USER README.

/\*

\* class Object

\*

\* This class is used to for the production of the player object. it is responsible for the position of the player, as well is its rotation and scale

\* based upon the Matrix3 of GlobalTranslate. it also contains the inital setup process of the game object, the update function that updates the screen with

\* each frame and a draw function, which allows the texture to be drawn as a part of the object class.

\* it also contains the setParent function, which allows for the use of a matrix3 hierachy, which in this case, is Object being the tank base, followed by the tankTurret

\* and then the turret itself. It contains position, scale and rotation, which is needed for the local translate, with is then converting into a global translate.

\* it also contains a origin value, which will contain the centre of the image.

\*

\* @author - Zackary Direen, Academy of Interactive Entertainment, 2017

\*/

/\* Object();

\*

\* This is the default constructor for an Object class.

\*

\*/

/\* void SetUp(aie::Texture\* texture, aie::Renderer2D\* Renderer, shape::shapeType shapePick, float PosX = 0, float PosY = 0, float ScaleX = 1, float ScaleY = 1, float OriginX = 0.5f, float OriginY = 0.5f);

\*

\* This is the setup function for the Object, which will set the texture, the renderer it will use.

\* shapetype, position, scale and the origin of the object.

\*

\* @param aie::Texture\* - A texture that will be used for a particular object

\* @param aie::Renderer2D\* - A pointer to the Renderer of the bootstrap.

\* @param shape::shapeType - A enum value of a shapetype.

\* @param float PosX - a float value that stores position.x defaulted to 0.

\* @param float PosY - a float value that stores position.y defaulted to 0.

\* @param float ScaleX - a float value that stores scale.x defaulted to 1.

\* @param float ScaleY - a float value that stores scale.y defaulted to 1.

\* @param float OriginX - a float value that stores origin.x defaulted to 0.5.

\* @param float OriginY - a float value that stores origin.y defaulted to 0.5.

\*

\* @returns void

\*/

/\* void setParent(Object\* newParent);

\*

\* This function sets the parent of an object which is important for the matrix hierarchy.

\*

\* @param Object\* newParent - An object pointer that the user wants to be the parent.

\* @returns void

\*/

/\* virtual void Update(float DT);

\*

\* This virtual function Updates the values of the object class based upon the parameter of deltaTime (which is game time.)

\* This function gets redefined when an object calls its own update function.

\*

\* @param float DT - This is deltatime.

\* @returns void

\*/

/\* virtual void Draw();

\*

\* This virtual function draws an object to the game screen when it is called based on its sprite, this draw will usually happen in update.

\* This function gets redefined when an object calls its own draw function.

\*

\* @param float DT - This is deltatime.

\* @returns void

\*/

/\* virtual CalMatrix();

\*

\* This function will update the GlobalTranslate based on the localTranslate, which is then based on the objects

\* scale, position and rotation. this needs to be done so that an object can be drawn and updated correctly.

\*

\* @returns void

\*/

/\* ~Object();

\*

\* This is the default destructor for an Object class.

\*

\*/

/\*

\* class Circle

\*

\* This class is based on the object class and has the same functions as an Object class, it however has its own Constructor based on the same parameters as

\* the object class, it also has its own Update and Draw function, as well as a bool variable called bounceUp and a vector2 called velocity.

\*

\*

\* @author - Zackary Direen, Academy of Interactive Entertainment, 2017

\*/

/\* Circle();

\*

\* This is the default constructor for a Circle object.

\*

\*/

/\* Circle(aie::Texture\* texture, aie::Renderer2D\* Renderer);

\*

\* This is a constructor that takes on the texture and the Renderer as parameters.

\*

\* @param aie::Texture\* - A texture that will be used for a particular object

\* @param aie::Renderer2D\* - A pointer to the Renderer of the bootstrap.

\*/

/\* void Update(float DT, aie::Input\* Input);

\*

\* This function Updates the values of a circle based upon the parameter of deltaTime (which is game time.) as we as input, which allows the user to input

\* keyboard commands.

\*

\* @param float DT - This is deltatime.

\* @param aie::Input\* - This allows for user input.

\* @returns void

\*/

/\* void Draw();

\*

\* This function draws a circle object to the game screen when it is called based on its sprite, this draw will usually happen in update.

\* This function gets redefined when an object calls its own draw function.

\*

\* @returns void

\*/

/\* ~Circle();

\*

\* This is the default destructor for a Circle object.

\*

\*/

/\*

\* class Player : public Object

\*

\* This class is the player object.

\* it is responsible for the tankbase as well as the tankTurret and the turret, as well is its rotation and scale

\* based upon the Matrix3 of GlobalTranslate. it also contains the inital setup process of the player, the update function that updates the screen with

\* each frame and a draw function, which allows the texture to be drawn as a part of the object class.

\* It contains position, scale and rotation which is the same values of Object, which is needed for the local translate, with is then converting into a global translate.

\* it also contains a origin value of the object, which will contain the centre of the sprite. it also contains 3 objects.

\* which are the tank base, the tank turret base and the tank turret and a boolean that checks if the tank turret is attached.

\*

\* @author - Zackary Direen, Academy of Interactive Entertainment, 2017

\*/

/\* Player();

\*

\* This is the default constructor for a Player object.

\*

\*/

/\* void SetUp(aie::Texture\* texture, aie::Renderer2D\* Renderer, aie::Texture \* turretTexture, aie::Texture \* Barreltexture, aie::Texture \* bullet)

\* This is the setup function for the Object, which will set the texture, the renderer it will use.

\* shapetype, position, scale and the origin of the object.

\*

\* @param aie::Texture\* - A texture that will be used for the tank base.

\* @param aie::Renderer2D\* - A pointer to the Renderer of the bootstrap.

\* @param aie::Texture\* turretTexture - A texture that will be used for the tank turretBase.

\* @param aie::Texture\* BarrelTexture - A texture that would be used for the tank turret.

\* @param aie::Texture\* bullet - a texture that wouldve been used for a bullet.

\* @returns void

\*/

/\* void Update(float DT, aie::Input\* Input);

\*

\* This function updates the values of the player on the parameter of deltaTime (which is game time.) as we as input, which allows the user to input

\* keyboard commands.

\*

\* @param float DT - This is deltatime.

\* @param aie::Input\* - This allows for user input.

\* @returns void

\*/

/\* void Draw();

\*

\* This function draws the player object to the game screen when it is called based on its sprite, this draw will usually happen in update.

\*

\* @returns void

\*/

/\* ~Player();

\*

\* This is the default destructor for a Player object.

\*

\*/

/\*

\* class Application2D : public aie::Application

\*

\* This class is the Application2D object.

\* it is responsible for the startup and shutdown of the game, as well as updating each frame

\* and drawing each opject within the gameworld.

\* @author - Zackary Direen, Academy of Interactive Entertainment, 2017

\*/

/\* Application2D();

\*

\* This function is the constructor.

\*

\*

\*/

/\* virtual ~Application2D();

\*

\* This function is the destructor.

\* It is meant to be overridden.

\*

\*/

/\* virtual bool startup();

\*

\* This function is for the startup of the application.

\* It is meant to be overridden.

\*

\*

\* @returns bool

\*/

/\* virtual void shutdown();

\*

\* This function is for the shutdown of the application.

\* It is meant to be overridden.

\*

\*

\* @returns void

\*/

/\* virtual void update();

\*

\* This function is for the updating of the application based upon deltatime.

\* It is meant to be overridden.

\*

\*

\* @param float deltaTime - gametime.

\* @returns void

\*/

/\* virtual void draw();

\*

\* This function is for the drawing the game objects in the application.

\* It is meant to be overridden.

\*

\*

\* @returns void

\*/

/\*

\* struct ArraySquare

\*

\* This struct allows for the storing of an array of Vector2's that will store points like a square.

\* @author - Zackary Direen, Academy of Interactive Entertainment, 2017

\*/

/\*

\* struct CollisionResults

\*

\* This struct allows for the storing of collision results.

\* @author - Zackary Direen, Academy of Interactive Entertainment, 2017

\*/

/\*

\* class shape

\*

\* This class is used to draw squares around game objects to allow for collisions to happen

\* and to be detected.

\* @author - Zackary Direen, Academy of Interactive Entertainment, 2017

\*/

/\* shape();

\*

\* This function is the default constructor.

\*

\*

\*/

/\* shape( Vector2 scale, Matrix3 \*Global);

\*

\* This function is the constructor that takes in a scale and the globalTranslate.

\*

\* @param Vector2 - scale

\* @param Matrix3\* - GlobalTranslate

\*/

/\* void SetShape(Vector2 scale, Matrix3 \*Global, shapeType shapeSelection,Vector2 Origion = { 0,0 });

\*

\* This function sets the shape based upon the paramaters recieved.

\*

\* @param Vector2 - scale

\* @param Matrix3\* - GlobalTranslate

\* @param shapeType - shapeSelection

\* @param Vector2 - the Origion, defaulted to 0,0

\*

\* @returns void

\*/

/\* void ReCalcShape();

\*

\* This function recalculates the shape based on the points of ArraySquare.

\*

\*@returns void

\*/

/\* ArraySquare getNormals();

\*

\* This function gets the normals of the points within ArraySquare and returns them normalised.

\*

\* @returns ArraySquare- normalised points

\*/

/\* ArraySquare getNormalsUnNormalized();

\*

\* This function gets the normals of the points within ArraySquare and returns them as if they were unnormalised.

\*

\* @returns ArraySquare- unnormalised points

\*/

/\* Vector2 ProjectPolygon(Vector2 AxisNorm);

\*

\* This function projects the polygon base upon the x and y axis that will be used for collision detection.

\*

\* @param Vector2 - AxisNorm are the Axis that have been normalised.

\* @return Vector2 - returns the projection.

\*/

/\* float IntervalDistance(Vector2 minMax1, Vector2 minMax2);

\*

\* This function checks the interval distance between two shapes and returns the distance between them.

\*

\* @param Vector2 - minMax1 is the min and max dot product of shape1.

\* @param Vector2 - minMax2 is the min and max dot product of shape2.

\*

\* @returns float

\*/

/\* void Update();

\*

\* This function updates each shape object to be able to redraw the shape to a new location in the game world.

\* @returns void

\*/

/\* void DrawDebug( aie::Renderer2D\* Renderer);

\*

\* This function draws the shape around the game object for the shape to be visually seen.

\* @param aie::Renderer2D\*- the aie renderer

\*

\* @returns void

\*/

/\* bool isColliding(shape other);

\*

\* This function tests to see if two shapes are colliding with each other and will return true if they

\* are and false if they arent.

\* @param shape - other is the 2nd shape thats being tested

\* @returns bool

\*/

/\* CollisionResults isCollidingExtra(shape other);

\*

\* This function allows for the direction and for the push out vector to be calculated. this function is called only if two shapes

\* have been found to be colliding.

\* @param shape - other is the 2nd shape thats being tested

\* @returns CollisionResults

\*/

/\*

\* class Wall : public Object

\*

\* This class is used to create Wall objects based on the type object.

\* @author - Zackary Direen, Academy of Interactive Entertainment, 2017

\*/

/\* Wall();

\*

\* This function is the default constructor.

\*

\*

\*/

/\* ~Wall();

\*

\* This function is the default destructor.

\*

\*

\*/