**Programming Project 2 - Sample Class Structure**

***Class Diagrams:***

Class : AwesomeSquare (the player class)

Instance Variables

int row; - stores the row the awesomeSquare is in within 2D array

int col; - stores the column the awesomeSquare is in within 2D array

sf::Texture texture; - texture for awesomeSquare player

sf::Sprite sprite; - sprite used to represent awesomeSquare player

int score; - players score

int lives; - contains players lives

int direction; - the direction the player is facing in

bool alive; - is the player alive

Functions

draw() - if alive draw the awesomeSquare using its row & column

move(maze) - moves the player via keyboard input– changes its row or column, boundary checking can be done within the move function

kick() - a function that checks if the player is trying to kick a square

die() - if lives 0 game over & call respawn function else take away 1 life

respawn() - causes the player to reappear on the maze at starting position

kill() - if player kills an evil square score increases

increaseScore() – increase player’s score

Class : EvilSquare (The enemy class)

Instance variables

int row; - stores the row of the enemy within 2D array

int col; - stores the column of the enemy within 2D array

sf::Texture texture; - texture for enemy

sf::Sprite sprite; - sprite used to represent enemy

int direction; - the direction the enemy is facing

bool alive; - checks if the enemy is alive

Functions

draw() - if alive draw the evil square

move() - moves the evil square in the direction it is facing until it reaches a worldSquare or boundary, then change direction. Boundary checking can be done in the move function.

die() - if the enemy is hit by a worldSquare pushed by the player, the enemy will die.

respawn() - causes the enemy to reappear on the maze at starting position

Class : WorldSquare (The individual cell)

Instance variables

bool containsSquare - does it contain a world square or not

sf::Texture worldSqTexture; - the texture for the worldSquare

sf::Texture emptySqTexture; - the texture for the empty square

sf::Sprite sprite; - sprite used to represent the square

Functions

draw() - if containsSquare is true draw the worldSqTexture else draw emptySqTexture.

kicked() - moves the worldSquare to the next cell.

Class : Game (The controlling class)

Instance variables

static const int MAX\_ROWS = 12; // the maximum number of rows

static const int MAX\_COLS = 12;

static const int MAX\_ENEMIES = 10; // the max number of evilSquares

static const int SQUARE\_SIZE; - global size of the squares

int gameMode; - to represent the different modes in the game, for example

game instruction mode and game play mode

sf::Font m\_font; - font for writing text

sf::Text m\_message; - text to write on the screen

AwesomeSquare player; - object of type player

EvilSquare enemies[MAX\_ENEMIES]; - 1D array of evil square enemies

WorldSquare maze[MAX\_ROWS][ MAX\_COLS]; - a 2D array of world squares to represent the maze

Functions

initialize() - initializes variables – such as maze layout

update() - updates the game objects e.g. moving, collisions

draw() - draws everything that’s active

collisions() - will control all the collisions in the game

userInput() - keyboard handling

restart() - resets and initializes the game objects in order to allow another game to be played.

**Inner workings of the 2D/1D array**

The 2D array of worldSquares represents the game board of the game. The 2D maze contains information about where the worldSquares are located. The game board is divided into rows and columns. Each cell of the gameboard contains a worldSquare or an empty square.

worldSquare[ , ] maze;

A diagram of the 2D array is shown on the next page. When the AwesomeSquare player is moving around the maze, you will need to check this 2D array to see if the cell he is about to move into is empty or not. If the player kicks a worldSquare you will need to update the 2D array.

**Diagram of what the maze could look like:**

**Row**

**Column**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |

**Legend:**  world Square

Empty Space

If the bool containsSquare is true, the worldSquare texture is drawn which is a green.

If the bool containsSquare is false, the emptySquare texture is drawn which is a blue.

|  |  |  |  |
| --- | --- | --- | --- |
| **row**  **col**  **direction**  **alive** | **row**  **col**  **direction**  **alive** | **row**  **col**  **direction**  **alive** | **row**  **col**  **direction**  **alive** |

1D Array of Evil Squares

The game contains a 1D array of evil squares called enemies. Each object in the array will contain information about the individual evil squares.

EvilSquare[] enemies;

Each EvilSquare object within the array has row and column data, sprite body, direction and whether they are alive or not etc. The row and column data allow you to find out where they are located within the 2D maze. You can then check the direction the evilSquare object is moving in and check the relevant cell within the 2D maze to see if it is empty or not. If the cell is not empty, they will need to move in a different direction.

The row and column will be used to determine the location of each enemy within the 2D maze, the alive bool will check if the enemy is currently alive in the game.

The purpose of the 1D array is to control multiple evil square objects that have different positions and states.