@author; Max Stewart //1086706, Elbert Alcantara //4435223, Sarang Han //5098495
Cosc345

We are currently on our beta version of developing our game.

Changes since the alpha release

GameStates.cpp– this class didn't exist at all before; implemented for tidiness of code and to shorten code

GamesStates::GameStates(…)…

@param data; data necessary for both easy state and game state
@param x; either 1 0r 2, 1 = easy level, 2 = hard level
Checks which game state is chosen by player
Loads and sets textures needed for both easy and hard game states.

Void GameStates::Setup()
@param x; position of player
Sets hasChosen=true so player can be drawn && sets players position and ensures tiles are allocated after player clicks start position in case player randomly starts on a bomb ; calculates how many mines are beside current tile <– code pulled from Player.cpp

Void GameStates::HandleInput()
This method handles player movement, links allocated number (whether its a bomb or not) to texture, and a function implemented to take into account the fact the outer most edges wont work for the lines in the hard state as the "tiles" are triangular shapes; /getGlobalBounds didn't work as is square

Void GameStates::Update()
@param dt, interpolation time
Moves player position, takes 6∗interpolation time seconds to move

Void GameStates::Draw()
@param dt, interpolation time in seconds
Hard state and easy state background is drawn here

Player.cpp– Player::Player method used to initialise player position, now in either Easy or Hard state to shorten code; PlayerMoving(), PlayerChosen(), Explode(…), MovePlayer(…) methods are added

Player::Player(GameDataRef data) : _data

@param GameDataRef data ; data necessary for player movement
This method sets and load texture necessary for in game player sprite , initialises variables needed for methods later on

bool Player::PlayerMoving()

@return returns true if player is moving

bool Player::PlayerChosen()
@return returns true if player has picked a start position

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 returns play position

Void Player::Explode()
@param newX, position of where “explosion” needs to be set in the x axis
@param newY, position of “ explosion” needs to be set in the y axis
Sets position of explosion sprite to where is needed

Void Player::MovePlayer()
@param newX, picked next position of player ; players next choice
@param newY, picked next position of player; players next choice
@param spriteShown ; where the sprites shown
This method accounts for out of bounds exceptions– checks if next move is valid or not

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//change

Main.cpp– changed “alpha” to “beta”
Creates an instance of Game ; has the width and height of what is defined in definitions.hpp And window is called ‘beta’

MainMenuState.cpp–Play button is swapped out to easy and hard buttons
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, easy and hard button

EasyState.cpp– Void EasyState method is added;
EasyState::EasyState added; code pulled from Player.cpp to shorten net number of lines

EasyState::EasyState(GameDataRef data) : _data(data)
This method loads & sets textures necessary for easy level
Debug printing; Creates an instance of new player

Void EasyState::Setup(int x)
@param x ; position at which tile the player clicked to start game
This method sets the players position at the start of the game; and ensures tiles are allocated after player clicks start position; calculates how many mines are beside current tile <–code pulled from Player.cpp ; swapped to shorten code

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 Mainmenustate // 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)
This method iterates through the grid (backboard of game) and allocates coinciding tile
@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

<u>No Changes since the alpha release</u>
We have used SFML (3rd party library) which needs to be installed correctly for our program to run

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.

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; also sets position of textures

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 9.8 (in definitions folder SPLASH_TITLE_SHOW_TIME is assigned to a float of 9.8) 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) according to a certain period of time

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