20.1.2019

PseudoChess

Documentation v.0.895432

Oliver Ruane
FRESHDUMB ENTERTAINMENT

Table of content

Overview

Two player game where both players first choose an individual starting position on a 10x10 board. The game then spawns 5 Units of a random type for each Player around these starting positions.

After that the Players take turns in moving their Units. There is a coin Pickup placed randomly on the board and the Players try to collect this Pickup, rewarding them with +1 to their Score. Then a new coin Pickup is spawned randomly.

The Player to first reach a Score of 10 wins the match and the game restarts.

Classes and Methods

1. Board

1.1. Tile Class

Description

Stores basic information of a simple game Tile.

Fields

Type	Name	Notes	Default Value	Scope
int	xIndex, yIndex	Stores the Indices of the Tile on the Array	0, 0	private
Unit*	`unitOnTile`	Stores Pointer to the Unit placed on the Tile *nullptr* when no Unit on Tile	*nullptr*	private
Pickup*	`pickupOnTile`	Stores Pointer to the Pickup placed on the Tile *nullptr* when no Pickup on Tile	*nullptr*	private

Methods

getUnitOnTile

inline Unit* getUnitOnTile() const { return unitOnTile; }

Description

Returns the field *unitOnTile*.

---

setUnitOnTile

inline void setUnitOnTile(Unit* _unit) { unitOnTile = _unit; }

Parameters

_unit	Unit to set unitOnTile

Description

Sets the field *unitOnTile* with the given Unit* from the Parameters.

---

getPickupOnTile

inline Pickup* getPickupOnTile() const { return pickupOnTile; }

Description

Returns the field *pickupOnTile*.

---

setPickupOnTile

inline void setPickupOnTile(Pickup* _pickup) { pickupOnTile = _pickup; }

Parameters

_pickup	Pickup to set pickupOnTile

Description

Sets the field *pickupOnTile* with the given Pickup* from the Parameters.

---

getXIndex

inline int getXIndex() const { return xIndex; }

Description

Returns the field *xIndex*.

---

getYIndex

inline int getYIndex() const { return yIndex; }

Description

Returns the field *yIndex*.

---

getIndex

inline Vector2 getIndex() const { return Vector2(xIndex, yIndex); }

Description

Returns the Indices *xIndex* and *yIndex* as a Vector2.

1.2. TileManager Class

Description

Stores basic information of a simple game Tile.

Fields

Type	Name	Notes	Default Value	Scope
Tile Array	Tiles	Stores all the tile objects of the board. Two-dimensional array with a size of BOARDSIZE * BOARDSIZE	-	private
const int	xBoardSize	Stores the horizontal size of the board.	*BOARDSIZE*	public
const int	yBoardSize	Stores the vertical size of the board.	*BOARDSIZE*	public

Methods

InitializeTiles

void InitializeTiles();

Description

Initializes the tile array Tiles with Tiles. Sets the *xIndex* and *yIndex* of the Tile to the appropriate values.

---

setUnitOnTile

Tile* getTile(int _xPos, int _yPos);

Parameters

_xPos	Horizontal position of the Tile
_yPos	Vertical position of the Tile

Description

Returns a Pointer to a Tile from the Tile Array.
Look for a Tile at Tiles[_xPos][_yPos] and returns the equivalent pointer. If the coordinates given are out of bounds returns *nullptr*.

---

placeGamePiece

void placeGamePiece(GamePiece& _gamePiece, int _xPos, int _yPos);

Parameters

_xPos	Horizontal position of the Tile
_yPos	Vertical position of the Tile
_gamePiece	Reference to the GamePiece to be placed on the Tile

Description

Places a GamePiece on a Tile by calling the *spawnGamePiece()* method from the GamePiece.

---

getClosestTile

Vector2 getClosestTile(int _xPixelPos, int _yPixelPos) const;

Parameters

_xPixelPos	Horizontal position of a pixel in the window
_xPixelPos	Horizontal position of a pixel in the window

Description

Returns a vector2 corresponding to the Tile that is closest to given pixel coordinates.
Used to get the closest Tile from the mouse curser.

---

findSpot

Vector2 findSpot(const Vector2& _originPos) const;

Parameters

_originPos	Starting position from where to look for a spot

Description

Returns one of the 8 fields around the _originPos on the board randomly, as a Vector2.
NOTE: Can return a Vector2 that is out of bounds of the Tile Array.

---

isInBounds (+1 overload)

bool isInBounds(const Vector2& _position) const;
bool isInBounds(const int _xPos, const int _yPos) const;

Parameters

_position **||** _xPos, _yPos	Position to check for out of bounds

Description

Checks if the given Vector2 (or individual X and Y positions) is in range of the Tiles Array and returns true if that's the case. Else it returns false.

---

isInBoundsAndNotEnemy

bool isInBoundsAndNotEnemy(const int _xPos, const int _yPos, const int _playerID) const;

Parameters

_xPos	X Position of the Tile to check
_yPos	Y Position of the Tile to check
_playerID	Player ID to check against enemies

Description

Checks if the tile given by its position is in bounds of the Tile Array. Then checks if the tile is occupied by a Unit with a different playerID than _playerID.

---

isFieldOccupied (+1 overload)

bool isFieldOccupied(Vector2 _position) const;
bool isFieldOccupied(const int _xPos,const int _yPos) const;

Parameters

_position **||** _xPos, _yPos	Position to check for out of bounds

Description

Checks if the tile given by its position (either as Vector2 or individual coordinates) is occupied by a Unit. If no Unit is on the tile returns false. If there is a Unit, calls *doesOccupieField()* from the Unit and returns its return value.

1.3. Vector2 Class

Description

Basic class to store scalars in a two dimensional Vector.

Fields

Type	Name	Notes	Default Value	Scope
int	xPos	X Scalar of the Vector	0	public
int	yPos	Y Scalar of the Vector	0	public

Methods

normalize

Vector2 normalize() const;

Description

Returns the normalized Vector2.
NOTE: Returns *this* Vector2 with ROUNDED X and Y Scalars meaning this function is not Mathematically correct as it does not always return a Vector with the length of 1.

---

operator+

inline Vector2 operator+(const Vector2 &B) const { … }

Parameters

B	Second Vector2 to add to *this* Vector

Description

Adds the Vector2 B to *this* Vector2 component-by-component and returns the result as a Vector2.

---

operator!=

inline bool operator!= (const Vector2 &B) const { return !(*this == B); }

Parameters

B	Second Vector2 to check for equality

Description

Checks if the Vector2 B is identical to *this* Vector2 using the operator== method and returns true if they are not.

---

operator==

inline bool operator== (const Vector2 &B) const { … }

Parameters

B	Second Vector2 to check for equality

Description

Checks if the Vector2 B is identical to *this* Vector2 component-by-component and returns true if they are identical.

---

operator- (+1 overload)

inline Vector2 operator-(const Vector2&B) const … }

inline Vector2 operator-() { … }

Parameters

B **||** NO PARAMETER	Vector2 to deduct from *this* Vector2

Description

Deducts the Vector2 B from *this* Vector2 component-by-component and returns the result as a Vector2. If no parameters are given *this* Vectors2 scalars are inverted component-by-component and returned as a Vector2

---

2. Game

2.1. Game Class

Description

Class that stores and handles information considering the Gameloop.
Also contains pointers to the other systems, like rendering and input handling.

Fields

Type	Name	Notes	Scope
int	Round	Current Round of the Game	private
Player*	Players	Array containing the Players of the Game	private
bool	isGameOver	Boolean value showing if the current match has ended by one of the players having won the match	private
bool	playerConfirmed	Boolean value to handle the hotseat mode False after every turn and waits for the player to confirm he is ready	private
int	currentActivePlayerID	ID of the Player whose turn it currently is Index of the Player object on the Player Array	private
Renderer*	rendererRef	Pointer to the Renderer object dealing with rendering the Game	private
bool	hasPickup	Boolean to handle the Pickup in the Game If false Game spawns another Pickup	private
TileManager	tileManager	TileManager object handling the board	private
gamePickup	gamePickup	Pointer to the Pickup object that needs to be collected from the Players	private
InputManager*	InputManagerRef	Pointer to the InputManager object handling input	private

Methods

isTheGameOver

inline bool isTheGameOver() const { return isGameOver; }

Description

Returns the value of *isGameover* field.

---

getActivePlayerID

inline int getActivePlayerID()const { return currentActivePlayerID; }

Description

Returns the ID of the Player whose turn is currently is.

---

hasPlayerConfirmed

inline bool hasPlayerConfirmed()const { return playerConfirmed; }

Description

Return the value of the *playerConfirmed* field.

---

getPlayers

inline Player* getPlayers() const { return Players; }

Description

Returns the Player Array containing all the Player objects.

---

getPickupPos

inline Vector2 getPickupPos() const { return gamePickup->getPosition(); }

Description

Returns the position of the *gamePickup* object.

---

gameHasPickup

inline bool gameHasPickup() const { return hasPickup; }

Description

Returns true if there is a Pickup on the board.

---

gameQuit

inline bool gameQuit() { return InputManagerRef->hasQuit(); }

Description

Returns true when the Game was quit via the close button of the window.

---

initGame

void initGame(SDL_Renderer* _rendererRef, SDL_Window* _windowRef);

Parameters

_rendererRef	Pointer to the SDL_Renderer needed to render the game
_windowRef	Pointer to the SDL_Window needed to render the game

Description

Initializes the game.
Creates *Players* and adds 5 *Units* of a random *UNIT_TYPE* to the *Players*. Sets *round* to 0 and resets the *srand()* seed.
Loads textures and font and instantiates the Renderer object.

---

restartGame

void restartGame();

Description

Resets all relevant values to the initial values to allow another match to be played after confirmation.

---

doZeroRound

void doZeroRound();

Description

Handles the “zero Round” where players can choose a starting position for their *Units*. After choosing a position Units are randomly spawned on and around the chosen position.

---

update

void update();

Description

Gameloop method being called every frame. Handles player input and rendering of the game via calls to the *Renderer* and *InpuManager*.

---

waitForPlayerConfirm

void waitForPlayerConfirm();

Description

Method being called after every turn waiting for a player to confirm with the ENTER key, after which the

game can continue.

---

doTurn
void doTurn();

Description
Handles the turn of a player, like selecting and moving Units, as well as collecting Pickups and checking for win condition.

---

spawnPickup
void spawnPickup();

Description
Spawns a Pickup at a random Tile not being occupied by a Unit.

---

isNewRound
bool isNewRound() const;

Description
Returns true when both players have performed their turn. This is the case when both Players have a internal *PlayerRound* differing from the *Round* of the Game.

2.2. GamePiece Interface

Description

Class handling Unit mechanics, such as moving patterns etc.

Fields

Type	Name	Notes	Scope
Vector2	position	Position of the GamePiece	private
bool	occupiesField	Stores whether the GamePiece occupies a Tile	private

Methods

setPosition(+1 overload)

inline void setPosition(int _xPosition, int _yPosition) { position = Vector2(_xPosition, _yPosition); }
inline void setPosition(Vector2 _position) { position = _position; }

Parameters

_xPosition **||** _position	X coordinate to set **||** Vector2 to set
_yPosition	Y coordinate to set

Description

Sets the *position* of *this* GamePiece.

---

setOccupiesField

inline void setOccupiesField(bool _value) { occupiesField = _value; }

Parameters

_value	Bool value to set

Description

Sets the Value of *occupiesField*.

---

doesOccupieField

inline bool doesOccupieField() const { return occupiesField; }

Description

Returns the value of *occupiesField*.

---

getPosition

inline Vector2 getPosition() const { return position; }

Description

Returns the value of *position*.

---

getXPos

inline int getXPos() const { return position.xPos; }

Description

Returns the *xPos* value of *position*.

---

getYPos

inline int getYPos() const { return position.xPos; }

Description

Returns the *yPos* value of *position*.

---

spawnGamePiece

virtual void spawnGamePiece(const Vector2& _position, TileManager& _tileManager) = 0;

Description

Purely virtual function *spawnGamePiece*.

---

isOnBoard

bool isOnBoard();

Description

Returns true when the *position* of *this* Unit is not equal to Vector2(-1,-1).

2.3. Pickup Class

Description

Class handling Pickup mechanics. Implements the GamePiece Interface.

Fields

Type	Name	Notes	Scope
View GamePiece Interface for more Info			

Methods

doPickUp

int doPickUp(Player& _player, TileManager& _tileManager);

Parameters

_player	Reference to the Player picking up the Pickup
_tileManager	Referance to the TileManager

Description

Removes the Pickup from the relevant Tile contained in _tileManager and increments the Score of *_player*.

---

spawnGamePiece

void spawnGamePiece(const Vector2& _position, TileManager& _tileManager);

Parameters

_tileManager	Reference to the TileManager containing the Tiles Array to place *this* Pickup onto
_Position	Position on the Tiles Array around which *this* Unit is placed

Description

Places *this* Pickup onto the Tile Array contained in _tileManager.

2.4. Player Class

Description

Player Class deals with initializing Units and storing Player specific information.

Fields

Type	Name	Notes	Scope
Int	playerRound	Number of turn *this* Player has taken	private
Unit*	selectedUnit	The Unit *this* Player has currently selected nullptr when none selected	private
int	score	Current score of *this* Player	private
Vector2	startingPos	Initial Position of the Player around which Units are initialized	private
std::vector<Unit>	PlayerUnits	List containing the Units of *this* Player	public

Methods

setStartingPos

inline void setStartingPos(const Vector2 _position) { startingPos = _position; }

Parameters

_position	Position to set

Description

Sets the value of *startingPos*.

---

getSelectedUnit

inline Unit* getSelectedUnit() { return selectedUnit; }

Description

Gets the value of *selectedUnit*.

---

setSelectedUnit

inline void setSelectedUnit(Unit* _unit) { selectedUnit = _unit; }

Parameters

_unit	Unit to set

Description

Sets the value of *selectedUnit*.

---

incrementScore

inline void incrementScore() { score++; }

Description

Increments the *score* of *this* Player by 1.

---

getScore

inline int getScore() const { return score; }

Description

Returns the value of *score*.

---

setPlayerRound

inline void setPlayerRound(const int _PlayerRound) { playerRound = _PlayerRound; }

Parameters

_PlayerRound	Int to set

Description

Sets the value of *playerRound*.

---

incrementPlayerRound

inline void incrementPlayerRound() { playerRound++; }

Description

Increments the value of *playerRound* by 1.

---

getPlayerRound

inline int getPlayerRound() const { return playerRound; }

Description

Returns the value of *playerRound*.

---

initUnits

void initUnits(TileManager& _tileManager);

Parameters

_tileManager	Reference to the TileManager

Description

Initializes the *PlayerUnits* via the _tileManager.

---

resetPlayer

void resetPlayer(TileManager& _tileManager);

Parameters

_tileManager	Reference to the TileManager

Description

Resets *this* Player to the initial stage, so the game can restart.

2.5. Unit Class

Description

Class handling Unit mechanics, such as moving patterns etc. Implements the GamePiece Interface.

Fields

Type	Name	Notes	Scope
int	controllingPlayerID	Index of the player controlling the Unit	private
bool	isSelected	Stores whether the Unit is selected or not	private
UNIT_TYPE	unitType	What kind of Unit *this* Unit is	private

Methods

getUnitType

inline UNIT_TYPE getUnitType()const { return unitType; }

Description

Returns the UNIT_TYPE of *this* Unit.

---

selectUnit

inline void selectUnit() { isSelected = true; }

Description

Sets *isSelected* to true.

---

deSelectUnit

inline void deSelectUnit() { isSelected = false; }

Description

Sets *isSelected* to false.

---

isUnitSelected

inline bool isUnitSelected() const { return isSelected; }

Description

Returns the Value of *isSelected*.

---

getControllingPlayerID

inline int getControllingPlayerID() const { return controllingPlayerID; }

Description

Returns the *controllingPlayerID*.

---

spawnGamePiece

void spawnGamePiece(const Vector2& _Position, TileManager& _tileManager);

Parameters

_tileManager	Reference to the TileManager containing the Tiles Array to place *this* Unit onto
_Position	Position on the Tiles Array around which *this* Unit is placed

Description

Places *this* Unit onto the Tile Array contained in _tileManager.
Checks if the _Position is occupied and tries to place Unit if not.
If it is occupied by another Unit it searches for an adjacent free Tile and places *this* Unit there.

---

checkMovement

bool checkMovement(const Vector2& _targetPosition, TileManager& _tileManager) const;

Parameters

_targetPosition	Desired move destination of *this* Unit
_tileManager	Reference to the TileManager object containing relevant information

Description

Depending on the *UNIT_TYPE* of *this* Unit calls the equivalent *UNIT_TYPEMovement()* method.

UNIT_TYPEMovement

bool pawnMovement(const Vector2& _targetPosition, TileManager& _tileManager) const;
bool kingMovement(const Vector2& _targetPosition, TileManager& _tileManager) const;
bool rookMovement(const Vector2& _targetPosition, TileManager& _tileManager) const;
bool rogueMovement(const Vector2& _targetPosition, TileManager& _tileManager) const;
bool bishopMovement(const Vector2& _targetPosition, TileManager& _tileManager) const;;

Parameters

_targetPosition	Desired move destination of *this* Unit

_tileManager	Reference to the TileManager object containing relevant information

Description

Checks if the Tile at _targetPosition is a valid spot for the specific movement pattern of *this* *UNIT_TYPE*.
Also checks if the Tile at _targetPosition is occupied and returns true if the Tile is valid and not occupied.

---

moveUnit

void moveUnit(const Vector2& _moveTarget, TileManager& _tileManager);

Parameters

_moveTarget	Move destination of *this* Unit
_tileManager	Reference to the TileManager object containing relevant information

Description

Moves *this* Unit to the Tile at _moveTarget.
Sets the unitOnTile on the Tile at the old Position to nullptr.

---

3. InputManager Class

Description

Manages Input events and stores them to be accessed by the Game Class.

Fields

Type	Name	Notes	Scope
Vector2	mousPos	Stores the current pixelposition of the cursor	private
SDL_bool	quit	Stores whether the close button on the window has been clicked	private
SDL_Event	Event	Event object storing event information	private
bool	leftButtonDown	Boolean representing the state of the left mouse button	private
bool	enter	Boolean storing true on Enter keypress until press was processed	public

Methods

pollEvents

void pollEvents();

Description

Polls the SDL_Events from the SDL eventsystem to be processed.

---

getMouseAxis

inline Vector2 getMouseAxis() { return mousePos; }

Description

Returns the value of the field *mousePos*.

---

getMouseButtonDown

inline bool getMouseButtonDown() { return leftButtonDown; }

Description

Returns the value of the field *leftButtonDown*.

---

hasQuit

inline SDL_bool hasQuit() { return quit; }

Description

Returns the value of the field *quit*.

---

4. Renderer Class

Description

Class containing textures to render and handling rendering of the game.

Fields

Type	Name	Notes	Scope
static const int	BorderSize	Size of the border around the board in pixels	private
SDL_Renderer*	rendererRef	Pointer to the SDL_Renderer needed to render the game	private
SDL_Window*	windowRef	Pointer to the SDL_Window needed to render the game	private
UILoader*	UILoaderRef	Pointer to the UILoader object	private
SDL_Texture*	PawnImg	Pointer to Texture of the Pawn Unit	private
SDL_Texture*	RogueImg	Pointer to Texture of the Rogue Unit	private
SDL_Texture*	RookImg	Pointer to Texture of the Rook Unit	private
SDL_Texture*	KingImg	Pointer to Texture of the King Unit	private
SDL_Texture*	BishopImg	Pointer to Texture of the Bishop Unit	private
SDL_Texture*	CoinImg	Pointer to Texture of the Coin Pickup	private
SDL_Texture*	TargetImg	Pointer to Texture of the Tile targeting UI symbol	private

Methods

renderPresent

void renderPresent();

Description

Calls the *SDL_RenderPresent()* method passing the *rendererRef*.

---

drawTiles

void drawTiles(Color _color, TileManager& _tileManager)const;

Parameters

_tileManager	Reference to the TileManager containing the Tiles Array to be drawn

Description

Draws the *Tiles* from the given _tileManager by subsequent calls of the *drawSingleTile()* method.

---

drawBorder

void drawBorder(Color _color)const;

Parameters

_color	Color of the border

Description

Draws a border around the board in the given color.

---

drawSingleTile

void drawSingleTile(const Color _color, const Vector2 _position)const;

Parameters

_color	Color of the border
_position	Position used to calculate where to draw the Tile

Description

Draws a single Tile.

---

drawBoard

void drawBoard(const TileManager& _tileManager)const;

Parameters

_tileManager	Reference to the TileManager

Description

Draws the Board by calling the *drawBorder()* method and the *drawTiles()* method.

---

drawUnits

void drawUnits(const Game& _gameRef)const;

Parameters

_gameRef	Reference to the Game object containing Unit information

Description

Draws all the Units from every Player contained in *_gameRef* by multiple calls of the *drawSingleUnit()* method.

---

drawPickup

void drawPickup(const Game& _gameRef) const;

Parameters

_gameRef	Reference to the Game object containing Pickup information

Description

Draws the Pickup in *_gameRef*.

---

drawSingleUnit

void drawSingleUnit(const Unit& _unitToDraw) const;

Parameters

_unitToDraw	Reference to the Unit object to draw

Description

Draws a Texture depending on the *UNIT_TYPE* and *Position* of the *_unitToDraw*.

---

drawUnitMovement

void drawUnitMovement(const Unit& _Unit, TileManager& _tileManager) const;

Parameters

_Unit	Reference to the Unit object selected
_tileManager	Reference to the TileManager object containg Unit information needed

Description

Depending on *UNIT_TYPE* of the _Unit calls the required *drawUNIT_TYPEMovementTiles()* Method.

---

drawUNIT_TYPEMovement

void drawPawnMovementTiles (const Unit& _unit, TileManager& _tileManager)const;
void drawKingMovementTiles (const Unit& _unit, TileManager& _tileManager)const;
void drawRogueMovementTiles (const Unit& _unit, TileManager& _tileManager)const;
void drawRookMovementTiles (const Unit& _unit, TileManager& _tileManager)const;
void drawBishopMovementTiles (const Unit& _unit, TileManager& _tileManager)const;

Parameters

_Unit	Reference to the Unit object selected
_tileManager	Reference to the TileManager object containg Unit information needed

Description

Depending on *UNIT_TYPE* of the _Unit an overlay is drawn to show possible movement of the *_Unit*.

---

drawEverything

void drawEverything(const Game& _gameRef, TileManager& _tileManager)const;

Parameters

_gameRef	Reference to the Game object containing information needed
_tileManager	Reference to the TileManager object containg Unit information needed

Description

Main method dealing with drawing of everything. Calls other methods of the Renderer to draw the Board, UI, Units, Pickups and pop-ups.

---

createTextures

void createTextures();

Description

Creates SDL_Textures* from the SDL_Surfaces* in the UILoader and stores them, to be used in Rendering.
Frees the SDL_Surfaces memory afterwards by calling the *SDL_FreeSurface()* method.

---

drawUIArea

void drawUIArea()const;

Description

Draws the basic UI Area right hand side of the board.

---

drawUIArea

void drawPlayerReadyUI(const Game& _gameRef) const;

Parameters

_gameRef	Reference to the Unit object selected

Description

Draws a pop-up window asking a Player to confirm when he is ready.

---

drawUIElement (+1 overload)

void drawUIElement(SDL_Texture* _textureToRender, int _xPos, int _yPos, int _width, int _height) const;
void drawUIElement(const std::string& input, int _xPos, int _yPos, int _width, int _height) const;

Parameters

_textureToRender **||** input	UI Texture **or** Text to render
_xPos	X Position of UI Element in pixels
_yPos	Y Position of UI Element in pixels
_width	Width of UI Element in pixels
_height	Height of UI Element in pixels

Description

Draws a given Texture **or** Text at the *_xPos* and *_yPos* with a *_width* and *_height*.

---

drawTargetUI

void drawTargetUI(int _xPos, int _yPos, int _width, int _height) const;

Parameters

_xPos	X Position of UI Element in pixels
_yPos	Y Position of UI Element in pixels
_width	Width of UI Element in pixels
_height	Height of UI Element in pixels

Description

Draws the Target UI at the *_xPos* and *_yPos* with a *_width* and *_height* by calling the *drawUIElement()* method passing the *TargetImg*.

---

drawScore

void drawScore(const Game& _gameRef) const;

Parameters

_gameRef	Reference to the Game object containing player scores

Description

Draws the Players scores in the UI Area on the right hand side.

---

drawTurnIndicator

void drawTurnIndicator(const Game& _gameRef)const;

Parameters

_gameRef	Reference to the Game object

Description

Draws an indicator showing whose players turn it currently is in the UI Area on the right hand side.

---

drawWinScreen

void drawWinScreen(const Game& _gameRef)const;

Parameters

_gameRef	Reference to the Game object

Description

Draws a win screen for the relevant player.

---

5. UILoaderClass

Description

Class containing textures to render and handling rendering of the game.

Fields

Type	Name	Notes	Scope
SDL_Surface*[5]	UnitSurf	SDL_Surfaces containing the Unit textures	public
SDL_Surface*[1]	PickupSurf	SDL_Surfaces containing the Pickup textures	public
SDL_Surface*	TargetSurf	SDL_Surface containing the Target texture	public
TTF_Font*	font	Contains the text font	public

Methods

loadTexturesToSurface

void loadTexturesToSurface();

Description

Loads the .bmp files as SDL_Surfaces and stores them.

---

loadFont

void loadFont(const char* _fontLocation);

Parameters

_fontLocation	Path to the font data

Description

Loads the .font and stores it.

---

getTextSurface

SDL_Surface* getTextSurface(const char* _text);

Parameters

_text	Text to print to surface

Description

Returns a Pointer to a SDL_Surface with the _text written on.

Class Diagram