https://github.com/itcgames/RWM_1617_P3_Stephen_Ennis

Introduction

This document provides example content of a user guide for a component to be implemented in the Real World Modelling problem 3. The guide for your component is to be comprehensive, providing complete set of usage examples, complete component features and a complete description of the demonstration application.

How to use this component

Setup

```
#include "Camera2D.h"
Camera2D m_camera;
SDL_Renderer *renderer;

//initialise the Camera2D with the SDL renderer
m_camera.init(windowWidth, windowHeight, renderer);

//Update camera effects and motion in the update loop
m_camera.update(deltaTime)

//draw camera parallax scroll and fade effect in the game's render function
m_camera.render()
```

Converting between world and screen coordinates (*IMPORTANT*)

The most important functions are the worldToScreen function for SDL_Rect and Camera2D::Point. These will take a point/rect that is in world coordinates and transform it to what it needs to be to appear to be in “Screen” coordinates. In this way it will seem as if there is a camera in the world that is moving around. All rects and points are drawn at different locations as the camera moves despite the world location of them not changing. Call this method when you are drawing at a location. screenToWorld does the reverse and can be used to get the real world coordinates/rect of a point/rect on the screen. (e.g mouse position to where it's actually pointing in the world)

```
//from the worldToScreen function on a rect
SDL_Rect worldRect = { 0, 0, 50, 50}; //location of our object
SDL_RenderCopy(renderer, &m_camera.worldToScreen(worldRect)); //draw it at the return
//-------------------------------------------------------------------------------------------
-----------------------------
Camera2D::Point screenPoint= getMousePoint() //we get the position of the mouse
Camera2D::Point worldPoint =m_camera.screenToWorld (screenPoint); //point in world coords
//from screenToWorld function on a point
```

Pan

Link to GDD

Demo Video: 00:00 - 00:10

The camera responds to user input when the directional arrow keys are pressed. The camera can pan horizontally, vertically or diagonally with default movement.

```
int xDir = 0; //don't move on the x
int yDir = 1; //move positively on the y
m_camera.pan(xDir, yDir); //move camera in this direction, if zooming to fit then this takes
//priority

xDir = -1; //move negatively on the x
yDir = 1;

m_camera.pan(xDir, yDir);

m_camera.panX(1); //pan right
m_camera.panY(-1); //pan down
```

The range of movement can be restricted to a SDL_Rect using the restrictCentre function. This will prevent the camera's centre going past the bounds of the passed in SDL_rect

```
SDL_rect bounds = {0, 0, 1000, 1000};
m_camera.restrictCentre(bounds);
m_camera.unrestrict(); //to clear this restriction
```

Creating custom motion properties

Demo Video: 00:10 - 00:42

Can set custom camera max speed and acceleration rate. If a parameter is not provided then it is defaulted.

```
float accelerationRate = 100.f; //rate to accelerate at when starting to move
float maxVelocity = 200.f; //don't speed up more than this
float drag = 0.9f; // slow the camera down when not presing a key
m_camera.setMotionProps(accelerationRate, maxVelocity, drag);
```

Zoom

Link to GDD (video)

Demo Video: 01:07 - 01:10

Demo Video: 01:12 - 01:18

Camera can be zoomed in or out at a certain speed. This is called every time the relevant key is pressed down.

```
Int zoomDir = 1; //1 in, -1 out
m_camera.zoom(zoomDir);
```

Can directly set the zoom

```
m_camera.setZoom(1.5f);

//Use getZoomMin() and getZoomMax() to get the zoom min and max
```

Setting zoom properties

Demo Video: 01:10 - 01:12 (set zoom min max and it automatically adjusts zoom to fit)

The zoom properties can be customized. If not provided then they are defaulted.

```
float zoomSpeed = 0.01f; //speed the camera zooms in or out (smaller due to no deltaTime)
float zoomToSpeed = 1.f; //when using zoom to (no deltaTime)
float minZoom = 2.f; //minimum level of zoom
float maxZoom = 0.5f //maximum threshold you can zoom in until;
m_camera.setZoomProps(zoomSpeed, zoomToSpeed, minZoom, maxZoom);
//the min and max can be set separately
m_camera.setZoomMinMax(minZoom, -1); //-1 means unlimited. If changes theses causes the
current zoom to be past a threshold then zoom will automatically adjust
```

Creating zoom action

Demo Video: 01:18 - 01:27

This creates a camera action which will continue without user input until it reaches its end. Can specify a time to complete in, 0 is instant. If not specified it will have a default time.

```
m_camera.zoomTo(2.5f); //zoom in from current zoom amount to the specified amount or to the
                                        //min/max threshold
Camera2D::Vector target(1.5f, 1.f);
m_camera.zoomTo(target); //can also zoom using vector for different aspect ratio

m_ccamera.zoomTo(0.5f, 10.f); //10 seconds
```

Zooming to fit

Link to GDD (video)

Demo Video: 00:42 - 01:07

The camera can zoom to fit a group of points or rects iif the min zoom threshold is big enough. If zooming in and required zoom is smaller than max zoom then max zoom will be what is ended up at.

```
m_camera.setZoomMinMax(-1, -1); //unlimited max and min zoom to ensure we can fit all
//objects

Bool keepZoomRatio = false; //change ratio so we might end up zooming more on x/y in order
```

```
to get a perfect fit

vector<Camera2D::Point> points;
points.push_back(Camera2D::Point(0,10));
points.push_back(Camera2D::Point(10,5));
m_camera.zoomToFit(points, keepZoomRatio); //camera zoom will end up large enough to fit all
points

vector<SDL_Rect> rects;
boxes.push_back(SDL_Rect(10, 10, 100, 50));
boxes.push_back(SDL_Rect(100, 5, 10, 50));
m_camera.zoomToFit(rects, keepZoomRatio); //each box will fit inside the camera

m_camera.resetZoomRatio(); //will reset the zoom ratio if it has changed. Resets by getting
middle of x and y zoom difference and zooming both components to there

//similarly to zoomTo, you can specify a time to the 3rd parameter of zoomToFit for a time
//to complete the action in
m_camera.zoomToFit(rects, keepZoomRatio, 2.f); //2 seconds
```

Creating custom parallax effect

[Link to GDD (video)]

[Demo Video: 01:27 - 01:59]

Parallax scrolling can be achieved by adding layers to a Parallax effect which is then added to the camera. EffectsLink to GDD (video) can be started by passing the effect object or starting to default effect with an enum.

```
Camera2D::ParallaxEffect pe;
int numberOfTextures = 5;
float scrollSpeed = 2.f;
pe.addLayer(Layer("/path/to/textures", numberOfTextures, scrollSpeed));
pe.addLayer(Layer("/path/to/textures2", 3, 1.f));
pe.setName("parallaxEffect"); // identifier of this parallaxEffect

m_camera.addEffect(pe); //add the effect to the camera

//can also add the name when adding effect, like so
m_camera.addEffect(pe, "parallaxEffect");
```

Starting/ stopping effects

Effects can be stopped at any time. Specific or generic effects can be started at any point.

```
m_camera.startEffect(Camera2D::Effect::Parallax); //if there effects stored then start the
first stored
Camera2D::ParallaxEffect pe;
m_camera.startEffect(pe); //start the created effect
m_camera.startEffect("parallaxEffect"); //can be started by name

m_camera.endEffect(Camera2D::Effect::Parallax, true); //end the current effect if there is
one, true tells it to also remove it from being stored. Default value here is false

m_camera.endEffects(); //stop all effects
m_camera.endEffect(Camera2D::Effect::Parallax, bool remove); //stop current Parallax
```

```
effect,remove defaults to false but set to true to also remove from storage
m_camera.endEffect(const std::string& name, bool remove);

void removeEffect(const std::string& name);//removes given effect
void removeEffect(Effect::Type type); //removes first occurrence of effect type
```

Modifying Effects

Effects that have been stored in the camera can be modified

```
m_camera.findShake("shakeEffect")->setMagnitude(0.5f);
//can change magnitude,duration,speed,range of shake
//can change scroll speeds of layers of parallax

//findShake/findParallax to find specific pointer type
//findEffect to find base pointer (which of course could be down cast)
```

Camera shake

[Link to GDD (video)]

[Demo Video: 01:59 - 02:23]

The camera is able to shake. This causes it to move short distances in random directions in uniform manner. (perlin noise) This effect can be started at any time and it's magntiude and duration can be changed.

```
ShakeEffect se;

float duration = 2.f; //length of shake
float speed = 2.f;
float magnitude = 1.f; //how strong the shake is
//float range; //can be provided to change range used to choose noise parameter, defaults to
1000.f if not provided
se.setProps(duration, speed, magnitude); //set shake properties

m_camera.addEffect(se, "shake"); //add effect and give it a name for id

//can be started and ended like the Parallax Effect
```

Manipulating camera bounds and position

The camera bounds and centre can be got and changed easility.

```
BoundingBox bounds(0, 0, 1920, 1080);
int x = 0;
int y = 100;
int width = 1280;
int height = 720;

m_camera.setBounds(bounds);
m_camera.setCentre(x + width * 0.5f, y + height * 0.5f);
m_camera.setSize(width, height);

//size and bounds and centre are got with
m_camera.getBounds(); //return SDL_rect of the camera bounds
```

```
m_camera.getCentre(); //return Camera2D::Vector of camera centre x and y position
m_camera.getSize();   //return Camera2D::Vector of camera bounds width and height
```

Constraints

We can limit the motion of the camera.

```
m_camera.setAllowedHorizontal(true); //true to allow horizontal motion, false to stop
m_camera.setAllowedVertical(false); //true to allow vertical motion, false to stop
m_camera.lockMotion(); //stop all motion
//m_camera.unlockMotion() to resume

//to get current values:
m_camera.getAllowedHorizontal()
m_camera.getAllowedVertical()

m_camera.resetMotion() // stops motion but forces still apply
```

Culling

There is an intersects function to check if a Camera2D::Point/Vector or a SDL_rect is intersecting with the camera. With this information the user can then not draw the object in order to cull offscreen objects. The point/rect they are testing must first be converted into screen Coordinates with the worldToScreen function.

```
SDL_Rect worldRect = { 0, 0, 10, 10 }; //rect in our world
SDL_Rect screenRect = m_camera->worldToScreen(worldRect ); //apply camera transform to //get
in screen coords
if (m_camera->intersects(r)) //check if it intersects with camera
    //draw object as it is on screen

//above is done the same for a Camera2D::Vector/Camera2D::Point
```

Attractors/Repulsors

Link to GDD (video)
Demo Video: 02:23 - 02:52

An attractor or repulsor can be used to influence a camera's position. An attractor will try to draw the camera towards it. A repulsor is the opposite and will push the camera away from it.

```
m_camera.setAttractorStopVel(20.f); //set velocity range at which attractors zero out the
camera velocity

Camera2D::Attractor attractor;
```

```
Camera2D::Vector2 v(10, 20);
Float affectRange = 400.f;
attractor.setProps(v, affectRange); //set position and range
attractor.setStrength(2.f); //set strength
m_camera.addInfluencer(attractor, "attractor1"); //add and give a name

m_camera.removeInfluencer("attractor1"); //remove
void removeAllInfluencer(Camera2D::Influencer::Type::Attractor); //remove all attractors

Influencer* findInfluencer(const std::string& name); //return a influencer point for given
name id in order to modify it
```

Demonstration of this component

This section describes the features of an application which demonstrates the feature set of this component.

Demo Game Feature

- Using directional arrow keys pans the camera
- y/g changes max velocity of camera motion
- u/h changes acceleration rate of camera
- Left click draws a blue box at mouse pointer
- Right draws a red point at mouse pointer
- Press z to zoom to fit blue boxes
- Press x to zoom to fit red points
- Press c to zoom to fit red points but don't maintain aspect ratio
- Pressing Q clears rectangles and points
- Mouse scroll wheel zooms in and out and resets aspect ratio
- Pressing v sets a limit to 0.5f for max zoom instead of starting unlimited min and unlimited max
- Pressing [zooms to the min zoom threshold or 3.f if there is no limit
- Pressing] again zooms to the max zoom threshold
- Pressing P starts parallax effect
- Pressing P while there is a parallax effect stops it
- Pressing O increases the parallax speed, and K decreases it
- Similarly pressing S for shake starts and stops the Shake effect
- Pressing I increases shake magnitude and J decreases it
- Pressing 1 constrains horizontal motion, pressing again frees it
- Pressing 2 constrains vertical motion, pressing again frees it
- Pressing 3 locks the camera, pressing it again unlocks it
- Pressing A places an attractor at the hovered mouse position
- Pressing R places a repulsor at the hovered mouse position
- Pressing W clears attractors and repulsors