SmallBluryPeople

Generated by Doxygen 1.8.9.1

Sun May 14 2017 18:47:00

Contents

Chapter 1

README

Folder for geometry files

Chapter 2

README

Folder for header files.

Chapter 3

README

Folder for compiled object files.

Chapter 4

SmallBlurryPeopleCVA3

CA2 Group project

Welcome to the Small Blury People Game repository!

Information on how to install, run and play the game can be found on the wiki.

Felix Marrington-Reeve

Ben Hawkyard

Quentin Corker-Marin

Rosie Emery

Chapter 5

README

Folder for glsl shaders.

Chapter 6

README

Folder for source code files.

Chapter 7

Namespace Index

7.1 Namespace List

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

Chapter 8

Hierarchical Index

8.1 Class Hierarchy

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

Chapter 9

Class Index

9.1 Class List

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

Chapter 10

File Index

10.1 File List

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

Chapter 11

Namespace Documentation

11.1 gameUtils Namespace Reference

This package contains usefule tools for building the maps for the game.

Variables

- list **__all__** = ["noise", "mapViewer", "helperFunctions"]

11.1.1 Detailed Description

This package contains usefule tools for building the maps for the game.

It is split into 3 modules: noise, mapViewer and helperFunctions.

The noise module contains tools for generating simpex and fractal noise.

The mapVieper module contains tools for virwing the map without the need for running the game

The helperFunctions module contains odd utility functions for creating maps

11.2 helperFunctions Namespace Reference

This module contains functions that I found useful when writing the maps that ship with the game and I thought would be useful to other people writing their own custom maps.

11.2.1 Detailed Description

This module contains functions that I found useful when writing the maps that ship with the game and I thought would be useful to other people writing their own custom maps.

11.3 mapViewer Namespace Reference

implementation of the MapViewer class that can be used to show generated maps as an image without having to run the entire game.

11.3.1 Detailed Description

implementation of the MapViewer class that can be used to show generated maps as an image without having to run the entire game.

Used in early development of maps

11.4 The Namespace Reference

Noise module has classes that implement simplex noise and fractal noise functions.

11.4.1 Detailed Description

Noise module has classes that implement simplex noise and fractal noise functions.

Chapter 12

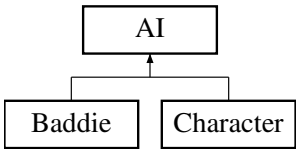
Class Documentation

12.1 AI Class Reference

Parent class for ingame characters and enemies, containing position, states and targets.

`#include <AI.hpp>`

Inheritance diagram for AI:



Public Member Functions

- AI (TerrainHeightTracer ∗_height_tracer, Grid ∗_grid)
  ctor, sets reference to grid and initialised values
- ∼AI ()=default
  destructor
- virtual void idleState ()=0
  idleState, randomly moves AI while not active
- virtual void update ()=0
  update, updates AI based on its current state
- bool move ()
  move, moves AI along its path
- void findPath (ngl::Vec2 _target)
  findPath, pathfinding function to get nodes for pathfinding
- void findPath (int _target_id)
  findPath, pathfinding function to get nodes for pathfinding
- std::vector< ngl::Vec2 > getPath (int _target_id)
  getPath, path finds to a tile and returns the path
- ngl::Vec2 calcAimVec (float ∗dist)
  calcAimVec, calculate vector towards next point
- bool setTarget (ngl::Vec2 _target_pos)
  setTarget, set a new target position based on a position
- bool setTarget (int _tile_id)
  setTarget, set a new target based on the grid tile id

- float getHealth ()

  gethealth, get AI's health
- void takeHealth (float m_amount)

  takeHealth, take health away from a AI
- ngl::Vec3 getPos ()

  getPos, get AI's position
- ngl::Vec2 getPos2D ()

  getPos2D, get 2D position
- void updateRot ()

  updateRot update m_rot, eg if direction has changed
- float getRot ()

  getRot return rotation of AI
- bool isIdle ()

  isIdle, returns whether the AI is idle

Protected Attributes

- Grid * m_grid

  m_grid, grid pointer to reference for pathfinding
- TerrainHeightTracer * m_height_tracer

  m_height_tracer, for grid height
- ngl::Vec2 m_pos

  m_pos, AI's position ///
- float m_offset

  m_offset, offset AI on current tile, to avoid clipping between AIs
- int m_target_id

  m_target_id, id of target tile on grid
- float m_health

  m_health, how much health the character has: 1 = full health, 0 = no health/dead
- float m_rot

  m_rot character's rotation to face current direction (degrees)
- float m_speed

  m_speed, max speed of character
- std::vector< ngl::Vec2 > m_path

  m_path, vector of target positions for movement
- QTime m_action_timer

  m_action_timer, timer for characters actions such as chopping wood and building
- bool m_idle

  m_idle, set when character is idle, for checking if idle

12.1.1 Detailed Description

Parent class for ingame characters and enemies, containing position, states and targets.

Definition at line 19 of file AI.hpp.

12.1.2 Constructor & Destructor Documentation

12.1.2.1 AI::AI (TerrainHeightTracer * *_height_tracer*, Grid * *_grid*)

ctor, sets reference to grid and initialised values

Parameters

in	*_grid,pointer*	to the grid to reference for pathfinding
in	*_height_↩ tracer,pointer*	to TerrainHeightTracer to get the height of the AI at its position

Definition at line 8 of file AI.cpp.

12.1.3 Member Function Documentation

12.1.3.1 ngl::Vec2 AI::calcAimVec (float * *dist*)

calcAimVec, calculate vector towards next point

Returns

vec2 of direction vector character is aiming towards

Definition at line 76 of file AI.cpp.

12.1.3.2 void AI::findPath (ngl::Vec2 *_target*)

findPath, pathfinding function to get nodes for pathfinding

Parameters

in	*_↩ target,coordinate*	to path find to

Definition at line 58 of file AI.cpp.

12.1.3.3 void AI::findPath (int *_target_id*)

findPath, pathfinding function to get nodes for pathfinding

Parameters

in	*_target_id,tile*	to path find to

Definition at line 64 of file AI.cpp.

12.1.3.4 float AI::getHealth ( ) [inline]

gethealth, get AI's health

Returns

m_health, AI's health value

Definition at line 83 of file AI.hpp.

12.1.3.5 std::vector< ngl::Vec2 > AI::getPath (int *_target_id*)

getPath, path finds to a tile and returns the path

Parameters

in	*_target_id,tile*	to path find to

Returns

vector of vec2's that store the path to the target tile

Definition at line 70 of file AI.cpp.

12.1.3.6 ngl::Vec3 AI::getPos ( )

getPos, get AI's position

Returns

m_pos, AI's position

Definition at line 120 of file AI.cpp.

12.1.3.7 ngl::Vec2 AI::getPos2D ( )

getPos2D, get 2D position

Returns

m_pos, AI's position

Definition at line 127 of file AI.cpp.

12.1.3.8 float AI::getRot ( ) [inline]

getRot return rotation of AI

Returns

m_rot, character's rotation

Definition at line 107 of file AI.hpp.

12.1.3.9 bool AI::isIdle ( ) [inline]

isIdle, returns whether the AI is idle

Returns

m_idle, the boolean stored in the AI determining if it is idle or not

Definition at line 112 of file AI.hpp.

12.1.3.10 bool AI::move ( )

move, moves AI along its path

Returns

a boolean determing if its reached its target

Definition at line 22 of file AI.cpp.

12.1.3.11 bool AI::setTarget (ngl::Vec2 *_target_pos*)

setTarget, set a new target position based on a position

Parameters

in	*_target_pos,the*	position to pathfind to

Returns

a boolean determining whether the target is viable

Definition at line 94 of file AI.cpp.

12.1.3.12 bool AI::setTarget (int *_tile_id*)

setTarget, set a new target based on the grid tile id

Parameters

in	*_tile_id,the*	tile id to pathfind to

Returns

a boolean determining whether the target is viable

Definition at line 99 of file AI.cpp.

12.1.3.13 void AI::takeHealth (float *m_amount*) [inline]

takeHealth, take health away from a AI

Parameters

m_↩ amount,amount	of health to take away

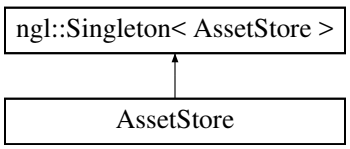
Definition at line 88 of file AI.hpp.

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

- include/AI.hpp
- src/AI.cpp

12.2 AssetStore Class Reference

Inheritance diagram for AssetStore:



Public Member Functions

- ngl::Obj * **getModel** (const std::string &_id)
- GLuint **getTexture** (const std::string &_id)
- void **loadMesh** (const std::string &_id, const std::string &_path)
- void **loadTexture** (const std::string &_id, const std::string &_path)

Friends

- class **Singleton**< **AssetStore** >

12.2.1 Detailed Description

Definition at line 15 of file AssetStore.hpp.

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

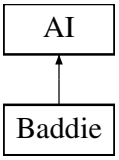
- include/AssetStore.hpp
- src/AssetStore.cpp

12.3 Baddie Class Reference

The Baddie class is the ingame enemy, with attacking and tracking states.

`#include <Baddie.hpp>`

Inheritance diagram for Baddie:



Public Member Functions

- Baddie (ngl::Vec2 _pos, TerrainHeightTracer *_height_tracer, Grid *_grid, std::vector< Character > *_characters)

  Baddie create baddie with a position on the grid and a references to the characters.
- void update ()

  update calculate behaviours for baddie
- void trackingState ()

  trackingState, baddie intiates fight with character and follows it
- void invadedState (int _target)

  invadedState, character intiates fight with baddie
- void stopSearching ()

  setInvade, stop character looking for an enemy
- void fight ()

  fight, baddies actions when fighting
- void idleState ()

  idleState, sets baddie to not be in a fithing state
- bool findNearestTarget ()

  findNearestTarget, finds character nearest to baddie
- float getScale ()

  getScale, returns baddie's scale for drawing
- void addScale (float _scale)

  addScale, adds onto scale of baddie
- float getID ()

  getID, return baddie ID

Additional Inherited Members

12.3.1 Detailed Description

The Baddie class is the ingame enemy, with attacking and tracking states.

Definition at line 22 of file Baddie.hpp.

12.3.2 Constructor & Destructor Documentation

12.3.2.1 Baddie::Baddie (ngl::Vec2 *_pos,* TerrainHeightTracer * *_height_tracer,* Grid * *_grid,* std::vector< Character > * *_characters*)

Baddie create baddie with a position on the grid and a references to the characters.

Parameters

_pos	spawn position
_height_tracer	height tracer to use
_grid	pointer to grid for pathfinding and positions
_characters	reference to character vector in scene

Definition at line 11 of file Baddie.cpp.

12.3.3 Member Function Documentation

12.3.3.1 void Baddie::addScale (float *_scale*) [inline]

addScale, adds onto scale of baddie

Parameters

_scale,amount	to add onto scale

Definition at line 70 of file Baddie.hpp.

12.3.3.2 float Baddie::getID ( ) [inline]

getID, return baddie ID

Returns

m_id, baddies ID

Definition at line 75 of file Baddie.hpp.

12.3.3.3 float Baddie::getScale ( ) [inline]

getScale, returns baddie's scale for drawing

Returns

m_scale, scale of baddie

Definition at line 65 of file Baddie.hpp.

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

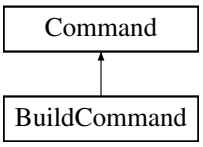
- include/Baddie.hpp
- src/Baddie.cpp

12.4 BuildCommand Class Reference

The BuildCommand class used to tell character to build a given building type.

`#include <Commands.hpp>`

Inheritance diagram for BuildCommand:



Public Member Functions

- BuildCommand (Character *_character, TileType _building)

  BuildCommand constructor for building command.
- virtual void execute ()

  execute tells character to build given building

12.4.1 Detailed Description

The BuildCommand class used to tell character to build a given building type.

Definition at line 159 of file Commands.hpp.

12.4.2 Constructor & Destructor Documentation

12.4.2.1 BuildCommand::BuildCommand (Character * *_character*, TileType *_building*)

BuildCommand constructor for building command.

Parameters

_character	character to instruct
_building	building wanted

Definition at line 31 of file Commands.cpp.

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

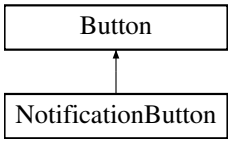
- include/Commands.hpp
- src/Commands.cpp

12.5 Button Class Reference

The Button class defines a button with a position that can be activated if clicked.

`#include <Button.hpp>`

Inheritance diagram for Button:



Public Member Functions

- Button (Action _action, XAlignment _x_align, YAlignment _y_align, ngl::Vec2 _window_res, ngl::Vec2 _offset, ngl::Vec2 _size, const std::string &_text)

  Button default constructor, takes alignment information.
- void updatePos (ngl::Vec2 _window_res)

  updatePos update the button's position based on the offset and screen res
- bool isInside (ngl::Vec2 _pos)

  isInside check if given position is inside the button, used for mouse position
- Action getAction () const

  getAction get command from button
- int getID ()

  getID get the button id
- ngl::Vec2 getPos ()

  getPos get position of button
- ngl::Vec2 getSize ()

  getSize get size of button
- const std::string & getText ()

  getText get button text
- void setText (const std::string &_text)

  setText change the text on a button
- bool isPassive (bool character_selected)

  isPassive whether button is clickable
- void move (ngl::Vec2 _move_vec)

  move move a button by a given amount

Static Public Member Functions

- static void resetIdCounter ()

  resetIdCounter set id counter of buttons to 0, NEED TO WIPE ALL BUTTONS BEFORE DOING THIS

Protected Attributes

- const int m_id

  m_id unique button id
- ngl::Vec2 m_offset

  m_offset offset in x and y from the button's alignment
- ngl::Vec2 m_pos

  m_pos button absolute position
- ngl::Vec2 m_size

  m_size size of button, in x and y
- XAlignment m_x_align

  m_x_align alignment of button left/center/right
- YAlignment m_y_align

m_y_align alignment of button top/center/bottom

- Action m_action

  m_action action that button does upon activation

- std::string m_text

  m_text button text displayed on button, or used for setting buttons to store key

Static Protected Attributes

- static int m_id_counter

  m_id_counter keeps track of number of buttons

12.5.1 Detailed Description

The Button class defines a button with a position that can be activated if clicked.

Definition at line 27 of file Button.hpp.

12.5.2 Constructor & Destructor Documentation

12.5.2.1 Button::Button (Action *_action,* XAlignment *_x_align,* YAlignment *_y_align,* ngl::Vec2 *_window_res,* ngl::Vec2 *_offset,* ngl::Vec2 *_size,* const std::string & *_text*)

Button default constructor, takes alignment information.

Parameters

_action	functionality of the button
_x_align	which side the button is bound to, either left, right or center
_y_align	which level the button is bound to, either up, down or center
_window_res	window resolution for calculating postion from offset
_offset	button offset from given alignments
_size	size of the button

Definition at line 6 of file Button.cpp.

12.5.3 Member Function Documentation

12.5.3.1 int Button::getID ( )

getID get the button id

Returns

id of the button

Definition at line 71 of file Button.cpp.

12.5.3.2 ngl::Vec2 Button::getPos ( )

getPos get position of button

Returns

position of button as vec2

Definition at line 76 of file Button.cpp.

12.5.3.3 ngl::Vec2 Button::getSize ( )

getSize get size of button

Returns

vec2 of x and y size

Definition at line 81 of file Button.cpp.

12.5.3.4 const std::string & Button::getText ( )

getText get button text

Returns

text of button, or "\0"

Definition at line 86 of file Button.cpp.

12.5.3.5 bool Button::isInside (ngl::Vec2 *_pos*)

isInside check if given position is inside the button, used for mouse position

Parameters

_pos	position to check button position and size against

Returns

true for inside, false for outside

Definition at line 53 of file Button.cpp.

12.5.3.6 bool Button::isPassive (bool *character_selected*)

isPassive whether button is clickable

Parameters

character_selected	whether there is currently a selected character

Returns

true if button has no action

Definition at line 96 of file Button.cpp.

12.5.3.7 void Button::move (ngl::Vec2 *_move_vec*)

move move a button by a given amount

Parameters

_move_vec	the vector to move the button by

Definition at line 128 of file Button.cpp.

12.5.3.8 void Button::setText (const std::string & *_text*)

setText change the text on a button

Parameters

_text	string to set the text to

Definition at line 91 of file Button.cpp.

12.5.3.9 void Button::updatePos (ngl::Vec2 *_window_res*)

updatePos update the button's position based on the offset and screen res

Parameters

screen_res	resolution of the window

Definition at line 19 of file Button.cpp.

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

- include/Button.hpp
- src/Button.cpp

12.6 Camera Class Reference

Public Member Functions

- Camera ()
  Default camera constructor.
- ~Camera ()=default
  Default camera destructor.
- void calculateViewMat ()
  Calculates the view matrix from the transformation of the camera about the pivot and the pivot in the world.
- void calculateProjectionMat ()
  Calculate the projection matrix. Unless you are changing the FOV or screen aspect, this shouldn't be called much.
- void updateSmoothCamera ()
  Updates all of the smooth camera values.
- ngl::Mat4 getV () const
  Get view matrix.
- ngl::Mat4 getP () const
  Get projection matrix.
- ngl::Mat4 getVP () const
  Get the view-project matrix (this is calculated in calculateViewMat and calculateProjectionMat.
- ngl::Vec3 getPos () const
  Gets the world space position of the camera. I find this useful for fancy shaders.
- ngl::Vec3 getPivot () const
  Returns the world space position of the pivot.

- float **getAspect** () const
- void setMinPitch (const float _p)

  Setters to control the range of the cameras pitch.
- void **setMaxPitch** (const float _p)
- void setMinDolly (const float _d)

  Setters to control the range of the cameras dollying.
- void **setMaxDolly** (const float _d)
- void setAspect (const float _aspect)

  Set aspect ratio (x / y).
- void setFOV (const float _fov)

  Set horizontal (?) field of view.
- void setInitPos (const ngl::Vec3 &_pos)

  Sets the initial position of the camera, prior to transformation about and of the pivot.
- void setInitPivot (const ngl::Vec3 &_pivot)

  Sets the initial pivot position, it may be transformed later.
- void setUp (const ngl::Vec3 &_up)

  Sets the up vector. [0, 1, 0] is usually pretty appropriate.
- void clearTransforms ()

  Empties the transformation stacks.
- ngl::Vec3 back ()

  Gets the back vector of the camera. Just ignore the transpose stuff, it works I swear.
- ngl::Vec3 forwards ()

  Gets the forwards vector of the camera. Just ignore the transpose stuff, it works I swear.
- ngl::Vec3 up ()

  Gets the up vector of the camera. Just ignore the transpose stuff, it works I swear.
- ngl::Vec3 right ()

  Gets the right vector of the camera. Just ignore the transpose stuff, it works I swear.
- std::array< ngl::Vec3, 8 > calculateCascade (float _start, float _end)

  Returns the 8 corners of a frustum starting at _start units from the camera, and ending _end units away. These vertices are in world space.
- void moveRight (const float _d)

  Moves the camera right, or forwards, relative to its orientation.
- void **moveForward** (const float _d)
- void **move** (const ngl::Vec3 _d)
- void moveScreenSpace (const ngl::Vec3 _d)

  moveScreenSpace moves camera in screenspace: x for left/right, z for forwards/backwards, y for up/down (world space, because it's more intuitive for input)
- void **dolly** (const float _d)
- void **setPos** (const ngl::Vec3 _p)
- ngl::Vec3 **getTargPos** () const
- void rotate (const float _pitch, const float _yaw)

  Rotates the camera.
- float **getTargetDolly** () const
- void **setFocalDepth** (const float _d)
- float **getFocalDepth** () const
- void **immediateTransform** (const ngl::Mat4 &_mat)

12.6.1 Detailed Description

Definition at line 17 of file Camera.hpp.

12.6.2 Member Function Documentation

12.6.2.1 void Camera::moveScreenSpace (const ngl::Vec3 _d)

moveScreenSpace moves camera in screenspace: x for left/right, z for forwards/backwards, y for up/down (world space, because it's more intuitive for input)

Parameters

_d	vector with x, y, z movement components

Definition at line 135 of file Camera.cpp.

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

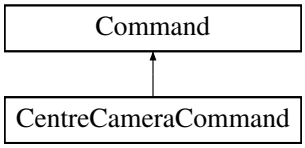
- include/Camera.hpp
- src/Camera.cpp

12.7 CentreCameraCommand Class Reference

The CentreCameraCommand class sends command to scene to reset camera.

`#include <Commands.hpp>`

Inheritance diagram for CentreCameraCommand:



Public Member Functions

- CentreCameraCommand (Scene *_scene)

  CentreCameraCommand constructor for centre camera command.
- virtual void execute ()

  execute send command to scene to centre the camera

12.7.1 Detailed Description

The CentreCameraCommand class sends command to scene to reset camera.

Definition at line 187 of file Commands.hpp.

12.7.2 Constructor & Destructor Documentation

12.7.2.1 CentreCameraCommand::CentreCameraCommand (Scene * *_scene*)

CentreCameraCommand constructor for centre camera command.

Parameters

_scene	scene to send instruction to

Definition at line 45 of file Commands.cpp.

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

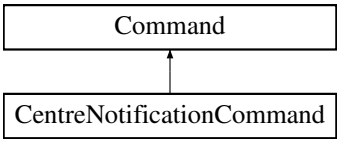
- include/Commands.hpp
- src/Commands.cpp

12.8 CentreNotificationCommand Class Reference

The CentreNotificationCommand class to focus camera on a notification's position.

`#include <Commands.hpp>`

Inheritance diagram for CentreNotificationCommand:



Public Member Functions

- CentreNotificationCommand (Scene *_scene, ngl::Vec2 _map_pos)

  CentreNotificationCommand constructor for center notification command.
- virtual void execute ()

  execute send instruction to move camera

12.8.1 Detailed Description

The CentreNotificationCommand class to focus camera on a notification's position.

Definition at line 425 of file Commands.hpp.

12.8.2 Constructor & Destructor Documentation

12.8.2.1 CentreNotificationCommand::CentreNotificationCommand (Scene * *_scene*, ngl::Vec2 *_map_pos*)

CentreNotificationCommand constructor for center notification command.

Parameters

_scene	current scene being used
_map_pos	position to move camera to

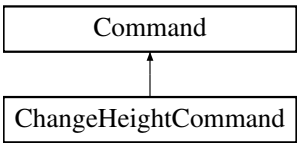
Definition at line 162 of file Commands.cpp.

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

- include/Commands.hpp
- src/Commands.cpp

12.9 ChangeHeightCommand Class Reference

Inheritance diagram for ChangeHeightCommand:



Public Member Functions

- **ChangeHeightCommand** (int _dir)
- virtual void **execute** ()

12.9.1 Detailed Description

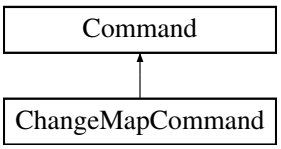
Definition at line 489 of file Commands.hpp.

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

- include/Commands.hpp
- src/Commands.cpp

12.10 ChangeMapCommand Class Reference

Inheritance diagram for ChangeMapCommand:



Public Member Functions

- **ChangeMapCommand** (int _dir)
- virtual void **execute** ()

12.10.1 Detailed Description

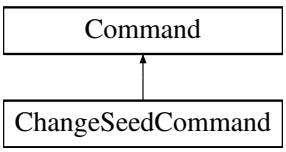
Definition at line 467 of file Commands.hpp.

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

- include/Commands.hpp
- src/Commands.cpp

12.11 ChangeSeedCommand Class Reference

Inheritance diagram for ChangeSeedCommand:



Public Member Functions

- **ChangeSeedCommand** (int _dir)
- virtual void **execute** ()

12.11.1 Detailed Description

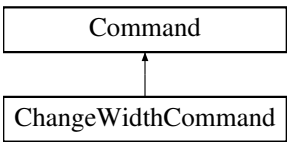
Definition at line 500 of file Commands.hpp.

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

- include/Commands.hpp
- src/Commands.cpp

12.12 ChangeWidthCommand Class Reference

Inheritance diagram for ChangeWidthCommand:



Public Member Functions

- **ChangeWidthCommand** (int _dir)
- virtual void **execute** ()

12.12.1 Detailed Description

Definition at line 478 of file Commands.hpp.

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

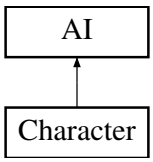
- include/Commands.hpp
- src/Commands.cpp

12.13 Character Class Reference

Information for ingame characters, containing position, states and targets.

`#include <Character.hpp>`

Inheritance diagram for Character:



Public Member Functions

- Character (TerrainHeightTracer *_height_tracer, Grid *_grid, std::string _name, std::vector< Baddie > *_↩ baddies)

  ctor, sets reference to grid and initialised values

- ~Character ()=default

  destructor
- void setState (int _target_id)

  setState, creates state stack for the character to execute
- void setForageState ()

  setForageState, sets character for foraging instead of chopping
- void isBaddie ()

  isBaddie, checks if an empty square has been selected or a enemy
- void buildState (TileType _building)

  buildState, tell character to start building on current square
- void moveState ()

  moveState, character moves to its target
- void attackState ()

  attackState, character intiates fight with enemy
- void invadedState (int _target)

  invadedState, enemy intiates fight with character
- void chopState ()

  chopState, character collects wood from a tree
- void fishState ()

  fishState, character tries to catch a fish
- void forageState ()

  forageState, character finds tree to get berries
- void storeState ()

  storeState, character stores whatever is in its inventory
- void sleepState ()

  sleepState, recovers character stamina and makes them inactive
- void eatBerriesState ()

  eatBerries, character eats berries and restores stamina
- void eatFishState ()

  eatFish, character eats fish and restores stamina
- void idleState ()

  idleState, randomly moves character while not active
- void clearState ()

  clearState, removes any actions in the state stack
- void update ()

  update, updates character based on its current state
- int getID ()

  getID, get the unique character id
- std::string getName ()

  getName, get character's name
- ngl::Vec3 getColour ()

  getColour, get character's colour
- State getState ()

  getState get character's current state
- float getStamina ()

  getStamina get character's stamina
- float getHunger ()

  getHunger get character's hunger
- std::vector< float > getAttributes ()

  getAttributes, gets character attributes such as chopping speed
- void setActive (bool _selection)

setActive, set's whether the character is active

- bool isActive ()

  isActive, returns whether the character is active

- bool isInside ()

  isInside, returns whether the character is inside a house or storehouse

- bool isSleeping ()

  isSleeping, returns whether the character is sleeping

Static Public Member Functions

- static void setWorldInventory (Inventory *_world_inventory)

  setWorldInventory initialize world inventory for character class

- static Inventory * getWorldInventory ()

  getWorldInventory access the world inventory

Additional Inherited Members

12.13.1 Detailed Description

Information for ingame characters, containing position, states and targets.

Definition at line 61 of file Character.hpp.

12.13.2 Constructor & Destructor Documentation

12.13.2.1 Character::Character (TerrainHeightTracer * *_height_tracer,* Grid * *_grid,* std::string *_name,* std::vector< Baddie > * *_baddies*)

ctor, sets reference to grid and initialised values

Parameters

in	*_height_tracer,pointer*	to height tracer, used for getting height at positions
in	*_grid,pointer*	to the grid to reference for pathfinding
in	*_world_inventory,pointer*	to global inventory
in	*_name,name*	for character
in	*_baddies,pointer*	to vector of baddies in the game

Definition at line 18 of file Character.cpp.

12.13.3 Member Function Documentation

12.13.3.1 void Character::buildState (TileType *_building*)

buildState, tell character to start building on current square

Parameters

in	*_building*	type of building to build

Definition at line 160 of file Character.cpp.

12.13.3.2 std::vector< float > Character::getAttributes ( )

getAttributes, gets character attributes such as chopping speed

Returns

a vector of m_chopping_speed, m_building_speed, m_fishing_catch, m_forage_amount, m_attack_power

Definition at line 1286 of file Character.cpp.

12.13.3.3 ngl::Vec3 Character::getColour ( ) [inline]

getColour, get character's colour

Returns

m_colour, character's colour

Definition at line 170 of file Character.hpp.

12.13.3.4 float Character::getHunger ( ) [inline]

getHunger get character's hunger

Returns

character's hunger value 0-1

Definition at line 185 of file Character.hpp.

12.13.3.5 int Character::getID ( ) [inline]

getID, get the unique character id

Returns

m_id, character's id

Definition at line 160 of file Character.hpp.

12.13.3.6 std::string Character::getName ( ) [inline]

getName, get character's name

Returns

m_name, character's name

Definition at line 165 of file Character.hpp.

12.13.3.7 float Character::getStamina ( ) [inline]

getStamina get character's stamina

Returns

character's stamina value 0-1

Definition at line 180 of file Character.hpp.

12.13.3.8 State Character::getState ( )

getState get character's current state

Returns

character's current state

Definition at line 1318 of file Character.cpp.

12.13.3.9 Inventory * Character::getWorldInventory ( ) [static]

getWorldInventory access the world inventory

Returns

pointer to world inventory

Definition at line 75 of file Character.cpp.

12.13.3.10 bool Character::isActive ( ) [inline]

isActive, returns whether the character is active

Returns

m_active, the boolean stored in the character determining if it is active or not

Definition at line 200 of file Character.hpp.

12.13.3.11 bool Character::isInside ( ) [inline]

isInside, returns whether the character is inside a house or storehouse

Returns

boolean OR operation between storing and sleeping so that if it is in a house or storehouse it returns true

Definition at line 205 of file Character.hpp.

12.13.3.12 bool Character::isSleeping ( ) [inline]

isSleeping, returns whether the character is sleeping

Returns

m_sleeping, the boolean stored in the character determining if it is sleeping in a house

Definition at line 210 of file Character.hpp.

12.13.3.13 void Character::setActive (bool *_selection*)

setActive, set's whether the character is active

Parameters

in	*_selection,a*	boolean determing whether the character is active or not

Definition at line 1309 of file Character.cpp.

12.13.3.14 void Character::setWorldInventory (Inventory * *_world_inventory*) [static]

setWorldInventory initialize world inventory for character class

Parameters

_world_inventory	pointer to world inventory

Definition at line 69 of file Character.cpp.

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

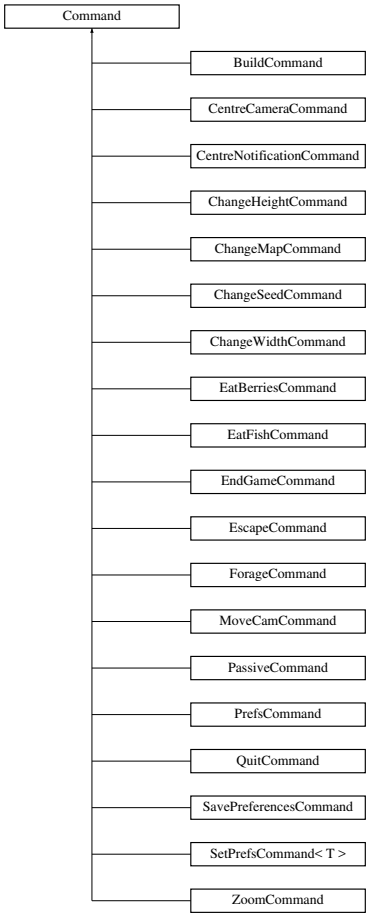
- include/Character.hpp
- src/Character.cpp

12.14 Command Class Reference

The Command class utility class for creating level of indirection between buttons and actions they perform, base class is abstract.

`#include <Commands.hpp>`

Inheritance diagram for Command:



Public Member Functions

- virtual void **execute** ()=0

12.14.1 Detailed Description

The Command class utility class for creating level of indirection between buttons and actions they perform, base class is abstract.

Definition at line 114 of file Commands.hpp.

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

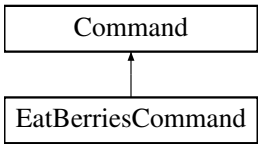
- include/Commands.hpp

12.15 EatBerriesCommand Class Reference

The EatBerriesCommand class used to tell character to find stored berries to eat.

`#include <Commands.hpp>`

Inheritance diagram for EatBerriesCommand:



Public Member Functions

- EatBerriesCommand (Character *_character)

  EatBerriesCommand constructor for eat berries command.
- virtual void execute ()

  execute tells character to find berries to eat

12.15.1 Detailed Description

The EatBerriesCommand class used to tell character to find stored berries to eat.

Definition at line 379 of file Commands.hpp.

12.15.2 Constructor & Destructor Documentation

12.15.2.1 EatBerriesCommand::EatBerriesCommand (Character * *_character*)

EatBerriesCommand constructor for eat berries command.

Parameters

_character	character to instruct

Definition at line 136 of file Commands.cpp.

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

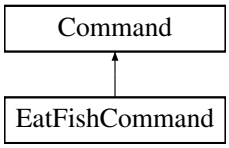
- include/Commands.hpp
- src/Commands.cpp

12.16 EatFishCommand Class Reference

The EatFishCommand class used to tell character to find stored fish to eat.

`#include <Commands.hpp>`

Inheritance diagram for EatFishCommand:



Public Member Functions

- EatFishCommand (Character *_character)

  EatFishCommand constructor for eat fish command.
- virtual void execute ()

  execute tells character to find stored fish to eat

12.16.1 Detailed Description

The EatFishCommand class used to tell character to find stored fish to eat.

Definition at line 402 of file Commands.hpp.

12.16.2 Constructor & Destructor Documentation

12.16.2.1 EatFishCommand::EatFishCommand (Character * *_character*)

EatFishCommand constructor for eat fish command.

Parameters

_character	character to instruct

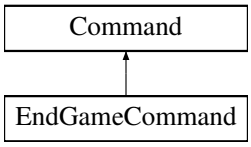
Definition at line 149 of file Commands.cpp.

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

- include/Commands.hpp
- src/Commands.cpp

12.17 EndGameCommand Class Reference

Inheritance diagram for EndGameCommand:



Public Member Functions

- **EndGameCommand** (Scene ∗_scene)
- virtual void **execute** ()

12.17.1 Detailed Description

Definition at line 511 of file Commands.hpp.

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

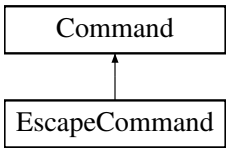
- include/Commands.hpp
- src/Commands.cpp

12.18 EscapeCommand Class Reference

The EscapeCommand class sends command to scene to toggle pause or escape current state.

`#include <Commands.hpp>`

Inheritance diagram for EscapeCommand:



Public Member Functions

- EscapeCommand (Scene ∗_scene)

  PauseCommand constructor for pause command.
- virtual void execute ()

  execute send command to scene to toggle pause

12.18.1 Detailed Description

The EscapeCommand class sends command to scene to toggle pause or escape current state.

Definition at line 210 of file Commands.hpp.

12.18.2 Constructor & Destructor Documentation

12.18.2.1 EscapeCommand::EscapeCommand (Scene ∗ *_scene*)

PauseCommand constructor for pause command.

Parameters

_scene	scene to send instruction to

Definition at line 59 of file Commands.cpp.

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

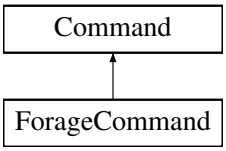
- include/Commands.hpp
- src/Commands.cpp

12.19 ForageCommand Class Reference

The ForageCommand class used to tell character to start foraging.

`#include <Commands.hpp>`

Inheritance diagram for ForageCommand:



Public Member Functions

- ForageCommand (Character *_character)

  ForageCommand constructor for forage command.
- virtual void execute ()

  execute tells character to forage

12.19.1 Detailed Description

The ForageCommand class used to tell character to start foraging.

Definition at line 356 of file Commands.hpp.

12.19.2 Constructor & Destructor Documentation

12.19.2.1 ForageCommand::ForageCommand (Character * *_character*)

ForageCommand constructor for forage command.

Parameters

_character	character to instruct

Definition at line 123 of file Commands.cpp.

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

- include/Commands.hpp
- src/Commands.cpp

12.20 gameUtils.noise.fractalNoise Class Reference

A class for generating fractal noise patterns in 2d using simplex noise.

Public Member Functions

- def __init__ (self, _seed)

  init *is the ctor that sets the seed, and shuffles the permutation table*
- def dot (self, a, x, y)

  dot is a custom dot product function used by the simplexNoise function
- def simplex (self, x_in, y_in)

  simplex computes values of 2d simplex noise for given positions
- def fractal (self, x, y, octaves, persistence, scale, low, high)

  fractal computes 2d fractal noise values using layered simplex noise

Public Attributes

- **p**
- **grad3**

12.20.1 Detailed Description

A class for generating fractal noise patterns in 2d using simplex noise.

The user can make calls to both fractal and simplex fiunctions, incase they only want a single leyer of noise. The simplex noise function was implemented from explanations and examples found at http://weber.itn.↩liu.se/~stegu/simplexnoise/simplexnoise.pdf

Definition at line 13 of file noise.py.

12.20.2 Constructor & Destructor Documentation

12.20.2.1 def gameUtils.noise.fractalNoise.__init__ (*self, _seed*)

init is the ctor that sets the seed, and shuffles the permutation table

Parameters

self	is the class instance
the	seed value used top generate the permutation table which controlls the gradient vector direction for each vertex in the grid

Definition at line 20 of file noise.py.

12.20.3 Member Function Documentation

12.20.3.1 def gameUtils.noise.fractalNoise.dot (*self, a, x, y*)

dot is a custom dot product function used by the simplexNoise function

Parameters

a	is a gradient vector
x	is the x component of second vector
y	is tge y component of second vector

Returns

the dot product of a and (x, y)

Definition at line 39 of file noise.py.

12.20.3.2 def gameUtils.noise.fractalNoise.fractal (*self, x, y, octaves, persistence, scale, low, high*)

fractal computes 2d fractal noise values using layered simplex noise

Parameters

x	is the x component of position
y	is the y component of position
octaves	is the number of layers of noise
persistence	is how much the weight of each layer is reduced by
scale	is the initial frequency
low	is the minimum value of output
high	is the maximum value of output

Returns

fractal noise value for the position (x, y) scaled to the range low-high

Definition at line 118 of file noise.py.

12.20.3.3 def gameUtils.noise.fractalNoise.simplex (*self, x_in, y_in*)

simplex computes values of 2d simplex noise for given positions

Parameters

x_in	is the x component of the position
y_in	is the y component of the position

Returns

the value of the simplex noise at (x_in, y_in)

Definition at line 47 of file noise.py.

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

- python/gameUtils/noise.py

12.21 Framebuffer Class Reference

Public Member Functions

- ~Framebuffer ()

  Destroys the framebuffer and all associated textures.

- void initialise (int _w, int _h)

  Initialises the framebuffer to a given width and height. All textures added later will conform to these dimensions.
- void **activeColourAttachments** ()
- void **activeColourAttachments** (const std::vector< GLenum > _bufs)
- void **activeReadAttachment** (const GLenum _buf)
- void **addTexture** (const std::string &_identifier, GLuint _tex, GLenum _attachment)
- void **addRenderbufferMultisampled** (const std::string &_identifier, GLuint _tex, GLenum _attachment)
- void **addTexture** (const std::string &_identifier, GLenum _format, GLenum _iformat, GLenum _attachment, GLint _type=GL_FLOAT)
- void **addDepthAttachment** (const std::string &_identifier)
- void **bind** ()
- void **bindTexture** (const GLint _shaderID, const std::string &_tex, const char *_uniform, int _target)
- bool **checkComplete** ()
- void **clear** ()
- void **unbind** ()
- GLuint **get** (const std::string _id)
- GLuint **getID** ()

12.21.1 Detailed Description

Definition at line 9 of file Framebuffer.hpp.

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

- include/Framebuffer.hpp
- src/Framebuffer.cpp

12.22 Grid Class Reference

The Grid class holds information about what is contained in each cell of the map. The Grid class is a wrapper around a std::vector of Tile enums. The Tile enums illustrate what is contained within each cell of the map, which can be used by other classes for rendering and path finding.

`#include <Grid.hpp>`

Public Member Functions

- Grid (Inventory *_world_inventory)

  default ctor that sets the grid to a default 50 by 50 set of empty tiles and runs the initialiser
- void updateScript (std::string _script_path, int _new_w=100, int _new_h=100, int _new_seed=8)

  updateScript loads the specified script and runs it to create a new map
- void **printTrees** ()
- void **printTypes** ()
- TileType getTileType (int _x, int _y)

  getTileType gets the tile type at the requested coordinates
- TileType getTileType (ngl::Vec2 _coord)

  getTileType gets the tile type from a ngl::Vec2
- TileType getTileType (int _id)

  getTileType gets the tile type at the given tile id
- int getTileHeight (int _x, int _y)

  getTileHeight gets the tile height at the given position
- int getTileHeight (int _id)

getTileHeight gets the tile height using an integer tile id

- float getInterpolatedHeight (float _x, float _y)

  getInterpolatedHeight interpolates the height between tiles at integer coordinates
- bool isTileTraversable (int _x, int _y)

  isTileTraversable returns a bool that indicated if a character can walk across this tile
- bool isTileTraversable (int _id)

  isTileTraversable returns a bool that indicated if a character can walk across this tile
- std::vector< ngl::Vec2 > getTreePositions (int _x, int _y)

  getTreePositions returns a vector of tree positions on the tile
- int cutTileTrees (int _id, int _goal_amount)

  cutTileTrees reduces the amount of trees on the tile by the requested amount, with a limit of 0 when the tile reaches 0 trees it is converted from a tree tile to an empty tile
- void setTileType (int _id, TileType _type)

  setTileType sets the tile type at the given id
- void setTileType (int _x, int _y, TileType _type)

  setTileType sets the tile type at the given coordinate
- void addBuildState (int _id, float _value, TileType _type)

  addBuildState adds to the build state of the tile, which represents the stage of completion of the building
- void addBuildState (int _x, int _y, float _value, TileType _type)

  addBuildState adds to the build state of the tile, which represents the stage of completion of the building
- float getBuildState (int _id)

  getBuildState gets the build state of the tile with the given id
- float getBuildState (int _x, int _y)

  getBuildState gets the build state of the tile with the given id
- int getNumTrees (int _x, int _y)

  getNumTrees gets the number of trees at the tile
- void houseAdded ()

  houseAdded increment number of houses in scene
- void houseDestroyed ()

  houseDestroyed decrement number of houses in scene
- int getNumHouses ()

  getNumHouses get the number of houses on the grid
- ngl::Vec2 idToCoord (int _tileId)

  converts a tile id to a coordinate, the tile id is the one dimensional coordinate of the tile
- int coordToId (ngl::Vec2 _coord)

  converts a coordinate to a tile id
- int getW ()

  returns grid width
- int getH ()

  returns grid height
- int getGlobalMountainHeight ()

  getGlobalMountainHeight returns the height limit that divides mountains from normal terrain set in the python script
- int getGlobalWaterLevel ()

  getGlobalWaterLevel returns the water level, set in the python script
- bool hasChanges ()

  hasChanges returns a boolean flag that indicated if changes have been made to the grid. changes include cutting down trees and building new houses, this is used to re-build the mesh instances
- void resetHasChanges ()

  resetHasChanges resets the has changes flag to false
- ngl::Vec2 getSpawnPoint ()

  getSpawnPoint retrieves the spawn point set by the python script

- bool checkID (int _id)

  checkID checks that the id asked for is valid
- bool checkCoord (double _x, double _y)

  checkCoord checks that the coordinate asked for is valid
- bool checkCoord (ngl::Vec2 _p)

  checkCoord checks that the coordinate asked for is valid
- std::vector< ngl::Vec2 > getStoreHouses ()

  getStoreHouseIds returns the vector of storehouse positions
- std::vector< ngl::Vec2 > **getChangedTiles** ()
- void **resetChangedTiles** ()

12.22.1 Detailed Description

The Grid class holds information about what is contained in each cell of the map. The Grid class is a wrapper around a std::vector of Tile enums. The Tile enums illustrate what is contained within each cell of the map, which can be used by other classes for rendering and path finding.

Definition at line 23 of file Grid.hpp.

12.22.2 Member Function Documentation

12.22.2.1 void Grid::addBuildState (int *_id*, float *_value*, TileType *_type*)

addBuildState adds to the build state of the tile, which represents the stage of completion of the building

Parameters

_id	is id of the tile of interest
_value	the increment value of the build state
_type	the type of building being built

Definition at line 244 of file Grid.cpp.

12.22.2.2 void Grid::addBuildState (int *_x*, int *_y*, float *_value*, TileType *_type*)

addBuildState adds to the build state of the tile, which represents the stage of completion of the building

Parameters

_x	is the x component of the tile of interest
_y	is the y component of the tile of interest
_value	is the value to increase the tiles build state by
_type	is the type of building being built

Definition at line 255 of file Grid.cpp.

12.22.2.3 bool Grid::checkCoord (double *_x*, double *_y*)

checkCoord checks that the coordinate asked for is valid

Parameters

_x	is the x coordinate

_y	is the y coordinate

Returns

true if the coordinate is valid

Definition at line 314 of file Grid.cpp.

12.22.2.4 bool Grid::checkCoord (ngl::Vec2 *_p*)

checkCoord checks that the coordinate asked for is valid

Parameters

_p	the coordinate as an ngl vec2

Returns

true if the coordinate is valid

Definition at line 326 of file Grid.cpp.

12.22.2.5 bool Grid::checkID (int *_id*)

checkID checks that the id asked for is valid

Parameters

_id	the id to check

Returns

true if the id is valid

Definition at line 302 of file Grid.cpp.

12.22.2.6 int Grid::coordToId (ngl::Vec2 *_coord*)

converts a coordinate to a tile id

Parameters

the	tile id to convert

Returns

a ngl::vec2 that is the 2d coordinate of the tile

Definition at line 111 of file Grid.cpp.

12.22.2.7 int Grid::cutTileTrees (int *_id,* int *_goal_amount*)

cutTileTrees reduces the amount of trees on the tile by the requested amount, with a limit of 0 when the tile reaches 0 trees it is converted from a tree tile to an empty tile

Parameters

_id	is the id of the tile of interest (aka its position in the 1D array of tiles
_goal_amount	is the desired number of trees to cut

Returns

the number of trees cut, not necessarily the goal amount in cases where the tile has no trees left

Definition at line 224 of file Grid.cpp.

12.22.2.8 float Grid::getBuildState (int *_id*)

getBuildState gets the build state of the tile with the given id

Parameters

_id	is the id of the tile of interest

Returns

the build state [0 ... 1] of the tile

Definition at line 266 of file Grid.cpp.

12.22.2.9 float Grid::getBuildState (int *_x*, int *_y*)

getBuildState gets the build state of the tile with the given id

Parameters

_x	is the x coordinate of the tile of interest
_y	is the y coordinate of the tile of interest

Returns

Definition at line 271 of file Grid.cpp.

12.22.2.10 int Grid::getGlobalMountainHeight ( )

getGlobalMountainHeight returns the height limit that divides mountains from normal terrain set in the python script

Returns

the height of the mountains

Definition at line 140 of file Grid.cpp.

12.22.2.11 int Grid::getGlobalWaterLevel ( )

getGlobalWaterLevel returns the water level, set in the python script

Returns

Definition at line 145 of file Grid.cpp.

12.22.2.12 int Grid::getH ( )

returns grid height

Returns

int height of grid

Definition at line 135 of file Grid.cpp.

12.22.2.13 float Grid::getInterpolatedHeight (float _x, float _y)

getInterpolatedHeight interpolates the height between tiles at integer coordinates

Parameters

_x	is the x component for the interpolated height
_y	is the y component for the interpolated height

Returns

the height value interpolated from the 4 surrounding tiles

Definition at line 150 of file Grid.cpp.

12.22.2.14 int Grid::getNumHouses ( ) [inline]

getNumHouses get the number of houses on the grid

Returns

m_num_houses, number of houses on grid

Definition at line 190 of file Grid.hpp.

12.22.2.15 int Grid::getNumTrees (int _x, int _y)

getNumTrees gets the number of trees at the tile

Parameters

_x	is the x coordinate of the tile of interest
_y	is the y coordinate of the tile of interest

Returns

the number of trees remaining on the tile

Definition at line 292 of file Grid.cpp.

12.22.2.16 ngl::Vec2 Grid::getSpawnPoint ( )

getSpawnPoint retrieves the spawn point set by the python script

Returns

the spwan point as an ngl::Vec2

Definition at line 297 of file Grid.cpp.

12.22.2.17 std::vector< ngl::Vec2 > Grid::getStoreHouses ( )

getStoreHouseIds returns the vector of storehouse positions

Returns

m_store_houses

Definition at line 355 of file Grid.cpp.

12.22.2.18 int Grid::getTileHeight (int *_x*, int *_y*)

getTileHeight gets the tile height at the given position

Parameters

_x	is the x component of the position
_y	is the y component of the position

Returns

the height value of the tile

Definition at line 190 of file Grid.cpp.

12.22.2.19 int Grid::getTileHeight (int *_id*)

getTileHeight gets the tile height using an integer tile id

Parameters

_id	is the id of the tile (aka its position in the 1D tile array)

Returns

the height value of the tile

Definition at line 195 of file Grid.cpp.

12.22.2.20 TileType Grid::getTileType (int *_x*, int *_y*)

getTileType gets the tile type at the requested coordinates

Parameters

_x	is the x component of the coordinate
_y	is the y component of the coordinate

Returns

the TileType of the tile at position _x, _y

Definition at line 180 of file Grid.cpp.

12.22.2.21 TileType Grid::getTileType (ngl::Vec2 *_coord*)

getTileType gets the tile type from a ngl::Vec2

Parameters

_coord	the coordinate as Vec2

Returns

the TileType at the given position

Definition at line 175 of file Grid.cpp.

12.22.2.22 TileType Grid::getTileType (int *_id*)

getTileType gets the tile type at the given tile id

Parameters

_id	is the id of the tile (aka its position in the 2D tile array)

Returns

the TileType at the given id

Definition at line 185 of file Grid.cpp.

12.22.2.23 std::vector< ngl::Vec2 > Grid::getTreePositions (int *_x*, int *_y*)

getTreePositions returns a vector of tree positions on the tile

Parameters

_x	is the x coordinate of the tile of interest
_y	is the y coordinate of the tile of interest

Returns

a vector of ngl::Vec2, one for each tree in the tile

Definition at line 287 of file Grid.cpp.

12.22.2.24 int Grid::getW ( )

returns grid width

Returns

int width of grid

Definition at line 130 of file Grid.cpp.

12.22.2.25 bool Grid::hasChanges ( )

hasChanges returns a boolean flag that indicated if changes have been made to the grid. changes include cutting down trees and building new houses, this is used to re-build the mesh instances

Returns

Definition at line 276 of file Grid.cpp.

12.22.2.26 ngl::Vec2 Grid::idToCoord (int *_tileId*)

converts a tile id to a coordinate, the tile id is the one dimensional coordinate of the tile

Parameters

the	tile id to convert

Returns

a ngl::vec2 that is the 2d coordinate of the tile

Definition at line 104 of file Grid.cpp.

12.22.2.27 bool Grid::isTileTraversable (int *_x*, int *_y*)

isTileTraversable returns a bool that indicated if a character can walk across this tile

Parameters

_x	is the x coordinate of the tile of interest
_y	is the y coordinate of the tile of interest

Returns

a bool value that is true if a character can walk on the tile

Definition at line 200 of file Grid.cpp.

12.22.2.28 bool Grid::isTileTraversable (int *_id*)

isTileTraversable returns a bool that indicated if a character can walk across this tile

Parameters

_id	is the id of the tile of interest (aka its position in the 1 array of tiles)

Returns

a bool value that is true if a character can walk on the tile

Definition at line 212 of file Grid.cpp.

12.22.2.29 void Grid::setTileType (int *_id*, TileType *_type*)

setTileType sets the tile type at the given id

Parameters

_id	is the id of the tile of interest (aka its position in the 1D array of tiles)
_type	the new type of the tile

Definition at line 232 of file Grid.cpp.

12.22.2.30 void Grid::setTileType (int *_x*, int *_y*, TileType *_type*)

setTileType sets the tile type at the given coordinate

Parameters

_x	is the x coordinate of the tile of interest
_y	is the y coordinate of the tile of interest
_type	is the new type of the tile

Definition at line 238 of file Grid.cpp.

12.22.2.31 void Grid::updateScript (std::string *_script_path*, int *_new_w* = 100, int *_new_h* = 100, int *_new_seed* = 8)

updateScript loads the specified script and runs it to create a new map

Parameters

_script_path	is the file path to the python script
_new_w	is the width of the map to generate
_new_h	is the new height of the map
_new_seed	is the seed value for random number generation in the map script

Definition at line 35 of file Grid.cpp.

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

- include/Grid.hpp
- src/Grid.cpp

12.23 GridTile Class Reference

stores all of the data associated with each tile in the map

`#include <GridTile.hpp>`

Public Member Functions

- GridTile (int _id)

  ctor that takes a tile id that represents its position on the map
- bool isTraversable ()

  isTraversable returns a bool to indicate if the tile can be walked across, currently trees and buildings are not traversable
- TileType getType ()

  getType gets the type of the tile
- void setType (TileType _t)

  setType sets the type of the tile
- int getHeight ()

  getHeight gets the height stored in the tile
- void setHeight (int _height)

  setHeight sets the height of the tile
- int getNumTrees ()

  getNumTrees returns the number of trees on a tile
- void setNumTrees (int _num_trees)

  setNumTrees sets the number of trees stored in the tile
- int cutTrees (int _goal_amount)

  cutTrees try to reduce the number of treesm returning the number of trees cut
- void addBuildState (float _value, TileType _type)

  setBuildState adds to the build state for a building tile
- float getBuildState ()

getBuildState gets the completion of a building in the range [0 ... 1]

- std::vector< ngl::Vec2 > getTreePositions ()

  getTreePositions gets the positions of the trees on a tile

12.23.1 Detailed Description

stores all of the data associated with each tile in the map

Definition at line 26 of file GridTile.hpp.

12.23.2 Constructor & Destructor Documentation

12.23.2.1 GridTile::GridTile (int *_id*)

ctor that takes a tile id that represents its position on the map

Parameters

the	id of the tile

Definition at line 5 of file GridTile.cpp.

12.23.3 Member Function Documentation

12.23.3.1 void GridTile::addBuildState (float *_value*, TileType *_type*)

setBuildState adds to the build state for a building tile

Parameters

_value	is the amount to increment the build state variable by
_type	is the type of building being built

Definition at line 75 of file GridTile.cpp.

12.23.3.2 int GridTile::cutTrees (int *_goal_amount*)

cutTrees try to reduce the number of treesm returning the number of trees cut

Parameters

_goal_amount	is the desired abount of trees to cut

Returns

the actualy amount of trees cut (for instances where 3 are reuqested but ony one is there for example)

Definition at line 58 of file GridTile.cpp.

12.23.3.3 float GridTile::getBuildState ( ) [inline]

getBuildState gets the completion of a building in the range [0 ... 1]

Returns

the tiles build state

Definition at line 96 of file GridTile.hpp.

12.23.3.4 int GridTile::getHeight ( )

getHeight gets the height stored in the tile

Returns

the height of the tile

Definition at line 38 of file GridTile.cpp.

12.23.3.5 int GridTile::getNumTrees ( )

getNumTrees returns the number of trees on a tile

Returns

the number of trees left on the tile

Definition at line 48 of file GridTile.cpp.

12.23.3.6 std::vector< ngl::Vec2 > GridTile::getTreePositions ( )

getTreePositions gets the positions of the trees on a tile

Returns

a vector of Vec2's that store the trees positions as offsets of the tiles position [-0.4 ... 0.4]

Definition at line 99 of file GridTile.cpp.

12.23.3.7 TileType GridTile::getType ( )

getType gets the type of the tile

Returns

the TileType stored in the tile

Definition at line 28 of file GridTile.cpp.

12.23.3.8 bool GridTile::isTraversable ( )

isTraversable returns a bool to indicate if the tile can be walked across, currently trees and buildings are not traversable

Returns

bool returned

Definition at line 19 of file GridTile.cpp.

12.23.3.9 void GridTile::setHeight (int *_height*)

setHeight sets the height of the tile

Parameters

_height	is the height to be set

Definition at line 43 of file GridTile.cpp.

12.23.3.10 void GridTile::setNumTrees (int *_num_trees*)

setNumTrees sets the number of trees stored in the tile

Parameters

_num_trees	is the number of trees to be set

Definition at line 53 of file GridTile.cpp.

12.23.3.11 void GridTile::setType (TileType *_t*)

setType sets the type of the tile

Parameters

_t	is the type to be set

Definition at line 33 of file GridTile.cpp.

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

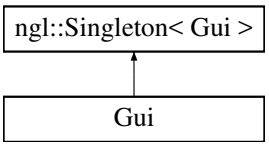
- include/GridTile.hpp
- src/GridTile.cpp

12.24 Gui Class Reference

The Gui class contains button positions and managing their use.

`#include <Gui.hpp>`

Inheritance diagram for Gui:



Public Member Functions

- void init (Scene *_scene, ngl::Vec2 _res, const std::string &_shader_name)
  init initialise or reset Gui with specific resolution
- void setResolution (ngl::Vec2 _res)
  setResolution set the window width and window height
- void initGL ()
  initGL initialise gl values
- void click ()
  click check if user has clicked inside any button area
- std::shared_ptr< Command > generateCommand (Action _action)
  generateCommand create a command to be executed

- int executeAction (Action _action)

  executeAction generate a command and execute it
- bool mousePos (ngl::Vec2 _pos)

  mousePos update mouse position
- void wipeButtons ()

  wipeButtons used to clear buttons, and button id counter is reset to 0
- void pause ()

  pause create pause menu and set pause uniform
- void unpause ()

  unpause leave pause menu and set pause uniform
- void createStartMenuButtons ()

  createStartMenuButtons creates the set of buttons for start menu
- void createSceneButtons ()

  createSceneButtons creates the set of default buttons for the scene
- void createPauseButtons ()

  createPauseButtons create the set of buttons for the pause menu
- void createPrefsButtons ()

  createPrefsButtons create the set of buttons for the preferences menu
- void createEndGameButtons (const std::string &_message)

  createEndGameButtons create set of buttons for the game over screen
- void addButton (Action _action, XAlignment _x_align, YAlignment _y_align, ngl::Vec2 _offset, ngl::Vec2 _size, const std::string &_text)

  addButton add a button to the button vector
- void addNotification (const std::string &_text, ngl::Vec2 _map_pos)

  addNotification add a notificaion to the button vector
- void removeButton (std::shared_ptr< Button > button)

  removeButton delete a button from the vector
- void updateButtonArrays ()

  updateButtonArrays updates positions and passes them to openGL, useful for changing buttons or resizing screen
- void updateNotifications ()

  updateNotifications update age of notifications and delete any if necessary
- void drawButtons ()

  drawButtons draw buttons to screen
- void mouseDown ()

  mouseDown register mouse as clicked
- void mouseUp ()

  mouseUp register mouse as not clicked
- void bindTextureToShader (const GLuint _tex, const char *_uniform, int _target)

  bindTextureToShader bind the given texture to the button shader
- void updateText ()

  updateText send m_button_text to shader
- void updateActiveCharacter ()

  updateActiveCharacter if active character changes, this function updates button text
- void notify (const std::string &_text, ngl::Vec2 _pos)

  notify create a notification command
- void moveNotifications (ngl::Vec2 _move_vec)

  moveNotifications move all notifications eg up or down
- int getButtonLength (const std::string &_text)

  getButtonLength find out how long a button should be to fit a string
- void scrollButton (int _dir)

  scrollButton send an increment or decrement command

- void mapChanged ()

  mapChanged update buttons which require name of current map
- void addMapButtons ()

  addMapButtons add map selection buttons to a menu
- void addMenuButtons ()

  addMenuButtons add buttons for menu

Friends

- class **Singleton**< **Gui** >

12.24.1 Detailed Description

The Gui class contains button positions and managing their use.

Definition at line 25 of file Gui.hpp.

12.24.2 Member Function Documentation

12.24.2.1 void Gui::addButton (Action *_action*, XAlignment *_x_align*, YAlignment *_y_align*, ngl::Vec2 *_offset*, ngl::Vec2 *_size*, const std::string & *_text*)

addButton add a button to the button vector

Parameters

_action	button's action when clicked
_x_align	horizonal alignment (to left/right/center of screen)
_y_align	vertical alignment (to top/bottom/center of screen)
_offset	offset from edge/center
_size	size of button
_text	button's text

Definition at line 388 of file Gui.cpp.

12.24.2.2 void Gui::addNotification (const std::string & *_text*, ngl::Vec2 *_map_pos*)

addNotification add a notificaion to the button vector

Parameters

_text	text of notification
_map_pos	position notification comes from

Definition at line 393 of file Gui.cpp.

12.24.2.3 void Gui::bindTextureToShader (const GLuint *_tex*, const char * *_uniform*, int *_target*)

bindTextureToShader bind the given texture to the button shader

Parameters

_tex	texture id to use

_uniform	name of texture in shader
_target	which number texture unit to use

Definition at line 619 of file Gui.cpp.

12.24.2.4 void Gui::click ( )

click check if user has clicked inside any button area

Parameters

pos	mouse position to compare buttons against

Definition at line 58 of file Gui.cpp.

12.24.2.5 int Gui::executeAction (Action *_action*)

executeAction generate a command and execute it

Parameters

_action	action to execute

Returns

0 for no errors, 1 for nothing executed

Definition at line 236 of file Gui.cpp.

12.24.2.6 std::shared_ptr< Command > Gui::generateCommand (Action *_action*)

generateCommand create a command to be executed

Returns

command to be executed

Definition at line 67 of file Gui.cpp.

12.24.2.7 int Gui::getButtonLength (const std::string & *_text*)

getButtonLength find out how long a button should be to fit a string

Parameters

_text	text on button

Returns

length of button needed

Definition at line 736 of file Gui.cpp.

12.24.2.8 void Gui::init (Scene * *_scene,* ngl::Vec2 *_res,* const std::string & *_shader_name*)

init initialise or reset Gui with specific resolution

Parameters

_res	resolution to initialise with
_shader_name	name of gui shader

Definition at line 25 of file Gui.cpp.

12.24.2.9 bool Gui::mousePos (ngl::Vec2 *_pos*)

mousePos update mouse position

Parameters

_pos	current mouse position

Definition at line 253 of file Gui.cpp.

12.24.2.10 void Gui::moveNotifications (ngl::Vec2 *_move_vec*)

moveNotifications move all notifications eg up or down

Parameters

_move_vec	

Definition at line 724 of file Gui.cpp.

12.24.2.11 void Gui::notify (const std::string & *_text*, ngl::Vec2 *_pos*)

notify create a notification command

Parameters

_text	text for notification
_pos	position on map that notification comes from

Definition at line 697 of file Gui.cpp.

12.24.2.12 void Gui::removeButton (std::shared_ptr< Button > *button*)

removeButton delete a button from the vector

Parameters

button	button to remove

Definition at line 401 of file Gui.cpp.

12.24.2.13 void Gui::scrollButton (int *_dir*)

scrollButton send an increment or decrement command

Parameters

_dir	positive for increment, negative for decrement

Definition at line 741 of file Gui.cpp.

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

- include/Gui.hpp
- src/Gui.cpp

12.25 ImGuiPlotArrayGetterData Struct Reference

Public Member Functions

- **ImGuiPlotArrayGetterData** (const float *values, int stride)

Public Attributes

- const float * **Values**
- int **Stride**

12.25.1 Detailed Description

Definition at line 7089 of file imgui.cpp.

The documentation for this struct was generated from the following file:

- src/imgui/imgui.cpp

12.26 Inventory Class Reference

The Inventory class for management of the global inventory as accessed by the character through storehouses.

`#include <Inventory.hpp>`

Public Member Functions

- Inventory ()
  ctor, sets up intial values of inventory items
- ~Inventory ()=default
  destructor
- int getWoodInventory ()
  getWoodInventory, get the amount of wood form the global inventory
- int getBerryInventory ()
  getBerryInventory, get the amount of berries from the global inventory
- int getFishInventory ()
  getFishInventory, get the amount of fishes from the global inventory
- bool addWood (int _amount)
  addWood, add wood to the global inventory
- bool addBerries (int _amount)
  addBerries, add berries to the global inventory
- bool addFish (int _amount)
  addFish, add fishes to the global inventory
- int takeWood (int _amount)
  takeWood, take wood from the global inventory
- int takeBerries (int _amount)
  takeBerries, take berries from the global inventory
- int takeFish (int _amount)
  takeFish, take fish from the global inventory
- int getMaxWood ()
  getMaxWood, returns maximum number of wood that can be stored

- int getMaxBerries ()

  getMaxBerries, returns maximum number of berries that can be stored
- int getMaxFish ()

  getMaxFish, returns maximum number of fish that can be stored
- void addStoreSpace ()

  addStoreSpace, adds onto maximum amount for storing items

12.26.1 Detailed Description

The Inventory class for management of the global inventory as accessed by the character through storehouses.

Definition at line 11 of file Inventory.hpp.

12.26.2 Member Function Documentation

12.26.2.1 bool Inventory::addBerries (int *_amount*)

addBerries, add berries to the global inventory

Parameters

in	*_amount,amount*	of berries being added to the inventory

Definition at line 26 of file Inventory.cpp.

12.26.2.2 bool Inventory::addFish (int *_amount*)

addFish, add fishes to the global inventory

Parameters

in	*_amount,amount*	of fishes being added to the inventory

Definition at line 39 of file Inventory.cpp.

12.26.2.3 bool Inventory::addWood (int *_amount*)

addWood, add wood to the global inventory

Parameters

_amount,amount	of wood being added to the inventory

Definition at line 13 of file Inventory.cpp.

12.26.2.4 int Inventory::getBerryInventory ( ) [inline]

getBerryInventory, get the amount of berries from the global inventory

Returns

m_berry_inventory, amount of berries in the inventory

Definition at line 31 of file Inventory.hpp.

12.26.2.5 int Inventory::getFishInventory ( ) [inline]

getFishInventory, get the amount of fishes from the global inventory

Returns

m_fish_inventory, amount of fishes in the inventory

Definition at line 36 of file Inventory.hpp.

12.26.2.6 int Inventory::getMaxBerries ( ) [inline]

getMaxBerries, returns maximum number of berries that can be stored

Returns

m_max_berries

Definition at line 79 of file Inventory.hpp.

12.26.2.7 int Inventory::getMaxFish ( ) [inline]

getMaxFish, returns maximum number of fish that can be stored

Returns

m_max_fish

Definition at line 84 of file Inventory.hpp.

12.26.2.8 int Inventory::getMaxWood ( ) [inline]

getMaxWood, returns maximum number of wood that can be stored

Returns

m_max_wood

Definition at line 74 of file Inventory.hpp.

12.26.2.9 int Inventory::getWoodInventory ( ) [inline]

getWoodInventory, get the amount of wood form the global inventory

Returns

m_wood_inventory, amount of wood in the inventory

Definition at line 26 of file Inventory.hpp.

12.26.2.10 int Inventory::takeBerries (int *_amount*)

takeBerries, take berries from the global inventory

Parameters

in	*_amount,amount*	of berries wanting to be taken

Returns

amount of berries available/the amount asked for

Definition at line 75 of file Inventory.cpp.

12.26.2.11 int Inventory::takeFish (int *_amount*)

takeFish, take fish from the global inventory

Parameters

in	*_amount,amount*	of fishes wanting to be taken

Returns

amount of fishes available/ the amount asked for

Definition at line 98 of file Inventory.cpp.

12.26.2.12 int Inventory::takeWood (int *_amount*)

takeWood, take wood from the global inventory

Parameters

in	*_amount,amount*	of wood wanting to be taken

Returns

amount of wood available/the amount asked for

Definition at line 52 of file Inventory.cpp.

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

- include/Inventory.hpp
- src/Inventory.cpp

12.27 IVal< T > Class Template Reference

Public Member Functions

- IVal (T _start, T _end, float _stepsize)
  Constructor, sets the start end and step size.
- void update ()
  Updates m_cur.
- T get () const
  Gets m_cur.
- T getStart () const
  Getter and setter for start.
- void **setStart** (const T &_start)

- T getEnd () const
  Getter and setter for end.
- void **setEnd** (const T &_end)
- void **incrEnd** (const T &_add)
- float getStepsize () const
  Getter and setter for step size.
- void **setStepsize** (const float _stepsize)
- void reset ()
  Resets the IVal, ready to be used again.

12.27.1 Detailed Description

template<typename T>class IVal< T >

Definition at line 11 of file IVal.hpp.

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

- include/IVal.hpp

12.28 Light Struct Reference

Public Member Functions

- **Light** (const ngl::Vec4 &_pos, const ngl::Vec3 &_col, const float _lum)

Public Attributes

- ngl::Vec4 **m_pos**
- ngl::Vec3 **m_col**
- float **m_lum**

12.28.1 Detailed Description

Definition at line 18 of file Light.hpp.

The documentation for this struct was generated from the following file:

- include/Light.hpp

12.29 light Class Reference

Contains vec4 position for transformation, a vec3 colour and a float opacity/brightness.

`#include <Light.hpp>`

12.29.1 Detailed Description

Contains vec4 position for transformation, a vec3 colour and a float opacity/brightness.

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

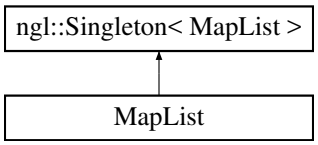
- include/Light.hpp

12.30 MapList Class Reference

The MapList class.

`#include <MapList.hpp>`

Inheritance diagram for MapList:



Public Member Functions

- MapList ()

  MapList constructor sets up map variables.
- std::string getCurrentMapName ()

  getCurrentMapName get name of map
- std::string getCurrentMapPath ()

  getCurrentMapPath get path of map
- void nextMap ()

  nextMap select the next map
- void prevMap ()

  prevMap select the previous map
- void addWidth (int x)

  addWidth change the width of the map
- void addHeight (int x)

  addHeight change the height of the map
- void addSeed (int x)

  addSeedchange the seed of the map
- int getW ()

  getW get map width
- int getH ()

  getH get map height
- int getSeed ()

  getSeed get map seed
- std::string getWString ()

  getWString get width as a string
- std::string getHString ()

  getHString get height as a string
- std::string getSeedString ()

  getSeedString get seed as a string

Friends

- class **Singleton**< **MapList** >

12.30.1 Detailed Description

The MapList class.

Definition at line 14 of file MapList.hpp.

12.30.2 Member Function Documentation

12.30.2.1 void MapList::addHeight (int *x*)

addHeight change the height of the map

Parameters

x	value to change height by

Definition at line 71 of file MapList.cpp.

12.30.2.2 void MapList::addSeed (int *x*)

addSeedchange the seed of the map

Parameters

x	value to change seed by

Definition at line 77 of file MapList.cpp.

12.30.2.3 void MapList::addWidth (int *x*)

addWidth change the width of the map

Parameters

x	value to change width by

Definition at line 65 of file MapList.cpp.

12.30.2.4 std::string MapList::getCurrentMapName ( )

getCurrentMapName get name of map

Returns

name of map as string

Definition at line 18 of file MapList.cpp.

12.30.2.5 std::string MapList::getCurrentMapPath ( )

getCurrentMapPath get path of map

Returns

path of map as string

Definition at line 34 of file MapList.cpp.

12.30.2.6 int MapList::getH ( )

getH get map height

Returns

m_H integer map height

Definition at line 88 of file MapList.cpp.

12.30.2.7 std::string MapList::getHString ( )

getHString get height as a string

Returns

string containing "Height: "+m_h

Definition at line 103 of file MapList.cpp.

12.30.2.8 int MapList::getSeed ( )

getSeed get map seed

Returns

m_seed integer map seed

Definition at line 93 of file MapList.cpp.

12.30.2.9 std::string MapList::getSeedString ( )

getSeedString get seed as a string

Returns

string containing "Seed: "+m_seed

Definition at line 108 of file MapList.cpp.

12.30.2.10 int MapList::getW ( )

getW get map width

Returns

m_w integer map width

Definition at line 83 of file MapList.cpp.

12.30.2.11 std::string MapList::getWString ( )

getWString get width as a string

Returns

string containing "Width: "+m_w

Definition at line 98 of file MapList.cpp.

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

- include/MapList.hpp
- src/MapList.cpp

12.31 gameUtils.mapViewer.mapViewer Class Reference

The map viewer class.

Public Member Functions

- def __init__ (self, _map, _dict, _w, _h)

  init *is the ctor that reads in all the data and sets colours for the different types*
- def setColours (self, default=(50, 200, tree=(30, 100, mountain=(100, 100, water=(80, 80)

  setColours sets the colours that different map elements will be drawn in
- def display

  display shows the map as an image on the screen using the python image libraray where 1 tile = 1 pixel

Public Attributes

- **map**
- **h**
- **w**
- **dict**
- **default**
- **tree**
- **mountain**
- **peak**
- **water**

12.31.1 Detailed Description

The map viewer class.

Definition at line 9 of file mapViewer.py.

12.31.2 Constructor & Destructor Documentation

12.31.2.1 def gameUtils.mapViewer.mapViewer.__init__ (*self, _map, _dict, _w, _h*)

init is the ctor that reads in all the data and sets colours for the different types

Parameters

self	is the instance of the class
_map	is the map data to be read to produce the image
_dict	is the dictionary that maps key strigs to integers that indicate tile types
_w	is the width of the map
_h	is the height of the map

Definition at line 20 of file mapViewer.py.

12.31.3 Member Function Documentation

12.31.3.1 def gameUtils.mapViewer.mapViewer.display (*self*, *displayType* = "types")

display shows the map as an image on the screen using the python image libraray where 1 tile = 1 pixel

Parameters

self	is the instance of the class
displayType	indicated what is to be drawn ("types", or "height")

Definition at line 52 of file mapViewer.py.

12.31.3.2 def gameUtils.mapViewer.mapViewer.setColours (*self*, *default* = (50, 200, *tree* = (30, 100, *mountain* = (100, 100, *water* = (80, 80)

setColours sets the colours that different map elements will be drawn in

Parameters

self	is the instance of the class
default	is the colour for empty tiles
tree	is the colour for tree tiles
mountain	is the colour for mountain tiles
water	is the colour for water tiles

Definition at line 40 of file mapViewer.py.

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

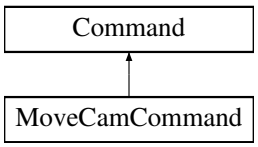
- python/gameUtils/mapViewer.py

12.32 MoveCamCommand Class Reference

The MoveCamCommand class moves the camera or stops the camera in the given direction.

`#include <Commands.hpp>`

Inheritance diagram for MoveCamCommand:



Public Member Functions

- MoveCamCommand (Scene *_scene, Direction _d, bool _stop)

MoveCamCommand constructor for MoveCamCommand.

- virtual void execute ()

  execute send command to scene to move/stop camera

12.32.1 Detailed Description

The MoveCamCommand class moves the camera or stops the camera in the given direction.

Definition at line 282 of file Commands.hpp.

12.32.2 Constructor & Destructor Documentation

12.32.2.1 MoveCamCommand::MoveCamCommand (Scene * *_scene*, Direction *_d*, bool *_stop*)

MoveCamCommand constructor for MoveCamCommand.

Parameters

_scene	scene to send instruction to
_d	direction of command
_stop	whether to stop in that direction or not

Definition at line 99 of file Commands.cpp.

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

- include/Commands.hpp
- src/Commands.cpp

12.33 Node Class Reference

Pathfinding node class.

`#include <Node.hpp>`

Public Member Functions

- Node (Grid *_grid, std::vector< Node > *nodes, ngl::Vec2 _pos, ngl::Vec2 _target_pos, int _parent_id)

  Node constructor used to set up position, target position and parent_id, and pointers to grid and vector of nodes.
- void close ()

  close turns the node "off", after it has been fully explored
- float manhattanDist (ngl::Vec2 _target_pos)

  manhattanDist get the x + y distance to given target
- float calcNewGCost (int _parent_id)

  calcNewGCost calculate a g_cost using a potential parent
- float getFCost ()

  getFCost get the node's current f cost
- bool isOpen ()

  isOpen check if node is still open or if it is closed
- int getID ()

  getID get node's position in vector
- ngl::Vec2 getPos ()

  getPos get the position of the node
- float getGCost ()

getGCost get the current g cost of the node

- void setParent (int _parent_id)

  setParent change parent of node

- int getParentID ()

  getParentID get the parent position in vector

- std::array< int, 4 > getNeighbours ()

  getNeighbours return list of neighbour positions in vector

12.33.1 Detailed Description

Pathfinding node class.

Definition at line 23 of file Node.hpp.

12.33.2 Constructor & Destructor Documentation

12.33.2.1 Node::Node (Grid * *_grid,* std::vector< Node > * *nodes,* ngl::Vec2 *_pos,* ngl::Vec2 *_target_pos,* int *_parent_id*)

Node constructor used to set up position, target position and parent_id, and pointers to grid and vector of nodes.

Parameters

_grid	used to reference traversability and size
nodes	for referencing other nodes in the vector
_pos	the position of the node being constructed
_target_pos	the target position for calculating manhatten distance for h cost
_parent_id	the position of the parent node in the vector

Definition at line 5 of file Node.cpp.

12.33.3 Member Function Documentation

12.33.3.1 float Node::calcNewGCost (int *_parent_id*)

calcNewGCost calculate a g_cost using a potential parent

Parameters

_parent_id	potential parent to check against

Returns

distance jumping node to node until reaching original position

Definition at line 26 of file Node.cpp.

12.33.3.2 float Node::getFCost ( )

getFCost get the node's current f cost

Returns

node's f cost

Definition at line 43 of file Node.cpp.

12.33.3.3 float Node::getGCost ( )

getGCost get the current g cost of the node

Returns

the current g cost of the node

Definition at line 63 of file Node.cpp.

12.33.3.4 int Node::getID ( )

getID get node's position in vector

Returns

m_id, which is the position in the vector

Definition at line 53 of file Node.cpp.

12.33.3.5 std::array< int, 4 > Node::getNeighbours ( )

getNeighbours return list of neighbour positions in vector

Returns

array of 4 potential neighbour positions, with -1 if not found

Definition at line 78 of file Node.cpp.

12.33.3.6 int Node::getParentID ( )

getParentID get the parent position in vector

Returns

parent position in vector

Definition at line 74 of file Node.cpp.

12.33.3.7 ngl::Vec2 Node::getPos ( )

getPos get the position of the node

Returns

the position of the node

Definition at line 58 of file Node.cpp.

12.33.3.8 bool Node::isOpen ( )

isOpen check if node is still open or if it is closed

Returns

true for open, false for closed

Definition at line 48 of file Node.cpp.

12.33.3.9 float Node::manhattanDist (ngl::Vec2 *_target_pos*)

manhattanDist get the x + y distance to given target

Parameters

_target_pos	the target to check position against

Returns

x dist + y dist

Definition at line 38 of file Node.cpp.

12.33.3.10 void Node::setParent (int *_parent_id*)

setParent change parent of node

Parameters

_parent_id	position of new parent in vector

Definition at line 68 of file Node.cpp.

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

- include/Node.hpp
- src/Node.cpp

12.34 NodeNetwork Class Reference

Wraps up a Node vector and uses it to find a path on the Grid object given in its constructor.

`#include <NodeNetwork.hpp>`

Public Member Functions

- NodeNetwork (Grid *_grid, ngl::Vec2 _pos, ngl::Vec2 _target_pos)

  NodeNetwork Constructor requires a grid and a start and end point for pathfinding.
- std::vector< ngl::Vec2 > findPath ()

  findPath returns a vector of 2d points that can be traversed
- void printNetwork ()

  printNetwork used for debugging node network
- std::vector< ngl::Vec2 > createFoundPath (Node _end_node)

  createFoundPath go through parent nodes and return list of positions of valid path
- void printPath (std::vector< ngl::Vec2 > &_path)

  printPath prints final path

Static Public Member Functions

- static bool raytrace (Grid *_grid, ngl::Vec2 _start_pos, ngl::Vec2 _end_pos)

  raytrace check whether straight path is possible between given points. Does not require instance of class. Algorithm from http://playtechs.blogspot.co.uk/2007/03/raytracing-on-grid.html

12.34.1 Detailed Description

Wraps up a Node vector and uses it to find a path on the Grid object given in its constructor.

Definition at line 16 of file NodeNetwork.hpp.

12.34.2 Constructor & Destructor Documentation

12.34.2.1 NodeNetwork::NodeNetwork (Grid * _grid, ngl::Vec2 _pos, ngl::Vec2 _target_pos)

NodeNetwork Constructor requires a grid and a start and end point for pathfinding.

Parameters

_grid	the grid used to check tile traversability
_pos	initial position of character
_target_pos	target position to find a path to

Definition at line 4 of file NodeNetwork.cpp.

12.34.3 Member Function Documentation

12.34.3.1 std::vector< ngl::Vec2 > NodeNetwork::createFoundPath (Node _end_node)

createFoundPath go through parent nodes and return list of positions of valid path

Parameters

_end_node	last node in path

Returns

vector of points in path

Definition at line 172 of file NodeNetwork.cpp.

12.34.3.2 std::vector< ngl::Vec2 > NodeNetwork::findPath ( )

findPath returns a vector of 2d points that can be traversed

Returns

vector of points in a path to target, or 0 length vector if no path is found

Definition at line 12 of file NodeNetwork.cpp.

12.34.3.3 void NodeNetwork::printPath (std::vector< ngl::Vec2 > & _path)

printPath prints final path

Parameters

_path	list of nodes that make up path

Definition at line 192 of file NodeNetwork.cpp.

12.34.3.4 bool NodeNetwork::raytrace (Grid * _grid, ngl::Vec2 _start_pos, ngl::Vec2 _end_pos) [static]

raytrace check whether straight path is possible between given points. Does not require instance of class. Algorithm from http://playtechs.blogspot.co.uk/2007/03/raytracing-on-grid.html

Parameters

_grid	grid to check
_start_pos	start of line
_end_pos	end of line

Returns

true for line of sight between _start_pos and _end_pos

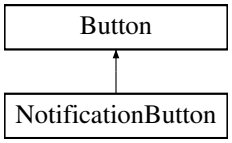
Definition at line 228 of file NodeNetwork.cpp.

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

- include/NodeNetwork.hpp
- src/NodeNetwork.cpp

12.35 NotificationButton Class Reference

Inheritance diagram for NotificationButton:



Public Member Functions

- **NotificationButton** (Action _action, XAlignment _x_align, YAlignment _y_align, ngl::Vec2 _window_res, ngl::Vec2 _offset, ngl::Vec2 _size, const std::string &_text, ngl::Vec2 _map_pos)
- void **incrementAge** ()
- int **getAge** ()
- ngl::Vec2 **getMapPos** ()

Additional Inherited Members

12.35.1 Detailed Description

Definition at line 134 of file Button.hpp.

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

- include/Button.hpp
- src/Button.cpp

12.36 ParticleSystem Struct Reference

Public Member Functions

- size_t **size** () const

Public Attributes

- std::vector< ngl::Vec3 > **m_pos**
- std::vector< ngl::Vec3 > **m_vel**
- std::vector< float > **m_scale**
- std::vector< float > **m_time**
- std::vector< float > **m_alpha**

12.36.1 Detailed Description

Definition at line 9 of file ParticleSystem.hpp.

The documentation for this struct was generated from the following file:

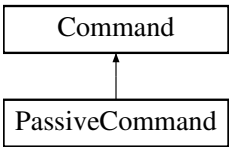
- include/ParticleSystem.hpp

12.37 PassiveCommand Class Reference

The PassiveCommand class for buttons which have no function, so cannot be clicked.

`#include <Commands.hpp>`

Inheritance diagram for PassiveCommand:



Public Member Functions

- PassiveCommand ()

  PassiveCommand basic constructor.
- virtual void execute ()

  execute function does nothing

12.37.1 Detailed Description

The PassiveCommand class for buttons which have no function, so cannot be clicked.

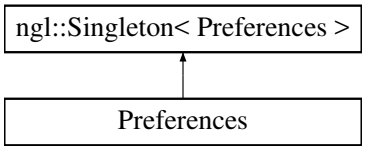
Definition at line 123 of file Commands.hpp.

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

- include/Commands.hpp
- src/Commands.cpp

12.38 Preferences Class Reference

Inheritance diagram for Preferences:



Public Member Functions

- void **init** ()
- void **save** ()
- int **getXRes** ()
- int **getYRes** ()
- bool **getDOP** ()
- bool **getShadows** ()
- bool **getAA** ()
- bool **getReflections** ()
- float **getTimeScale** ()
- int **getShadowMapRes** ()
- int **getWaterMapRes** ()
- float **getCharacterSpeed** ()
- std::string **getMapScriptPath** ()

Friends

- class **Singleton**< **Preferences** >

12.38.1 Detailed Description

Definition at line 7 of file Preferences.hpp.

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

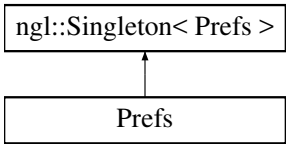
- include/Preferences.hpp
- src/Preferences.cpp

12.39 Prefs Class Reference

The Prefs class holds all of the preferences stored in the file preferences.conf the file is parsed by the PrefsParser class and the contentse are stored in 3 std::maps that link string keys to their corresponding values. The three maps store int, float and string preferences respectively. The Prefs class also has the capacity to restore default parameters and save the current state of the parameters to the same text file they were read from.

`#include <Prefs.hpp>`

Inheritance diagram for Prefs:



Public Member Functions

- void init (std::string _pref_file="preferences.conf")

  init initialises the Prefs class by running the PrefsParser to build the preferences from the given file
- void restoreDefaultPrefs ()

  restoreDefaultPrefs clears the three maps and fills them with default values
- IncType **getIncType** (std::string _key)
- float getFloatIncValue (const std::string &_key)

  getFloatIncValue get value for incrementing given preference
- float getFloatDecValue (const std::string &_key)

  getFloatDecValue get value for decrementing given preference
- float getFloatChangeValue (const std::string &_key, int _dir)

  getFloatChangeValue get float value required for changing given preference in given direction
- int getIntIncValue (const std::string &_key)

  getIntIncValue get value for incrementing given preference
- int getIntDecValue (const std::string &_key)

  getIntDecValue get value for decrementing given preference
- int getIntChangeValue (const std::string &_key, int _dir)

  getIntChangeValue get int value required for changing given preference in given direction
- void setIntPref (std::string _key, int _val)

  setIntPref adds an integer preference key value pair to the interger preferences map
- void setFloatPref (std::string _key, float _val)

  setFloatPref adds a float preference key value pair to the float preferences map
- void setStrPref (std::string _key, std::string _val)

  setStrPref adds an integer preference key value pair to the interger preferences map
- void **setBoolPref** (std::string _key, bool _val)
- int getIntPref (std::string _key)

  getIntPref retrieves an integer preference with the given key
- float getFloatPref (std::string _key)

  getFloatPref retrieves a float preference with the given key
- std::string getStrPref (std::string _key)

  getStrPref retrieves a string preference with the given key
- bool **getBoolPref** (std::string _key)
- const std::map< std::string, std::pair< int, IncType > > & getIntMap ()

  getIntMap retrieves the whole integer preference map
- const std::map< std::string, std::pair< float, IncType > > & getFloatMap ()

  getFloatMap retrieves the entire float preference map
- const std::map< std::string, std::pair< std::string, IncType > > & getStrMap ()

  getStrMap retrieves the entire string preference map
- const std::map< std::string, std::pair< bool, IncType > > & getBoolMap ()

  getBoolMap retrieves the entire string preference map
- void printPrefs ()

  printPrefs prints out all key-value pairs stored in the three maps
- void savePrefs ()

  savePrefs saves the preferences to the preferences.conf text file
- PrefType getTypeOfPref (const std::string &_key)

  getTypeOfPref find what type the preference is
- void setPref (std::string &_key, int _val)

  setPref generic template for setting preference
- void **setPref** (std::string &_key, float _val)
- void **setPref** (std::string &_key, const std::string &_val)

- void **setPref** (std::string &_key, bool _val)
- std::string getPrefValueString (const std::string &_key)

  getPrefValueString get text for value of string
- int getNumPrefs ()

  getNumPrefs get total number of preference options
- int getNumChangeablePrefs ()

  getNumChangeablePrefs get number of preferences that can be changed by the Gui
- std::string boolToString (bool _b)

  boolToString convert bool to string for Gui

Friends

- class **Singleton**< **Prefs** >

12.39.1 Detailed Description

The Prefs class holds all of the preferences stored in the file preferences.conf the file is parsed by the PrefsParser class and the contentse are stored in 3 std::maps that link string keys to their corresponding values. The three maps store int, float and string preferences respectively. The Prefs class also has the capacity to restore default parameters and save the current state of the parameters to the same text file they were read from.

To add a new preference, it only needs to be added to the preferences.conf text file and a default value defined in the function restoreDefaultPrefs()

Definition at line 43 of file Prefs.hpp.

12.39.2 Member Function Documentation

12.39.2.1 std::string Prefs::boolToString (bool *_b*)

boolToString convert bool to string for Gui

Returns

string, either "0" or "1"

Definition at line 416 of file Prefs.cpp.

12.39.2.2 const std::map< std::string, std::pair< bool, IncType > > & Prefs::getBoolMap ( )

getBoolMap retrieves the entire string preference map

Returns

the map containing all of the boolean preference key-value pairs

Definition at line 294 of file Prefs.cpp.

12.39.2.3 float Prefs::getFloatChangeValue (const std::string & *_key*, int *_dir*)

getFloatChangeValue get float value required for changing given preference in given direction

Parameters

_key	name of preference
_dir	positive or negative for up/down

Returns

value to add/subtract

Definition at line 105 of file Prefs.cpp.

12.39.2.4 float Prefs::getFloatDecValue (const std::string & *_key*)

getFloatDecValue get value for decrementing given preference

Parameters

_key	name of preference

Returns

float value to subtract from preference

Definition at line 100 of file Prefs.cpp.

12.39.2.5 float Prefs::getFloatIncValue (const std::string & *_key*)

getFloatIncValue get value for incrementing given preference

Parameters

_key	name of preference

Returns

float value to add to preference

Definition at line 95 of file Prefs.cpp.

12.39.2.6 const std::map< std::string, std::pair< float, IncType > > & Prefs::getFloatMap ( )

getFloatMap retrieves the entire float preference map

Returns

the map containing all of the integer preference key-value pairs

Definition at line 284 of file Prefs.cpp.

12.39.2.7 float Prefs::getFloatPref (std::string *_key*)

getFloatPref retrieves a float preference with the given key

Parameters

_key	is the string key associated with the value

Returns

the value of the preference

Definition at line 214 of file Prefs.cpp.

12.39.2.8 int Prefs::getIntChangeValue (const std::string & *_key*, int *_dir*)

getIntChangeValue get int value required for changing given preference in given direction

Parameters

_key	name of preference
_dir	positive or negative for up/down

Returns

value to add/subtract

Definition at line 156 of file Prefs.cpp.

12.39.2.9 int Prefs::getIntDecValue (const std::string & *_key*)

getIntDecValue get value for decrementing given preference

Parameters

_key	name of preference

Returns

int value to subtract to preference

Definition at line 151 of file Prefs.cpp.

12.39.2.10 int Prefs::getIntIncValue (const std::string & *_key*)

getIntIncValue get value for incrementing given preference

Parameters

_key	name of preference

Returns

int value to add to preference

Definition at line 146 of file Prefs.cpp.

12.39.2.11 const std::map< std::string, std::pair< int, IncType > > & Prefs::getIntMap ( )

getIntMap retrieves the whole integer preference map

Returns

the map containing all of the integer preference key-value pairs

Definition at line 279 of file Prefs.cpp.

12.39.2.12 int Prefs::getIntPref (std::string *_key*)

getIntPref retrieves an integer preference with the given key

Parameters

_key	is the string key associated with the value

Returns

the value of the preference

Definition at line 209 of file Prefs.cpp.

12.39.2.13 int Prefs::getNumChangeablePrefs ( )

getNumChangeablePrefs get number of preferences that can be changed by the Gui

Returns

number of prefs that can be changed by the Gui

Definition at line 411 of file Prefs.cpp.

12.39.2.14 int Prefs::getNumPrefs ( )

getNumPrefs get total number of preference options

Returns

total number of preference options

Definition at line 406 of file Prefs.cpp.

12.39.2.15 std::string Prefs::getPrefValueString (const std::string & *_key*)

getPrefValueString get text for value of string

Parameters

_key	preferences key value

Returns

string representing value, eg "10" if value is 10

Definition at line 352 of file Prefs.cpp.

12.39.2.16 const std::map< std::string, std::pair< std::string, IncType > > & Prefs::getStrMap ( )

getStrMap retrieves the entire string preference map

Returns

the map containing all of the integer preference key-value pairs

Definition at line 289 of file Prefs.cpp.

12.39.2.17 std::string Prefs::getStrPref (std::string *_key*)

getStrPref retrieves a string preference with the given key

Parameters

_key	is the string key associated with the value

Returns

the value of the preference

Definition at line 219 of file Prefs.cpp.

12.39.2.18 PrefType Prefs::getTypeOfPref (const std::string & *_key*)

getTypeOfPref find what type the preference is

Parameters

_key	name of preference

Returns

INT, FLOAT or STRING PrefType enum

Definition at line 299 of file Prefs.cpp.

12.39.2.19 void Prefs::init (std::string *_pref_file* = "preferences.conf")

init initialises the Prefs class by running the PrefsParser to build the preferences from the given file

Parameters

_pref_file	is the file name of the preferences file

Definition at line 16 of file Prefs.cpp.

12.39.2.20 void Prefs::setFloatPref (std::string *_key*, float *_val*)

setFloatPref adds a float preference key value pair to the float preferences map

Parameters

_key	is the string key, which is the phrase to the left of the equals sign in the conf file
_val	is the float value to be asociated with the string key

Definition at line 194 of file Prefs.cpp.

12.39.2.21 void Prefs::setIntPref (std::string *_key*, int *_val*)

setIntPref adds an integer preference key value pair to the interger preferences map

Parameters

_key	is the string key, which is the phrase to the left of the equals sign in the conf file
_val	is the interger value to be asociated with the string key

Definition at line 189 of file Prefs.cpp.

12.39.2.22 void Prefs::setPref (std::string & *_key*, int *_val*)

setPref generic template for setting preference

Parameters

_key	is the string key
_val	is the value to be associated with the string key, either int, float or string

Definition at line 332 of file Prefs.cpp.

12.39.2.23 void Prefs::setStrPref (std::string *_key*, std::string *_val*)

setStrPref adds an integer preference key value pair to the interger preferences map

Parameters

_key	is the string key, which is the phrase to the left of the equals sign in the conf file
_val	is the interger value to be asociated with the string key

Definition at line 199 of file Prefs.cpp.

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

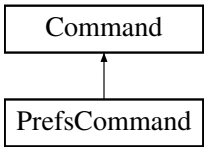
- include/Prefs.hpp
- src/Prefs.cpp

12.40 PrefsCommand Class Reference

The PrefsCommand class tell scene to show/hide preferences.

`#include <Commands.hpp>`

Inheritance diagram for PrefsCommand:



Public Member Functions

- PrefsCommand (Scene *_scene)

  PrefsCommand constructor for preferences command.
- virtual void execute ()

  execute send command to scene to toggle preferences

12.40.1 Detailed Description

The PrefsCommand class tell scene to show/hide preferences.

Definition at line 232 of file Commands.hpp.

12.40.2 Constructor & Destructor Documentation

12.40.2.1 PrefsCommand::PrefsCommand (Scene * *_scene*)

PrefsCommand constructor for preferences command.

Parameters

_scene	scene to send instruction to

Definition at line 72 of file Commands.cpp.

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

- include/Commands.hpp
- src/Commands.cpp

12.41 PrefsParser Class Reference

The PrefsParser class is resposible for reading in a preferences text file and parsing the text into integer, float and string preferences that it stores in the Prefs singelton class.

```
#include <PrefsParser.hpp>
```

Public Member Functions

- PrefsParser ()=default

  default constructor
- void parseFile (std::string _file_name)

  parseFile parses te given text file and stores the result in an instance of the Pefs singelton class

12.41.1 Detailed Description

The PrefsParser class is resposible for reading in a preferences text file and parsing the text into integer, float and string preferences that it stores in the Prefs singelton class.

The parsing is done using boost:spirit::classic which uses ruled defined in an EBNF format to decide what to do with different combinations of characters.

Definition at line 20 of file PrefsParser.hpp.

12.41.2 Member Function Documentation

12.41.2.1 void PrefsParser::parseFile (std::string *_file_name*)

parseFile parses te given text file and stores the result in an instance of the Pefs singelton class

Parameters

_file_name	is the path to the preferences.conf file

Definition at line 4 of file PrefsParser.cpp.

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

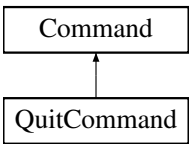
- include/PrefsParser.hpp
- src/PrefsParser.cpp

12.42 QuitCommand Class Reference

The QuitCommand class for quitting the game.

`#include <Commands.hpp>`

Inheritance diagram for QuitCommand:



Public Member Functions

- QuitCommand (Scene *_scene)

  QuitCommand constructor takes scene so it knows what to send "quit" message to.
- virtual void execute ()

  execute quits the given scene

12.42.1 Detailed Description

The QuitCommand class for quitting the game.

Definition at line 139 of file Commands.hpp.

12.42.2 Constructor & Destructor Documentation

12.42.2.1 QuitCommand::QuitCommand (Scene * *_scene*)

QuitCommand constructor takes scene so it knows what to send "quit" message to.

Parameters

_scene	scene which button refers to

Definition at line 18 of file Commands.cpp.

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

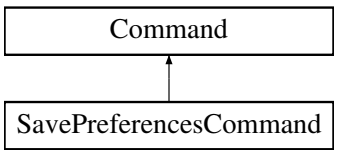
- include/Commands.hpp
- src/Commands.cpp

12.43 SavePreferencesCommand Class Reference

The SavePreferencesCommand class to write preferences out to config file.

`#include <Commands.hpp>`

Inheritance diagram for SavePreferencesCommand:



Public Member Functions

- SavePreferencesCommand ()
  SavePreferencesCommand constructor for save preferences command.
- virtual void execute ()
  execute writes out preferences to file

12.43.1 Detailed Description

The SavePreferencesCommand class to write preferences out to config file.

Definition at line 453 of file Commands.hpp.

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

- include/Commands.hpp
- src/Commands.cpp

12.44 Scene Class Reference

Public Member Functions

- Scene (ngl::Vec2 _viewport)
  constructor
- ~Scene ()=default
  deconstructor
- void draw ()
  draw
- void drawSky (bool _flipped=false)
  draws the sky
- void drawTerrain (bool _shouldClip=false)
  draws the terrain
- void drawMeshes ()
  draws the meshes
- void update ()
  update
- void quit ()
  quit close the program

- bool isActive ()

  isActive check whether program should still be running
- void mousePressEvent (const SDL_MouseButtonEvent &_event)

  Checks which mouse button has been pressed and sets relevenat mouse state to active.
- void mouseReleaseEvent (const SDL_MouseButtonEvent &_event)

  Checks which mouse button has been released and sets relevant mouse state to false, also calls mouseSelection for clicks.
- void wheelEvent (const SDL_MouseWheelEvent &_event)

  Sets the mouse pan according to the mouse wheel.
- void zoom (int _direction)

  zoom move the camera in or out
- void keyDownEvent (const SDL_KeyboardEvent &_event)

  keyDownEvent called when a key is pressed
- void keyUpEvent (const SDL_KeyboardEvent &_event)

  keyUpEvent called when a key is released
- void updateMousePos ()

  updateMousePos check where mouse is for gui
- void windowEvent (const SDL_WindowEvent &_event)

  windowEvent called upon window event such as resize to update resolution parameters
- void initialiseFramebuffers ()

  Initialises all of the framebuffers. Separated into its own unit so it can be called when the viewport resizes.
- void resize (const ngl::Vec2 &_dim)

  All the messy code to resize the viewport. Updates shaders, framebuffers etc.
- void centreCamera ()

  Centre camera on the active character.
- Character * getActiveCharacter ()

  getActiveCharacter get the active character in the scene
- std::string getActiveCharacterName ()

  getActiveCharacterName get the name of the active character
- void togglePause ()

  togglePause switch between paused and unpaused mode
- void startGame ()

  startGame leave main menu and start the game
- void startMove (Direction _d)

  startMove set movement flag in given direction to true
- void stopMove (Direction _d)

  stopMove set movement flag in given direction to false
- void prefsMode ()

  prefsMode show/hide preferences
- void escapeState ()

  escapeState leave current state
- GameState getState ()

  getState return the game state
- void focusCamToGridPos (ngl::Vec2 _pos)

  focusCamToGridPos send the target to the given position
- void baddiesSpawn ()

  baddiesSpawn manage baddie spawning
- void charactersSpawn ()

  charactersSpawn spawn characters at houses if there's space
- void endGame (const std::string &_message)

  endGame return to main menu

- int getPopulation ()

  getPopulation get the number of characters in the scene
- int getMaxPopulation ()

  getMaxPopulation get maximum possible population

12.44.1 Detailed Description

Definition at line 51 of file Scene.hpp.

12.44.2 Constructor & Destructor Documentation

12.44.2.1 Scene::Scene (ngl::Vec2 *_viewport*)

constructor

Parameters

_viewport	

Definition at line 29 of file Scene.cpp.

12.44.3 Member Function Documentation

12.44.3.1 void Scene::endGame (const std::string & *_message*)

endGame return to main menu

Parameters

_message	text for first button in main menu

Definition at line 2943 of file Scene.cpp.

12.44.3.2 void Scene::focusCamToGridPos (ngl::Vec2 *_pos*)

focusCamToGridPos send the target to the given position

Parameters

_pos	position for the camera to focus in on

Definition at line 2897 of file Scene.cpp.

12.44.3.3 Character * Scene::getActiveCharacter ( )

getActiveCharacter get the active character in the scene

Returns

a pointer to the active character

Definition at line 2724 of file Scene.cpp.

12.44.3.4 std::string Scene::getActiveCharacterName ( )

getActiveCharacterName get the name of the active character

Returns

active character's name, or an empty string if no active character

Definition at line 2740 of file Scene.cpp.

12.44.3.5 int Scene::getMaxPopulation ( ) [inline]

getMaxPopulation get maximum possible population

Returns

maximum population

Definition at line 212 of file Scene.hpp.

12.44.3.6 int Scene::getPopulation ( ) [inline]

getPopulation get the number of characters in the scene

Returns

size of the characters vector

Definition at line 207 of file Scene.hpp.

12.44.3.7 GameState Scene::getState ( )

getState return the game state

Returns

m_state

Definition at line 2892 of file Scene.cpp.

12.44.3.8 bool Scene::isActive ( )

isActive check whether program should still be running

Returns

m_active to see if program should quit if it's false

Definition at line 3002 of file Scene.cpp.

12.44.3.9 void Scene::keyDownEvent (const SDL_KeyboardEvent & *_event*)

keyDownEvent called when a key is pressed

Parameters

_event	SDL keyboard event structure

Definition at line 1937 of file Scene.cpp.

12.44.3.10 void Scene::keyUpEvent (const SDL_KeyboardEvent & *_event*)

keyUpEvent called when a key is released

Parameters

_event	SDL keyboard event structure

Definition at line 1967 of file Scene.cpp.

12.44.3.11 void Scene::mousePressEvent (const SDL_MouseButtonEvent & *_event*)

Checks which mouse button has been pressed and sets relevenat mouse state to active.

Parameters

_event,SDL	mouse event structure

Definition at line 1862 of file Scene.cpp.

12.44.3.12 void Scene::mouseReleaseEvent (const SDL_MouseButtonEvent & *_event*)

Checks which mouse button has been released and sets relevant mouse state to false, also calls mouseSelection for clicks.

Parameters

_event,SDL	mouse event structure

Definition at line 1890 of file Scene.cpp.

12.44.3.13 void Scene::startMove (Direction *_d*)

startMove set movement flag in given direction to true

Parameters

_d	direction to move

Definition at line 2829 of file Scene.cpp.

12.44.3.14 void Scene::stopMove (Direction *_d*)

stopMove set movement flag in given direction to false

Parameters

_d	direction to move

Definition at line 2834 of file Scene.cpp.

12.44.3.15 void Scene::wheelEvent (const SDL_MouseWheelEvent & *_event*)

Sets the mouse pan according to the mouse wheel.

Parameters

_event,SDL	mouse event structure

Definition at line 1918 of file Scene.cpp.

12.44.3.16 void Scene::windowEvent (const SDL_WindowEvent & *_event*)

windowEvent called upon window event such as resize to update resolution parameters

Parameters

_event	SDL window event

Definition at line 1996 of file Scene.cpp.

12.44.3.17 void Scene::zoom (int *_direction*)

zoom move the camera in or out

Parameters

_direction	direction, positive to zoom in, negative to zoom out

Definition at line 1931 of file Scene.cpp.

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

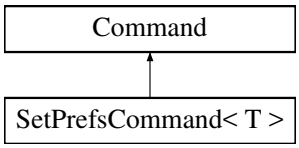
- include/Scene.hpp
- src/Scene.cpp

12.45 SetPrefsCommand< T > Class Template Reference

The SetPrefsCommand class used to set a preference.

`#include <Commands.hpp>`

Inheritance diagram for SetPrefsCommand< T >:



Public Member Functions

- SetPrefsCommand (const std::string &_key, const T &_val)

  SetPrefsCommand constructor for the set prefs command.
- virtual void execute ()

  execute send command to prefs to change one of the settings

12.45.1 Detailed Description

template<class T>class SetPrefsCommand< T >

The SetPrefsCommand class used to set a preference.

Definition at line 316 of file Commands.hpp.

12.45.2 Constructor & Destructor Documentation

12.45.2.1 template<class T > SetPrefsCommand< T >::SetPrefsCommand (const std::string & *_key*, const T & *_val*)

SetPrefsCommand constructor for the set prefs command.

Parameters

_key	preference key to use
_val	value to set it to

Definition at line 342 of file Commands.hpp.

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

- include/Commands.hpp

12.46 TerrainHeightTracer Class Reference

Public Member Functions

- **TerrainHeightTracer** (std::vector< ngl::Vec4 > _trimesh)
- float **getHeight** (double _x, double _y)

12.46.1 Detailed Description

Definition at line 7 of file TerrainHeightTracer.hpp.

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

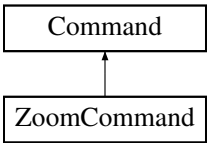
- include/TerrainHeightTracer.hpp
- src/TerrainHeightTracer.cpp

12.47 ZoomCommand Class Reference

The ZoomCommand class allows zooming in/out of a scene.

`#include <Commands.hpp>`

Inheritance diagram for ZoomCommand:



Public Member Functions

- ZoomCommand (Scene *_scene, int _direction)

  ZoomCommand constructor for zoom command.

- virtual void execute ()

  execute send command to scene to zoom

12.47.1 Detailed Description

The ZoomCommand class allows zooming in/out of a scene.

Definition at line 255 of file Commands.hpp.

12.47.2 Constructor & Destructor Documentation

12.47.2.1 ZoomCommand::ZoomCommand (Scene * *_scene*, int *_direction*)

ZoomCommand constructor for zoom command.

Parameters

_scene	scene to send instruction to
_direction	

Definition at line 85 of file Commands.cpp.

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

- include/Commands.hpp
- src/Commands.cpp

Chapter 13

File Documentation

13.1 include/AI.hpp File Reference

The AI refers to the grid for pathfinding and keeps track of a target for pathfinding.

```
#include "Grid.hpp"
#include "TerrainHeightTracer.hpp"
#include "ngl/Vec2.h"
#include <QTime>
#include <vector>
#include <SDL.h>
#include <iostream>
```

Classes

- class AI

  Parent class for ingame characters and enemies, containing position, states and targets.

13.1.1 Detailed Description

The AI refers to the grid for pathfinding and keeps track of a target for pathfinding.

13.2 include/Baddie.hpp File Reference

The enemy class that wanders around searching for characters, when one comes into range it follows and attacks the character.

```
#include "AI.hpp"
#include "ngl/Vec2.h"
#include "Grid.hpp"
#include "TerrainHeightTracer.hpp"
#include <vector>
```

Classes

- class Baddie

  The Baddie class is the ingame enemy, with attacking and tracking states.

13.2.1 Detailed Description

The enemy class that wanders around searching for characters, when one comes into range it follows and attacks the character.

13.3 include/Camera.hpp File Reference

The camera class is essentially a wrapper around a load of mat4s. Similar to NGLs default camera class but it represents both the camera and a lookat target. The user can transform both of these. When happy with the transformations they have set up, they can call methods to calculate the view and projection matrices.

```
#include <ngl/Mat4.h>
#include <ngl/Vec3.h>
#include <ngl/Vec2.h>
#include <vector>
#include "IVal.hpp"
```

Classes

- class Camera

13.3.1 Detailed Description

The camera class is essentially a wrapper around a load of mat4s. Similar to NGLs default camera class but it represents both the camera and a lookat target. The user can transform both of these. When happy with the transformations they have set up, they can call methods to calculate the view and projection matrices.

13.4 include/Character.hpp File Reference

Has multiple states for actions, responsible for updating itself.

```
#include "AI.hpp"
#include "Grid.hpp"
#include "Inventory.hpp"
#include "TerrainHeightTracer.hpp"
#include "ngl/Vec2.h"
#include <QTime>
#include <vector>
#include <stack>
#include <set>
#include <SDL.h>
```

Classes

- class Character

  Information for ingame characters, containing position, states and targets.

Enumerations

- enum State {
  CHOP_WOOD, **STORE**, **FISH**, **FORAGE**,
  CHECK_WOOD, **CHECK_BERRIES**, **CHECK_FISH**, **GET_WOOD**,
  GET_BERRIES, **GET_FISH**, **BUILD**, **SLEEP**,
  EAT_BERRIES, **EAT_FISH**, **MOVE**, **TRACK**,
  FIGHT, **REPEAT**, **IDLE** }

  The State enum, used for stack of states that are handled internally in a switch statment.
- enum CharInventory { **WOOD**, **FISH**, **BERRIES**, **NONE** }

  The CharInventory enum, used to define what the character is holding.

13.4.1 Detailed Description

Has multiple states for actions, responsible for updating itself.

13.5 include/Grid.hpp File Reference

header file for the Grid class

```
#include <vector>
#include <ngl/Vec2.h>
#include <ngl/Vec3.h>
#include "GridTile.hpp"
#include "Inventory.hpp"
```

Classes

- class Grid

  The Grid class holds information about what is contained in each cell of the map. The Grid class is a wrapper around a std::vector of Tile enums. The Tile enums illustrate what is contained within each cell of the map, which can be used by other classes for rendering and path finding.

13.5.1 Detailed Description

header file for the Grid class

13.6 include/GridTile.hpp File Reference

header file for the GridTile class

```
#include "ngl/Vec2.h"
#include <vector>
```

Classes

- class GridTile

  stores all of the data associated with each tile in the map

Enumerations

- enum **TileType** {
  NONE, **TREES**, **STUMPS**, **WATER**,
  MOUNTAINS, **HOUSE**, **STOREHOUSE**, **FOUNDATION_A**,
  FOUNDATION_B, **FOUNDATION_C**, **FOUNDATION_D** }

13.6.1 Detailed Description

header file for the GridTile class

13.7 include/Gui.hpp File Reference

The Gui is used for user interaction, and managing and drawing the buttons.

```
#include <vector>
#include <memory>
#include "ngl/Singleton.h"
#include "ngl/Vec2.h"
#include "Button.hpp"
#include "Commands.hpp"
#include "Scene.hpp"
```

Classes

- class Gui
  The Gui class contains button positions and managing their use.

Variables

- constexpr unsigned int BUTTON_TEXT_LENGTH = 1024
  BUTTON_TEXT_LENGTH.

13.7.1 Detailed Description

The Gui is used for user interaction, and managing and drawing the buttons.

13.8 include/Inventory.hpp File Reference

The world inventory, globally shared across storehouses.

Classes

- class Inventory
  The Inventory class for management of the global inventory as accessed by the character through storehouses.

13.8.1 Detailed Description

The world inventory, globally shared across storehouses.

13.9 include/IVal.hpp File Reference

When doing a smooth interpolation between two values, I usually have to define them in a header somewhere and then write cur += (targ - cur) / x This gets pretty tiresome and clutters the code up, especially in Scene.hpp which until recently suffered from my excessive variables used to track the camera transformation. I figure that wrapping this all up into a class lets me clean this up a bit.

Classes

- class IVal< T >

13.9.1 Detailed Description

When doing a smooth interpolation between two values, I usually have to define them in a header somewhere and then write cur += (targ - cur) / x This gets pretty tiresome and clutters the code up, especially in Scene.hpp which until recently suffered from my excessive variables used to track the camera transformation. I figure that wrapping this all up into a class lets me clean this up a bit.

13.10 include/Light.hpp File Reference

This class acts as a simple point light.

```
#include <ngl/Vec3.h>
```

Classes

- struct Light

13.10.1 Detailed Description

This class acts as a simple point light.

Author

Ben Hawkyard

Version

1.0

Date

19/01/17 Revision History : This is an initial version used for the program.

13.11 include/MapList.hpp File Reference

```
#include <vector>
#include <string>
#include "ngl/Singleton.h"
```

Classes

- class MapList

  The MapList class.

13.11.1 Detailed Description

contains MapList class

Author

Felix

13.12 include/Node.hpp File Reference

Utility class used for pathfinding, holds information position, cost, parent node etc.

```
#include "Grid.hpp"
```

Classes

- class Node

  Pathfinding node class.

Enumerations

- enum **Neighbour** { **UP**, **DOWN**, **LEFT**, **RIGHT** }

13.12.1 Detailed Description

Utility class used for pathfinding, holds information position, cost, parent node etc.

13.13 include/NodeNetwork.hpp File Reference

The NodeNetwork can be created as a temporary helper object for finding a path. All pathfinding logic is contained within it.

```
#include "Node.hpp"
#include "Grid.hpp"
#include <vector>
#include "ngl/Vec2.h"
```

Classes

- class NodeNetwork

  Wraps up a Node vector and uses it to find a path on the Grid object given in its constructor.

13.13.1 Detailed Description

The NodeNetwork can be created as a temporary helper object for finding a path. All pathfinding logic is contained within it.

13.14 src/Grid.cpp File Reference

source code for the Grid class

```
#include <iostream>
#include <iomanip>
#include <string>
#include <fstream>
#include <streambuf>
#include <Python.h>
#include "ngl/Random.h"
#include "ngl/NGLStream.h"
#include "Grid.hpp"
#include "Prefs.hpp"
#include "Utility.hpp"
```

13.14.1 Detailed Description

source code for the Grid class

Index