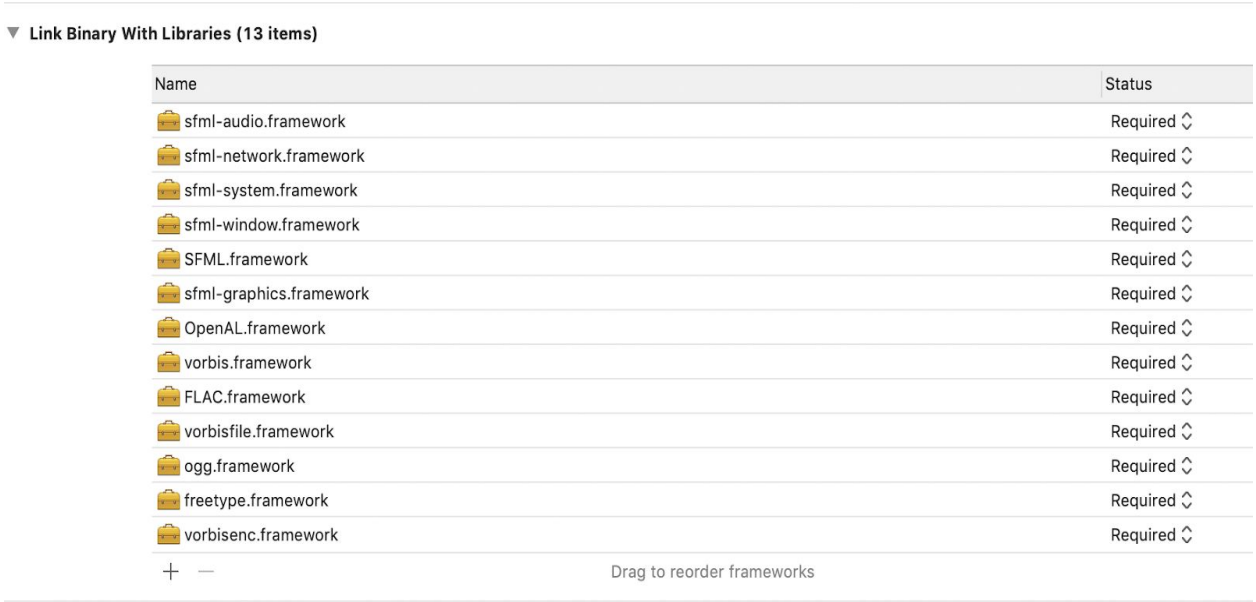
@author; Max Stewart //1086706, Elbert Alcantara //4435223, Sarang Han //5098495
Cosc 345

We are currently on our alpha version of developing our game.
We have used SFML (3rd party library) which needs to be installed correctly for our program to run

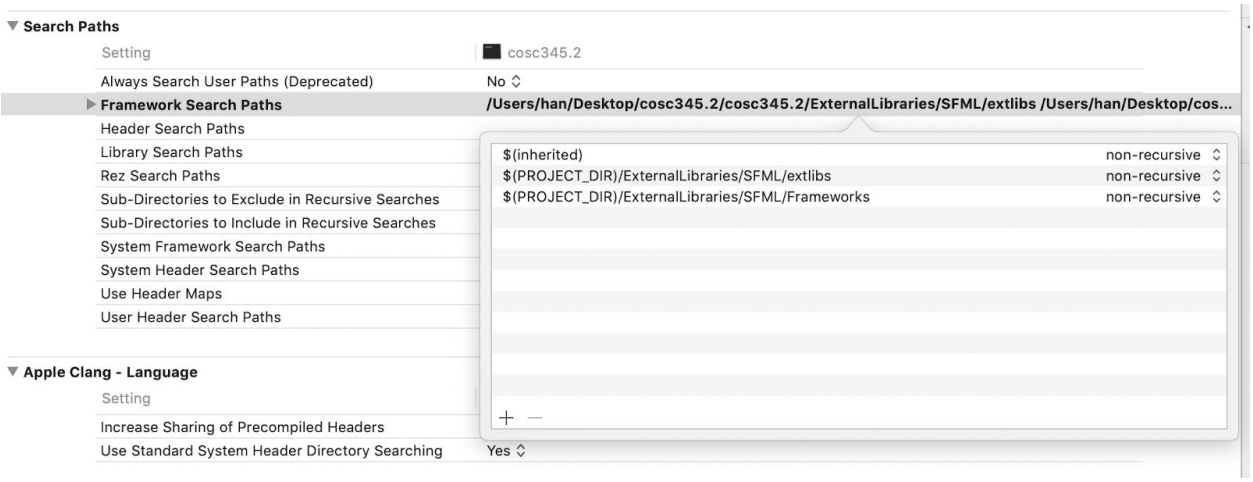
The files required to install SFML is in already present in our github repositories under extLib in the project folder
If not, they can be downloaded at
https://www.sfml-dev.org/download/sfml/2.5.1/
Once downloaded all SFML files need to be moved to the project root folder

For XCode build phases & build search paths need to be configured for SFML to work.
Build phases >> link binary with libraries >> all the .framework files in both extlib && framework folder (should be 13 in total)



Under compiled sources (still in build phase) all folders should be present, so it should look like this

After, under build settings there should be a search path subsection. framework search path settings should look like this;



https://www.sfml-dev.org/tutorials/2.5/start-osx.php
Link for SFML set up.

How to play:
Once loaded player needs to click which tile to start from (any tile in the first row). **If tile is not clicked, bombs will not be distributed, hence game won't start.** Left mouse click uncovers hidden tile / marks start point; right click- puts up a flag → like minesweeper.
Using keys: ASWD player traverses through the map to avoid bombs
A- left
D- right
S- down
W- up
If player hits bomb, it is game over.

Here is a quick run down on most most of the files present.

Definitions.hpp - we have this to change out our images easily when needed // only file path needs to be changed in this file.

AssetManager.cpp-
Void AssetManager::LoadTexture…….
@param std::string name ; assigns a string to represent file name that is defined in the definitions.cpp file
@param std::string fileName; file name that defines pathway of file thats loaded/set
Sets texture to private variables- needed for encapsulation

sf::Texture &AssetManager::GetTexture(std::string name)
@param std::string name ; string that is assigned to represent file name that is defined in the definitions.cpp file
Gets texture set to private variables- needed for encapsulation

Void AssetManager::LoadFont…..
@param std::string name ; assigns a string to represent file name that is defined in the definitions.cpp file
@param std::string fileName ; file name that defines pathway of file thats loaded/set
Sets font to private variables

sf::Font &AssetManager::GetFont(std::string name)
@param std string name ; string that is assignment to represent file name that is defined in the definitions.cpp file
Gets Font set to private variables- needed for encapsulation

InputManager.cpp -
@param sf::Sprite object ; sprite (icon) to be clicked
@param sf::Mouse::Button button ; any mouse click operation

@param sf::RenderWindow &window ; window that input manager is currently managing
@return bool IsSpriteClicked ; returns true if sprite clicked
This method finds the boundaries of the sprite and if a mouse button is clicked within those boundaries, returns true, if not, false

@param sf::RenderWindow &window ; window that input manager is currently working on
@return getMousePosition ; returns position of mouse
This method returns the position of mouse after mouse click has occurred.

MainMenuState.cpp-
MainMenuState::MainMenuState(GameDataRef data) : _data(data)
This method loads a new reference of _data which is concordant with the data MainMenuState needs.
This method loads and sets textures needed for MainMenuState. Loads the background & ‘Play’ icon. Sets position background and ‘play’ icon.
Sets and gets private variables declared in the coinciding .hpp file.

void MainMenuState::HandleInput()
Closes window if window is requested to be closed (no data present)
If sprite (the ‘play’ icon) is pressed, takes player to game screen from the main menu

Void MainMenuState::Update(float dt)
@param dt , interpolation times to update state;
No updated data in main menu state

Void MainMenuState::Draw(float dt)
@param dt ; loaded _data
Clears & displays window & what is loaded and set from MainMenuState::Init
Draw background and ‘play’ button

EasyState.cpp-
Easystate::EasyState(GameDataRef data) : _data(data)
This method loads a new reference of _data which is concordant with the data EasyState needs.
This method loads and sets textures needed for EasyState. Loads restart button, desert, tile and background & Also sets for restart button, surface (desert) and background
sets and gets private variables declared in the coinciding .hpp file
For loops goes through the array of ‘tiles’ and randomly sets where bombs are
Adds a new instance of player
Prints original array for debugging purposes

Void Easystate::SetUp(int x)
@param x ; position of player

This method sets players position to the tile the player has clicked, and makes hasChosen=true which needs to be true so player can be drawn &&
Sets all tiles to be clear of mines in the start, ‘tiles’ are distributed once player chooses/ clicks initial start position.
If upper = 0 → shows tile
If under = 0 → has no bomb
The nested for loop running through the array of ‘tiles’ calculates how many mines are beside (beside = sides && edges) the current tile, giving tile number.

void EasyState::HandleInput()
This method gets position of mouse click &&
Closes window if window is requested to be closed (nothing in _data) &&
If the ‘_restart’ texture is left clicked loads _data of Minmenustate // changes to main menu
&& if clicked tile has an assigned value of 0 it will display numbered tile underneath
&& includes method that links player movement from keyboard
Used AWSD
A- moves player to the left
W- moves player up
S- moves player down
D- moves player to the right

Void EasyState::Update(float dt)
@param dt , interpolation times to update state;
Updates player movement

Void EasyState::Draw(float dt)
@param dt ; loaded _data
Clears & displays window, tiles and surface (desert)
Sets position of tiles according to their assigned number

Void EasyState::RemoveUpperTile…
@param xPos ; int that represents grid position in the x axis
@param yPos ; int that represents grid position in the y axis
This method resets upper grid

Game.cpp-
@param int width ; width of game window
@param int height ; height of game window
@param std::string title ; title of game
Creates the games window and takes player to splash screen

Void Game::Run()
This method uses time to update and handle inputs between different frames
&& finds all timing values needed to run game.

SplashState.cpp-
SplashState::SplashState(GameDataRef data) : _data(data)
This method Loads a new instance of state , splashstate for splash screen &&
Loads and sets textures needed for splash screen

Void Splashstate::HandleInput()
Closes window if window is requested to be closed (nothing in _data)

Void SplashState::Update…
@param float dt ; interpolation times to update state
This method takes player to main menu screen after 3.0 (in definitions folder
`SPLASH_TITLE_SHOW_TIME` is assigned to a float of 3.0) seconds

Void SplashState::Draw..
@param float dt ; loaded _data
This method clears and displays window &&
Draws window and what was loaded and set from SplashState::SplashState(GameDataRef data) : _data(data)

StateMachine.cpp-
@param StateRef newState ; new state that has replaced old state
@param bool isReplacing ; returns true if new state is created / is in line for ‘stack’ if not, false
Sets variables needed to help ProcessStateChanges()

Void StateMachine::ProcessStateChanges()
If statements which check what is in front of ‘line’ (or stack) check if current state has been replaced or if a new state is ‘in line’ to be created, if a new state is in ‘line’ to be created, current state is ‘popped’ off the line and the ascending one is created/ replaces current state

StateRef &StateMachine::GetActiveState()
Returns current active state

Player.cpp-
Player::Player(GameDataRef data, int gridUnder[][22]) : _data(data)
@param GameDataRef data ; data necessary for player movement
@param int gridUnder ; initialised position player is in
This method sets and load texture necessary for in game player sprite , initialises variables needed for methods later on

@return bool ; returns true if player is moving, false if not

@return bool ; returns true if player has chose to start, false if not

Void Player::setPos(int x)
@param int x ; where player chooses to start in x axis. Y stays constant as always starting in top row then working down / across
This method sets players initial position

sf::Vector2i Player::GetPos()...
@return sf::Vector2i Player ; returns current position of player

Void Player:PrintTileArray()
Nested for loop prints array

@param int newX ; new position of player in x axis
@param int newY ; new position of player in y axis
@return int ; returns new position of player

@param int newX ; new position of player in x axis
@param int newY ; new position of player in y axis
@param int spriteShown ;
@return int ; returns 0 if player doesn't have new position , & returns a value if it has

Void Player::Draw()
Draws in game character

Void Player::move(float dt)
@param float dt ; vector difference of the movement of player
This method handles player movement

Main.cpp-
Creates an instance of Game ; has the width and height of what is defined in definitions.hpp
And window is called 'alpha'