**Classes and Methods**

**Vec2D struct methods and parameters**

**double *x*** – the x coordinate of a mathematical vector in a 2D space.

**double *y*** - the y coordinate of a mathematical vector in a 2D space.

**double DotProduct(*Vec2D u*)** – calculates the dot product between this vector and another vector u. Returns a double.

**double Magnitude()** – calculates the magnitude (length) of this vector. Returns a double.

**Vec2D GetNormal(*Vec2D u*)** – gets the normal vector between this vector and another vector u. Returns the normal vector.

**void Normalize()** – normalizes this vector (sets the length of this vector to 1).

**Game class methods and parameters**

**Protected members:**

**const char\* *GameName*** – The name of your game. This should be initialized by the user in their own subclass of the game class.

**Scene\* *GameScene*** – The Scene that this Game will be played in.

**std::vector<Sprite\*> *Sprites*** – List of Sprites pointers for your game. This should be initialized by the user in their own subclass of the game class.

**std::vector<std::string> *ImagePaths*** – Standard library vector of strings that are paths to the images for your sprites. This should be initialized by the user in their own subclass of the game class.

**std::vector<Vec2D> *InitialPositions*** – Standard library vector of mathematical vectors that represent the initial positions in the Window for your Game. This should be initialized by the user in their own subclass of the game class.

**Public methods:**

**Game()** – Standard game constructor

**Game(*Scene\* GameScene*)** – Overloaded game constructor that takes in a pointer to the Scene for your game. You should have an overloaded constructor for your Game subclass that also takes a Scene pointer, and in that class set the Scene for your game.

**virtual void InitGame(*const char\* GameName*)** – This is where you will initialize all of the above parameters, and then call Start on your GameScene. Should be overwritten by your own custom subclass.

**Scene class methods and parameters**

**Private parameters:**

**bool *bPlay*** – set to true by default. States whether the game is currently playing or not.

**bool *bPaused***– set to false by default. States if the player has paused the game.

**Sprite\* *Player*** – due to this engine not supporting multiplayer functionality, this is meant to be the Sprite that is controlled by the user.

**SDL\_Window\* *Window*** – the window that is to be displayed, a type from the SDL2 library.

**SDL\_Renderer\* *Renderer*** – set to nullptr by default. This is the rendering object that renders objects to the window, a standard SDL2 type.

**Vec2D *Size*** – A custom vector to describe the size of the screen.

**std::vector<Sprite\*> *Sprites*** – A standard library vector that lists all Sprites available for the current game.

**int *Framerate*** – Defines how many times per second the screen will refresh. This is by default set to 60.

**std::chrono::system\_clock::time\_point *CurrentTime*** – a chrono time point that gets the current time. Will constantly be updated every call.

**std::chrono::system\_clock::time\_point *LastTime*** – a chrono time point, initialized to be the CurrentTime. Gets updated every RefreshSeconds.

**std::chrono::duration<double> *DeltaTime*** – Will be the difference between CurrentTime and LastTime. Used to make sure that at least RefreshSeconds has passed.

**double *RefreshSeconds*** – defined to be 1/Framerate

**int *PCount*** – Initialized as 0. Counts the number of players to make sure the user doesn’t try to define more than 1 player.

**Public Methods:**

**Scene()** – Basic constructor that initializes most parameters.

**void Start(const char\* GameName, std::vector<Sprite\*> Sprites, std::vector<std::string> ImagePaths, std::vector<Vec2D> InitialPositions)** – GameName is a const char\* that defines the name of the game that should be put at top of the window. Sprites is a standard library vector of all Sprite pointers for the game. Private parameter Sprites will be set to this passed parameter. ImagePaths is a standard library vector of strings that define the location of the Images for each Sprite. InitialPositions is a standard library vector of custom vectors that lists where each Sprite should start on the Window. Start initializes the Window, Renderer, calls InitSprites, gets the player sprite, and then calls the Tick function which is where the main updates occur.

**void End()** – Destroys all SDL\_Textures (sprite images), the Window and Renderer, then calls SDL\_Quit.

**void Tick()** – Updates everything if DeltaTime >= RefreshSeconds, calls update for each sprite as long as the game is not paused, calls EventHandler, and checks for collisions. If the game is paused, then it calls the PausedEventHandler.

**void Clear()** – Calls SDL\_RenderClear(Renderer) to clear the renderer.

**void HideCursor()** – Calls SDL\_ShowCursor(SDL\_DISABLE)

**void ShowCursor()** - Calls SDL\_ShowCursor(SDL\_ENABLE)

**void Hide()** – calls SDL\_HideWindow(Window)

**void Show()** – calls SDL\_ShowWindow(Window) (doesn’t currently work).

**void EventHandler(SDL\_Event Event, bool& bPlay, bool& bPaused)** – takes an SDL\_Event, and then references to bPlay and bPaused (out parameters), and handles user input, some of which is defined by the engine, some is to be defined by the user.

**void PausedEventHandler(bool& bPlay, bool& bPaused)** – checks for different events based on if game is paused, that way game doesn’t update, but also user can’t input unwanted inputs during pause.

**void InitSprites(std::vector<std::string> ImagePaths, std::vector<Vec2D> InitialPositions)** – Calls SetImage, SetVertices, and Draw for each Sprite in Sprites.

**Sprite\* GetPlayerSprite()** – Find which Sprite in Sprites has bPlayer set to true.

**Sprite class methods and parameters**

**protected:**

**SDL\_Texture\* Image** – This is the actual Image, or picture, for your Sprite, as type SDL\_Texture\*.

**Vec2D Size** – Size of image as mathematical vector

**Vec2D Position** – A 2D vector representing where the Sprite is located in the Window (relative to the upper left corner).

**double LastAngle** – The last angle (in radians) that the image was facing before a rotation. Needed for certain calculations.

**double ImageAngle** – The direction that the image faces (in radians).

**double MoveAngle** – The direction that the Sprite moves in (in radians).

**double Speed** – The movement speed of a Sprite represented as a single number.

**double Scale** – a double value representing the scale of a Sprite, assuming Scale of x and Scale of y are the same.

**Vec2D D** – A 2D vector that represents the 2D difference in space between the Image coordinate (it’s upper left coordinate), and the Center vector.

**Vec2D Center** – A 2D vector representing the center of the sprite

**Vec2D Velocity** – A 2D vector representing dx, dy.

**Vec2D Acceleration** – A 2D vector representing ddx, ddy.

**Scene\* Scene** – A pointer to the scene that the Sprite is in

**Uint8\* BoundAction;**

**SDL\_Rect texture** – a SDL\_Rect (rectangle) that represents the ‘box’ that the Image of the sprite goes in (needed to move and rotate the Image).

**std::vector<Vec2D> vertices** – A list of the vertices around the Sprite.

**bool bCollisionEnabled** – A bool value representing if the collision of this Sprite is enabled.

**bool bPlayer** – A bool value representing if this Sprite is the player controlled Sprite. Defaults to false.

**bool bDraw** – A bool value representing if the Sprite should be drawn or not.

**double ConvertToDegrees(double Radians)** – Converts an angle from radians to degrees. Returns a double.

**double Health** – Represents the health of a Sprite, if they have any.

**private:**

**void VectorProjection(double Speed)** – Takes the speed of the Sprite and converts it to the velocity and calculates the move angle.

**double ConvertToRadians(double Degrees)** – Converts an angle that is in degrees to radians.

**public:**

**Sprite()** – standard constructor for a Sprite.

**void SetScale(double s)** – Sets the scale of the sprite.

**SDL\_Texture\* SetImage(SDL\_Renderer\* renderer, std::string ImagePath, Vec2D InitPosition)** – Loads image from ImagePath onto the renderer, sets initial position to InitPosition.

**void Draw(SDL\_Renderer\* renderer)** – Draws the Sprite onto the renderer.

**virtual void Update(SDL\_Renderer\* renderer)** – An update method to be overwritten by subclasses.

**virtual void PlayerInput(SDL\_Event Event, SDL\_Renderer\* renderer)** – A method to be overwritten by subclasses, handles specific player inputs.

**virtual void DefaultBehavior()** – Handles the default behavior of a given sprite, such as AI behavior, to be overwritten by subclasses.

**virtual void DealDamage(Sprite\* OtherSprite, double Amount)** – A method to handle how to deal with damage. To be overwritten by subclasses.

**void Hide()** – Hide a sprite. Turns off collisions and sets bDraw to false.

**void Show()** – Show a sprite. Turns collisions back on and sets bDraw set back to true.

**void SetSpeed(double Speed)** – Set speed to a specific number. A wrapper for VectorProjection.

**void SetImageAngle(double Degrees)** – Set the image angle to an angle in degrees.

**void SetMoveAngle(double Degrees)** – Set the move angle to an angle in degrees.

**void AddForce(Vec2D Force)** – Adds a force vector to this sprite.

**void SetBoundAction()**

**void CheckBounds();**

**bool SeparateAxisTheorem(std::stack<Vec2D> Normals, Sprite\* OtherSprite)** – Uses Normals to check against the OtherSprite and determine if there is a collision with respect to each normal. Returns a bool

**bool CollidesWith(Sprite\* OtherSprite)** – returns a bool value, checks if this Sprite collides with OtherSprite.

**double DistanceTo(Sprite\* OtherSprite)** – returns a double. Finds the distance between the OtherSprite and this Sprite.

**double AngleTo(Sprite\* OtherSprite)** – Finds the angle to this Sprite and OtherSprite.

**void SetVertices()** – Set the vertices for this Sprite.

**void UpdateVertices()** – Update vertices based on the Image Angle (rotation).

**void SetCollisionEnabled(bool Set)** – Sets bCollisionEnabled to Set.

**void MoveSprite()** – MoveSprite using Velocity and MoveAngle.

**std::stack<Vec2D> GetNormals()** – Calculates the normal vectors of this Sprite.

**void SetPlayerStatus(bool IsPlayer)** – Checks if this Sprite is the player.

**Vec2D GetPosition()** – Gets the position in a Vec2D of this Sprite.

**void TakeDamage(double Amount)** – this Sprites takes Amount of damage.