Tower Defense

Generated by Doxygen 1.9.3

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

File Index

3.1 File List

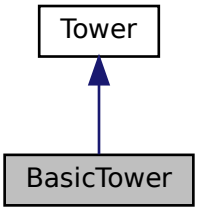
Here is a list of all documented files with brief descriptions:

Chapter 4

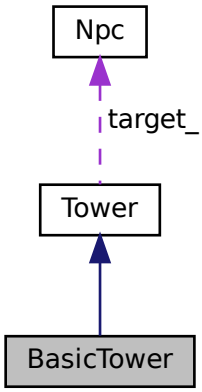
Class Documentation

4.1 BasicTower Class Reference

Inheritance diagram for BasicTower:



Collaboration diagram for BasicTower:



Public Member Functions

- **BasicTower** (std::string root_filepath, sf::Vector2f pos, float attack_speed, int damage, float radius, int soundEffectVolumeLevel)
- void upgrade ()
- virtual void render (sf::RenderTarget *target)

  Render tower on target (window)
- virtual void update (const float &dt, std::list< Npc * > enemies, const sf::Vector2f mousePos)

  Update tower with new information.
- void **initSoundEffect** ()

  Initializes shooting sound effect.
- void **setUpSprites** ()

  Loads required textures and sets up sprites.
- void rotateGun (const float &dt, std::list< Npc * > enemies)

  Rotates the gun part of the tower towards first enemy and targets that enemy for the next attack.
- virtual void attack (const float &dt)

  Attacks a target if the tower has one. Dealing damage and possible slowing the enemy.
- float **getDamage** ()
- float **getAttackSpeed** ()
- int **getTowerLevel** ()
- sf::Vector2f **getPosition** ()

Protected Attributes

- sf::Sprite **platform_**

  Sprite for the platform part of the tower.
- sf::Sprite **gun_**

  Sprite for the gun part of the tower.
- sf::Texture **sftexture_platform_**

  Texture for the platform part of the tower.
- sf::Texture **sftexture_gun_**

  Texture for the gun part of the tower.
- sf::Texture **sftexture_gunfire_**
- std::string **texture_gunfire_**
- sf::Sprite **gunfire_**
- Npc * **target_**

  Target for the next attack.
- sf::Clock **clock_**

  Measures time.
- sf::Vector2f **pos_**

  Position of this tower.
- std::string **root_filepath_**

  Root filepath.
- std::string **texture_platform_**

  Name of the platform texture.
- std::string **texture_gun_**

  Name of the gun texture.
- sf::CircleShape **radius_shape_**

  CircleShape to show the radius of the tower.
- float **attack_speed_**

  Attack speed (seconds)

- float **damage_**

  Damage.
- int **tower_level_** = 1
- float **radius_**

  Attack radius.
- float **slowing_parameter_**

  Slowing parameter.
- short unsigned **towerState_**

  Checks if the mouse is hovering the tower.
- sf::Transform **transform_**
- double **angle_**
- sf::Clock **gunfire_clock_**
- sf::Color **gunfire_color_**
- sf::SoundBuffer **buffer**

  Shoot sound effect.
- sf::Sound **sound**
- int **soundEffectVolumeLevel_**

4.1.1 Member Function Documentation

4.1.1.1 attack()

```
void Tower::attack (
            const float & dt )  [virtual], [inherited]
```

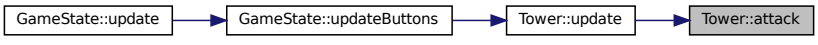
Attacks a target if the tower has one. Dealing damage and possible slowing the enemy.

Parameters

dt	Delta time

Reimplemented in BombTower, and SlowingTower.

Here is the caller graph for this function:



4.1.1.2 render()

```
void Tower::render (
            sf::RenderTarget * target )  [virtual], [inherited]
```

Render tower on target (window)

Parameters

target	Rendering target, i.e. game window

Reimplemented in BombTower, and SlowingTower.

4.1.1.3 rotateGun()

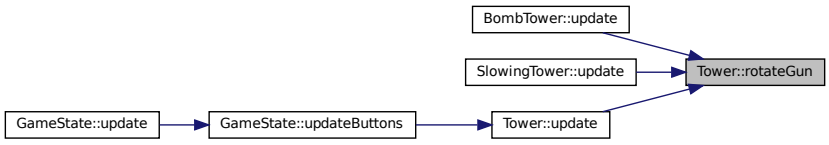
```
void Tower::rotateGun (
            const float & dt,
            std::list< Npc * > enemies )  [inherited]
```

Rotates the gun part of the tower towards first enemy and targets that enemy for the next attack.

Parameters

dt	Delta time
enemies	List of every enemy on the board currently

Here is the caller graph for this function:



4.1.1.4 update()

```
void Tower::update (
            const float & dt,
            std::list< Npc * > enemies,
            const sf::Vector2f mousePos )  [virtual], [inherited]
```

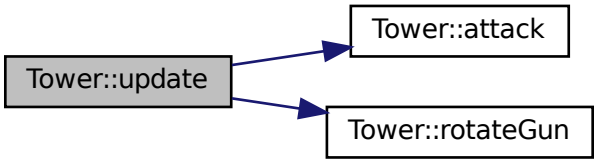
Update tower with new information.

Parameters

dt	Delta time
enemies	List of every enemy on the board currently
mousePos	Mouse position on screen

Reimplemented in BombTower, and SlowingTower.

Here is the call graph for this function:



Here is the caller graph for this function:

4.1.1.5 upgrade()

```
void BasicTower::upgrade ( )  [virtual]
```

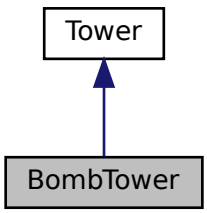
Reimplemented from Tower.

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

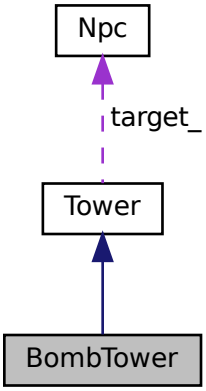
- src/basic_tower.hpp
- src/basic_tower.cpp

4.2 BombTower Class Reference

Inheritance diagram for BombTower:



Collaboration diagram for BombTower:



Public Member Functions

- **BombTower** (std::string root_filepath, sf::Vector2f pos, float attack_speed, int damage, float radius, int soundEffectVolumeLevel)
- void attack (const float &dt)

  Attacks a target if the tower has one. Dealing damage and possible slowing the enemy.
- void update (const float &dt, std::list< Npc * > enemies, const sf::Vector2f mousePos)

  Update tower with new information.
- void render (sf::RenderTarget *target)

  Render tower on target (window)
- void upgrade ()
- void **initSoundEffect** ()

  Initializes shooting sound effect.
- void **setUpSprites** ()

  Loads required textures and sets up sprites.
- void rotateGun (const float &dt, std::list< Npc * > enemies)

  Rotates the gun part of the tower towards first enemy and targets that enemy for the next attack.
- float **getDamage** ()
- float **getAttackSpeed** ()
- int **getTowerLevel** ()
- sf::Vector2f **getPosition** ()

Protected Attributes

- sf::Sprite **platform_**

  Sprite for the platform part of the tower.
- sf::Sprite **gun_**

  Sprite for the gun part of the tower.
- sf::Texture **sftexture_platform_**

  Texture for the platform part of the tower.

- sf::Texture **sftexture_gun_**
  Texture for the gun part of the tower.
- sf::Texture **sftexture_gunfire_**
- std::string **texture_gunfire_**
- sf::Sprite **gunfire_**
- Npc * **target_**
  Target for the next attack.
- sf::Clock **clock_**
  Measures time.
- sf::Vector2f **pos_**
  Position of this tower.
- std::string **root_filepath_**
  Root filepath.
- std::string **texture_platform_**
  Name of the platform texture.
- std::string **texture_gun_**
  Name of the gun texture.
- sf::CircleShape **radius_shape_**
  CircleShape to show the radius of the tower.
- float **attack_speed_**
  Attack speed (seconds)
- float **damage_**
  Damage.
- int **tower_level_** = 1
- float **radius_**
  Attack radius.
- float **slowing_parameter_**
  Slowing parameter.
- short unsigned **towerState_**
  Checks if the mouse is hovering the tower.
- sf::Transform **transform_**
- double **angle_**
- sf::Clock **gunfire_clock_**
- sf::Color **gunfire_color_**
- sf::SoundBuffer **buffer**
  Shoot sound effect.
- sf::Sound **sound**

Private Attributes

- std::list< Missile * > **missiles_**
- int **soundEffectVolumeLevel_**

4.2.1 Member Function Documentation

4.2.1.1 attack()

```
void BombTower::attack (
            const float & dt )  [virtual]
```

Attacks a target if the tower has one. Dealing damage and possible slowing the enemy.

Parameters

dt	Delta time

Reimplemented from Tower.

Here is the caller graph for this function:

4.2.1.2 render()

```
void BombTower::render (
            sf::RenderTarget * target )  [virtual]
```

Render tower on target (window)

Parameters

target	Rendering target, i.e. game window

Reimplemented from Tower.

4.2.1.3 rotateGun()

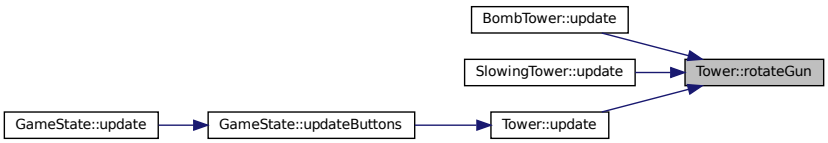
```
void Tower::rotateGun (
            const float & dt,
            std::list< Npc * > enemies )  [inherited]
```

Rotates the gun part of the tower towards first enemy and targets that enemy for the next attack.

Parameters

dt	Delta time
enemies	List of every enemy on the board currently

Here is the caller graph for this function:



4.2.1.4 update()

```
void BombTower::update (
            const float & dt,
            std::list< Npc * > enemies,
            const sf::Vector2f mousePos )  [virtual]
```

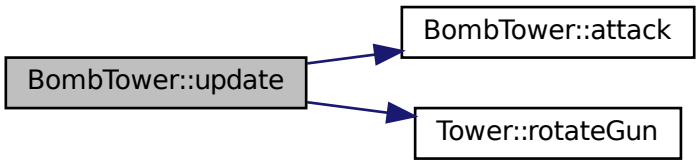
Update tower with new information.

Parameters

dt	Delta time
enemies	List of every enemy on the board currently
mousePos	Mouse position on screen

Reimplemented from Tower.

Here is the call graph for this function:



4.2.1.5 upgrade()

```
void BombTower::upgrade ( )  [virtual]
```

Reimplemented from Tower.

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

- src/bomb_tower.hpp
- src/bomb_tower.cpp

4.3 Button Class Reference

Button with visual parameters to initiate some action.

`#include <button.hpp>`

Public Member Functions

- Button (float x, float y, float width, float height, sf::Font *font, std::string text, sf::Color idleColor, sf::Color hoverColor, sf::Color activeColor, int text_size, bool box)

  Constructor.
- ~**Button** ()

  Destructor.
- const bool **isPressed** () const

  Accessor for determining whether a button is pressed.
- void update (const sf::Vector2f mousePos)

  Update button state based on mouse action.
- void render (sf::RenderTarget *target)

  Render button on target (window)
- void **changeText** (std::string text)

Private Attributes

- sf::Font * **font_**

  Button font.
- sf::RectangleShape **shape_**

  Button (rectangle) shape.
- sf::Text **buttonText_**

  Button text.
- int **text_size_**

  Button font size.
- bool **boxBoolean_**

  Whether to draw box or not.
- sf::Color **idleColor_**

  Button color when idle.
- sf::Color **hoverColor_**

  Button color when hovered over.
- sf::Color **activeColor_**

  Button color when pressed.
- short unsigned **buttonState_**

  Button state between idle (0), hover (1) and active (2)

4.3.1 Detailed Description

Button with visual parameters to initiate some action.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Button()

```
Button::Button (
            float x,
            float y,
            float width,
            float height,
            sf::Font * font,
            std::string text,
            sf::Color idleColor,
            sf::Color hoverColor,
            sf::Color activeColor,
            int text_size,
            bool box )
```

Constructor.

Parameters

x	X-coordinate of the button
y	Y-coordinate of the button
width	Width of the button
height	Height of the button
font	Font of the button
text	Text of the button
idleColor	Color of the button while idle
hoverColor	Color of the button while hovering over
activeColor	Color of the button when pressed
text_size	Text size of the button
box	Whether to include a box around the button

Note

(X,Y)-coordinates begin from the top-left corner (0,0)

4.3.3 Member Function Documentation

4.3.3.1 render()

```
void Button::render (
            sf::RenderTarget * target )
```

Render button on target (window)

Parameters

target	Rendering target, i.e. game window

4.3.3.2 update()

```
void Button::update (
            const sf::Vector2f mousePos )
```

Update button state based on mouse action.

Parameters

mousePos	Mouse position on screen

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

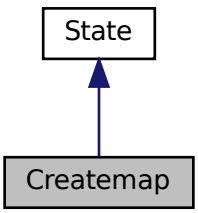
- src/button.hpp
- src/button.cpp

4.4 Createmap Class Reference

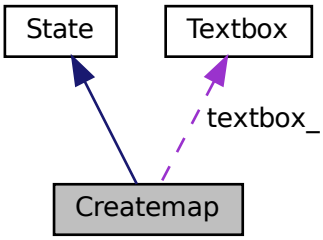
State that is used to display a map that can be edited.

`#include <createmap_state.hpp>`

Inheritance diagram for Createmap:



Collaboration diagram for Createmap:



Public Member Functions

- Createmap (sf::RenderWindow *window, std::map< std::string, int > *supportedKeys, std::string root_↩ filepath, std::stack< State * > *states, sf::Font *font, sf::Event *Event)

  Constructor.
- virtual ~**Createmap** ()

  Destructor.
- void **initSoundEffect** ()

  Initialize click sound effect.
- virtual void updateInput (const float &dt)

  Updates inputs.
- virtual void endState ()

  Called when ending this state.
- virtual void update (const float &dt)

  Updates objects.
- virtual void render (sf::RenderTarget *target=nullptr)

  Renders objects.
- void **updateButtons** ()

  Updates buttons.
- void renderButtons (sf::RenderTarget *target=nullptr)

  Renders all of the buttons.
- void **initVariables** ()

  Initializes some of the needed variables.
- void **SaveToFile** (std::string mapname)

  Saves the current map to a textfile.
- void **initTiles** ()

  Initializes default map containing only sand tiles.
- void ReplacePosition (int x, int y, std::string texturename, bool buildable)

  Changes the texture on the given position.
- void **initTextures** ()

  Initializes all the given textures.
- GameTile * TileAt (int x, int y)

  Returns the tile at that position.
- void NextTextureAt (int x, int y)

Changes the texture on the given position.

- void NextSpawnOrExit (int x, int y)

  Changes the texture on the given position.

- void ReplaceSpawnOrExit (int x, int y, std::string texturename)

  Changes the texture on the given position.

- virtual void **checkForQuit** ()

  Functions.

- const bool & **getQuit** () const
- virtual void **updateMousePosition** ()

Public Attributes

- bool **quit_**

Protected Attributes

- std::vector< sf::Texture > **textures**

  Container for textures.

- sf::RenderWindow * **window_**

  Pointer to a window.

- std::map< std::string, int > * **supportedKeys_**

  Pointer to supported keys.

- std::stack< State * > * **states**

  Pointer to a stack filled with state pointers (check game.hpp private)

- std::map< std::string, int > **keybinds_**
- sf::Vector2i **mousePosScreen**

  Mouse position variavles.

- sf::Vector2i **mousePosWindow**
- sf::Vector2f **mousePosView**

Private Member Functions

- void initKeyBinds ()

  Initializes keybinds.

- void **initButtons** ()

  Initializes buttons.

- const bool **getPressTimer** ()

  Gets the presstimer.

Private Attributes

- std::map< std::pair< int, int >, GameTile * > **Tiles**

  Map containing all of the tiles and their x and y values.
- std::map< std::pair< int, int >, GameTile * > **SpawnAndExit_**

  Map containing all of the spawn and exit tiles and their x and y values.
- std::string **root_filepath_**

  Root filepath.
- sf::Vector2u **mousePosGrid**

  Mouse position according to tiles starting from top left (0,0)
- std::vector< std::pair< bool, std::string > > **textures_**

  Vector containing names of all the textures and bool values if they are buildable.
- std::vector< std::pair< bool, std::string > > **spawnandexittextures_**

  Vector containing names of all the spawn and exit textures and bool values if they are buildable.
- sf::Clock **clock**

  Counts time.
- std::map< std::string, Button * > **mapcreateButtons_**

  Map containing all of the buttons.
- sf::Clock **pressTimer_**

  Timer.
- sf::Int32 **pressTimerMax_**

  Timer max.
- float **windowX_**

  Window's x length.
- float **windowY_**

  Window's y length.
- sf::Font * **font_**

  Font of the text.
- sf::SoundBuffer **buffer**
- sf::Sound **clickSound_**
- sf::Event * **Event_**
- Textbox * **textbox_**

4.4.1 Detailed Description

State that is used to display a map that can be edited.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Createmap()

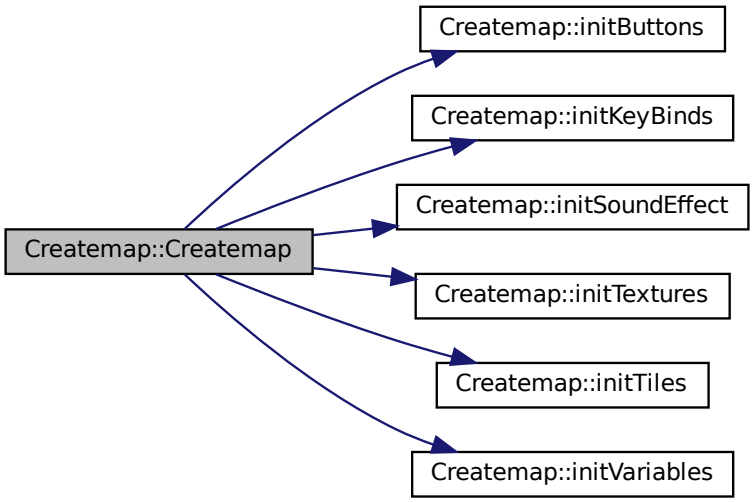
```
Createmap::Createmap (
            sf::RenderWindow * window,
            std::map< std::string, int > * supportedKeys,
            std::string root_filepath,
            std::stack< State * > * states,
            sf::Font * font,
            sf::Event * Event )
```

Constructor.

Parameters

window	Window object where objects are drawn
supportedKeys	Keys that are supported
root_filepath	Root filepath
states	Stack containing all of the current states
font	Font used for texts

Here is the call graph for this function:



4.4.3 Member Function Documentation

4.4.3.1 endState()

```
void Createmap::endState ( )  [virtual]
```

Called when ending this state.

Implements State.

4.4.3.2 initKeyBinds()

```
void Createmap::initKeyBinds ( )  [private], [virtual]
```

Initializes keybinds.

Implements State.

Here is the caller graph for this function:

4.4.3.3 NextSpawnOrExit()

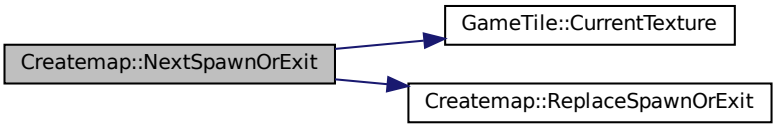
```
void Createmap::NextSpawnOrExit (
            int x,
            int y )
```

Changes the texture on the given position.

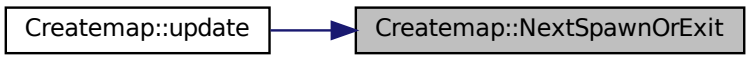
Parameters

x	X value of Mouse position
y	y value of Mouse position

Here is the call graph for this function:



Here is the caller graph for this function:



4.4.3.4 NextTextureAt()

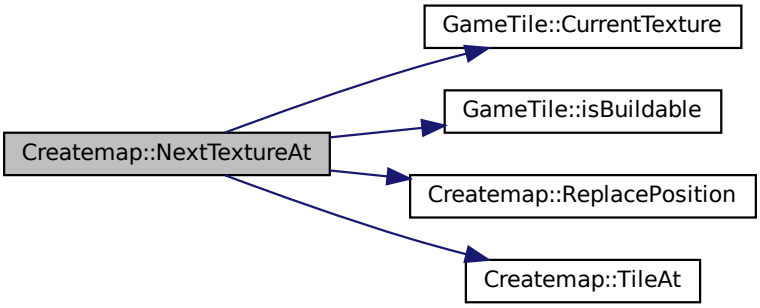
```
void Createmap::NextTextureAt (
            int x,
            int y )
```

Changes the texture on the given position.

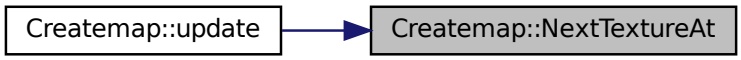
Parameters

x	X value of Mouse position
y	y value of Mouse position

Here is the call graph for this function:



Here is the caller graph for this function:



4.4.3.5 render()

```
void Createmap::render (
            sf::RenderTarget * target = nullptr )  [virtual]
```

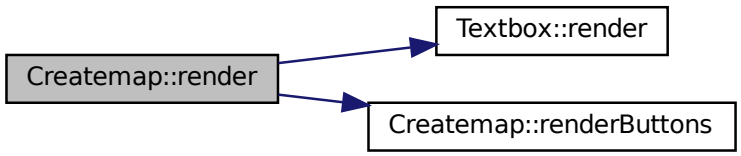
Renders objects.

Parameters

target	Rendering target, i.e. game window

Implements State.

Here is the call graph for this function:



4.4.3.6 renderButtons()

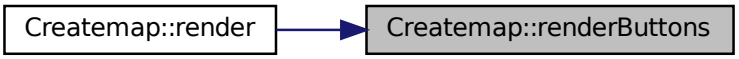
```
void Createmap::renderButtons (
            sf::RenderTarget * target = nullptr )
```

Renders all of the buttons.

Parameters

target	Rendering target, i.e. game window

Here is the caller graph for this function:



4.4.3.7 ReplacePosition()

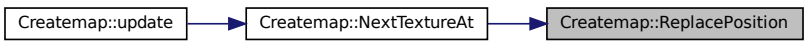
```
void Createmap::ReplacePosition (
            int x,
            int y,
            std::string texturename,
            bool buildable )
```

Changes the texture on the given position.

Parameters

x	X value of Mouse position
y	y value of Mouse position
texturename	Name of the texture that will replace the current texture
buildable	Tells if the next tile is going to be buildable tile or not

Here is the caller graph for this function:



4.4.3.8 ReplaceSpawnOrExit()

```
void Createmap::ReplaceSpawnOrExit (
            int x,
```

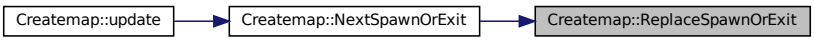
```
            int y,
            std::string texturename )
```

Changes the texture on the given position.

Parameters

x	X value of Mouse position
y	y value of Mouse position
texturename	Name of the texture that will replace the current texture

Here is the caller graph for this function:



4.4.3.9 TileAt()

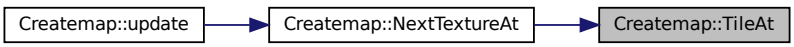
```
GameTile * Createmap::TileAt (
            int x,
            int y )
```

Returns the tile at that position.

Parameters

x	X value of Mouse position
y	y value of Mouse position

Here is the caller graph for this function:



4.4.3.10 update()

```
void Createmap::update (
            const float & dt )  [virtual]
```

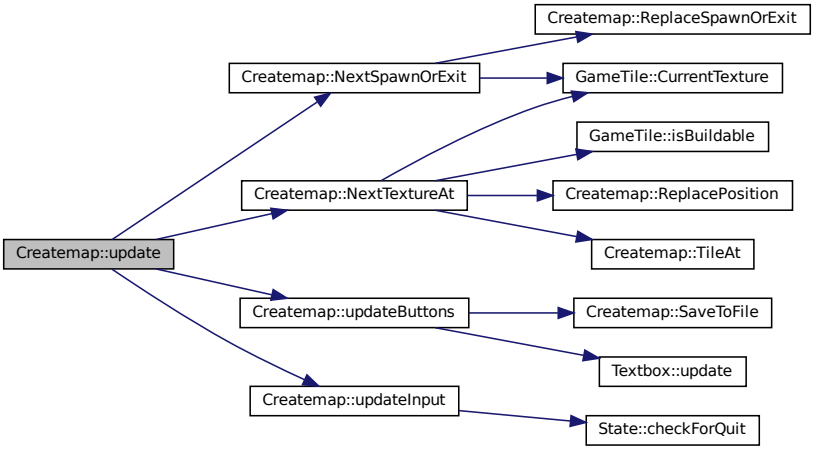
Updates objects.

Parameters

dt	Delta time

Implements State.

Here is the call graph for this function:



4.4.3.11 updateInput()

```
void Createmap::updateInput (
            const float & dt )  [virtual]
```

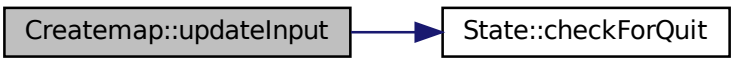
Updates inputs.

Parameters

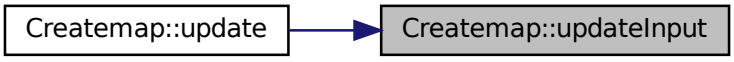
dt	Delta time

Implements State.

Here is the call graph for this function:



Here is the caller graph for this function:



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

- src/createmap_state.hpp
- src/createmap_state.cpp

4.5 Game Class Reference

Game header.

```
#include <game.hpp>
```

Public Member Functions

- **Game** (std::string &root_filepath)
  Constructor/Destructors.
- void **endApplication** ()
  Functions.
- void **updateDt** ()
  Update Functions.
- void **updateSFMLEvents** ()
- void **update** ()
- void **render** ()
  Render Functions.
- void **run** ()
  Core Functions.

Private Member Functions

- void **initVariables** ()
  Initializing.
- void **initWindow** ()
- void **initStates** ()
- void **initKeys** ()

Private Attributes

- std::string **root_filepath_**

  Variables.
- sf::RenderWindow * **window**
- sf::Event **sfEvent**
- std::vector< sf::VideoMode > **videoModes_**
- sf::ContextSettings **windowSettings_**
- bool **fullscreen_**
- float **dt**

  Timing.
- sf::Clock **dtClock**
- sf::Clock **quitclock_**
- std::stack< State * > **states**

  States stack.
- std::map< std::string, int > **supportedKeys**

  Map of supported keys.

4.5.1 Detailed Description

Game header.

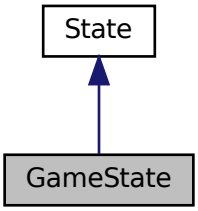
Include needed headers

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

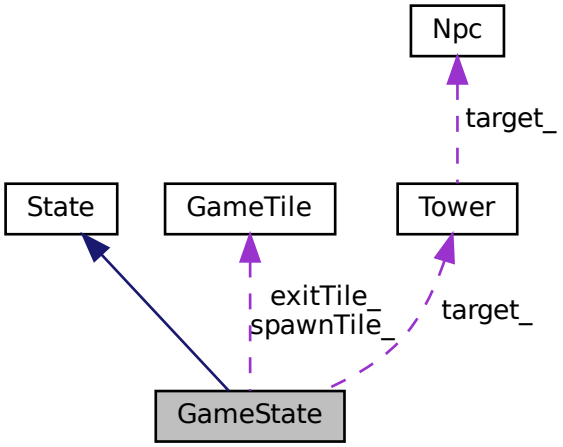
- src/game.hpp
- src/game.cpp

4.6 GameState Class Reference

Inheritance diagram for GameState:



Collaboration diagram for GameState:



Public Member Functions

- GameState (sf::RenderWindow *window, std::map< std::string, int > *supportedKeys, std::string root_↩ filepath, std::stack< State * > *states, std::string mapfile, sf::Font *font, int volumeLevel, int soundEffect↩ VolumeLevel)

  Constructor.
- virtual ~**GameState** ()

  Destructor.
- virtual void updateInput (const float &dt)

  Updates inputs.
- virtual void endState ()

  Called when ending this state.
- virtual void update (const float &dt)

  Updates objects.
- virtual void render (sf::RenderTarget *target=nullptr)

  Renders objects.
- void **loadmap** ()

  Reads the file given from the Maps folder and initializes the tiles.
- void **initSpawnLocation** ()

  Initializes the spawnlocation.
- void **initExitLocation** ()

  Initializes the exitlocation.
- void **initRoad** ()

  Initializes the road for enemies.
- void **initMusic** ()

  Initializes game state background music.
- Tower * **towerAt** (int x, int y)

  Returns tower, if there is one, at that gridlocation.
- void **buildRoad** (std::pair< int, int > current, std::pair< int, int > previous)

Builds the road recursively using this function.

- GameTile * **TileAt** (int x, int y)

  Returns tile, if there is one, at that gridlocation.
- std::vector< GameTile * > **getNeighbours** (GameTile *current)

  Returns every neighbour (not diagonals) of this tile.
- void **changeMoney** (int money)

  Changes the money player has.
- void **changeHealth** (int health)

  Changes the health player has.
- void **changeScore** (int score)

  Changes the score player has.
- bool **lostGame** ()

  Checks if the player has 0 or less health.
- sf::Vector2f getSpawn ()

  Returns the spawn position.
- sf::Vector2f **getExit** ()

  Returns the exit position.
- std::vector< GameTile * > **getRoad** ()

  Returns all roadtiles inside a vector.
- void **messageForPlayer** (std::string message)

  Puts message on the screen for the player for 5 seconds.
- void **updateButtons** ()

  Updates buttons.
- void **initVariables** ()

  Initializes variables.
- const bool **getPressTimer** ()

  Gets press timer.
- void **renderSoonToBeTower** (sf::RenderTarget *target)

  Renders ghost towers if they are selected.
- void **initGhostTowers** ()

  Initializes ghost towers.
- void **buyTower** ()

  Buys the selected tower and puts at the wanted location.
- void **updateTargetedTowerInfo** ()

  Updates info about selected tower to the UI.
- void **updateGameLogic** ()

  Handles the Round system logic.
- void **spawnEnemies** ()

  Handles the enemy spawning.
- virtual void **checkForQuit** ()

  Functions.
- const bool & **getQuit** () const
- virtual void **updateMousePosition** ()

Public Attributes

- bool **quit_**

Protected Attributes

- std::vector< sf::Texture > **textures**

  Container for textures.
- sf::RenderWindow * **window_**

  Pointer to a window.
- std::map< std::string, int > * **supportedKeys_**

  Pointer to supported keys.
- std::stack< State * > * **states**

  Pointer to a stack filled with state pointers (check game.hpp private)
- std::map< std::string, int > **keybinds_**
- sf::Vector2i **mousePosScreen**

  Mouse position variavles.
- sf::Vector2i **mousePosWindow**
- sf::Vector2f **mousePosView**

Private Member Functions

- void initKeyBinds ()

  Initializes keybinds.

Private Attributes

- std::string **root_filepath_**

  Root filepath.
- std::string **mapfile_**

  Name of the map.
- std::map< std::pair< int, int >, GameTile * > **Tiles**

  All tiles in the map and their gridlocations.
- std::map< std::pair< int, int >, GameTile * > **SpawnAndExit_**

  All spawn and exit tiles in the map and their gridlocations.
- sf::Vector2f **spawnlocation_**

  Spawn position.
- sf::Vector2f **exitlocation_**

  Exit position.
- sf::Text **money_**

  Current money as a text that can be drawn.
- sf::Text **health_**

  Current health as a text that can be drawn.
- sf::Text **score_**

  Current score as a text that can be drawn.
- int **int_money_** = 0

  Current money.
- int **int_health_** = 0

  Current health.
- int **int_score_** = 0

  Current score.
- GameTile * **spawnTile_**

  Spawn tile.
- GameTile * **exitTile_**

Exit tile.

- sf::Font * **font_**

  Font.

- std::vector< GameTile * > **roadTiles_**

  All road tiles.

- sf::RectangleShape **rect_menu_**

  RectangleShape used as sidebar.

- sf::Vector2u **mousePosGrid**

  Tracks the current mouseposition on grid.

- sf::Clock **clock**

  Timer.

- std::map< std::pair< int, int >, Tower * > **towers_**

  All towers and their gridlocations.

- std::list< Npc * > **enemies_**

  All enemies currently on the road.

- sf::Text **message_**

  Message to be displayed for the player.

- sf::Clock **message_timer_**

  Message timer.

- std::map< std::string, Button * > **towerbuttons_**
- std::map< std::string, Button * > **targeted_towerbuttons_**
- bool **basic_tower_flag_** = false
- bool **bomb_tower_flag_** = false
- bool **slowing_tower_flag_** = false
- sf::Clock **pressTimer_**
- sf::Int32 **pressTimerMax_**
- sf::Clock **timer_**
- sf::Int32 **timerMax_**
- float **windowX_**
- float **windowY_**
- sf::Sprite **basic_ghost_**
- sf::Sprite **bomb_ghost_**
- sf::Sprite **slowing_ghost_**
- sf::Texture **basic_texture_ghost_**
- sf::Texture **bomb_texture_ghost_**
- sf::Texture **slowing_texture_ghost_**
- sf::Clock **shoptimer_**
- Tower * **target_** = nullptr
- int **upgradeprice_**
- int **sellprice_**
- sf::Text **damage_text_**
- sf::Text **attack_speed_text_**
- sf::Text **tower_level_text**
- unsigned int **currentRound_** = 0
- int **enemiestobespawnedremaining_** = 0
- sf::Clock **roundtimer_**
- sf::Text **roundtimer_text_**
- int **roundtimer_int_**
- bool **roundover_flag_**
- std::map< std::string, Button * > **roundoverbuttons_**
- sf::Clock **spawntimer_**
- sf::Text **currentwave_**
- sf::Text **enemiesleft_**

- sf::Text **you_have_lost_**
- bool **game_over_** = false
- sf::Music **gamingStateMusic_**

  Music object and volume.
- int **volumeLevel_**
- int **soundEffectVolumeLevel_**

4.6.1 Constructor & Destructor Documentation

4.6.1.1 GameState()

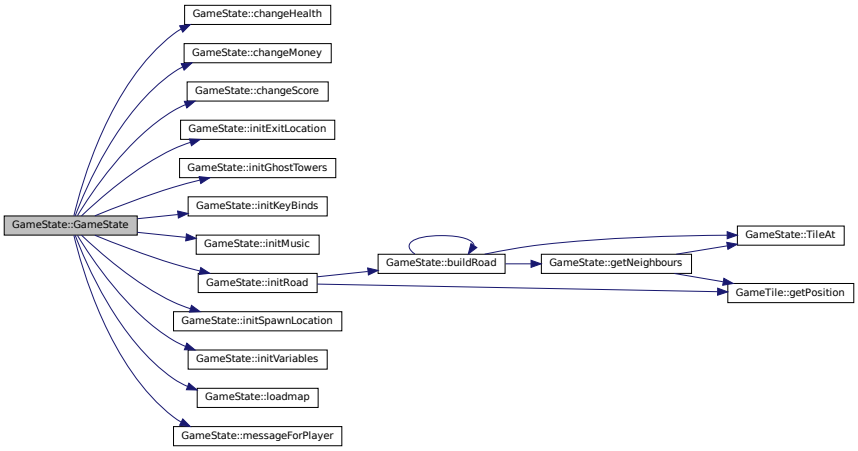
```
GameState::GameState (
            sf::RenderWindow * window,
            std::map< std::string, int > * supportedKeys,
            std::string root_filepath,
            std::stack< State * > * states,
            std::string mapfile,
            sf::Font * font,
            int volumeLevel,
            int soundEffectVolumeLevel )
```

Constructor.

Parameters

window	Window object where objects are drawn
supportedKeys	Keys that are supported
root_filepath	Root filepath
states	Stack containing all of the current states
mapfile	Name of the map to be loaded
font	Font used for text

Here is the call graph for this function:



4.6.2 Member Function Documentation

4.6.2.1 endState()

```
void GameState::endState ( )  [virtual]
```

Called when ending this state.

Implements State.

4.6.2.2 getSpawn()

```
sf::Vector2f GameState::getSpawn ( )
```

Returns the spawn position.

Osku addition.

4.6.2.3 initKeyBinds()

```
void GameState::initKeyBinds ( )  [private], [virtual]
```

Initializes keybinds.

Implements State.

Here is the caller graph for this function:

4.6.2.4 render()

```
void GameState::render (
            sf::RenderTarget * target = nullptr )  [virtual]
```

Renders objects.

Parameters

target	Rendering target, i.e. game window

Implements State.

Here is the call graph for this function:

4.6.2.5 update()

```
void GameState::update (
            const float & dt )  [virtual]
```

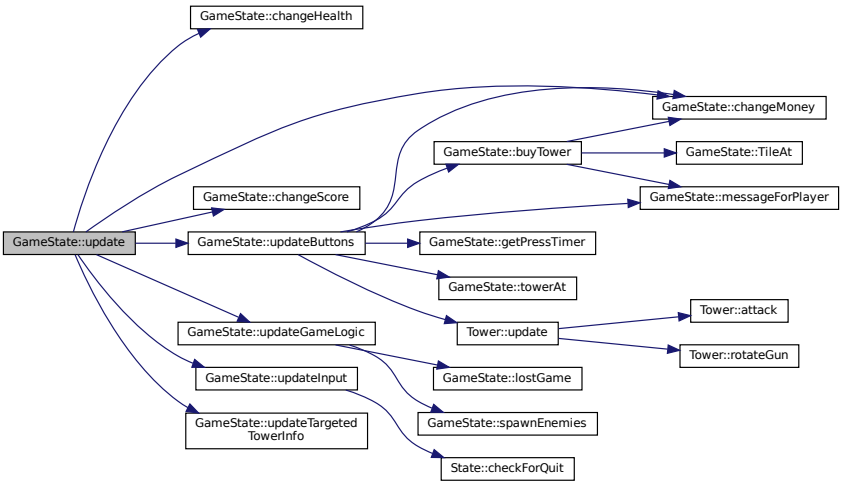
Updates objects.

Parameters

dt	Delta time

Implements State.

Here is the call graph for this function:



4.6.2.6 updateInput()

```
void GameState::updateInput (
            const float & dt )  [virtual]
```

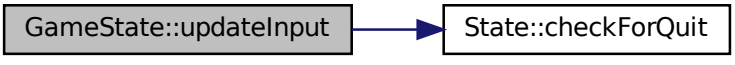
Updates inputs.

Parameters

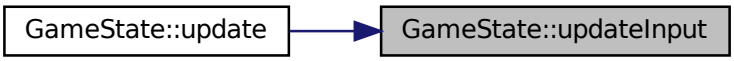
dt	Delta time

Implements State.

Here is the call graph for this function:



Here is the caller graph for this function:



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

- src/gaming_state.hpp
- src/gaming_state.cpp

4.7 GameTile Class Reference

GameTile object that is used for a construction of a map.

`#include <gametile.hpp>`

Public Member Functions

- GameTile (std::string root_filepath, std::string texturename, sf::Vector2f pos, bool buildable, int squaresize)
  Constructor.
- ~**GameTile** ()
  Destructor.
- void **setUpSprite** ()
  Sets up the sprite.
- std::string **CurrentTexture** ()
  Returns the name of the texture.
- sf::Vector2f **getCenterPosition** ()
  Returns the center position of the tile.
- sf::Vector2f **getPosition** ()
  Returns the top left corner position of this tile.
- int **getSquareSize** ()
  Returns the squaresize of the tile.
- bool **isExit** ()
  Checks if the tile is exit tile.
- bool **isSpawn** ()
  Checks if the tile is spawn tile.
- bool **isRoad** ()
  Checks if the tile is a road tile.
- bool **isBuildable** ()
  Checks if the tile is a buildable tile.
- void render (sf::RenderTarget *target)
  Renders the tile.

Protected Attributes

- sf::Vector2f **pos_**

  Top left corner position of the tile.
- sf::Texture **texture_**

  Texture object.
- sf::Sprite **sprite_**

  Sprite of this tile.
- std::string **root_filepath_**

  Root filepath.
- std::string **texturename_**

  Name of the texture.
- bool **buildable_**

  Value telling if the tile is buildable or not.
- int **squaresize_**

  Squaresize of the tile.

4.7.1 Detailed Description

GameTile object that is used for a construction of a map.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 GameTile()

```
GameTile::GameTile (
            std::string root_filepath,
            std::string texturename,
            sf::Vector2f pos,
            bool buildable,
            int squaresize )
```

Constructor.

Parameters

root_filepath	Root filepath
texturename	Name of the texture used for the tile
pos	Position of the tile
buildable	Value telling if the tile is buildable
squaresize	Size of the tile

Here is the call graph for this function:

4.7.3 Member Function Documentation

4.7.3.1 render()

```
void GameTile::render (
            sf::RenderTarget * target )
```

Renders the tile.

Parameters

target	Rendering target, i.e. game window

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

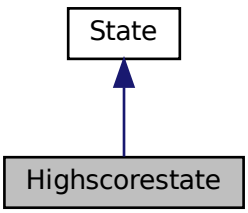
- src/gametile.hpp
- src/gametile.cpp

4.8 Highscorestate Class Reference

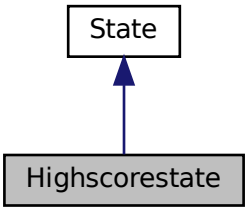
State that shows all of the highscores on the screen.

`#include <highscore_state.hpp>`

Inheritance diagram for Highscorestate:



Collaboration diagram for Highscorestate:



Public Member Functions

- Highscorestate (sf::RenderWindow *window, std::map< std::string, int > *supportedKeys, std::string root_filepath, std::stack< State * > *states, sf::Font *font)

  Constructor.
- virtual ~**Highscorestate** ()

  Destructor.
- void **initSoundEffect** ()

  Initialize sound effects.
- virtual void updateInput (const float &dt)

  Updates inputs.
- virtual void endState ()

  Called when ending this state.
- virtual void update (const float &dt)

  Updates objects.
- virtual void render (sf::RenderTarget *target=nullptr)

  Renders objects.
- void **updateButtons** ()

  Updates buttons.
- void renderButtons (sf::RenderTarget *target=nullptr)

  Renders all of the buttons.
- void **Readhighscores** ()

  Reads the file containing the highscores.
- virtual void **checkForQuit** ()

  Functions.
- const bool & **getQuit** () const
- virtual void **updateMousePosition** ()

Static Public Member Functions

- static void Addhighscore (const std::string name, int highscore)

  Adds a new highscore to the textfile.

Public Attributes

- bool **quit_**

Protected Attributes

- std::vector< sf::Texture > **textures**

  Container for textures.
- sf::RenderWindow * **window_**

  Pointer to a window.
- std::map< std::string, int > * **supportedKeys_**

  Pointer to supported keys.
- std::stack< State * > * **states**

  Pointer to a stack filled with state pointers (check game.hpp private)
- std::map< std::string, int > **keybinds_**
- sf::Vector2i **mousePosScreen**

  Mouse position variavles.
- sf::Vector2i **mousePosWindow**
- sf::Vector2f **mousePosView**

Private Member Functions

- void **initVariables** ()

  Initializes some of the needed variables.
- void initKeyBinds ()

  Initializes keybinds.
- void **initHighscores** ()

  Initializes the highscores.
- void **initButtons** ()

  Initializes the buttons.
- const bool **getPressTimer** ()

  Gets the presstimer.

Private Attributes

- std::string **root_filepath_**

  Root filepath.
- std::list< std::pair< int, std::string > > **highscores_**

  List containing all of the highscores and names.
- sf::RectangleShape **background_**

  Rectangle background.
- sf::Font * **font_**

  Font of the text.
- std::vector< sf::Text > **Texts_**

  Vector containing all of the texts to be drawn.
- std::map< std::string, Button * > **highscroeButtons_**

  Map containing all of the buttons.
- sf::Clock **pressTimer_**

  Timer.

- sf::Int32 **pressTimerMax_**

  Timer max.
- sf::SoundBuffer **buffer**
- sf::Sound **clickSound_**
- float **windowX_**

  Window's x length.
- float **windowY_**

  Window's y length.

4.8.1 Detailed Description

State that shows all of the highscores on the screen.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Highscorestate()

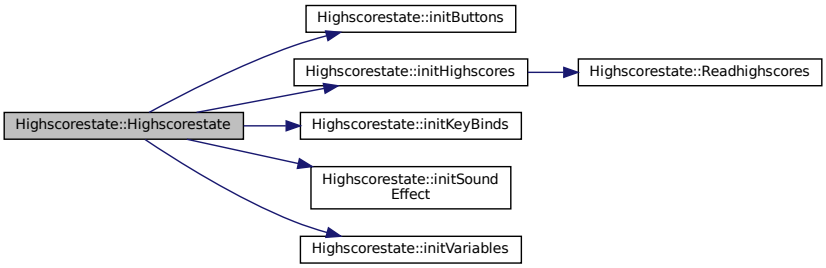
```
Highscorestate::Highscorestate (
            sf::RenderWindow * window,
            std::map< std::string, int > * supportedKeys,
            std::string root_filepath,
            std::stack< State * > * states,
            sf::Font * font )
```

Constructor.

Parameters

window	Window object where objects are drawn
supportedKeys	Keys that are supported
root_filepath	Root filepath
states	Stack containing all of the current states
font	Font used for texts

Here is the call graph for this function:



4.8.3 Member Function Documentation

4.8.3.1 Addhighscore()

```
void Highscorestate::Addhighscore (
            const std::string name,
            int highscore )  [static]
```

Adds a new highscore to the textfile.

Parameters

name	Name of the person
highscore	Score achieved

4.8.3.2 endState()

```
void Highscorestate::endState ( )  [virtual]
```

Called when ending this state.

Implements State.

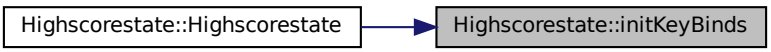
4.8.3.3 initKeyBinds()

```
void Highscorestate::initKeyBinds ( )  [private], [virtual]
```

Initializes keybinds.

Implements State.

Here is the caller graph for this function:



4.8.3.4 render()

```
void Highscorestate::render (
            sf::RenderTarget * target = nullptr )  [virtual]
```

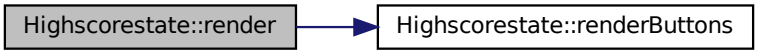
Renders objects.

Parameters

target	Rendering target, i.e. game window

Implements State.

Here is the call graph for this function:



4.8.3.5 renderButtons()

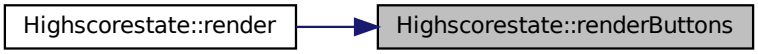
```
void Highscorestate::renderButtons (
            sf::RenderTarget * target = nullptr )
```

Renders all of the buttons.

Parameters

target	Rendering target, i.e. game window

Here is the caller graph for this function:



4.8.3.6 update()

```
void Highscorestate::update (
            const float & dt )  [virtual]
```

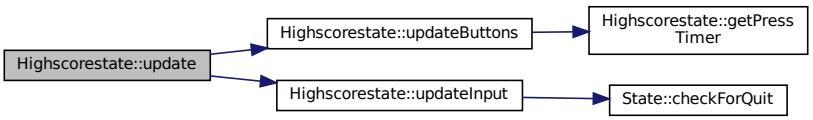
Updates objects.

Parameters

dt	Delta time

Implements State.

Here is the call graph for this function:



4.8.3.7 updateInput()

```
void Highscorestate::updateInput (
            const float & dt )  [virtual]
```

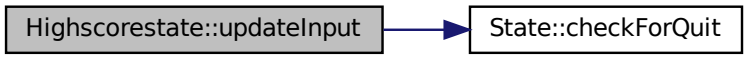
Updates inputs.

Parameters

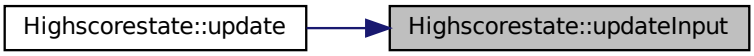
dt	Delta time

Implements State.

Here is the call graph for this function:



Here is the caller graph for this function:

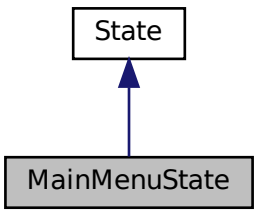


The documentation for this class was generated from the following files:

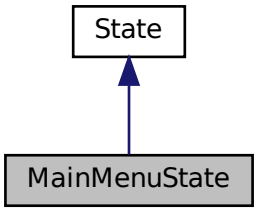
- src/highscore_state.hpp
- src/highscore_state.cpp

4.9 MainMenuState Class Reference

Inheritance diagram for MainMenuState:



Collaboration diagram for MainMenuState:



Public Member Functions

- **MainMenuState** (sf::RenderWindow *window, std::map< std::string, int > *supportedKeys, std::string root_filepath, std::stack< State * > *states, sf::Event *Event)

  Constructor and destructor.
- int **getSFXVolume** ()

  Functions.
- virtual void updateInput (const float &dt)
- virtual void endState ()
- virtual void update (const float &dt)
- virtual void render (sf::RenderTarget *target=nullptr)
- void **updateButtons** ()
- void **renderButtons** (sf::RenderTarget *target=nullptr)
- void **updateAnimation** (const float &dt)

  Main menu animation.
- virtual void **checkForQuit** ()

  Functions.
- const bool & **getQuit** () const
- virtual void **updateMousePosition** ()

Public Attributes

- bool **quit_**

Protected Attributes

- std::vector< sf::Texture > **textures**

  Container for textures.
- std::map< std::string, int > * **supportedKeys_**

  Pointer to supported keys.
- std::stack< State * > * **states**

  Pointer to a stack filled with state pointers (check game.hpp private)
- std::map< std::string, int > **keybinds_**
- sf::Vector2i **mousePosScreen**

  Mouse position variavles.
- sf::Vector2i **mousePosWindow**
- sf::Vector2f **mousePosView**

Private Member Functions

- void **initVariables** ()

  Functions.
- void **initBackground** ()
- void initKeyBinds ()

  Functions.
- void **initFonts** ()
- void **initButtons** ()
- void **initMusic** ()
- void **initBGAnimation** ()
- const bool **getTimer** ()

  Timer function.
- const bool **getPressTimer** ()

  Button press timer.

Private Attributes

- sf::RenderWindow * **window_**

  Variables.
- std::string **root_filepath_**
- sf::Texture **backgroundTexture_**

  Background variables.
- sf::RectangleShape **background_**
- sf::RectangleShape **backgroundText_**
- sf::RectangleShape **backgroundShader_**
- sf::Text **backgroundTextUsingFont_**
- sf::Font **font_**
- sf::Music **backgroundMusic_**
- std::map< std::string, Button * > **buttons_**
- float **windowX_**
- float **windowY_**
- sf::Event * **Event_**
- sf::Texture **rocketTexture_**

  Rocket animation.
- sf::Texture **rocketExhaustTX_**
- sf::Sprite **rocket_**
- sf::Sprite **rocketExhaust_**
- bool **passState_**
- float **rocketMoveSpeed_**
- sf::Clock **timer_**
- sf::Int32 **timerMax_**
- sf::Clock **pressTimer_**

  Press timer.
- sf::Int32 **pressTimerMax_**
- int **volumeLevel_**

  Volume settings.
- int **soundEffectVolumeLevel_**
- sf::Text **volumeLevelText_**
- sf::Text **soundEffectvolumeLevelText_**
- sf::SoundBuffer **buffer**
- sf::Sound **clickSound_**

4.9.1 Member Function Documentation

4.9.1.1 endState()

```
void MainMenuState::endState ( )  [virtual]
```

Implements State.

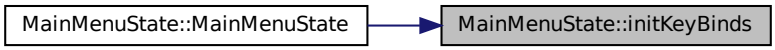
4.9.1.2 initKeyBinds()

```
void MainMenuState::initKeyBinds ( )  [private], [virtual]
```

Functions.

Implements State.

Here is the caller graph for this function:



4.9.1.3 render()

```
void MainMenuState::render (
            sf::RenderTarget * target = nullptr )  [virtual]
```

Implements State.

4.9.1.4 update()

```
void MainMenuState::update (
            const float & dt )  [virtual]
```

Implements State.

4.9.1.5 updateInput()

```
void MainMenuState::updateInput (
            const float & dt )  [virtual]
```

Implements State.

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

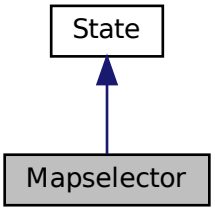
- src/main_menu_state.hpp
- src/main_menu_state.cpp

4.10 Mapselector Class Reference

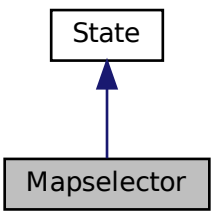
Header for GameStates class.

```
#include <mapselector_state.hpp>
```

Inheritance diagram for Mapselector:



Collaboration diagram for Mapselector:



Public Member Functions

- **Mapselector** (sf::RenderWindow *window, std::map< std::string, int > *supportedKeys, std::string root↩ _filepath, std::stack< State * > *states, sf::Font *font, sf::Event *Event, int volumeLevel, int soundEffect↩ VolumeLevel)
- virtual void updateInput (const float &dt)

  Functions.
- virtual void endState ()
- virtual void update (const float &dt)
- virtual void render (sf::RenderTarget *target=nullptr)
- void **openFolder** ()
- void **initVariables** ()
- void **initSoundEffect** ()
- const bool **getPressTimer** ()
- virtual void **checkForQuit** ()

  Functions.
- const bool & **getQuit** () const
- virtual void **updateMousePosition** ()

Public Attributes

- bool **quit_**

Protected Attributes

- std::vector< sf::Texture > **textures**

  Container for textures.
- sf::RenderWindow * **window_**

  Pointer to a window.
- std::map< std::string, int > * **supportedKeys_**

  Pointer to supported keys.
- std::stack< State * > * **states**

  Pointer to a stack filled with state pointers (check game.hpp private)
- std::map< std::string, int > **keybinds_**
- sf::Vector2i **mousePosScreen**

  Mouse position variavles.
- sf::Vector2i **mousePosWindow**
- sf::Vector2f **mousePosView**

Private Member Functions

- void initKeyBinds ()

  Functions.

Private Attributes

- std::string **root_filepath_**
- sf::Font * **font_**
- float **windowX_**
- float **windowY_**
- sf::RectangleShape **backgroundColor_**
- int **scrollUpperBoundary_**
- int **scrollLowerBoundary_**
- int **volumeLevel_**
- int **soundEffectVolumeLevel_**
- sf::SoundBuffer **buffer**
- sf::Sound **clickSound_**
- sf::Clock **pressTimer_**

  Press timer.
- sf::Int32 **pressTimerMax_**
- sf::Clock **timer_**
- sf::Int32 **timerMax_**
- std::map< std::string, Button * > **buttons_**
- sf::Event * **Event_**
- sf::Clock **clock_**

4.10.1 Detailed Description

Header for GameStates class.

4.10.2 Member Function Documentation

4.10.2.1 endState()

```
void Mapselector::endState ( )  [virtual]
```

Implements State.

4.10.2.2 initKeyBinds()

```
void Mapselector::initKeyBinds ( )  [private], [virtual]
```

Functions.

Implements State.

4.10.2.3 render()

```
void Mapselector::render (
            sf::RenderTarget * target = nullptr )  [virtual]
```

Implements State.

4.10.2.4 update()

```
void Mapselector::update (
            const float & dt )  [virtual]
```

Implements State.

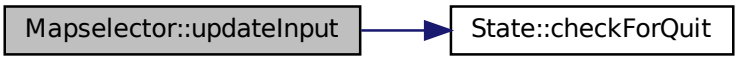
4.10.2.5 updateInput()

```
void Mapselector::updateInput (
            const float & dt )  [virtual]
```

Functions.

Implements State.

Here is the call graph for this function:



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

- src/mapselector_state.hpp
- src/mapselector_state.cpp

4.11 Missile Class Reference

Public Member Functions

- **Missile** (std::string root_filepath, sf::Vector2f spawn, sf::Vector2f target, int damage, int speed, int sound↩ EffectVolumeLevel)
- void **move** (const float &dt)
- void **setUpSprites** ()
- void **render** (sf::RenderTarget *target)
- bool **hasExploded** ()
- void **dealDamage** (std::list< Npc * > enemies)
- void **update** (std::list< Npc * > enemies)
- void **initSoundEffect** ()

  Initializes explostion sound effect.

Public Attributes

- sf::Clock **clock_**

Private Attributes

- std::string **missile_texture_name_**
- std::string **explosion_texture_name_**
- sf::Texture **missile_texture_**
- sf::Texture **explosion_texture_**
- sf::Sprite **missile_sprite_**
- sf::Sprite **explosion_sprite_**
- sf::Vector2f **spawn_**
- sf::Vector2f **target_**
- std::list< Npc * > **enemies_**
- int **damage_**
- int **blast_radius_**
- int **speed_**
- sf::Vector2f **direction_**
- bool **explosion_** = false
- std::string **root_filepath_**
- sf::SoundBuffer **buffer**

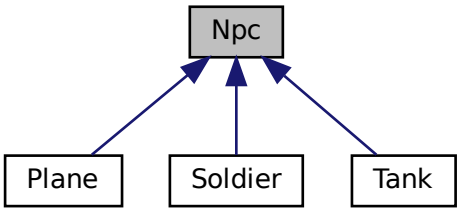
  Explostion sound effect.
- sf::Sound **sound**
- int **soundEffectVolumeLevel_**

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

- src/missile.hpp
- src/missile.cpp

4.12 Npc Class Reference

Inheritance diagram for Npc:



Public Member Functions

- **Npc** (std::string root_filepath, sf::Vector2f sLocation, sf::Vector2f eLocation, std::vector< GameTile * > *r↩ Tiles, int newTileCount=0)

  Constructor/Destructors.
- virtual void Update (const float &dt)

  Functions.
- virtual void **Render** (sf::RenderTarget *target)
- virtual void **MoveTo** (const float &dt)
- virtual void **initNpc** ()
- virtual std::pair< int, int > **FindDirection** (sf::Vector2f nextLocation)
- virtual void **Rotate** (int x, int y)
- virtual sf::Vector2f **getPosition** ()
- virtual int **getHitpoints** ()
- virtual bool **hasReachedEnd** ()
- virtual void **dealDamage** (int damage)
- virtual void **slowMovement** (float slow)
- virtual int **getTileCount** ()
- virtual int **getWorth** ()

Protected Attributes

- float **movementSpeed**
- float **movementspeedmemory_**
- int **hitpoints** = 1
- sf::Vector2f **spawnLocation**
- sf::Vector2f **exitLocation**
- sf::Vector2f **nextLocation**
- std::vector< GameTile * > * **roadTiles**
- int **tileCount** = 0
- sf::Sprite **sotilastektuuri**
- sf::Texture **stexture_**
- int **worth_** = 1
- sf::Clock **slowcooldown_**
- bool **end_** = false
- std::string **root_filepath_**

4.12.1 Member Function Documentation

4.12.1.1 Update()

```
void Npc::Update (
            const float & dt )  [virtual]
```

Functions.

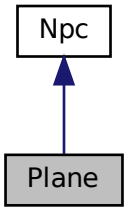
Reimplemented in Plane, Soldier, and Tank.

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

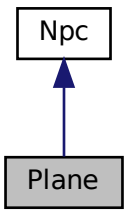
- src/npc.hpp
- src/npc.cpp

4.13 Plane Class Reference

Inheritance diagram for Plane:



Collaboration diagram for Plane:



Public Member Functions

- **Plane** (std::string root_filepath, sf::Vector2f sLocation, sf::Vector2f eLocation, std::vector< GameTile * > *rTiles)

  Constructor/Destructors.
- virtual void Update (const float &dt)

  Functions.
- virtual void Render (sf::RenderTarget *target)
- virtual void MoveTo (const float &dt)
- virtual void initNpc ()
- std::pair< int, int > FindDirection (sf::Vector2f nextLocation)
- virtual void Rotate (int x, int y)
- sf::Vector2f getPosition ()
- int getHitpoints ()
- bool hasReachedEnd ()
- void dealDamage (int damage)
- void slowMovement (float slow)
- virtual int **getTileCount** ()
- virtual int **getWorth** ()

Protected Attributes

- float **movementSpeed**
- float **movementspeedmemory_**
- int **hitpoints** = 1
- sf::Vector2f **spawnLocation**
- sf::Vector2f **exitLocation**
- sf::Vector2f **nextLocation**
- std::vector< GameTile * > * **roadTiles**
- int **tileCount** = 0
- sf::Sprite **planeTexture**
- sf::Texture **texture_**
- sf::Clock **slowcooldown_**
- bool **end_** = false
- std::string **root_filepath_**
- sf::Sprite **sotilastektuuri**
- sf::Texture **stexture_**
- int **worth_** = 1

4.13.1 Member Function Documentation

4.13.1.1 dealDamage()

```
void Plane::dealDamage (
            int damage )  [virtual]
```

Reimplemented from Npc.

4.13.1.2 FindDirection()

```
std::pair< int, int > Plane::FindDirection (
            sf::Vector2f nextLocation )  [virtual]
```

Reimplemented from Npc.

4.13.1.3 getHitpoints()

```
int Plane::getHitpoints ( )  [virtual]
```

Reimplemented from Npc.

4.13.1.4 getPosition()

```
sf::Vector2f Plane::getPosition ( )  [virtual]
```

Reimplemented from Npc.

4.13.1.5 hasReachedEnd()

```
bool Plane::hasReachedEnd ( )  [virtual]
```

Reimplemented from Npc.

4.13.1.6 initNpc()

```
void Plane::initNpc ( )  [virtual]
```

Reimplemented from Npc.

4.13.1.7 MoveTo()

```
void Plane::MoveTo (
            const float & dt )  [virtual]
```

Reimplemented from Npc.

4.13.1.8 Render()

```
void Plane::Render (
            sf::RenderTarget * target )  [virtual]
```

Reimplemented from Npc.

4.13.1.9 Rotate()

```
void Plane::Rotate (
            int x,
            int y )  [virtual]
```

Reimplemented from Npc.

4.13.1.10 slowMovement()

```
void Plane::slowMovement (
            float slow )  [virtual]
```

Reimplemented from Npc.

4.13.1.11 Update()

```
void Plane::Update (
            const float & dt )  [virtual]
```

Functions.

Reimplemented from Npc.

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

- src/plane.hpp
- src/plane.cpp

4.14 Slimeball Class Reference

Public Member Functions

- **Slimeball** (std::string root_filepath, sf::Vector2f spawn, sf::Vector2f target, int damage, int speed, int slowing_parameter)
- void **move** (const float &dt)
- void **setUpSprites** ()
- void **render** (sf::RenderTarget *target)
- bool **hasExploded** ()
- void **dealDamage** (std::list< Npc * > enemies)
- void **update** (std::list< Npc * > enemies)

Public Attributes

- sf::Clock **clock_**

Private Attributes

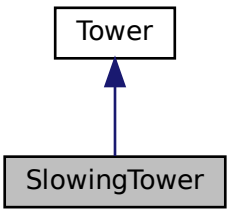
- std::string **missile_texture_name_**
- std::string **explosion_texture_name_**
- sf::Texture **missile_texture_**
- sf::Texture **explosion_texture_**
- sf::Sprite **missile_sprite_**
- sf::Sprite **explosion_sprite_**
- sf::Vector2f **spawn_**
- sf::Vector2f **target_**
- std::list< Npc * > **enemies_**
- int **damage_**
- int **blast_radius_**
- int **speed_**
- int **slowing_parameter_**
- sf::Vector2f **direction_**
- bool **explosion_** = false
- std::string **root_filepath_**

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

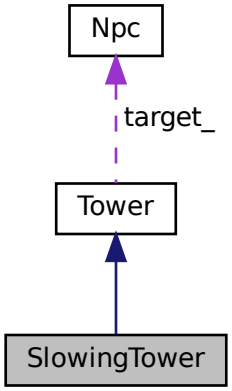
- src/slimeball.hpp
- src/slimeball.cpp

4.15 SlowingTower Class Reference

Inheritance diagram for SlowingTower:



Collaboration diagram for SlowingTower:



Public Member Functions

- **SlowingTower** (std::string root_filepath, sf::Vector2f pos, float attack_speed, int damage, float radius, int soundEffectVolumeLevel)
- void attack (const float &dt)

  Attacks a target if the tower has one. Dealing damage and possible slowing the enemy.
- void update (const float &dt, std::list< Npc * > enemies, const sf::Vector2f mousePos)

  Update tower with new information.
- void render (sf::RenderTarget *target)

  Render tower on target (window)
- void upgrade ()
- void **initSoundEffect** ()

  Initializes shooting sound effect.
- void **setUpSprites** ()

  Loads required textures and sets up sprites.
- void rotateGun (const float &dt, std::list< Npc * > enemies)

  Rotates the gun part of the tower towards first enemy and targets that enemy for the next attack.
- float **getDamage** ()
- float **getAttackSpeed** ()
- int **getTowerLevel** ()
- sf::Vector2f **getPosition** ()

Protected Attributes

- sf::Sprite **platform_**

  Sprite for the platform part of the tower.
- sf::Sprite **gun_**

  Sprite for the gun part of the tower.
- sf::Texture **sftexture_platform_**

  Texture for the platform part of the tower.

- sf::Texture **sftexture_gun_**
  Texture for the gun part of the tower.
- sf::Texture **sftexture_gunfire_**
- std::string **texture_gunfire_**
- sf::Sprite **gunfire_**
- Npc * **target_**
  Target for the next attack.
- sf::Clock **clock_**
  Measures time.
- sf::Vector2f **pos_**
  Position of this tower.
- std::string **root_filepath_**
  Root filepath.
- std::string **texture_platform_**
  Name of the platform texture.
- std::string **texture_gun_**
  Name of the gun texture.
- sf::CircleShape **radius_shape_**
  CircleShape to show the radius of the tower.
- float **attack_speed_**
  Attack speed (seconds)
- float **damage_**
  Damage.
- int **tower_level_** = 1
- float **radius_**
  Attack radius.
- float **slowing_parameter_**
  Slowing parameter.
- short unsigned **towerState_**
  Checks if the mouse is hovering the tower.
- sf::Transform **transform_**
- double **angle_**
- sf::Clock **gunfire_clock_**
- sf::Color **gunfire_color_**

Private Attributes

- std::list< Slimeball * > **missiles_**
- sf::SoundBuffer **buffer**
  Shoot sound effect.
- sf::Sound **sound**
- int **soundEffectVolumeLevel_**

4.15.1 Member Function Documentation

4.15.1.1 attack()

```
void SlowingTower::attack (
            const float & dt )  [virtual]
```

Attacks a target if the tower has one. Dealing damage and possible slowing the enemy.

Parameters

dt	Delta time

Reimplemented from Tower.

Here is the caller graph for this function:

4.15.1.2 render()

```
void SlowingTower::render (
            sf::RenderTarget * target )  [virtual]
```

Render tower on target (window)

Parameters

target	Rendering target, i.e. game window

Reimplemented from Tower.

4.15.1.3 rotateGun()

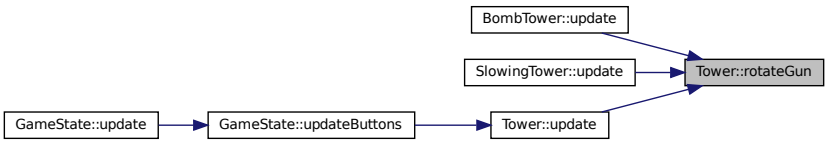
```
void Tower::rotateGun (
            const float & dt,
            std::list< Npc * > enemies )  [inherited]
```

Rotates the gun part of the tower towards first enemy and targets that enemy for the next attack.

Parameters

dt	Delta time
enemies	List of every enemy on the board currently

Here is the caller graph for this function:



4.15.1.4 update()

```
void SlowingTower::update (
            const float & dt,
            std::list< Npc * > enemies,
            const sf::Vector2f mousePos )  [virtual]
```

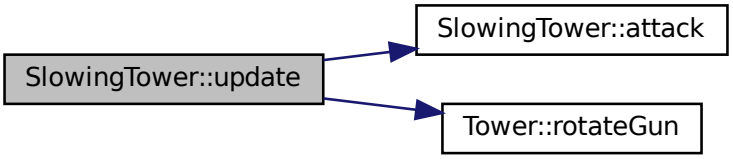
Update tower with new information.

Parameters

dt	Delta time
enemies	List of every enemy on the board currently
mousePos	Mouse position on screen

Reimplemented from Tower.

Here is the call graph for this function:



4.15.1.5 upgrade()

```
void SlowingTower::upgrade ( )  [virtual]
```

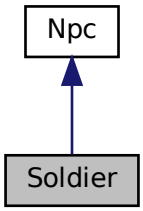
Reimplemented from Tower.

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

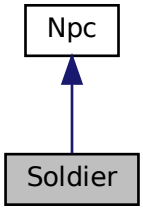
- src/slowing_tower.hpp
- src/slowing_tower.cpp

4.16 Soldier Class Reference

Inheritance diagram for Soldier:



Collaboration diagram for Soldier:



Public Member Functions

- **Soldier** (std::string root_filepath, sf::Vector2f sLocation, sf::Vector2f eLocation, std::vector< GameTile * > *rTiles, int newTileCount=0)

  Constructor/Destructors.
- virtual void Update (const float &dt)

  Functions.
- virtual void Render (sf::RenderTarget *target)
- virtual void MoveTo (const float &dt)
- virtual void initNpc ()
- std::pair< int, int > FindDirection (sf::Vector2f nextLocation)
- virtual void Rotate (int x, int y)
- sf::Vector2f getPosition ()

- int getHitpoints ()
- bool hasReachedEnd ()
- void dealDamage (int damage)
- void slowMovement (float slow)
- virtual int **getTileCount** ()
- virtual int **getWorth** ()

Protected Attributes

- float **movementSpeed**
- float **movementspeedmemory_**
- int **hitpoints** = 1
- sf::Vector2f **spawnLocation**
- sf::Vector2f **exitLocation**
- sf::Vector2f **nextLocation**
- std::vector< GameTile * > * **roadTiles**
- int **tileCount** = 0
- sf::Sprite **soldierTexture**
- sf::Texture **texture_**
- sf::Clock **slowcooldown_**
- bool **end_** = false
- std::string **root_filepath_**
- sf::Sprite **sotilastektuuri**
- sf::Texture **stexture_**
- int **worth_** = 1

4.16.1 Member Function Documentation

4.16.1.1 dealDamage()

```
void Soldier::dealDamage (
            int damage )  [virtual]
```

Reimplemented from Npc.

4.16.1.2 FindDirection()

```
std::pair< int, int > Soldier::FindDirection (
            sf::Vector2f nextLocation )  [virtual]
```

Reimplemented from Npc.

4.16.1.3 getHitpoints()

```
int Soldier::getHitpoints ( )  [virtual]
```

Reimplemented from Npc.

4.16.1.4 getPosition()

```
sf::Vector2f Soldier::getPosition ( )  [virtual]
```

Reimplemented from Npc.

4.16.1.5 hasReachedEnd()

```
bool Soldier::hasReachedEnd ( )  [virtual]
```

Reimplemented from Npc.

4.16.1.6 initNpc()

```
void Soldier::initNpc ( )  [virtual]
```

Reimplemented from Npc.

4.16.1.7 MoveTo()

```
void Soldier::MoveTo (
            const float & dt )  [virtual]
```

Reimplemented from Npc.

4.16.1.8 Render()

```
void Soldier::Render (
            sf::RenderTarget * target )  [virtual]
```

Reimplemented from Npc.

4.16.1.9 Rotate()

```
void Soldier::Rotate (
            int x,
            int y )  [virtual]
```

Reimplemented from Npc.

4.16.1.10 slowMovement()

```
void Soldier::slowMovement (
            float slow )  [virtual]
```

Reimplemented from Npc.

4.16.1.11 Update()

```
void Soldier::Update (
            const float & dt )  [virtual]
```

Functions.

Reimplemented from Npc.

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

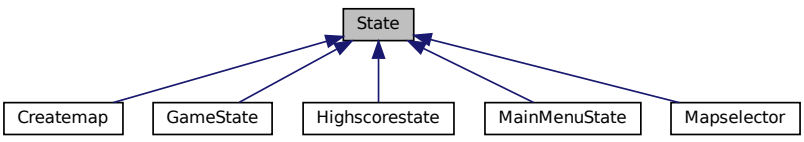
- src/soldier.hpp
- src/soldier.cpp

4.17 State Class Reference

State header.

```
#include <state.hpp>
```

Inheritance diagram for State:



Public Member Functions

- **State** (sf::RenderWindow *window, std::map< std::string, int > *supportedKeys, std::string root_filepath, std::stack< State * > *states)

  Constructor and destructor.
- virtual void **checkForQuit** ()

  Functions.
- const bool & **getQuit** () const
- virtual void endState ()=0
- virtual void updateInput (const float &dt)=0
- virtual void update (const float &dt)=0
- virtual void render (sf::RenderTarget *target=nullptr)=0
- virtual void **updateMousePosition** ()

Public Attributes

- bool **quit_**

Protected Member Functions

- virtual void initKeyBinds ()=0

  Functions.

Protected Attributes

- std::vector< sf::Texture > **textures**

  Container for textures.
- sf::RenderWindow * **window_**

  Pointer to a window.
- std::map< std::string, int > * **supportedKeys_**

  Pointer to supported keys.
- std::stack< State * > * **states**

  Pointer to a stack filled with state pointers (check game.hpp private)
- std::map< std::string, int > **keybinds_**
- std::string **root_filepath_**
- sf::Vector2i **mousePosScreen**

  Mouse position variavles.
- sf::Vector2i **mousePosWindow**
- sf::Vector2f **mousePosView**

4.17.1 Detailed Description

State header.

Include headers

4.17.2 Member Function Documentation

4.17.2.1 endState()

```
virtual void State::endState ( )  [pure virtual]
```

Implemented in Createmap, GameState, and Highscorestate.

4.17.2.2 initKeyBinds()

```
virtual void State::initKeyBinds ( )  [protected], [pure virtual]
```

Functions.

Implemented in Createmap, GameState, Highscorestate, MainMenuState, and Mapselector.

4.17.2.3 render()

```
virtual void State::render (
            sf::RenderTarget * target = nullptr )  [pure virtual]
```

Implemented in Createmap, GameState, and Highscorestate.

4.17.2.4 update()

```
virtual void State::update (
            const float & dt )  [pure virtual]
```

Implemented in Createmap, GameState, and Highscorestate.

4.17.2.5 updateInput()

```
virtual void State::updateInput (
            const float & dt )  [pure virtual]
```

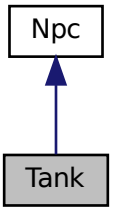
Implemented in Createmap, GameState, Highscorestate, and Mapselector.

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

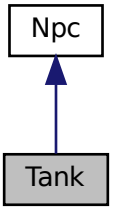
- src/state.hpp
- src/state.cpp

4.18 Tank Class Reference

Inheritance diagram for Tank:



Collaboration diagram for Tank:



Public Member Functions

- **Tank** (std::string root_filepath, sf::Vector2f sLocation, sf::Vector2f eLocation, std::vector< GameTile * > *rTiles)

  Constructor/Destructors.
- virtual void Update (const float &dt)

  Functions.
- virtual void Render (sf::RenderTarget *target)
- virtual void MoveTo (const float &dt)
- virtual void initNpc ()
- std::pair< int, int > FindDirection (sf::Vector2f nextLocation)
- virtual void Rotate (int x, int y)
- sf::Vector2f getPosition ()
- int getHitpoints ()
- bool hasReachedEnd ()
- void dealDamage (int damage)
- void slowMovement (float slow)
- int getTileCount ()
- virtual int **getWorth** ()

Protected Attributes

- float **movementSpeed**
- float **movementspeedmemory_**
- int **hitpoints** = 1
- sf::Vector2f **spawnLocation**
- sf::Vector2f **exitLocation**
- sf::Vector2f **nextLocation**
- std::vector< GameTile * > * **roadTiles**
- int **tileCount** = 0
- sf::Sprite **tankTexture**
- sf::Texture **texture_**
- sf::Clock **slowcooldown_**
- bool **end_** = false
- std::string **root_filepath_**
- sf::Sprite **sotilastektuuri**
- sf::Texture **stexture_**
- int **worth_** = 1

4.18.1 Member Function Documentation

4.18.1.1 dealDamage()

```
void Tank::dealDamage (
            int damage )  [virtual]
```

Reimplemented from Npc.

4.18.1.2 FindDirection()

```
std::pair< int, int > Tank::FindDirection (
            sf::Vector2f nextLocation )  [virtual]
```

Reimplemented from Npc.

4.18.1.3 getHitpoints()

```
int Tank::getHitpoints ( )  [virtual]
```

Reimplemented from Npc.

4.18.1.4 getPosition()

```
sf::Vector2f Tank::getPosition ( )  [virtual]
```

Reimplemented from Npc.

4.18.1.5 getTileCount()

```
int Tank::getTileCount ( )  [virtual]
```

Reimplemented from Npc.

4.18.1.6 hasReachedEnd()

```
bool Tank::hasReachedEnd ( )  [virtual]
```

Reimplemented from Npc.

4.18.1.7 initNpc()

```
void Tank::initNpc ( )  [virtual]
```

Reimplemented from Npc.

4.18.1.8 MoveTo()

```
void Tank::MoveTo (
            const float & dt )  [virtual]
```

Reimplemented from Npc.

4.18.1.9 Render()

```
void Tank::Render (
            sf::RenderTarget * target )  [virtual]
```

Reimplemented from Npc.

4.18.1.10 Rotate()

```
void Tank::Rotate (
            int x,
            int y )  [virtual]
```

Reimplemented from Npc.

4.18.1.11 slowMovement()

```
void Tank::slowMovement (
            float slow )  [virtual]
```

Reimplemented from Npc.

4.18.1.12 Update()

```
void Tank::Update (
            const float & dt )  [virtual]
```

Functions.

Reimplemented from Npc.

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

- src/tank.hpp
- src/tank.cpp

4.19 Textbox Class Reference

Header for the button class.

`#include <textbox.hpp>`

Public Member Functions

- Textbox (float x, float y, float width, float height, sf::Font *font, std::string text, sf::Color idleColor, sf::Color hoverColor, sf::Color activeColor, int text_size, bool box, sf::Event *Event)

  Constructor.
- ~**Textbox** ()

  Destructor.
- const bool **isPressed** () const

  Accessor for determining whether a button is pressed.
- std::string **GetText** ()
- void update (const sf::Vector2f mousePos)

  Update button state based on mouse action.
- void render (sf::RenderTarget *target)

  Render button on target (window)

Private Attributes

- sf::Font * **font_**

  Button font.
- sf::RectangleShape **shape_**

  Button (rectangle) shape.
- sf::Text **buttonText_**

  Button text.
- int **text_size_**

  Button font size.
- bool **boxBoolean_**

  Whether to draw box or not.
- sf::Color **idleColor_**

  Button color when idle.
- sf::Color **hoverColor_**

  Button color when hovered over.
- sf::Color **activeColor_**

  Button color when pressed.
- std::string **text_**
- sf::Clock **clock_**
- sf::Event * **Event_**
- short unsigned **buttonState_**

  Button state between idle (0), hover (1) and active (2)

4.19.1 Detailed Description

Header for the button class.

Add independent includers Button with visual parameters to initiate some action

4.19.2 Constructor & Destructor Documentation

4.19.2.1 Textbox()

```
Textbox::Textbox (
            float x,
            float y,
            float width,
            float height,
            sf::Font * font,
            std::string text,
            sf::Color idleColor,
            sf::Color hoverColor,
            sf::Color activeColor,
            int text_size,
            bool box,
            sf::Event * Event )
```

Constructor.

Parameters

x	X-coordinate of the button
y	Y-coordinate of the button
width	Width of the button
height	Height of the button
font	Font of the button
text	Text of the button
idleColor	Color of the button while idle
hoverColor	Color of the button while hovering over
activeColor	Color of the button when pressed
text_size	Text size of the button
box	Whether to include a box around the button

Note

(X,Y)-coordinates begin from the top-left corner (0,0)

4.19.3 Member Function Documentation

4.19.3.1 render()

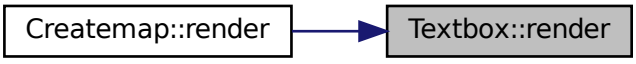
```
void Textbox::render (
            sf::RenderTarget * target )
```

Render button on target (window)

Parameters

target	Rendering target, i.e. game window

Here is the caller graph for this function:



4.19.3.2 update()

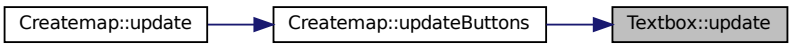
```
void Textbox::update (
            const sf::Vector2f mousePos )
```

Update button state based on mouse action.

Parameters

mousePos	Mouse position on screen

Here is the caller graph for this function:



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

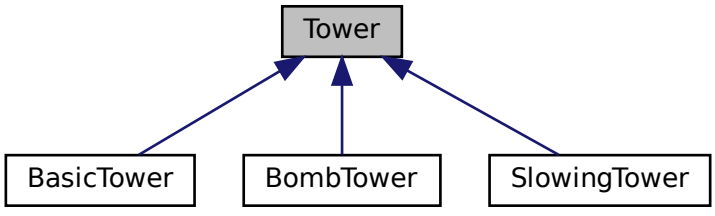
- src/textbox.hpp
- src/textbox.cpp

4.20 Tower Class Reference

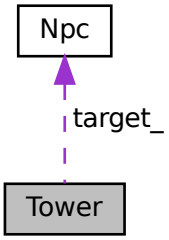
Tower class for all kinds of towers.

```
#include <tower.hpp>
```

Inheritance diagram for Tower:



Collaboration diagram for Tower:



Public Member Functions

- Tower (std::string root_filepath, sf::Vector2f pos, float attack_speed, float damage, float radius, int soundEffectVolumeLevel, float slowing_parameter=0)

  Constructor.
- virtual ~**Tower** ()

  Destructor.
- virtual void render (sf::RenderTarget *target)

  Render tower on target (window)
- virtual void update (const float &dt, std::list< Npc * > enemies, const sf::Vector2f mousePos)

  Update tower with new information.
- void **initSoundEffect** ()

  Initializes shooting sound effect.
- void **setUpSprites** ()

  Loads required textures and sets up sprites.
- void rotateGun (const float &dt, std::list< Npc * > enemies)

  Rotates the gun part of the tower towards first enemy and targets that enemy for the next attack.
- virtual void attack (const float &dt)

  Attacks a target if the tower has one. Dealing damage and possible slowing the enemy.
- virtual void **upgrade** ()
- float **getDamage** ()
- float **getAttackSpeed** ()
- int **getTowerLevel** ()
- sf::Vector2f **getPosition** ()

Protected Attributes

- sf::Sprite **platform_**

  Sprite for the platform part of the tower.
- sf::Sprite **gun_**

  Sprite for the gun part of the tower.
- sf::Texture **sftexture_platform_**

  Texture for the platform part of the tower.
- sf::Texture **sftexture_gun_**

Texture for the gun part of the tower.

- sf::Texture **sftexture_gunfire_**
- std::string **texture_gunfire_**
- sf::Sprite **gunfire_**
- Npc * **target_**

  Target for the next attack.
- sf::Clock **clock_**

  Measures time.
- sf::Vector2f **pos_**

  Position of this tower.
- std::string **root_filepath_**

  Root filepath.
- std::string **texture_platform_**

  Name of the platform texture.
- std::string **texture_gun_**

  Name of the gun texture.
- sf::CircleShape **radius_shape_**

  CircleShape to show the radius of the tower.
- float **attack_speed_**

  Attack speed (seconds)
- float **damage_**

  Damage.
- int **tower_level_** = 1
- float **radius_**

  Attack radius.
- float **slowing_parameter_**

  Slowing parameter.
- short unsigned **towerState_**

  Checks if the mouse is hovering the tower.
- sf::Transform **transform_**
- double **angle_**
- sf::Clock **gunfire_clock_**
- sf::Color **gunfire_color_**
- sf::SoundBuffer **buffer**

  Shoot sound effect.
- sf::Sound **sound**
- int **soundEffectVolumeLevel_**

4.20.1 Detailed Description

Tower class for all kinds of towers.

4.20.2 Constructor & Destructor Documentation

4.20.2.1 Tower()

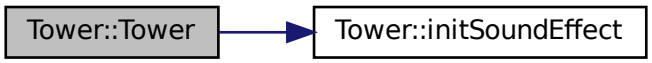
```
Tower::Tower (
            std::string root_filepath,
            sf::Vector2f pos,
            float attack_speed,
            float damage,
            float radius,
            int soundEffectVolumeLevel,
            float slowing_parameter = 0 )
```

Constructor.

Parameters

root_filepath	Root filepath
pos	Position for the tower placement
attack_speed	Attack speed (seconds)
damage	Damage that tower deals per attack to a target
radius	Attack radius
slowing_parameter	Possible parameter if you want this tower to slow the targeted enemy

Here is the call graph for this function:



4.20.3 Member Function Documentation

4.20.3.1 attack()

```
void Tower::attack (
            const float & dt ) [virtual]
```

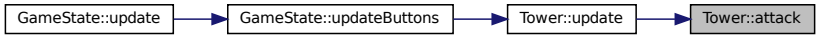
Attacks a target if the tower has one. Dealing damage and possible slowing the enemy.

Parameters

dt	Delta time

Reimplemented in BombTower, and SlowingTower.

Here is the caller graph for this function:



4.20.3.2 render()

```
void Tower::render (
            sf::RenderTarget * target )  [virtual]
```

Render tower on target (window)

Parameters

target	Rendering target, i.e. game window

Reimplemented in BombTower, and SlowingTower.

4.20.3.3 rotateGun()

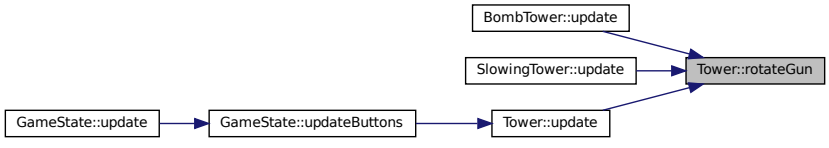
```
void Tower::rotateGun (
            const float & dt,
            std::list< Npc * > enemies )
```

Rotates the gun part of the tower towards first enemy and targets that enemy for the next attack.

Parameters

dt	Delta time
enemies	List of every enemy on the board currently

Here is the caller graph for this function:



4.20.3.4 update()

```
void Tower::update (
            const float & dt,
            std::list< Npc * > enemies,
            const sf::Vector2f mousePos )  [virtual]
```

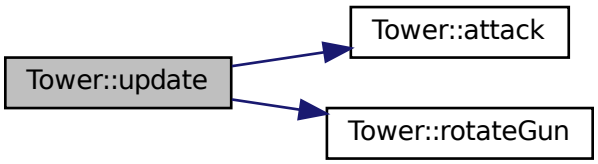
Update tower with new information.

Parameters

dt	Delta time
enemies	List of every enemy on the board currently
mousePos	Mouse position on screen

Reimplemented in BombTower, and SlowingTower.

Here is the call graph for this function:



Here is the caller graph for this function:

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

- src/tower.hpp
- src/tower.cpp

Chapter 5

File Documentation

5.1 basic_tower.hpp

```
1 #ifndef BASIC_TOWER_H
2 #define BASIC_TOWER_H
3
4 #include "tower.hpp"
5 #include "tower.cpp"
6
7 class BasicTower : public Tower {
8 public:
9     BasicTower(std::string root_filepath, sf::Vector2f pos, float attack_speed, int damage, float radius,
       int soundEffectVolumeLevel);
10     virtual ~BasicTower();
11     void upgrade();
12
13 private:
14
15
16 };
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32 #endif
```

5.2 bomb_tower.hpp

```
1 #ifndef BOMB_TOWER_H
2 #define BOMB_TOWER_H
3
4 #include "tower.hpp"
5 #include "missile.hpp"
6 #include "missile.cpp"
7
8 #include <list>
9
10 class BombTower : public Tower {
11 public:
12     BombTower(std::string root_filepath, sf::Vector2f pos, float attack_speed, int damage, float radius,
       int soundEffectVolumeLevel);
13     virtual ~BombTower();
14     void attack(const float& dt);
15     void update(const float& dt, std::list<Npc*> enemies, const sf::Vector2f mousePos);
16     void render(sf::RenderTarget* target);
17     void upgrade();
```

```
18
19 private:
20     std::list<Missile*> missiles_;
21     int soundEffectVolumeLevel_;
22
23 };
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39 #endif
```

5.3 button.hpp

```
1
2
3 #ifndef BUTTON_H
4 #define BUTTON_H
5
7 #include <iostream>
8 #include <ctime>
9 #include <cstdlib>
10 #include <sstream>
11
12 #include <SFML/System.hpp>
13 #include <SFML/Window.hpp>
14 #include <SFML/Graphics.hpp>
15 #include <SFML/Audio.hpp>
16
17 enum button_states{BUTTON_IDLE = 0, BUTTON_HOVER, BUTTON_ACTIVE};
18
19
21 class Button
22 {
23 public:
24     // Constructor and destructor
25
41     Button(float x, float y, float width, float height, sf::Font* font, std::string text,
42      sf::Color idleColor, sf::Color hoverColor, sf::Color activeColor, int text_size, bool box);
43
45     ~Button();
46
47
48     // Accessor
49
51     const bool isPressed() const;
52
53
54     // Functions
55
59     void update(const sf::Vector2f mousePos);
60
64     void render(sf::RenderTarget* target);
65
66     void changeText(std::string text);
67
68
69
70 private:
71     // Variables
72
74     sf::Font* font_;
76     sf::RectangleShape shape_;
78     sf::Text buttonText_;
80     int text_size_;
82     bool boxBoolean_;
83
84
85     // Color variables
86
88     sf::Color idleColor_;
90     sf::Color hoverColor_;
```

```
92      sf::Color activeColor_;
93
94
95      // Button state
96
98      short unsigned buttonState_;
99
100 };
101
102 #endif
```

5.4 createmap_state.hpp

```
1  #ifndef CREATEMAPSTATE_H
2  #define CREATEMAPSTATE_H
3
4  #include "state.hpp"
5  #include "gametile.hpp"
6  #include "gametile.cpp"
7  #include "button.hpp"
8  #include "textbox.hpp"
9  #include "textbox.cpp"
10
11 #include <fstream>
12 #include <iostream>
13 #include <vector>
14 #include <map>
15
17 class Createmap : public State
18 {
19 public:
27      Createmap(sf::RenderWindow* window, std::map<std::string, int>* supportedKeys, std::string
        root_filepath, std::stack<State*>* states, sf::Font* font, sf::Event* Event);
28
30      virtual ~Createmap();
31
33      void initSoundEffect();
34
38      virtual void updateInput(const float& dt);
39
41      virtual void endState();
42
46      virtual void update(const float& dt);
47
51      virtual void render(sf::RenderTarget* target = nullptr);
52
54      void updateButtons();
55
59      void renderButtons(sf::RenderTarget* target = nullptr);
60
62      void initVariables();
63
65      void SaveToFile(std::string mapname);
66
68      void initTiles();
69
76      void ReplacePosition(int x, int y, std::string texturename, bool buildable);
77
79      void initTextures();
80
85      GameTile* TileAt(int x, int y);
86
91      void NextTextureAt(int x, int y);
92
97      void NextSpawnOrExit(int x, int y);
98
104      void ReplaceSpawnOrExit(int x, int y, std::string texturename);
105
106 private:
108      std::map<std::pair<int, int>, GameTile *> Tiles;
109
111      std::map<std::pair<int, int>, GameTile *> SpawnAndExit_;
112
114      std::string root_filepath_;
115
117      sf::Vector2u mousePosGrid;
118
120      std::vector<std::pair<bool, std::string» textures_;
121
123      std::vector<std::pair<bool, std::string» spawnandexittextures_;
124
126      sf::Clock clock;
127
```

```
129     std::map<std::string, Button*> mapcreateButtons_;
130
132     sf::Clock pressTimer_;
133
135     sf::Int32 pressTimerMax_;
136
138     float windowX_;
139
141     float windowY_;
142
144     sf::Font* font_;
145
146     // Click sound effect
147     sf::SoundBuffer buffer;
148     sf::Sound clickSound_;
149
151     void initKeyBinds();
152
154     void initButtons();
155
157     const bool getPressTimer();
158
159     sf::Event* Event_;
160
161     Textbox* textbox_;
162 };
163
164 #endif
```

5.5 game.hpp

```
1
2
3 #ifndef GAME_H
4 #define GAME_H
5
7 #include "main_menu_state.hpp"
8 #include "main_menu_state.cpp"
9
10 class Game
11 {
12 public:
14     Game(std::string& root_filepath);
15     virtual ~Game();
16
18     void endApplication();
19
21     void updateDt();
22     void updateSFMLEvents();
23     void update();
24
26     void render();
27
29     void run();
30
31 private:
33     void initVariables();
34     void initWindow();
35     void initStates();
36     void initKeys();
37
39     std::string root_filepath_;
40     sf::RenderWindow *window;
41     sf::Event sfEvent;
42     std::vector<sf::VideoMode> videoModes_;
43     sf::ContextSettings windowSettings_;
44     bool fullscreen_;
45
47     float dt;
48     sf::Clock dtClock;
49     sf::Clock quitclock_;
50
52     std::stack<State*> states;
53
55     std::map<std::string, int> supportedKeys;
56 };
57
58 #endif
```

5.6 gametile.hpp

```
1 #ifndef GAMETILE_H
2 #define GAMETILE_H
3
4 #include "state.hpp"
5
7 class GameTile
8 {
9 public:
17      GameTile(std::string root_filepath, std::string texturename, sf::Vector2f pos, bool buildable, int
        squaresize);
18
20      ~GameTile();
21
23      void setUpSprite();
24
26      std::string CurrentTexture();
27
29      sf::Vector2f getCenterPosition();
30
32      sf::Vector2f getPosition();
33
35      int getSquareSize();
36
38      bool isExit();
39
41      bool isSpawn();
42
44      bool isRoad();
45
47      bool isBuildable();
48
52      void render(sf::RenderTarget* target);
53
54 protected:
56      sf::Vector2f pos_;
57
59      sf::Texture texture_;
60
62      sf::Sprite sprite_;
63
65      std::string root_filepath_;
66
68      std::string texturename_;
69
71      bool buildable_;
72
74      int squaresize_;
75
76 };
77
78 #endif
```

5.7 gaming_state.hpp

```
1 #ifndef GAMESTATE_H
2 #define GAMESTATE_H
3
4 #include "state.hpp"
5 #include "state.cpp"
6 #include "gametile.hpp"
7 #include <algorithm>
8 #include "npc.hpp"
9 #include "basic_tower.hpp"
10 #include "basic_tower.cpp"
11 #include "bomb_tower.hpp"
12 #include "bomb_tower.cpp"
13 #include "slowing_tower.hpp"
14 #include "slowing_tower.cpp"
15 #include <list>
16 #include "button.hpp"
17 #include <iomanip>
18 #include <sstream>
19
20 class GameState : public State
21 {
22 public:
23
32      GameState(sf::RenderWindow* window, std::map<std::string, int>* supportedKeys, std::string
        root_filepath, std::stack<State*>* states, std::string mapfile, sf::Font *font, int volumeLevel, int
        soundEffectVolumeLevel);
33
```

```
35      virtual ~GameState();
36
40      virtual void updateInput(const float& dt);
41
43      virtual void endState();
44
48      virtual void update(const float& dt);
49
53      virtual void render(sf::RenderTarget* target = nullptr);
54
56      void loadmap();
57
59      void initSpawnLocation();
60
62      void initExitLocation();
63
65      void initRoad();
66
68      void initMusic();
69
71      Tower* towerAt(int x, int y);
72
74      void buildRoad(std::pair<int, int> current, std::pair<int, int> previous);
75
77      GameTile* TileAt(int x, int y);
78
80      std::vector<GameTile*> getNeighbours(GameTile* current);
81
83      void changeMoney(int money);
84
86      void changeHealth(int health);
87
89      void changeScore(int score);
90
92      bool lostGame();
93
95      sf::Vector2f getSpawn();
96
98      sf::Vector2f getExit();
99
101      std::vector<GameTile*> getRoad();
102
104      void messageForPlayer(std::string message);
105
107      void updateButtons();
108
110      void initVariables();
111
113      const bool getPressTimer();
114
116      void renderSoonToBeTower(sf::RenderTarget* target);
117
119      void initGhostTowers();
120
122      void buyTower();
123
125      void updateTargetedTowerInfo();
126
128      void updateGameLogic();
129
131      void spawnEnemies();
132
133
134  private:
135
137      std::string root_filepath_;
138
140      std::string mapfile_;
141
143      std::map<std::pair<int, int>, GameTile *> Tiles;
144
146      std::map<std::pair<int, int>, GameTile *> SpawnAndExit_;
147
149      sf::Vector2f spawnlocation_;
150
152      sf::Vector2f exitlocation_;
153
155      sf::Text money_;
156
158      sf::Text health_;
159
161      sf::Text score_;
162
164      int int_money_ = 0;
165
167      int int_health_ = 0;
168
```

```
170     int int_score_ = 0;
171
173     GameTile* spawnTile_;
174
176     GameTile* exitTile_;
177
179     sf::Font *font_;
180
182     std::vector<GameTile*> roadTiles_;
183
185     sf::RectangleShape rect_menu_;
186
188     sf::Vector2u mousePosGrid;
189
191     sf::Clock clock;
192
194     std::map<std::pair<int, int>, Tower *> towers_;
195
197     std::list<Npc*> enemies_;
198
200     void initKeyBinds();
201
203     sf::Text message_;
204
206     sf::Clock message_timer_;
207
208     std::map<std::string, Button*> towerbuttons_;
209
210     std::map<std::string, Button*> targeted_towerbuttons_;
211
212     bool basic_tower_flag_ = false;
213     bool bomb_tower_flag_ = false;
214     bool slowing_tower_flag_ = false;
215
216     sf::Clock pressTimer_;
217     sf::Int32 pressTimerMax_;
218     sf::Clock timer_;
219     sf::Int32 timerMax_;
220     float windowX_;
221     float windowY_;
222
223     sf::Sprite basic_ghost_;
224     sf::Sprite bomb_ghost_;
225     sf::Sprite slowing_ghost_;
226
227     sf::Texture basic_texture_ghost_;
228     sf::Texture bomb_texture_ghost_;
229     sf::Texture slowing_texture_ghost_;
230
231     sf::Clock shoptimer_;
232
233     Tower* target_ = nullptr;
234
235     int upgradeprice_;
236     int sellprice_;
237
238     sf::Text damage_text_;
239     sf::Text attack_speed_text_;
240     sf::Text tower_level_text;
241
242     unsigned int currentRound_ = 0;
243     int enemiestobespawnedremaining_ = 0;
244
245     sf::Clock roundtimer_;
246     sf::Text roundtimer_text_;
247     int roundtimer_int_;
248     bool roundover_flag_;
249     std::map<std::string, Button*> roundoverbuttons_;
250     sf::Clock spawntimer_;
251     sf::Text currentwave_;
252
253     sf::Text enemiesleft_;
254     sf::Text you_have_lost_;
255     bool game_over_ = false;
256
258     sf::Music gamingStateMusic_;
259     int volumeLevel_;
260     int soundEffectVolumeLevel_;
261 };
262
263
264 #endif
```

5.8 highscore_state.hpp

```
1 #ifndef HIGHSCORE_H
2 #define HIGHSCORE_H
3
4 #include "state.hpp"
5 #include "button.hpp"
6
7 #include <iostream>
8 #include <fstream>
9 #include <list>
10 #include <utility>
11 #include <map>
12
14 class Highscorestate : public State
15 {
16 public:
24     Highscorestate(sf::RenderWindow* window, std::map<std::string, int>* supportedKeys, std::string
      root_filepath, std::stack<State*>* states, sf::Font* font);
25
27     virtual ~Highscorestate();
28
30     void initSoundEffect();
31
35     virtual void updateInput(const float& dt);
36
38     virtual void endState();
39
43     virtual void update(const float& dt);
44
48     virtual void render(sf::RenderTarget* target = nullptr);
49
51     void updateButtons();
52
56     void renderButtons(sf::RenderTarget* target = nullptr);
57
62     static void Addhighscore(const std::string name, int highscore);
63
65     void Readhighscores();
66
67 private:
69     std::string root_filepath_;
70
72     std::list<std::pair<int, std::string» highscores_;
73
75     sf::RectangleShape background_;
76
78     sf::Font* font_;
79
81     std::vector<sf::Text> Texts_;
82
84     std::map<std::string, Button*> highscroeButtons_;
85
87     sf::Clock pressTimer_;
88
90     sf::Int32 pressTimerMax_;
91
92     // Click sound effect
93     sf::SoundBuffer buffer;
94     sf::Sound clickSound_;
95
97     float windowX_;
98
100     float windowY_;
101
103     void initVariables();
104
106     void initKeyBinds();
107
109     void initHighscores();
110
112     void initButtons();
113
115     const bool getPressTimer();
116
117
118 };
119
120 #endif
```

5.9 main_menu_state.hpp

```
1 #ifndef MAINMENUSTATE_H
```

```
2 #define MAINMENUSTATE_H
3
4 #include "gaming_state.hpp"
5 #include "gaming_state.cpp"
6 #include "button.hpp"
7 #include "button.cpp"
8 #include "highscore_state.hpp"
9 #include "highscore_state.cpp"
10 #include "createmap_state.hpp"
11 #include "createmap_state.cpp"
12 #include "mapselector_state.hpp"
13 #include "mapselector_state.cpp"
14
15 class MainMenuState : public State
16 {
17 public:
19     MainMenuState(sf::RenderWindow* window, std::map<std::string, int>* supportedKeys, std::string
      root_filepath, std::stack<State*>* states, sf::Event *Event);
20     virtual ~MainMenuState();
21
23     int getSFXVolume();
24     virtual void updateInput(const float& dt);
25     virtual void endState();
26     virtual void update(const float& dt);
27     virtual void render(sf::RenderTarget* target = nullptr);
28     void updateButtons();
29     void renderButtons(sf::RenderTarget* target = nullptr);
30
32     void updateAnimation(const float& dt);
33
34
35
36 private:
38     sf::RenderWindow* window_;
39     std::string root_filepath_;
40
42     sf::Texture backgroundTexture_;
43     sf::RectangleShape background_;
44     sf::RectangleShape backgroundText_;
45     sf::RectangleShape backgroundShader_;
46     sf::Text backgroundTextUsingFont_;
47
48
49     sf::Font font_;
50     sf::Music backgroundMusic_;
51
52     std::map<std::string, Button*> buttons_;
53     float windowX_;
54     float windowY_;
55
56     sf::Event *Event_;
57
59     sf::Texture rocketTexture_;
60     sf::Texture rocketExhaustTX_;
61     sf::Sprite rocket_;
62     sf::Sprite rocketExhaust_;
63     bool passState_;
64     float rocketMoveSpeed_;
65     sf::Clock timer_;
66     sf::Int32 timerMax_;
67
69     sf::Clock pressTimer_;
70     sf::Int32 pressTimerMax_;
71
73     int volumeLevel_;
74     int soundEffectVolumeLevel_;
75     sf::Text volumeLevelText_;
76     sf::Text soundEffectvolumeLevelText_;
77     sf::SoundBuffer buffer;
78     sf::Sound clickSound_;
79
81     void initVariables();
82     void initBackground();
83     void initKeyBinds();
84     void initFonts();
85     void initButtons();
86     void initMusic();
87     void initBGAnimation(); // For background animation
88
90     const bool getTimer();
91
93     const bool getPressTimer();
94 };
95
96 #endif
```

5.10 mapselector_state.hpp

```
1
2
3 #ifndef MAPSELECTOR_H
4 #define MAPSELECTOR_H
5
6 #include "state.hpp"
7 #include "button.hpp"
8 #include <filesystem>
9
10 class Mapselector : public State
11 {
12 public:
13     Mapselector(sf::RenderWindow* window, std::map<std::string, int>* supportedKeys, std::string
       root_filepath, std::stack<State*>* states, sf::Font *font, sf::Event* Event, int volumeLevel, int
       soundEffectVolumeLevel);
14     virtual ~Mapselector();
15
17     virtual void updateInput(const float& dt);
18     virtual void endState();
19     virtual void update(const float& dt);
20     virtual void render(sf::RenderTarget* target = nullptr);
21     void openFolder();
22     void initVariables();
23     void initSoundEffect();
24     const bool getPressTimer();
25
26
27 private:
28     std::string root_filepath_;
29     sf::Font *font_;
30     float windowX_;
31     float windowY_;
32     sf::RectangleShape backgroundColor_;
33     int scrollUpperBoundary_;
34     int scrollLowerBoundary_;
35     int volumeLevel_;
36     int soundEffectVolumeLevel_;
37
38     // Click sound effect
39     sf::SoundBuffer buffer;
40     sf::Sound clickSound_;
41
43     sf::Clock pressTimer_;
44     sf::Int32 pressTimerMax_;
45     sf::Clock timer_;
46     sf::Int32 timerMax_;
47
48     std::map<std::string, Button*> buttons_;
49     sf::Event *Event_;
50     sf::Clock clock_;
51
53     void initKeyBinds();
54 };
55
56
57 #endif
```

5.11 missile.hpp

```
1 #ifndef MISSILE_H
2 #define MISSILE_H
3
4 #include <SFML/System.hpp>
5 #include <SFML/Window.hpp>
6 #include <SFML/Graphics.hpp>
7 #include <SFML/Audio.hpp>
8 #include <cmath>
9
10 class Missile {
11 public:
12     Missile(std::string root_filepath, sf::Vector2f spawn, sf::Vector2f target, int damage, int speed,
       int soundEffectVolumeLevel);
13     ~Missile();
14
15     void move(const float& dt);
16     void setUpSprites();
17     void render(sf::RenderTarget* target);
18     bool hasExploded();
19     void dealDamage(std::list<Npc*> enemies);
20     void update(std::list<Npc*> enemies);
21
```

```
23      void initSoundEffect();
24
25      sf::Clock clock_;
26
27
28
29  private:
30
31      std::string missile_texture_name_;
32      std::string explosion_texture_name_;
33
34      sf::Texture missile_texture_;
35      sf::Texture explosion_texture_;
36
37      sf::Sprite missile_sprite_;
38      sf::Sprite explosion_sprite_;
39
40      sf::Vector2f spawn_;
41      sf::Vector2f target_;
42      std::list<Npc*> enemies_;
43
44      int damage_;
45      int blast_radius_;
46      int speed_;
47
48      sf::Vector2f direction_;
49      bool explosion_ = false;
50
51      std::string root_filepath_;
52
54      sf::SoundBuffer buffer;
55      sf::Sound sound;
56      int soundEffectVolumeLevel_;
57
58  };
59
60
61  #endif
```

5.12 npc.hpp

```
1  #ifndef NPC_H
2  #define NPC_H
3
4  #include <iostream>
5  #include <ctime>
6  #include <cstdlib>
7  #include <fstream>
8  #include <sstream>
9  #include <stack>
10 #include <map>
11 #include <vector>
12 #include "gametile.hpp"
13
14 #include <SFML/System.hpp>
15 #include <SFML/Window.hpp>
16 #include <SFML/Graphics.hpp>
17 #include <SFML/Audio.hpp>
18
19 class Npc
20 {
21 public:
23      Npc(std::string root_filepath,sf::Vector2f sLocation,sf::Vector2f eLocation,std::vector<GameTile*>*
        rTiles, int newTileCount = 0);
24      ~Npc();
25
27      virtual void Update(const float& dt);
28      virtual void Render(sf::RenderTarget* target);
29      virtual void MoveTo(const float& dt);
30      virtual void initNpc();
31      virtual std::pair<int,int> FindDirection(sf::Vector2f nextLocation);
32      virtual void Rotate(int x,int y);
33      virtual sf::Vector2f getPosition();
34      virtual int getHitpoints();
35      virtual bool hasReachedEnd();
36      virtual void dealDamage(int damage);
37      virtual void slowMovement(float slow);
38      virtual int getTileCount();
39      virtual int getWorth();
40
41 protected:
42
43      float movementSpeed;
```

```
44      float movementspeedmemory_;
45      int hitpoints = 1;
46      sf::Vector2f spawnLocation;
47      sf::Vector2f exitLocation;
48      sf::Vector2f nextLocation;
49      std::vector<GameTile*>* roadTiles;
50      int tileCount = 0;
51      sf::Sprite sotilastektuuri;
52      sf::Texture stexture_;
53      int worth_ = 1;
54
55      sf::Clock slowcooldown_;
56      bool end_ = false;
57
58      std::string root_filepath_;
59  private:
60
61  };
62
63  #endif
```

5.13 plane.hpp

```
1  #ifndef PLANE_H
2  #define PLANE_H
3
4  #include <iostream>
5  #include <ctime>
6  #include <cstdlib>
7  #include <fstream>
8  #include <sstream>
9  #include <stack>
10 #include <map>
11 #include <vector>
12 #include "gametile.hpp"
13 #include "npc.hpp"
14
15 #include <SFML/System.hpp>
16 #include <SFML/Window.hpp>
17 #include <SFML/Graphics.hpp>
18 #include <SFML/Audio.hpp>
19
20 class Plane : public Npc
21 {
22 public:
24      Plane(std::string root_filepath,sf::Vector2f sLocation,sf::Vector2f eLocation,std::vector<GameTile*>*
        rTiles);
25      ~Plane();
26
28      virtual void Update(const float& dt);
29      virtual void Render(sf::RenderTarget* target);
30      virtual void MoveTo(const float& dt);
31      virtual void initNpc();
32      std::pair<int,int> FindDirection(sf::Vector2f nextLocation);
33      virtual void Rotate(int x,int y);
34      sf::Vector2f getPosition();
35      int getHitpoints();
36      bool hasReachedEnd();
37      void dealDamage(int damage);
38      void slowMovement(float slow);
39
40 protected:
41
42      float movementSpeed;
43      float movementspeedmemory_;
44      int hitpoints = 1;
45      sf::Vector2f spawnLocation;
46      sf::Vector2f exitLocation;
47      sf::Vector2f nextLocation;
48      std::vector<GameTile*>* roadTiles;
49      int tileCount = 0;
50      sf::Sprite planeTexture;
51      sf::Texture texture_;
52
53      sf::Clock slowcooldown_;
54      bool end_ = false;
55
56      std::string root_filepath_;
57  private:
58
59  };
60
61  #endif
```

5.14 slimeball.hpp

```
1 #ifndef SLIMEBALL_H
2 #define SLIMEBALL_H
3
4 #include <SFML/System.hpp>
5 #include <SFML/Window.hpp>
6 #include <SFML/Graphics.hpp>
7 #include <SFML/Audio.hpp>
8 #include <cmath>
9
10 class Slimeball {
11 public:
12     Slimeball(std::string root_filepath, sf::Vector2f spawn, sf::Vector2f target, int damage, int speed,
      int slowing_parameter);
13     ~Slimeball();
14
15     void move(const float& dt);
16     void setUpSprites();
17     void render(sf::RenderTarget* target);
18     bool hasExploded();
19     void dealDamage(std::list<Npc*> enemies);
20     void update(std::list<Npc*> enemies);
21
22     sf::Clock clock_;
23
24
25
26 private:
27
28     std::string missile_texture_name_;
29     std::string explosion_texture_name_;
30
31     sf::Texture missile_texture_;
32     sf::Texture explosion_texture_;
33
34     sf::Sprite missile_sprite_;
35     sf::Sprite explosion_sprite_;
36
37     sf::Vector2f spawn_;
38     sf::Vector2f target_;
39     std::list<Npc*> enemies_;
40
41     int damage_;
42     int blast_radius_;
43     int speed_;
44     int slowing_parameter_;
45
46     sf::Vector2f direction_;
47     bool explosion_ = false;
48
49     std::string root_filepath_;
50
51
52 };
53
54
55 #endif
```

5.15 slowing_tower.hpp

```
1 #ifndef SLOWING_TOWER_H
2 #define SLOWING_TOWER_H
3
4 #include "tower.hpp"
5 #include "slimeball.hpp"
6 #include "slimeball.cpp"
7
8
9 class SlowingTower : public Tower {
10 public:
11     SlowingTower(std::string root_filepath, sf::Vector2f pos, float attack_speed, int damage, float
      radius, int soundEffectVolumeLevel);
12     virtual ~SlowingTower();
13     void attack(const float& dt);
14     void update(const float& dt, std::list<Npc*> enemies, const sf::Vector2f mousePos);
15     void render(sf::RenderTarget* target);
16     void upgrade();
17
19     void initSoundEffect();
20
21 private:
22     std::list<Slimeball*> missiles_;
```

```
23
25      sf::SoundBuffer buffer;
26      sf::Sound sound;
27      int soundEffectVolumeLevel_;
28
29 };
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45 #endif
```

5.16 soldier.hpp

```
1 #ifndef SOLDIER_H
2 #define SOLDIER_H
3
4 #include <iostream>
5 #include <ctime>
6 #include <cstdlib>
7 #include <fstream>
8 #include <sstream>
9 #include <stack>
10 #include <map>
11 #include <vector>
12 #include "gametile.hpp"
13 #include "npc.hpp"
14
15 #include <SFML/System.hpp>
16 #include <SFML/Window.hpp>
17 #include <SFML/Graphics.hpp>
18 #include <SFML/Audio.hpp>
19
20 class Soldier : public Npc
21 {
22 public:
24      Soldier(std::string root_filepath,sf::Vector2f sLocation,sf::Vector2f
        eLocation,std::vector<GameTile*>* rTiles, int newTileCount = 0);
25      ~Soldier();
26
28      virtual void Update(const float& dt);
29      virtual void Render(sf::RenderTarget* target);
30      virtual void MoveTo(const float& dt);
31      virtual void initNpc();
32      std::pair<int,int> FindDirection(sf::Vector2f nextLocation);
33      virtual void Rotate(int x,int y);
34      sf::Vector2f getPosition();
35      int getHitpoints();
36      bool hasReachedEnd();
37      void dealDamage(int damage);
38      void slowMovement(float slow);
39
40 protected:
41
42      float movementSpeed;
43      float movementspeedmemory_;
44      int hitpoints = 1;
45      sf::Vector2f spawnLocation;
46      sf::Vector2f exitLocation;
47      sf::Vector2f nextLocation;
48      std::vector<GameTile*>* roadTiles;
49      int tileCount = 0;
50      sf::Sprite soldierTexture;
51      sf::Texture texture_;
52
53      sf::Clock slowcooldown_;
54      bool end_ = false;
55
56      std::string root_filepath_;
57 private:
58
59 };
```

```
60
61 #endif
```

5.17 state.hpp

```
1
2 #ifndef STATE_H
3 #define STATE_H
4
6 #include <iostream>
7 #include <ctime>
8 #include <cstdlib>
9 #include <fstream>
10 #include <sstream>
11 #include <stack>
12 #include <map>
13 #include <vector>
14
15 #include <SFML/System.hpp>
16 #include <SFML/Window.hpp>
17 #include <SFML/Graphics.hpp>
18 #include <SFML/Audio.hpp>
19
20
21 class State
22 {
23 public:
25     State(sf::RenderWindow* window, std::map<std::string, int>* supportedKeys, std::string root_filepath,
        std::stack<State*>* states);
26     virtual ~State();
27
29     virtual void checkForQuit();
30     const bool& getQuit() const;
31     virtual void endState() = 0;
32     virtual void updateInput(const float& dt) = 0;
33     virtual void update(const float& dt) = 0;
34     virtual void render(sf::RenderTarget* target = nullptr) = 0;
35     virtual void updateMousePosition();
36     bool quit_;
37
38
39
40 protected:
42     std::vector<sf::Texture> textures;
44     sf::RenderWindow* window_;
46     std::map<std::string, int>* supportedKeys_;
47
49     std::stack<State*>* states;
50
51     std::map<std::string, int> keybinds_;
52     std::string root_filepath_;
53
55     sf::Vector2i mousePosScreen;
56     sf::Vector2i mousePosWindow;
57     sf::Vector2f mousePosView;
58
60     virtual void initKeyBinds() = 0;
61
62
63
64
65 private:
66 };
67
68 #endif
```

5.18 tank.hpp

```
1 #ifndef TANK_H
2 #define TANK_H
3
4 #include <iostream>
5 #include <ctime>
6 #include <cstdlib>
7 #include <fstream>
8 #include <sstream>
9 #include <stack>
10 #include <map>
11 #include <vector>
```

```
12 #include "gametile.hpp"
13 #include "npc.hpp"
14
15 #include <SFML/System.hpp>
16 #include <SFML/Window.hpp>
17 #include <SFML/Graphics.hpp>
18 #include <SFML/Audio.hpp>
19
20 class Tank : public Npc
21 {
22 public:
24     Tank(std::string root_filepath,sf::Vector2f sLocation,sf::Vector2f eLocation,std::vector<GameTile*>*
      rTiles);
25     ~Tank();
26
28     virtual void Update(const float& dt);
29     virtual void Render(sf::RenderTarget* target);
30     virtual void MoveTo(const float& dt);
31     virtual void initNpc();
32     std::pair<int,int> FindDirection(sf::Vector2f nextLocation);
33     virtual void Rotate(int x,int y);
34     sf::Vector2f getPosition();
35     int getHitpoints();
36     bool hasReachedEnd();
37     void dealDamage(int damage);
38     void slowMovement(float slow);
39     int getTileCount();
40
41 protected:
42
43     float movementSpeed;
44     float movementspeedmemory_;
45     int hitpoints = 1;
46     sf::Vector2f spawnLocation;
47     sf::Vector2f exitLocation;
48     sf::Vector2f nextLocation;
49     std::vector<GameTile*>* roadTiles;
50     int tileCount = 0;
51     sf::Sprite tankTexture;
52     sf::Texture texture_;
53
54     sf::Clock slowcooldown_;
55     bool end_ = false;
56
57     std::string root_filepath_;
58 private:
59
60 };
61
62 #endif
```

5.19 textbox.hpp

```
1
2
3 #ifndef TEXTBOX_H
4 #define TEXTBOX_H
5
7 #include <iostream>
8 #include <ctime>
9 #include <cstdlib>
10 #include <sstream>
11
12 #include <SFML/System.hpp>
13 #include <SFML/Window.hpp>
14 #include <SFML/Graphics.hpp>
15 #include <SFML/Audio.hpp>
16
17
18
19
21 class Textbox
22 {
23 public:
24     // Constructor and destructor
25
41     Textbox(float x, float y, float width, float height, sf::Font* font, std::string text,
42      sf::Color idleColor, sf::Color hoverColor, sf::Color activeColor, int text_size, bool box, sf::Event
      *Event);
43
45     ~Textbox();
46
47
```

```
48      // Accessor
49
51      const bool isPressed() const;
52
53      std::string GetText();
54      // Functions
55
59      void update(const sf::Vector2f mousePos);
60
64      void render(sf::RenderTarget* target);
65
66
67
68  private:
69      // Variables
70
72      sf::Font* font_;
74      sf::RectangleShape shape_;
76      sf::Text buttonText_;
78      int text_size_;
80      bool boxBoolean_;
81
82
83      // Color variables
84
86      sf::Color idleColor_;
88      sf::Color hoverColor_;
90      sf::Color activeColor_;
91
92      std::string text_;
93
94      sf::Clock clock_;
95
96
97      // Button state
98      sf::Event *Event_;
99
101      short unsigned buttonState_;
102
103 };
104
105 #endif
```

5.20 tower.hpp

```
1  #ifndef TOWER_H
2  #define TOWER_H
3
4  #include <SFML/System.hpp>
5  #include <SFML/Window.hpp>
6  #include <SFML/Graphics.hpp>
7  #include <SFML/Audio.hpp>
8  #include <list>
9  #include <cmath>
10
11 enum tower_states{TOWER_IDLE = 0, TOWER_HOVER};
12
14 class Tower {
15 public:
24      Tower(std::string root_filepath, sf::Vector2f pos, float attack_speed, float damage, float radius,
        int soundEffectVolumeLevel, float slowing_parameter = 0);
25
27      virtual ~Tower();
28
32      virtual void render(sf::RenderTarget* target);
33
39      virtual void update(const float& dt, std::list<Npc*> enemies, const sf::Vector2f mousePos);
40
42      void initSoundEffect();
43
45      void setUpSprites();
46
51      void rotateGun(const float& dt, std::list<Npc*> enemies);
52
56      virtual void attack(const float& dt);
57
58      virtual void upgrade();
59
60      float getDamage();
61
62      float getAttackSpeed();
63
64      int getTowerLevel();
```

```
65
66      sf::Vector2f getPosition();
67
68  protected:
69
71      sf::Sprite platform_;
72
74      sf::Sprite gun_;
75
77      sf::Texture sftexture_platform_;
78
80      sf::Texture sftexture_gun_;
81
82      sf::Texture sftexture_gunfire_;
83
84      std::string texture_gunfire_;
85
86      sf::Sprite gunfire_;
87
89      Npc* target_;
90
92      sf::Clock clock_;
93
95      sf::Vector2f pos_;
96
98      std::string root_filepath_;
99
101      std::string texture_platform_;
102
104      std::string texture_gun_;
105
107      sf::CircleShape radius_shape_;
108
110      float attack_speed_;
111
113      float damage_;
114
115      int tower_level_ = 1;
116
118      float radius_;
119
121      float slowing_parameter_;
122
124      short unsigned towerState_;
125
126      sf::Transform transform_;
127      double angle_;
128
129      sf::Clock gunfire_clock_;
130      sf::Color gunfire_color_;
131
133      sf::SoundBuffer buffer;
134      sf::Sound sound;
135      int soundEffectVolumeLevel_;
136
137
138  };
139
140
141  #endif
```

Index