiektu.
- **getState(const short unsigned state)**: Sprawdza stan ruchu obiektu na podstawie wartości z enuma **movement_states**.
- **stopVelocity()**: Zatrzymuje prędkość obiektu we wszystkich kierunkach.
- **stopVelocityX()**: Zatrzymuje prędkość obiektu w osi X.
- **stopVelocityY()**: Zatrzymuje prędkość obiektu w osi Y.
- **move(const float dir_x, const float dir_y, const float& dtime)**: Przemieszcza obiekt na podstawie podanego kierunku i czasu delta.
- **update(const float& dtime)**: Aktualizuje stan ruchu obiektu na podstawie przyspieszenia i spowolnienia.

Klasa **PauseMenu**, która reprezentuje menu pauzy w grze. Oto opis klasy i jej metod:

Klasa PauseMenu:

Prywatne składowe:

- **font**: Referencja do czcionki używanej do wyświetlania tekstu w menu.
- **menuText**: Tekst wyświetlany w menu.
- **background**: Prostokąt reprezentujący tło menu pauzy.
- **container**: Prostokąt reprezentujący kontener na przyciski w menu.
- **buttons**: Mapa przechowująca przyciski w menu.

Publiczne metody:

- **Konstruktor**: Inicjalizuje **PauseMenu** z podanym oknem **window** i czcionką **font**.
- **Destruktor**: Zwolnienie pamięci.
- **getButtons()**: Zwraca mapę przycisków w menu.
- **isButtonPressed(const std::string key)**: Sprawdza, czy przycisk o podanej nazwie został naciśnięty.
- **addButton(const std::string key, float y, const std::string text)**: Dodaje przycisk do menu o podanej nazwie, pozycji **y** i tekście.
- **update(const sf::Vector2i& mousePositionWindow)**: Aktualizuje stan menu na podstawie pozycji myszy.
- **render(sf::RenderTarget& target)**: Renderuje menu na podanym obiekcie renderującym **target**.

klasa **Player**, która reprezentuje gracza w grze. Oto opis klasy i jej metod:

Klasa Player:

Prywatne składowe:

- **attacking**: Flaga określająca, czy gracz wykonuje atak.

Prywatne metody:

- **initVariables()**: Inicjalizuje zmienne wewnętrzne gracza.
- **initComponents()**: Inicjalizuje komponenty gracza.

Publiczne metody:

- **Konstruktor**: Inicjalizuje gracza z podanymi współrzędnymi **x** i **y** oraz teksturą **texture_sheet**.
- **Destruktor**: Zwolnienie pamięci.
- **updateAttack()**: Aktualizuje stan ataku gracza.
- **updateAnimation(const float& dtime)**: Aktualizuje animację gracza na podstawie czasu delta.
- **setPosition(const float x, const float y)**: Ustawia pozycję gracza na podanych współrzędnych **x** i **y**.
- **move(const float dir_x, const float dir_y, const float& dtime)**: Przesuwa gracza na podstawie podanego kierunku i czasu delta.
- **stopVelocity()**: Zatrzymuje prędkość gracza we wszystkich kierunkach.
- **stopVelocityX()**: Zatrzymuje prędkość gracza w osi X.
- **stopVelocityY()**: Zatrzymuje prędkość gracza w osi Y.
- **update(const float& dtime)**: Aktualizuje stan gracza na podstawie czasu delta.
- **render(sf::RenderTarget& target)**: Renderuje gracza na podanym obiekcie renderującym **target**.

Klasa bazową **Entity**, która reprezentuje ogólny obiekt w grze. Oto opis klasy i jej metod:

Klasa Entity:

Prywatne metody:

- **initVariables()**: Inicjalizuje zmienne wewnętrzne obiektu.

Chronione składowe:

- **sprite**: Obiekt **sf::Sprite** reprezentujący wygląd obiektu.
- **hitboxComponent**: Wskaźnik do komponentu obszaru kolizji.
- **movementComponent**: Wskaźnik do komponentu ruchu.
- **animationComponent**: Wskaźnik do komponentu animacji.
- **attributeComponent**: Wskaźnik do komponentu atrybutów.

Publiczne metody:

- **Konstruktor**: Inicjalizuje **Entity**.
- **Destruktor**: Zwolnienie pamięci zaalokowanej dla komponentów.
- **setTexture(sf::Texture& texture)**: Ustawia teksturę obiektu.
- **createHitboxComponent(sf::Sprite& sprite, float offset_x, float offset_y, float width, float height)**: Tworzy komponent obszaru kolizji.
- **createMovementComponent(const float maxVelocity, const float acceleration, const float deceleration)**: Tworzy komponent ruchu.
- **createAnimationComponent(sf::Texture& texture_sheet)**: Tworzy komponent animacji.
- **createAttributeComponent()**: Tworzy komponent atrybutów.
- **getPosition() const**: Zwraca aktualną pozycję obiektu.
- **getGridPosition(const int gridSizeI) const**: Zwraca pozycję obiektu na siatce gry.
- **getGlobalBounds() const**: Zwraca globalne granice obiektu.
- **getNextPositionBounds(const float& dtime) const**: Zwraca granice następnej pozycji obiektu na podstawie czasu delta.
- **setPosition(const float x, const float y)**: Ustawia pozycję obiektu na podanych współrzędnych.
- **move(const float dir_x, const float dir_y, const float& dtime)**: Przesuwa obiekt na podstawie podanego kierunku i czasu delta.
- **stopVelocity()**, **stopVelocityX()**, **stopVelocityY()**: Zatrzymują prędkość obiektu.
- **update(const float& dtime)**: Aktualizuje stan obiektu na podstawie czasu delta.
- **render(sf::RenderTarget& target)**: Renderuje obiekt na podanym obiekcie renderującym.

Klasa **TileMap**, która reprezentuje mapę kafelków w grze. Oto opis klasy i jej metod:

Klasa TileMap:

Prywatne składowe:

- **clear()**: Metoda prywatna do czyszczenia mapy.
- **gridSizeF, gridSizeI**: Rozmiar siatki kafelków w formie zmiennoprzecinkowej i całkowitoliczbowej.
- **layers**: Liczba warstw mapy.
- **maxSizeWorldGrid, maxSizeWorldF**: Maksymalny rozmiar mapy w siatce kafelków i pikselach.
- **map**: Wektor przechowujący kafelki w formie tablicy wielowymiarowej.
- **deferredRenderStack**: Stos przechowujący kafelki do renderowania w kolejności opóźnionej.

- **textureFile**: Ścieżka do pliku tekstury kafelków.
- **tileSheet**: Tekstura przechowująca kafelki.
- **collisionBox**: Prostokąt reprezentujący obszar kolizji na mapie.
- **fromX, toX, fromY, toY, layer**: Zmienne pomocnicze do renderowania tylko widocznych kafelków.

Publiczne metody:

- **Konstruktor**: Inicjalizuje **TileMap** z podanym rozmiarem siatki kafelków, szerokością, wysokością i ścieżką do pliku tekstury.
- **Destruktor**: Zwolnienie pamięci.
- **getTileSheet()**: Zwraca wskaźnik do tekstury kafelków.
- **getLayerSize(const int x, const int y, const int layer)**: Zwraca rozmiar warstwy w punkcie o podanych współrzędnych.
- **addTile(const int x, const int y, const int z, const sf::IntRect& texture_rect, const bool& collision, const short& type)**: Dodaje kafelek do mapy w określonym miejscu i warstwie.
- **removeTile(const int x, const int y, const int z)**: Usuwa kafelek z mapy w określonym miejscu i warstwie.
- **saveToFile(const std::string file_name)**: Zapisuje mapę do pliku.
- **loadFromFile(const std::string file_name)**: Wczytuje mapę z pliku.
- **updateCollision(Entity* entity, const float& dtime)**: Aktualizuje kolizje z obiektami na mapie.
- **update()**: Aktualizuje stan mapy.
- **render(sf::RenderTarget& target, const sf::Vector2i& gridPosition)**: Renderuje mapę na podanym obiekcie renderującym.
- **renderDeferred(sf::RenderTarget& target)**: Renderuje mapę z opóźnieniem.

Przestrzeń nazw **gui**, zawiera klasy obsługujące elementy interfejsu użytkownika (GUI) w grze. Oto opis klas i ich metod:

Przestrzeń nazw gui:

Enum button_states:

- **BUTTON_IDLE**: Stan przycisku - brak interakcji.
- **BUTTON_HOVER**: Stan przycisku - kursor myszy nad przyciskiem.
- **BUTTON_PRESSED**: Stan przycisku - przycisk został naciśnięty.

Klasa Button:

- **Konstruktor**: Inicjalizuje przycisk z podanymi parametrami.

- **Destruktor**: Zwolnienie pamięci.
- **isPressed()**: Sprawdza, czy przycisk został naciśnięty.
- **getText()**: Zwraca tekst przycisku.
- **getId()**: Zwraca identyfikator przycisku.
- **setText(const std::string text)**: Ustawia tekst przycisku.
- **setId(const short unsigned id)**: Ustawia identyfikator przycisku.
- **update(const sf::Vector2i& mousePositionWindow)**: Aktualizuje stan przycisku na podstawie pozycji myszy.
- **render(sf::RenderTarget& target)**: Renderuje przycisk na podanym obiekcie renderującym.

Klasa DropDownList:

- **Konstruktor**: Inicjalizuje listę rozwijaną z podanymi parametrami.
- **Destruktor**: Zwolnienie pamięci.
- **getActiveElementId()**: Zwraca identyfikator aktywnego elementu na liście.
- **getKeytime()**: Sprawdza, czy upłynął czas między klawiszami.
- **updateKeytime(const float& dtime)**: Aktualizuje czas między klawiszami.
- **update(const sf::Vector2i& mousePositionWindow, const float& dtime)**: Aktualizuje stan listy rozwijanej.
- **render(sf::RenderTarget& target)**: Renderuje listę rozwijaną na podanym obiekcie renderującym.

Klasa TextureSelector:

- **Konstruktor**: Inicjalizuje selektor tekstur z podanymi parametrami.
- **Destruktor**: Zwolnienie pamięci.
- **getActive()**: Zwraca stan aktywności selektora tekstur.
- **getTextureRect()**: Zwraca prostokąt tekstury wybrany przez selektor.
- **getKeytime()**: Sprawdza, czy upłynął czas między klawiszami.
- **updateKeytime(const float& dtime)**: Aktualizuje czas między klawiszami.
- **update(const sf::Vector2i& mousePositionWindow, const float& dtime)**: Aktualizuje stan selektora tekstur.
- **render(sf::RenderTarget& target)**: Renderuje selektor tekstur na podanym obiekcie renderującym.

Klasa **Game**, która zarządza główną logiką gry. Oto opis klasy i jej metod:

Klasa Game:

Prywatne składowe:

- **gfxSettings**: Ustawienia graficzne gry.
- **stateData**: Dane stanu gry.
- **window**: Wskaźnik na obiekt okna SFML.
- **event**: Obiekt zdarzenia SFML.
- **dtime_Clock**: Zegar do pomiaru czasu delta.
- **dtime**: Czas delta między klatkami.
- **state**: Stos wskaźników na obiekty stanów gry.
- **supportedKeys**: Mapa obsługiwanych klawiszy.
- **gridSize**: Rozmiar siatki gry.

Prywatne metody:

- **initVariables()**: Inicjalizuje zmienne gry.
- **initGraphicsSettings()**: Inicjalizuje ustawienia graficzne.
- **initWindow()**: Inicjalizuje okno gry.
- **initKeys()**: Inicjalizuje obsługiwane klawisze.
- **initStateData()**: Inicjalizuje dane stanu gry.
- **initState()**: Inicjalizuje stan gry.

Publiczne metody:

- **Konstruktor**: Inicjalizuje obiekt gry.
- **Destruktor**: Zwolnienie pamięci.
- **exitApp()**: Zamyka aplikację.
- **updatedtime()**: Aktualizuje czas delta.
- **updateEvents()**: Aktualizuje zdarzenia.
- **update()**: Aktualizuje stan gry.
- **render()**: Renderuje stan gry.
- **run()**: Uruchamia pętlę gry.

Klasa **GraphicsSettings** definiuje ustawienia graficzne gry, takie jak tytuł okna, rozdzielczość, tryb pełnoekranowy, synchronizacja pionowa, limit klatek na sekundę oraz ustawienia kontekstu SFML. Oto opis klasy i jej metod:

Klasa GraphicsSettings:

Publiczne składowe:

- **title**: Tytuł okna gry.

- **resolution**: Rozdzielczość ekranu.
- **fullscreen**: Flaga określająca, czy gra jest w trybie pełnoekranowym.
- **verticalSync**: Flaga określająca, czy synchronizacja pionowa jest włączona.
- **frameRateLimit**: Limit klatek na sekundę.
- **contextSettings**: Ustawienia kontekstu SFML.
- **videoModes**: Wektor zawierający dostępne tryby wideo.

Publiczne metody:

- **Konstruktor**: Tworzy obiekt **GraphicsSettings**.
- **Destruktor**: Niszczy obiekt **GraphicsSettings**.
- **saveToFile(const std::string path)**: Zapisuje ustawienia graficzne do pliku o podanej ścieżce.
- **loadFromFile(const std::string path)**: Wczytuje ustawienia graficzne z pliku o podanej ścieżce.

Klasa **EditorState** jest klasą dziedziczącą po klasie **State** i odpowiada za logikę i interakcję podczas edycji poziomów w grze. Oto opis klasy i jej metod:

Klasa EditorState:

Prywatne składowe:

- **view**: Widok SFML dla edytora.
- **font**: Czcionka używana w edytorze.
- **cursorText**: Tekst wyświetlany wskazujący na pozycję kursora.
- **pmenu**: Wskaźnik na menu pauzy.
- **buttons**: Mapa przycisków w edytorze.
- **tileMap**: Wskaźnik na mapę kafelków.
- **sidebar**: Pasek boczny w edytorze.
- **selectorRect**: Prostokąt selektora w edytorze.
- **textureSelector**: Selektor tekstur w edytorze.
- **textureRect**: Prostokąt tekstury aktualnie wybrany przez selektor.
- **collision**: Flaga określająca, czy wybrana tekstura ma kolizję.
- **type**: Typ wybranego elementu na mapie.
- **cameraSpeed**: Prędkość kamery w edytorze.
- **layer**: Warstwa aktualnie edytowana na mapie.

Prywatne metody:

- **initVariables()**: Inicjalizuje zmienne edytora.
- **initView()**: Inicjalizuje widok edytora.
- **initBackground()**: Inicjalizuje tło edytora.
- **initFonts()**: Inicjalizuje czcionki edytora.
- **initText()**: Inicjalizuje teksty edytora.
- **initKeybinds()**: Inicjalizuje przypisania klawiszy edytora.
- **initPauseMenu()**: Inicjalizuje menu pauzy edytora.
- **initButtons()**: Inicjalizuje przyciski edytora.
- **initGui()**: Inicjalizuje interfejs graficzny edytora.
- **initTileMap()**: Inicjalizuje mapę kafelków edytora.

Publiczne metody:

- **Konstruktor**: Tworzy obiekt **EditorState**.
- **Destruktor**: Niszczy obiekt **EditorState**.
- **updateKeybinds(const float& dtime)**: Aktualizuje przypisania klawiszy edytora.
- **updateEditorInput(const float& dtime)**: Aktualizuje wejście edytora.
- **upadteButtons()**: Aktualizuje przyciski edytora.
- **updateGui(const float& dtime)**: Aktualizuje interfejs graficzny edytora.
- **updatePauseMenuButtons()**: Aktualizuje przyciski menu pauzy edytora.
- **renderButtons(sf::RenderTarget& target)**: Renderuje przyciski edytora.
- **update(const float& dtime)**: Aktualizuje stan edytora.
- **renderGui(sf::RenderTarget& target)**: Renderuje interfejs graficzny edytora.
- ***render(sf::RenderTarget* target = nullptr)**: Renderuje edytor na podanym obiekcie renderującym.

Klasa **GameState** jest klasą dziedziczącą po klasie **State** i odpowiada za logikę i interakcję podczas rozgrywki w grze. Oto opis klasy i jej metod:

Klasa GameState:

Prywatne składowe:

- **view**: Widok SFML dla stanu gry.
- **renderTexture**: RenderTexture SFML do odroczonego renderowania.
- **renderSprite**: Sprite SFML do wyświetlania renderowanej tekstury.
- **font**: Czcionka używana w grze.
- **pmenu**: Wskaźnik na menu pauzy.

- **player**: Wskaźnik na gracza.
- **tileMap**: Wskaźnik na mapę kafelków.

Prywatne metody:

- **initDeferredRender()**: Inicjalizuje odroczone renderowanie.
- **initView()**: Inicjalizuje widok stanu gry.
- **initKeybinds()**: Inicjalizuje przypisania klawiszy stanu gry.
- **initFonts()**: Inicjalizuje czcionki stanu gry.
- **initTextures()**: Inicjalizuje tekstury stanu gry.
- **initPauseMenu()**: Inicjalizuje menu pauzy stanu gry.
- **initPlayers()**: Inicjalizuje gracza.
- **initTileMap()**: Inicjalizuje mapę kafelków stanu gry.

Publiczne metody:

- **Konstruktor**: Tworzy obiekt **GameState**.
- **Destruktor**: Niszczy obiekt **GameState**.
- **updateView(const float& dtime)**: Aktualizuje widok stanu gry.
- **updateKeybinds(const float& dtime)**: Aktualizuje przypisania klawiszy stanu gry.
- **updatePlayerKeybinds(const float& dtime)**: Aktualizuje przypisania klawiszy dla gracza.
- **updatePauseMenuButtons()**: Aktualizuje przyciski menu pauzy stanu gry.
- **updateTileMap(const float& dtime)**: Aktualizuje mapę kafelków stanu gry.
- **update(const float& dtime)**: Aktualizuje stan gry.
- ***render(sf::RenderTarget* target = nullptr)**: Renderuje stan gry na podanym obiekcie renderującym.

Klasa **MainMenuState** jest klasą dziedziczącą po klasie **State** i odpowiada za logikę i interakcję w menu głównym gry. Oto opis klasy i jej metod:

Klasa MainMenuState:

Prywatne składowe:

- **backgroundTexture**: Tekstura tła menu głównego.
- **background**: Prostokąt SFML reprezentujący tło menu głównego.
- **font**: Czcionka używana w menu głównym.
- **music**: Muzyka odtwarzana w tle menu głównego.
- **buttons**: Mapa przycisków w menu głównym.

Prywatne metody:

- **initVariables()**: Inicjalizuje zmienne menu głównego.
- **initBackground()**: Inicjalizuje tło menu głównego.
- **initFonts()**: Inicjalizuje czcionki menu głównego.
- **initKeybinds()**: Inicjalizuje przypisania klawiszy menu głównego.
- **initButtons()**: Inicjalizuje przyciski menu głównego.
- **initMusic()**: Inicjalizuje muzykę menu głównego.

Publiczne metody:

- **Konstruktor**: Tworzy obiekt **MainMenuState**.
- **Destruktor**: Niszczy obiekt **MainMenuState**.
- **updateKeybinds(const float& dtime)**: Aktualizuje przypisania klawiszy menu głównego.
- **updateButtons()**: Aktualizuje przyciski menu głównego.
- **playMusic()**: Odtwarza muzykę w tle menu głównego.
- **stopMusic()**: Zatrzymuje odtwarzanie muzyki w tle menu głównego.
- **renderButtons(sf::RenderTarget& target)**: Renderuje przyciski menu głównego.
- **update(const float& dtime)**: Aktualizuje stan menu głównego.
- ***render(sf::RenderTarget* target = nullptr)**: Renderuje menu główne na podanym obiekcie renderującym.

Klasa **SettingsState** jest klasą dziedziczącą po klasie **State** i odpowiada za wyświetlanie ustawień gry oraz interakcję z nimi. Oto opis klasy i jej metod:

Klasa SettingsState:

Prywatne składowe:

- **backgroundTexture**: Tekstura tła stanu ustawień.
- **background**: Prostokąt SFML reprezentujący tło stanu ustawień.
- **font**: Czcionka używana w ustawieniach.
- **buttons**: Mapa przycisków w ustawieniach.
- **dropdownLists**: Mapa list rozwijalnych w ustawieniach.
- **optionsText**: Tekst wyświetlany w ustawieniach.
- **modes**: Wektor zawierający dostępne tryby wyświetlania.

Prywatne metody:

- **initVariables()**: Inicjalizuje zmienne ustawień.
- **initBackground()**: Inicjalizuje tło ustawień.

- **initFonts()**: Inicjalizuje czcionki ustawień.
- **initKeybinds()**: Inicjalizuje przypisania klawiszy ustawień.
- **initGui()**: Inicjalizuje interfejs graficzny ustawień.
- **initText()**: Inicjalizuje teksty wyświetlane w ustawieniach.

Publiczne metody:

- **Konstruktor**: Tworzy obiekt **SettingsState**.
- **Destruktor**: Niszczy obiekt **SettingsState**.
- **updateKeybinds(const float& dtime)**: Aktualizuje przypisania klawiszy w ustawieniach.
- **updateGui(const float& dtime)**: Aktualizuje interfejs graficzny w ustawieniach.
- **renderGui(sf::RenderTarget& target)**: Renderuje interfejs graficzny w ustawieniach.
- **update(const float& dtime)**: Aktualizuje stan ustawień.
- ***render(sf::RenderTarget* target = nullptr)**: Renderuje ustawienia na podanym obiekcie renderującym.

Klasa **State** jest abstrakcyjną klasą bazową dla różnych stanów gry. Oto opis klasy i jej metod:

Klasa State:

Prywatne składowe:

- **StateData**: Obiekt przechowujący dane o stanie gry.
- **states**: Wskaźnik na stos stanów gry.
- **window**: Wskaźnik na obiekt okna SFML.
- **supportedKeys**: Wskaźnik na mapę obsługiwanych klawiszy.
- **keybinds**: Mapa przypisanych klawiszy.
- **exit**: Flaga określająca, czy stan powinien zostać zakończony.
- **paused**: Flaga określająca, czy stan jest wstrzymany.
- **keytime**: Aktualny czas między wciśnięciami klawiszy.
- **keytimeMax**: Maksymalny czas między wciśnięciami klawiszy.
- **gridSize**: Rozmiar siatki używanej w grze.
- **mousePositionScreen**: Pozycja myszy na ekranie.
- **mousePositionWindow**: Pozycja myszy w oknie gry.
- **mousePositionView**: Pozycja myszy w widoku gry.
- **mousePositionGrid**: Pozycja myszy na siatce gry.
- **texture**: Mapa tekstur używanych w stanie.

Metody:

- **Konstruktor**: Tworzy obiekt **State** i inicjalizuje jego dane.
- **Destruktor**: Niszczy obiekt **State**.
- **getExit()**: Zwraca flagę **exit**, określającą, czy stan powinien zostać zakończony.
- **getKeytime()**: Zwraca aktualny czas między wciśnięciami klawiszy.
- **endState()**: Ustawia flagę **exit** na **true**, kończąc stan.
- **pauseState()**: Ustawia flagę **paused** na **true**, wstrzymując stan.
- **unpauseState()**: Ustawia flagę **paused** na **false**, wznowienie stanu.
- ***updateMousePosition(sf::View* view = nullptr)**: Aktualizuje pozycję myszy.
- **updateKeytime(const float& dtime)**: Aktualizuje czas między wciśnięciami klawiszy.
- **updateKeybinds(const float& dtime)**: Abstrakcyjna metoda do aktualizacji przypisanych klawiszy.
- **update(const float& dtime)**: Abstrakcyjna metoda do aktualizacji stanu gry.
- ***render(sf::RenderTarget* target = nullptr)**: Abstrakcyjna metoda do renderowania stanu na określonym obiekcie renderującym.

Klasa **Tile** reprezentuje pojedynczy kafelek w grze, który może być używany do tworzenia mapy poziomów. Oto opis klasy i jej metod:

Klasa Tile:

Prywatne składowe:

Brak prywatnych składowych.

Chronione składowe:

- **shape**: Kształt prostokątny SFML reprezentujący wygląd kafelka.
- **collision**: Flaga określająca, czy kafelek jest kolizyjny.
- **type**: Typ kafelka.

Metody:

- **Konstruktor**: Tworzy obiekt **Tile**.
- **Destruktor**: Niszczy obiekt **Tile**.
- **getType()**: Zwraca typ kafelka.
- **getCollision()**: Zwraca flagę określającą, czy kafelek jest kolizyjny.
- **getPosition()**: Zwraca pozycję kafelka.
- **getGlobalBounds()**: Zwraca globalne granice kafelka.

- **intersects(const sf::FloatRect bounds)**: Sprawdza, czy kafelek przecina się z określonym prostokątem.
- **getAsString()**: Zwraca kafelek w formie ciągu znaków.
- **update()**: Aktualizuje stan kafelka.
- **render(sf::RenderTarget& target)**: Renderuje kafelek na podanym obiekcie renderującym.

Typ wyliczeniowy TileTypes:

- **DEFAULT**: Domyślny typ kafelka.
- **DAMAGING**: Kafelek, który powoduje obrażenia.
- **DOODAD**: Kafelek dekoracyjny, nie mający żadnej funkcji kolizyjnej.