# The Village Pillager

2

Generated by Doxygen 1.8.14

# Contents

# Chapter 1

# Hierarchical Index

## 1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# Chapter 2

# Class Index

## 2.1 Class List

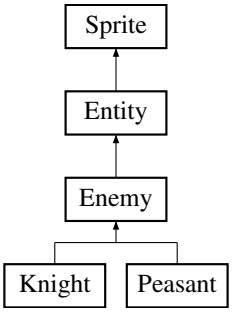
Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 3

# Class Documentation

## 3.1 Enemy Class Reference

Inheritance diagram for Enemy:




### Public Member Functions

- **Enemy** (int hp, int immunity, unsigned points, sf::Vector2f speed, sf::Vector2f position, sf::Vector2f scale, sf::Texture &texture)
- void hit (int attack_type)

  *Removes health from the object if it is not immune.*
- void update (Entity *player, sf::RenderWindow &window, sf::Time tick)

  *Moves the enemies towards the player character.*
- unsigned **get_points** ()

### Protected Attributes

- int **immunity**
- unsigned **points**

#### 3.1.1 Member Function Documentation

#### 3.1.1.1 hit()

```
void Enemy::hit (
            int attack_type )
```

Removes health from the object if it is not immune.

#### 3.1.1.2 update()

```
void Enemy::update (
            Entity * player,
            sf::RenderWindow & window,
            sf::Time tick )
```

Moves the enemies towards the player character based on the time elapsed since the last update.

The documentation for this class was generated from the following files:

- entities.h
- entities.cc

## 3.2 Engine Class Reference

### Public Member Functions

- Engine ()

  *Constructor for Engine.*
- void run ()

  *The function that initiates the game and switches between states.*

### Public Attributes

- sf::RenderWindow **window**

### Friends

- class **WinState**
- class **GameOver**
- class **MenuState**
- class **PlayState**

### 3.2.1 Constructor & Destructor Documentation

#### 3.2.1.1 Engine()

```
Engine::Engine ( )
```

Engine's constructor is responsible for creating the window with the correct resolution. It's also responsible for enabling and disabling window's settings. Finally it creates a map to contain the textures for all the background sprites.

### 3.2.2 Member Function Documentation

#### 3.2.2.1 run()

```
void Engine::run ( )
```

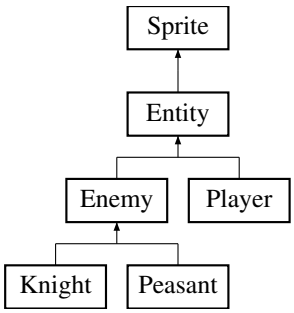
Adds all relevant background textures to its map. Uses stateNum to determine which state should be active. Calls the appropriate switch-function to create and run the desired state.

The documentation for this class was generated from the following files:

- state.h
- state.cc

## 3.3 Entity Class Reference

Inheritance diagram for Entity:




### Public Member Functions

- **Entity** (int hp, sf::Vector2f speed, sf::Vector2f position, sf::Vector2f scale, sf::Texture &texture)
- int **get_hp** ()

### Protected Attributes

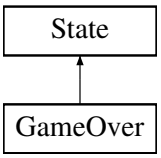
- sf::Vector2f const **speed**
- int **hp**
- sf::Vector2f **scale**
- sf::Clock **timer**
- sf::Clock **immunity_timer**
- bool **marked_for_destruction**

The documentation for this class was generated from the following files:

- entities.h
- entities.cc

## 3.4 GameOver Class Reference

Inheritance diagram for GameOver:




### Public Member Functions

- **GameOver** (sf::Texture &background, sf::RenderWindow &window)
- void update (sf::Event &event_queue, sf::RenderWindow &window, int &stateNum)

  *A function to update the game while in GameOverState.*

### Additional Inherited Members

### 3.4.1 Member Function Documentation

#### 3.4.1.1 update()

```
void GameOver::update (
            sf::Event & event_queue,
            sf::RenderWindow & window,
            int & stateNum )
```

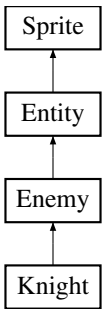
Draws the background image. Recieves keyboard input to switch to MenuState. Triggers switching of states by changing the value of stateNum.

The documentation for this class was generated from the following files:

- state.h
- state.cc

## 3.5 Knight Class Reference

Inheritance diagram for Knight:




### Public Member Functions

- **Knight** (int hp, sf::Vector2f speed, sf::Vector2f position, sf::Vector2f scale, sf::Texture &texture, sf::Texture &sword_t)
- void update (Entity *player, sf::RenderWindow &window, sf::Time tick)

  *Moves the Knight towards the Player object.*

### Additional Inherited Members

### 3.5.1 Member Function Documentation

#### 3.5.1.1 update()

```
void Knight::update (
            Entity * player,
            sf::RenderWindow & window,
            sf::Time tick )
```

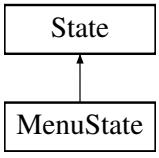
Moves the Knight towards the Player object based on the time elapsed since the last update. Triggers a heavy attack if the Player object is within reach. Draws itself and it's sword.

The documentation for this class was generated from the following files:

- entities.h
- entities.cc

## 3.6 MenuState Class Reference

Inheritance diagram for MenuState:



### Public Member Functions

- **MenuState** (sf::Texture &background, sf::RenderWindow &window)
- void update (sf::Event &event, sf::RenderWindow &window, int &stateNum)

  *A function to update the game while in MenuState.*

### Additional Inherited Members

### 3.6.1 Member Function Documentation

#### 3.6.1.1 update()

```
void MenuState::update (
            sf::Event & event_queue,
            sf::RenderWindow & window,
            int & stateNum )
```

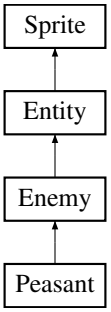
Draws the background image. Recieves keyboard input to either switch to the PlayState or close the game. Trigger switching of states by changing the value of stateNum.

The documentation for this class was generated from the following files:

- state.h
- state.cc

## 3.7 Peasant Class Reference

Inheritance diagram for Peasant:




### Public Member Functions

- **Peasant** (sf::Vector2f speed, sf::Vector2f position, sf::Vector2f scale, sf::Texture &texture)

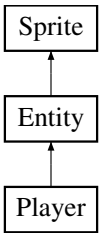
### Additional Inherited Members

The documentation for this class was generated from the following files:

- entities.h
- entities.cc

## 3.8 Player Class Reference

Inheritance diagram for Player:




### Public Member Functions

- **Player** (int hp, sf::Vector2f speed, sf::Vector2f position, sf::Vector2f scale, sf::Texture &player_t, sf::Texture &sword_t, sf::Texture &health)
- void collision (std::vector< Enemy * > Enemy)

  *Checks if the player character is colliding with an enemy and updates the player accordingly.*
- void hit (int attack_mode)

  *Subtracts hp from the Player object if it's not immune.*
- void player_update (sf::Time time, sf::Event &event_queue, sf::RenderWindow &window, std::vector< Enemy * > &enemies, int &stateNum)

  *Updates the player character.*
- void draw_player (sf::RenderWindow &window)

  *Draws the player character.*
- void process_input (sf::Event &event_queue, int &stateNum, sf::RenderWindow &window, sf::Time tick)

  *Handels the inputs from the keyboard and other events.*
- void jump ()

  *Moves the Player object in the y-axis based on a timer.*

### Additional Inherited Members

### 3.8.1 Member Function Documentation

#### 3.8.1.1 collision()

```
void Player::collision (
            std::vector< Enemy *> enemies )
```

Checks if any enemy intersects with the player character if it isn't currently immune. If intersection occurs; knockback is applied on both the player character and the enemy. The player character's health is reduced and immunity is gained.

#### 3.8.1.2 draw_player()

```
void Player::draw_player (
            sf::RenderWindow & window )
```

Draws the player character. If the player character is currently immune to damage the player character blinks with a frequency of 5 blinks per second.

#### 3.8.1.3 hit()

```
void Player::hit (
            int attack_mode )
```

Subtracts hp from the Player object if it's not immune.

#### 3.8.1.4 jump()

```
void Player::jump ( )
```

The Player object is moved in the y-axis based on a timer. The y-position is determined by a second-degree equation y = (x - 0)∗(x - z)∗a where x is the elapsed time since the jump started, z is the time when the jump is finished and a is the amplitude of the jump.

#### 3.8.1.5 player_update()

```
void Player::player_update (
            sf::Time time,
            sf::Event & event_queue,
            sf::RenderWindow & window,
            std::vector< Enemy *> & enemies,
            int & stateNum )
```

Calls subfunctions related to updating the player characters position, scale, health. Ends the playstate if the player's health reaches 0.

#### 3.8.1.6 process_input()

```
void Player::process_input (
            sf::Event & event_queue,
            int & stateNum,
            sf::RenderWindow & window,
            sf::Time tick )
```

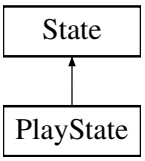
Changes the current state based on player input. Updates the player characters position and scale based on input and time passed since last function call.

The documentation for this class was generated from the following files:

- entities.h
- entities.cc

## 3.9 PlayState Class Reference

Inheritance diagram for PlayState:




### Public Member Functions

- **PlayState** (sf::Texture &background, sf::RenderWindow &window)
- void addEnemy (Enemy *entity)

  *Adds an Enemy object to a vector.*
- void setPlayer (Player *player)

  *Adds a pointer to the Player object in PlayState.*
- void update (sf::Time time, sf::Event &event, sf::RenderWindow &window, int &stateNum, sf::Text &score, std::vector< std::vector< Enemy *>> waves)

  *Updates all objects belonging to PlayState and then displays them. Changes the state if win or lose conditions are met.*

### Friends

- class **Engine**

### Additional Inherited Members

### 3.9.1 Member Function Documentation

#### 3.9.1.1 addEnemy()

```
void PlayState::addEnemy (
            Enemy * enemy )
```

Inserts the Enemy object at the beginning of the vector.

#### 3.9.1.2 setPlayer()

```
void PlayState::setPlayer (
            Player * entity )
```

Adds a pointer to the Player object in PlayState.

#### 3.9.1.3 update()

```
void PlayState::update (
            sf::Time time,
            sf::Event & event,
            sf::RenderWindow & window,
            int & stateNum,
            sf::Text & score,
            std::vector< std::vector< Enemy *>> waves )
```

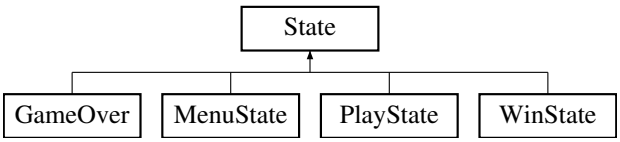
Updates all objects belonging to PlayState and then displays them. It adds Enemy objects to the playfield at specific intervals. Erases Enemy objects when their health is reduced to 0 and adds the objects points to the variable total_points. Changes the state if win or lose conditions are met.

The documentation for this class was generated from the following files:

- state.h
- state.cc

## 3.10 State Class Reference

Inheritance diagram for State:




### Public Member Functions

- State (sf::Texture &background, sf::RenderWindow &window)

  *State's constructor.*
- void window_resize (sf::RenderWindow &window)

  *Controls the scaling of the window.*

### Protected Attributes

- sf::Sprite **bg**

### Friends

- class **Engine**

### 3.10.1 Constructor & Destructor Documentation

#### 3.10.1.1 State()

```
State::State (
            sf::Texture & background,
            sf::RenderWindow & window )
```

Sets texture for the background sprite and causes the dimensionens of the window to keep a ratio of 16:9 regardless of scaling.

### 3.10.2 Member Function Documentation

#### 3.10.2.1 window_resize()

```
void State::window_resize (
            sf::RenderWindow & window )
```

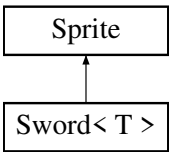
Keeps the window at a constant ratio of 16:9 regardless of scaling.

The documentation for this class was generated from the following files:

- state.h
- state.cc

## 3.11 Sword< T > Class Template Reference

Inheritance diagram for Sword< T >:



### Public Member Functions

- **Sword** (sf::Vector2f scale, sf::Texture &texture, float speed=1)
- void update (std::vector< T > enemies, Entity *holder)

  *Updates the swords position and scale relative to its holder and calls attack functions if the sword is in an attack mode.*
- void strike (std::vector< T > enemies)

  *Detects collision during attack.*
- void light_attack (std::vector< T > enemies, float orientation)

  *The sword swings back and forth quickly.*
- void heavy_attack (std::vector< T > enemies, float orientation)

  *The sword moves in a predetermined pattern relative to the holder.*

### Friends

- class **Player**
- class **Knight**

## 3.11.1 Member Function Documentation

#### 3.11.1.1 heavy_attack()

```
template<typename T>
void Sword< T >::heavy_attack (
            std::vector< T > enemies,
            float orientation )
```

The sword follows a specific animation pattern relative to the holders position and scale. During the attack-frames, when the sword is thrust forward, the function calls the strike function to resolve potential hits. To change the total time of the animation; tweak the swords speed variable.

#### 3.11.1.2 light_attack()

```
template<typename T>
void Sword< T >::light_attack (
            std::vector< T > enemies,
            float orientation )
```

The sword swings back and forth between 0 and 45 degrees and checks for collision with enemies relative to the holder. If collision with an enemy-object occurs during the forward swing a function to deal damage to the object is called.

#### 3.11.1.3 strike()

```
template<typename T>
void Sword< T >::strike (
            std::vector< T > enemies )
```

Goes through a list of enemies and checks collision. If collision occurs the enemy is knocked back and the hit function is called on the object.

#### 3.11.1.4 update()

```
template<typename T>
void Sword< T >::update (
            std::vector< T > enemies,
            Entity * holder )
```

Updates the swords position and scale relative to its holder and calls attack functions if the sword is in an attack mode.

The documentation for this class was generated from the following files:

- entities.h
- entities.cc

## 3.12 Wave Class Reference

### Public Member Functions

- **Wave** (sf::Vector2f p_speed, sf::Vector2f k_speed, float playheight, sf::Vector2f scale, sf::Texture &p_texture, sf::Texture &k_texture)

### Public Attributes

- std::vector< std::vector< Enemy * > > **waves**

### Friends

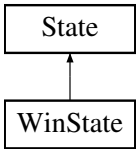
- class **PlayState**

The documentation for this class was generated from the following files:

- state.h
- state.cc

## 3.13 WinState Class Reference

Inheritance diagram for WinState:



### Public Member Functions

- **WinState** (sf::Texture &background, sf::RenderWindow &window)
- void update (sf::Event &event_queue, sf::RenderWindow &window, int &stateNum)

  *A function to update the game while in WinState.*

### Additional Inherited Members

## 3.13.1 Member Function Documentation

#### 3.13.1.1 update()

```
void WinState::update (
            sf::Event & event_queue,
            sf::RenderWindow & window,
            int & stateNum )
```

Draws the background image. Recieves keyboard input to switch to MenuState. Triggers switching of states by changing the value of stateNum.

The documentation for this class was generated from the following files:

- state.h
- state.cc

# Index