Super Cow DLC

# Классы

## Game

### Game();

Инициализация игрового окна, управления и состояния главного меню. Хранит состояния, которые обновляет и рендерит.

### void initWindow();

Устанавливает параметры игрового окна, заданные в файле "Config/window.ini".

*Формат “window.ini”:*

Super Cow DLC //название окна

1280 720 //разрешение

0 //fullscreen

120 //framerate\_limit

0 //vertical\_sync\_enables

0 //antialiasing\_level

### void initStates();

Создает состояние MainMenuState и добавляет его в стек состояний.

### void initKeys();

Устанавливает поддерживаемые элементы управления, заданные в файле "Config/supported\_keys.ini"

*Формат “supported\_keys.ini”:*

A 0 //Клавиша, id

D 3

W 22

S 18

…

### void updateDT();

Обновление delta time.

### void updateSFMLEvents();

Отслеживание события закрытия окна.

### void update();

Обновление верхнего состояния в стеке, при отсутствии состояний закрывание окна.

### void render();

Очищение окна, рендеринг верхнего состояния, отображение окна.

### void run();

Пока активно окно, запуск функций обновления и рендеринга.

## State

### State(sf::RenderWindow\* window, std::map<std::string, int>\* supportedKeys, std::stack<State\*>\* states);

Родительский класс для других состояний.

### const bool& getQuit() const;

Возвращает булевое значение, если состояние завершено возвращает *true*, в ином случае *false.*

### void endState();

Устанавливает значение флага выхода из состояния *true*.

### virtual void updateMousePosition();

Обновление позиции мыши относительно окна и экрана.

## MainMenuState

### MainMenuState(sf::RenderWindow\* window, std::map<std::string, int>\* supportedKeys, std::stack<State\*>\* states);

Инициализация фона, шрифтов, элементов управления, кнопок. Содержит всю информацию о состоянии главного меню.

### void initVariables();

Инициализация переменных (отсутствуют))).

### void initBackground();

Инициализация фона текстурой из файла "Resources/Images/Backgrounds/bg1.png".

### void initFonts();

Инициализация шрифтов, шрифты указан по директории "Fonts".

### void initKeybinds();

Инициализация элементов управления, заданных в файле "Config/mainmenustate\_keybinds.ini".

*Формат "Config/mainmenustate\_keybinds.ini":*

CLOSE Escape //Тег элемента, клавиша

…

### void initButtons();

Инициализация кнопок. Подробнее в классе Button.

### void updateButtons();

Обновление состояния кнопок.  
Кнопка NEW GAME создает состояние GameState.

Кнопка QUIT завершает состояние MainMenuState.

### void update(const float& dt);

Выполнение функций обновления. const float& dt – delta time.

### void renderButtons(sf::RenderTarget\* target = NULL);

Отображение кнопок.

### void render(sf::RenderTarget\* target = NULL);

Выполнение функций рендеринга.

## GameState

### GameState(sf::RenderWindow\* window, std::map<std::string, int>\* supportedKeys, std::stack<State\*>\* states);

Инициализация элементов управления, текстур, игрового персонажа, музыки и платформ. Основное состояние игры.

### void initKeybinds();

Инициализация элементов управления, заданных в файле "Config/gamestate\_keybinds.ini".

*Формат "gamestate\_keybinds.ini":*

CLOSE Escape //Тег элемента, клавиша

MOVE\_UP W

MOVE\_DOWN S

MOVE\_RIGHT D

…

### void initTextures();

Инициализация текстур, размещенных по директории “Resources/Images/Sprites”.

### void initPlayers();

Инициализация игрока, создается класс Player.

### void initMusic();

Инициализация музыки, расположенной по директории “Resources/Sounds”.

### void initPlatforms();

Инициализация платформ. Платформы описаны в файлах с расширением *level,* размещенных по директории "Resources/Levels"

*Формат файла level:*

300 400 200 50 GRASS //размещение по х, размещение по у, ширина, высота, тектура

600 500 200 70 STONE

200 300 400 100 WATER

…

### void hp\_upd();

Обновление здоровья

### void updateInput(const float& dt);

Обработка команд пользователя.

### void update(const float& dt);

Выполнение всех обновлений.

### void render(sf::RenderTarget\* target = NULL);

Выполенение всего рендеринга.

### void renderPlatforms(sf::RenderTarget\* target = NULL);

Отображение платформ.

## Entity

### void setTexture(sf::Texture& texture);

Устанавливает текстуру для объекта.

### **void createMovementComponent(const float maxVelocity, const float acceleration, const float deceleration, HeatBox\* heatbox);**

Создает класс MovementCompontnt, который позволит создать управление персонажем.

### **void createAnimationComponent(sf::Texture& texture);**

Создает класс AnimationCompontnt, который позволит создать анимацию персонажей с помощью texture.

### 

### **virtual void setPosition(const float x, const float y);**

Устанавливает спрайт в заданную позицию по координатам x и y.

### **virtual void move(const float x, const float y, const float& dt);**

Вызывает функцию движение у MovementComponent.

### **virtual void update(const float& dt);**

Обновление

### **virtual void render(sf::RenderTarget\* target);**

Рендеринг

## Player

### **Player(float x, float y, sf::Texture& texture);**

### Инициализация класса игрока.

### **virtual void update(const float& dt);**

Обновление.

### 

### **void initVariables();**

Инициализация переменных.

### **void initComponents();**

Инициализация AnimationComponent и MovementComponent.

## Heatbox

### void setMaxHP(int hp);

Устанавливает максимальное значение здоровья.

### **void getDamage(int damage);**

Уменьшает количество здоровья на значение damage.

### 

### **void setHitboxSize(const sf::Vector2f& size);**

Устанавливает размер хитбокса.

### **void setHitboxPosition(**const **sf::Vector2f& position);**

Устанавливает координаты хитбокса

### **void move(const float dir\_x, const float dir\_y, const float& dt);**

Передвигает хитбокс.

### **void setVelocity(float newvelocity);**

Обновляет значение velocity по x.

### **void setVelocityy(float newvelocity);**

Установка значения velocity по y.

### **void render(sf::RenderTarget& target);**

Выполнение функции рендеринга

## MovementComponent

## 

### virtual ~MovementComponent();

### 

### **const sf::Vector2f& getVelocity() const;**

Возвращает текущее значение скорости.

### 

### **void move(const float x, const float y, const float& dt);**

Описывается поведение для движения, используются ускорения и замедления для более плавного движения.

### 

### **void falling();**

Осуществляет перемещение персонажа по оси Y к большим значениям.

### 

### **void update(const float& dt);**

Обновление.

## AnimationComponent

### Animation(sf::Sprite& sprite, sf::Texture& texture\_sheet, float animation\_timer, int start\_frame\_x, int start\_frame\_y, int frames\_x, int frames\_y, int width, int height)

Инициализация класса анимации.

### **void play(const float& dt)**

Проигрывание анимации.

### 

### **void addAnimation(const std::string key, float animation\_timer, int start\_frame\_x, int start\_frame\_y, int frames\_x, int frames\_y, int width, int height);**

Создание новой анимации, пример нужной текстуры приведен ниже:



### 

### **void startAnimation(const std::string animation);**

Старт анимации.

### 

### **void pauseAnimation(const std::string animation);**

Приостановка анимации.

### **void resetAnimation(const std::string animation);**

Возобновление анимации.

### 

### **void play(const std::string key, const float& dt);**

Проигрывание анимации.

## Platform

### Platform(float x, float y, float width, float height, sf::Color idleColor);

Инициализация класса платформы.

### void checkCollision(std::queue<sf::FloatRect>& queue, HeatBox\* hitbox);

Проверка на соприкосновение платформы и персонажа. При наличии коллизии координаты хитбокса добавляются в очередь.

### 

### void render(sf::RenderTarget\* target);

Выполнение функции рендеринга.