Designed by: struct_by_lighting{};

Module: graphics

Responsible for all graphical representation on the screen, such as loading media files, setting up coordinates for the game layout, and positioning images throughout the screen accordingly.

--- Interface ---

Uses:

SDL2/SDL.h
string
board.h
linkedList.h

Constants:

CONNECT4_WINDOW_OFFSET_Y: int
Prevents the application window to open beyond the boundaries of the client's screen

SCALE: float
Allows the game screen to be scaled

SCREEN_WIDTH: int
Window width size for the game

SCREEN_HEIGHT: int
Window height size for the game

TOKEN_WIDTH: int
Width of the tokens used for the game

TOKEN_HEIGHT: int
Height of the tokens used for the game

GRID_OFFSET_Y: int
Offset of column lines for the game playing board

GRID_OFFSET_X: int
Offset of row lines for the game playing board

GRID_WIDTH: int
Width of the game board

GRID_HEIGHT: int
Height of the game board

MAINMENU_SETUP_BUTTON_LEFT: int
Calculates the left position the setup button

MAINMENU_SETUP_BUTTON_RIGHT: int
Calculates the right position the setup button

MAINMENU_SETUP_BUTTON_TOP: int
Calculates the top position the setup button

MAINMENU_SETUP_BUTTON_BOTTOM: int
Calculates the bottom position the setup button

MAINMENU_QUIT_BUTTON_LEFT: int
Calculates the left position the quit button

MAINMENU_QUIT_BUTTON_RIGHT: int
Calculates the right position the quit button

MAINMENU_QUIT_BUTTON_TOP: int
Calculates the top position the quit button

MAINMENU_QUIT_BUTTON_BOTTOM: int
Calculates the bottom position the quit button

SETUP_BOTTOM_BUTTONS_OFFSET: int
Calculates the bottom position of all the buttons in the bottom of the setup game mode screen

SETUP_CLICKY_TOKENS_OFFSET: int
Calculates the position of the button that changes the current token colour in play(ie. Going from blue's turn to red's turn and vice versa)

SETUP_2PLAYER_BUTTON_WIDTH: int
Calculates the width of the '2 player' game mode button within the setup screen

SETUP_2PLAYER_BUTTON_HEIGHT: int
Calculates the height of the ‘2 player’ game mode button within the setup screen

SETUP_1PLAYER_BUTTON_WIDTH: int
Calculates the width of the ‘1 player’ game mode button within the setup screen

SETUP_1PLAYER_BUTTON_HEIGHT: int
Calculates the height of the ‘1 player’ game mode button within the setup screen

SETUP_MENU_BUTTON_WIDTH: int
Calculates the width of the ‘menu” button within the setup screen

SETUP_MENU_BUTTON_HEIGHT: int
Calculates the height of the ‘menu” button within the setup screen

Global Declarations:

extern List<FallingToken> *gFallingTokens
A list that represents all the falling tokens during gameplay

Defined Structures:

FallingToken
Holds information about each token, which will help gravity to be simulated on each token individually.

TextureWrapper
Provides a way to store dimensions to a texture.

TokenLocation   (incomplete type)
Allows a tokens to be defined by its color and position within the game board.
(the structure definition for this struct is visible in the interface)

```
struct TokenLocation {
   int row;
   int column;
  Token colour;
};
```

Access programs:

drawFallingToken:
Return Type: void
Parameters: FallingToken *token
Given a token, determine the position to drop it.

clearFallingToken:
Return Type: void
Parameters: FallingToken *fallingToken
Finds the position of the falling token.

updateFallingToken:
Return Type: void
Parameters: FallingToken *fallingToken, float dt
*Updates the position of a falling token depending on the time (*dt*) it has been airborne.*

mainMenuRender:
Return Type: void
Parameters: none
Rendering for the main menu

transitionSetupRender:
Return Type: void
Parameters: none
Positions and renders the buttons to be used in the setup game mode to allow players to either start a game from setup mode, or return to main menu.

setupRender:
Return Type: void
Parameters: none
Rendering for the setup game mode

init:
Return Type: boolean value
Parameters: none
Returns true if the program has been initialized and window has been created successfully.

loadMedia:
Return Type: boolean value
Parameters: none
Returns true if all media (images) required for the game has been successfully accessed.

close_sdl:
Return Type: void
Parameters: none
Ends the program properly and closes the window

dropToken:
Return Type: boolean
Parameters: Board b, Token tokenColour, integer col
Returns true if a token and been successfully dropped onto the given board, at the specified column.

deleteStillToken:
Return Type: void
Parameters: Falling Token *fallingToken
If falling token has reached the lowest possible position, stop the gravity simulation on that token.

setHighlightedTokenList:
Return Type: void
Parameters: List<TokenLocation> *highlightedTokenList
List of token location that are highlighted in the setup game mode.

--- Implementation ---

Uses:

SDL2/SDL.h
string
board.h
linkedList.h
graphics.h
stdlib.h

Type Declarations:

TextureWrapper:
Any object of this struct will be able to use SDL's texture function, and make it easier to calculate positioning for these textures by the use of the width and height.

```
struct TextureWrapper {
   SDL_Texture *texture;
   int width;
   int height;
};
```

FallingToken:
Allows manipulation of multiple falling tokens at the same time, independently of each other.

```
struct FallingToken {
   int x;          //distance from left to right
   int y;          //distance from top to bottom
   int v;           // velocity
   int yFinal;   // final position
   bool isFalling;
   Token token;  // enum {BLUE, RED, EMPTY}
};
```

Global Variables

*gFallingTokens: List<FallingToken>
This list keeps track of all the tokens that are falling in the game.

Variables

(Any variables beginning with a g (ie gVarName) is a global variable without external linkage (meaning its internal to module))

gRenderHighlighted: bool
Flag for tracking whether token highlighting should be rendered or not.

*gWindow: SDL_Window
Represents the window to be used for the game, any change with the window itself (height, width, border, and full screen) will be made through this.

*gConnect4Board: TextureWrapper
Represents the image of the game board.

*gRedToken: TextureWrapper
Represents the image of the red token.

*gBlueToken: TextureWrapper
Represents the image of the blue token.

*gMainMenu: TextureWrapper
Represents the image of the main menu.

*gOnePlayerButton : TextureWrapper
Represents the one player button

*gTwoPlayerButton: TextureWrapper
Represents the two player image button.

*gMenuButton: TextureWrapper
Represents the menu image button.

*gGlow: TextureWrapper
Token highlighting texture.

*gInvalidMessage: TextureWrapper
Invalid board message display

*gInvalidTokenMessage: TextureWrapper
Invalid game setup message display.

*gRenderer: SDL_Rendered
Responsible for rendering any image onto an SDL_Window.

*gHighlightedTokens: List<TokenLocation>
This list contains all currently highlighted tokens in game.

Local Programs:

static void highlightToken(TokenLocation *tokenToHighlight)

Given a tokenLocation pointer as argument, this function find the row and column of the respective token and highlights it. The TextureWrapper gGlow is used to highlight the tokens (it's simply a white disk with low alpha value that allows tokens to look 'highlighted').

```
static void highlightToken(TokenLocation *tokenToHighlight) {

   //find location of token
   SDL_Rect fillRect = {GRID_OFFSET_X +
                  TOKEN_WIDTH*tokenToHighlight->column,
                  GRID_OFFSET_Y + TOKEN_HEIGHT * tokenToHighlight->row,
                  TOKEN_WIDTH, TOKEN_HEIGHT};

   TextureWrapper *tokenColour =
                  (tokenToHighlight->colour == RED) ?  gRedToken : gBlueToken;

   SDL_RenderCopy(gRenderer, tokenColour->texture, NULL, &fillRect);

   SDL_SetRenderDrawColor(gRenderer, 0xFF, 0xFF, 0xFF, 0x66);
   SDL_SetRenderDrawBlendMode(gRenderer, SDL_BLENDMODE_BLEND);

   SDL_RenderFillRect(gRenderer, &fillRect);
   SDL_SetRenderDrawColor( gRenderer, 128, 128, 128, 0xFF );
   SDL_SetRenderDrawBlendMode(gRenderer, SDL_BLENDMODE_NONE);
   displayBoard();   // get the board ready

   // render the glow texture wrapper (gGlow) to the back buffer of the window.
   SDL_RenderCopy(gRenderer, gGlow->texture, NULL, &fillRect);
}
```

static void freeTexture(TextureWrapper *myTexture)

Responsible for de-allocating any Texture Wrapper. Given the argument of a pointer to TextureWrapper, destroy the texture with SDL_DestroyTexture, and set it to NULL.

```
static void freeTexture(TextureWrapper *myTexture) {
  if(myTexture != NULL) {
    if(myTexture->texture != NULL) {
      SDL_DestroyTexture(myTexture->texture);    //destroy texture
      myTexture->texture = NULL;
    }
    free(myTexture);   //built in to C standard library, free() releases from memeory
  }
}
```

static TextureWrapper* loadTexture(std::string path)

Load a media file from the given directory path and transform it into a texture, by using SDLs built in function as describe below

```
static TextureWrapper *loadTexture(std::string path) {

            Validate the path given (call SDL_GetError with unsuccessful)

            Create a new surface with SDL_CreateRGBSurface based on the scale    size of
            the game

            Create a new texture (newTexture) from SDL_CreateTextureFromSurface

            Allocate memory for newTexture

            Remove the surface created earlier, since its no longer needed, this can   be
            done with SLD_FreeSurface

            Return newTexture

}
```

static bool compareXPosition(FallingToken *listItem, FallingToken *item)

Returns true if the two tokens are in the same column, false otherwise

```
static bool compareXPosition(FallingToken *listItem, FallingToken *item) {

  return listItem->x == item->x;

}
```

static void freeTokenLocation(TokenLocation *tokenLocation)

Calls the standard library free() function, while removes the pointer of TokenLocation from memory.

```
static void freeTokenLocation(TokenLocation *tokenLocation) {
  free(tokenLocation);
}
```

Access Programs:

void mainMenuRender()

Renders the main menu background texture on the screen by calling local function displayMainMenu(). DisplayMainMenu() just copies a global texture gMainMenu onto gRenderer. MainMenuRender() also presents the renderer (causing the texture to be displayed on the screen).

```
void mainMenuRender() {
  displayMainMenu();
  SDL_RenderPresent(gRenderer);
}
```

void drawFallingToken(FallingToken *fallingToken)

The function creates a new tokenWrapper variable from the color, and position (x, y) of the falling token pointer passed in as argument.
This allows us to render the token onto the screen with SDL_RenderCopy (SDL's built-in rendering function).

```
void drawFallingToken(FallingToken *fallingToken) {
  TextureWrapper *tokenTexture;
  if (fallingToken->token == RED) {
    tokenTexture = gRedToken;
  }
  else {
    tokenTexture = gBlueToken;
  }

  SDL_Rect tokenRect;
  tokenRect.x = fallingToken->x;
  tokenRect.y = fallingToken->y;
  tokenRect.w = TOKEN_WIDTH;
  tokenRect.h = TOKEN_HEIGHT;

  SDL_RenderCopy( gRenderer, tokenTexture->texture, NULL, &tokenRect );
}
```

bool init()

Initializes all of SDL's graphical functions. Also creates a new window (gWindow) and renderer for the window (gRenderer). If everything is initialized successfully, it will return true.
If any initialization is unsuccessful the program will not start and will prompt the user with a message of what has gone wrong.

```
bool init() {

  bool success = true;

  if (SDL_Init(SDL_INIT_VIDEO) < 0) {
    printf("SDL could not initialize! SDL_Error: %s\n", SDL_GetError());
  }
  else {
    // create window
    gWindow = SDL_CreateWindow("Connect 4", SDL_WINDOWPOS_UNDEFINED,
        SDL_WINDOWPOS_UNDEFINED, SCREEN_WIDTH, SCREEN_HEIGHT, 0);
    if(gWindow == NULL) {
      printf("Window could not be created! SDL_Error: %s\n",
          SDL_GetError());
    }
    else {
      //Create renderer for window
      gRenderer = SDL_CreateRenderer( gWindow, -1,
          SDL_RENDERER_ACCELERATED | SDL_RENDERER_PRESENTVSYNC );
      if( gRenderer == NULL ) {
        printf( "Renderer could not be created! SDL Error: %s\n",
            SDL_GetError() );
        success = false;
      }
      else {
        //Initialize renderer color
        SDL_SetRenderDrawColor( gRenderer, 128, 128, 128, 0xFF );
      }
    }
  }
  return success;
}
```

bool loadMedia()

Any media (images, sounds, effects) must be declared in this function. If at least one media file is not loaded properly, the program will not start.
All images loaded are variables of the textureWrapper structure, which gives the image an SDL texture, a width and a height.

```
bool loadMedia() {

  bool success = true;

  gBackground = loadTexture("../misc/white_background.bmp");    // background image
  if (gBackground == NULL) {
    printf("Failed to load background!\n");
    success = false;
  }
   // code omitted: repeat the above 4 lines for any other media that requires to be loaded
   // Make sure a global texture has been declared in order to be properly assigned to the
   // media

  return success;
 }
```

void close_sdl()

When the program is closed, we must de-allocate all textureWrapper surface variables. To do so we first destroy the texture by using freeTexture, which in turn use SDL_DestroyTexture. Once the texture is destroyed we can safely set the variables to NULL. Next all windows must be destroyed and set to NULL, which can be done with SDL_DestroyWindow. Finally SDL_Quit can be called, which will terminate SDL safely.

```
void close_sdl() {
  // for all the global texture variables we apply the same set of function calls
 // to de-allocate  and destroy each texture

  freeTexture(gConnect4Board);
  gConnect4Board = NULL;
  // code omitted: repeat above two steps for all other global textures

  // Destroy window
  SDL_DestroyWindow(gWindow);
  gWindow = NULL;

  // Quit SDL subsystems
  SDL_Quit();
}
```

void deleteStillToken(FallingToken *fallingToken)

If a token is no longer falling (isFalling == false) then we can remove it from the list gFallingTokens. The deleteFromList function, will return a new list of FallingToken pointers, and set it as the new gFallingToken list.

```
void deleteStillToken(FallingToken *fallingToken) {
  if(fallingToken->isFalling == false) {
    gFallingTokens = List<FallingToken>::deleteFromList(fallingToken, gFallingTokens);
  }
}
```

void transitionSetupRender()

This function is the transition state from MAINMENU to SETUP. TransitionSetupRender() clears the background, displays the two setup tokens used to choose whether blue or red tokens are dropped in SETUP mode, and renders the Menu, One Player and Two Player buttons on the screen. These are all textures that need to be rendered only once, during the transition from MAINMENU to SETUP, and not in the SETUP state itself.

void setupRender()

This function does the rendering for the SETUP state. It displays the board, then renders highlighted tokens if they have not been rendered since the last press of the "Two Player" button, which is determined by checking a boolean value " gRenderHighlighted" that is local to the graphics module. SetupRender also presents draws the texture stored in gRenderer onto the screen (it draws the state of the game in SETUP).

void clearFallingToken(FallingToken *fallingToken)

This function just overwrites a fallingToken's previous position with the background. This is to save rendering the whole background each frame (we just erase where the token WAS before it dropped a frame's distance further).

void updateFallingTokens(FallingToken *fallingToken, float dt)

This function updates the position/velocity of fallingToken based on a macro-defined constant acceleration, and an input time-step "dt". The function also implements bouncing and damping: when a fallingToken reaches its slot in the board it bounces upward and its speed is reduced until its speed is below a certain value. When a fallingToken's speed is below this epsilon value it is marked to be deleted used a bool value "isFalling" in the fallingToken struct.

bool dropToken(Board b, Token tokenColour, int col)

Before dropping the token in a column, we must first check that the column is not full. If the drop is allowed to be made, create a new FallingToken pointer, and declare all the values required by the FallingToken struct. Then the falling token can then be inserted in the list gFallingTokens, where it can be accessed for gravity simulation.

```
bool dropToken(Board b, Token tokenColour, int col) {

  // Find the row where the token should land, and check that it is not full
  int row = board_dropPosition(b, col);
  if (row == -1) {
    return false;
  }

  TextureWrapper *token;
  if (tokenColour == RED) {
    token = gRedToken;
  }
  else if (tokenColour == BLUE) {
    token = gBlueToken;
  }

  FallingToken *newToken = (FallingToken *)malloc(sizeof(FallingToken));

  // Initial position of the token
  newToken->x = GRID_OFFSET_X + TOKEN_WIDTH * col;
  newToken->y = GRID_OFFSET_Y;
  // check if there is another falling token that is above
  // the top of the board; if so drop this next token ABOVE that token
  FallingToken *currentHighest =
    List<FallingToken>::reduceList(compareXPosition, newToken, gFallingTokens);
  if(currentHighest != NULL) {
    if(newToken->y + TOKEN_HEIGHT > currentHighest->y) {
      newToken->y = currentHighest->y - TOKEN_HEIGHT;
    }
  }

  // Velocity of token
  newToken->v = 0;
  // Final height of the token
  newToken->yFinal = GRID_OFFSET_Y + row * TOKEN_HEIGHT;
  newToken->isFalling = true;
  newToken->token = tokenColour;

  gFallingTokens = List<FallingToken>::addToList(newToken, gFallingTokens);
  return true;
}
```

void setHighlightedTokenList(List<TokenLocation> *highlightedTokenList)

This function takes a List of TokenLocations (row, column and colour) and turns the data structure gHighlightedTokens into that List by first free'ing the old list, then setting gHighlightedTokens to the head of highlightedTokenList. This function also sets the global variable gRenderHighlighted to indicate to setupRender that the highlighted tokens must be rendered on the next frame flip.

void displayBoard()

Determine position of where the board should be located, and render the texture wrapper of the board (gConnect4Board) in the back buffer of the window with SDL_RenderCopy.

```
void displayBoard() {
  // determine the position for the board
  SDL_Rect DestR;
  DestR.x = GRID_OFFSET_X - 1;
  DestR.y = GRID_OFFSET_Y - 1;
  DestR.w = gConnect4Board->width;
  DestR.h = gConnect4Board->height;
  SDL_RenderCopy( gRenderer, gConnect4Board->texture, NULL, &DestR );
}
```

void displaySetupTokens()

Determine the position of the blue token and red tokens that are used to switch the colors in game, and render the texture wrapper of each token (gRedToken & gBlueToken) in the back buffer of the window with SDL_RenderCopy.

```
void displaySetupTokens() {
  // determine the position for the setup tokens
  SDL_Rect tokenRect;
  tokenRect.x = SETUP_CLICKY_TOKENS_OFFSET;
  tokenRect.y = GRID_OFFSET_Y;
  tokenRect.w = TOKEN_WIDTH;
  tokenRect.h = TOKEN_HEIGHT;

  //Render texture to screen
  SDL_RenderCopy( gRenderer, gRedToken->texture, NULL, &tokenRect );

  tokenRect.x = SCREEN_WIDTH - SETUP_BOTTOM_BUTTONS_OFFSET -
TOKEN_WIDTH;
  tokenRect.y = GRID_OFFSET_Y;
  tokenRect.w = TOKEN_WIDTH;
  tokenRect.h = TOKEN_HEIGHT;

  //Render texture to screen
  SDL_RenderCopy( gRenderer, gBlueToken->texture, NULL, &tokenRect );
}
```

void displayMainMenu()

Render the main menu texture wrapper (gMainMenu) to the back buffer of the window with SDL_RenderCopy.

```
void displayMainMenu(void)
{
  SDL_RenderCopy(gRenderer, gMainMenu->texture, NULL, NULL);
}
```