GFC 2.7

Games Fundamental Classes
Cheat Sheet

Member Object Sprites

Variable definition
is typically placed within CMyGame class in your MyGame.h file

```
class CMyGame : public CGame
{
    CSprite mySprite;
```

Constructor call in *.cpp file
Constructors of member objects should be placed just before the class body of your game class – typically CMyGame class in MyGame.cpp file

```
CMyGame::CMyGame() : mySprite(50, 300, "rocket.bmp", CColor::Blue(), 0)
{
    // here follows the body of the CMyGame class constructor
```

Standard constructors for bitmap sprites — ***Note:*** *to construct two or more member objects, list them all separating with commas*

```
CSprite mySprite1, mySprite2, mySprite3, mySprite4;     // Variable definitions in the *.h file
CMyGame::CMyGame()                                      // Sample constructors in the *.cpp file:
: mySprite1(50, 300, "rocket.bmp", 0),                  // image loaded from rocket.bmp, placed at (50, 300)
  mySprite2(50, 300, 150, 100, "rocket.bmp", 0),        // width set to 150px, height to 100px (image may be deformed)
  mySprite3(50, 300, "rocket.bmp", CColor::Blue(), 0), // every blue pixel will be converted into transparent
  mySprite4(50, 300, 150, 100, 0)                       // without a filename: see deferred image loading below
```

Specialised classes of sprites: ovals, rectangles and texts

```
CSpriteOval myCircle;
CSpriteOval myEllipse;
CSpriteRect myRect;
CSpriteText myText;

myCircle (400, 100, 30, CColor::Red(), 0),       // creates a red circle with radius 30
myEllipse(400, 100, 100, 30, CColor::Red(), 0), // creates an ellipse 100 px wide, 30 px high
myRect(400, 100, 100, 50, CColor::Red(), CColor::Black(), 0), // black is colour of outline
myText(100, 100, "ARIAL.TTF", 36, "my text here", CColor::Red(), 0)  // text in Arial size 36
```

These operations are essential for sprites: **Updating**: `mySprite.Update(GetTime());` **Drawing**: `mySprite. Draw(g);`

Controlling Dynamic Sprites using Lists & Vectors

Variable definition

```
CSpriteList myList;
CSpriteVector myVec;
```

Adding a new sprite to a list

```
CSprite *p = new CSprite(50, 300, "rocket.bmp", CColor::Blue(), GetTime());
myList.push_back(p);
```

Deleting all sprites

```
myList.delete_all();
```

Updating all Sprites in a List/Vector

```
for (CSprite *pSprite : myList)
        pSprite->Update(GetTime());
```

Drawing all Sprites in a List/Vector

```
for (CSprite *pSprite: myList)
        pSprite->Draw(g);
```

Deleting all Sprites in a List/Vector

```
myList.delete_all();
```

Creating Sprites at Random Time

```
if (rand() % 60 == 0)    // a new sprite will be created, on average, once every 60 frames
{                        // at X coord 850 and Y coord random between 0 and 600
        CSprite *pSprite = new CSprite(CVector(850, rand() % 600), "enemy.bmp", CColor::Blue(), GetTime());
        pSprite->SetVelocity(-100, 0);               // newly created sprite will move to the left, 100px/sec.
        myList.push_back(pSprite);                   // and added to a list
}
```

Using Objects and Pointers

create a static object	`CSprite sprite( ... );`	create a dynamic object	`CSprite *p = new CSprite(...);`
function call with an object	`sprite.SetSpeed(100);`	function call with an object	`p->SetSpeed(100);`
delete an object	`//static obj can't be deleted`	delete a dynamic object	`delete p;`

Deferred Image Loading and Animated Sprites

Deferred Image Loading is used for sprites that often change their bitmap image or display animations. This technique allows for loading images (LoadImage or LoadAnimation) giving them an alias name and storing the image(s) internally within the sprite. The stored images can then be very quickly set for display (use SetImage for single images, SetAnimation for animated sequences).
To use a sprite sheet you have to specify the number of columns and rows, and then provide col/row coords:

```
// create a sprite with no image:
mySprite(50, 300, 150, 100, 0);
// load image with an alias name
mySprite.LoadImage("walk1.bmp", "pose1");
// load an image from a sprite sheet. Tile(0, 0) is lower left corner. Tile(3, 2) upper right.
mySprite.LoadImage("sheet.bmp", "pose2", CSprite::Sheet(4, 3).Tile(3, 2), CColor::Green());
// load an animated sequence from a sprite sheet
mySprite.LoadAnimation("sheet.bmp", "walk", CSprite::Sheet(4, 3).Row(2).From(0).To(3), CColor::Green());
mySprite.SetImage("pose1");      // set the current image to "pos1"
mySprite.SetAnimation("walk");  // set the current animation to "walk"
```

check if any animation is playing

```
if (mySprite.IsAnimationPlaying())
```

get the name of the currently playing animation (NULL if none)

```
const char *pName = mySprite.GetCurrentAnimation();
```

check if the named animation is playing

```
if (mySprite.IsAnimationPlaying("walk"))
```

get the frame index of the currently playing animation (-1 if none)

```
int iFrame = mySprite.GetCurrentAnimationFrame();
```

Sprite Position and Size

get sprite centre coordinates

```
CVector GetPosition();
CVector GetPos();
float GetX();
float GetY();
```

get / set sprite edge coordinates

```
float GetLeft();
float GetRight();
float GetTop();
float GetBottom();
```

get / set sprite size

```
CVector GetSize();
float GetWidth();   float GetW();
float GetHeight();  float GetH();
void SetSize(CVector v);
void SetSize(float w, float h);
```

move centre in absolute coordinates

```
void SetPosition(CVector v);
void SetPosition(float x, float y);
void SetPos(CVector v);
void SetPos(float x, float y);
```

```
void SetLeft(float x);
void SetRight(float x);
void SetTop(float y);
void SetBottom(float y);
```

bounding rectangle – position and size of the sprite

```
void GetBoundingRect(CRectangle &rect);
```

client rectangle – upper-left corner is (0, 0)

```
void GetClientRect(CRectangle &rect);
```

Pivot point – reference point for sprite positional functions, by default centre of the sprite point.

```
void SetPivot(CVector v);
void SetPivot(float x, float y);
```

move centre relative to the current position

```
void Move(CVector v);
void Move(float dx, float dy);
```

*convert between **global** and **local** coords*

```
void GtoL(CVector &p);
void LtoG(CVector &p);
```

Basic Motion

Speed (scalar only) [pixels per sec]

```
float GetSpeed();
void SetSpeed(float newV);
```

Direction (angle only) [degrees]

```
float GetDirection();
void SetDirection(float dir);
void SetDirection(CVector vec);
void SetDirection(float dX, float dY);
```

*Velocity (vector of motion = speed * direction vector)*

```
CVector GetVelocity();
void SetVelocity(CVector v);
void SetVelocity(float vx, float vy);
```

Rotation and Rotational (Angular) Velocity

Rotation [degrees]

```
float GetRotation();
void SetRotation(float newRot);
void SetRotation(float a, float b);
void Rotate(float rot);
```

Angular Velocity [degrees per sec.]

```
float GetOmega();
void SetOmega(float newOmega);
```

Normalised Velocity (direction of movement)

```
CVector GetNormalisedVelocity();
void SetNormalisedVelocity(CVector v);
void SetNormalisedVelocity(float vx, float vy);
```

Simple Dynamics

Mass	`float GetMass();`	`void SetMass(float mass);`
Acceleration	`void Accelerate(CVector a);`	`void Accelerate(float ax, float ay);`
Force	`void ApplyForce(CVector f);`	`void ApplyForce(float fx, float fy);`

Simple Bounce
(floor level = 600, restitution = 0.8)

```
CVector v = m_sprite.GetVelocity();              // velocity
CVector n = CVector(0, 1);                       // normal
if (collision && Dot(v, n) < 0)
        m_sprite.SetVelocity(k * Reflect(v, n)); // k is restitution factor
```

Testing for Collisions

```
// Game is over if pMyRocket hits any spaceship
for (CSprite *pShip : spaceships)
        if (pShip->HitTest(pPlayer))
                GameOver();
```

Bullets & Enemies

```
// remove every spaceship hit by a missile
// (and a missile as well
for (CSprite *pShip : spaceships)
        for (CSprite *pMissile : missiles)
                if (pShip->HitTest(pMissile))
                {
                        pShip->Delete();
                        pMissile->Delete();
                        nScore += 10;
                }
spaceships.delete_if(deleted);
missiles.delete_if(deleted);
```

Deleting from a Collection

Note: *Items must not be removed in for-each; they can only be marked!*

```
// remove any spaceship hit by the rocket
for (CSprite *pShip : spaceships)
        if (pShip->HitTestPixel(pPlayer))
                pShip->Delete();
spaceships.delete_if(deleted);
```

Game Utilities – CGame class functions (not CSprite!)

Window or Screen Size

```
CVectorI GetSize();
Sint16 GetWidth();
Sint16 GetHeight();
```

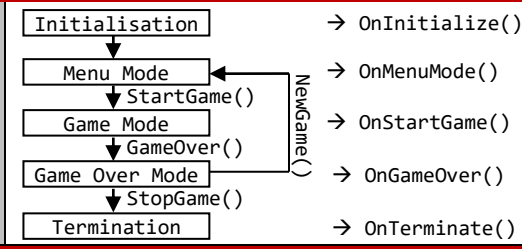
Time Information
Milliseconds from the start of the game (or entering the GameOver mode)

```
long GetTime();
long GetDeltaTime();
long GetTimeGameOver();
```

Game Level Id
Game level is automatically incremented with each new game

```
int GetId();
void IncrementId();
void ResetId();
```

Game Life Cycle



Controlling the game life cycle

```
void StartGame(); // → Game Mode
void GameOver();  // → Game Over mode
void NewGame();   // → Menu Mode
(restart)
void PauseGame(s);// pauses the game
void StopGame();  // quits the program
```

Testing the game mode

```
bool IsRunning();
bool IsPaused();
bool IsMenuMode();
bool IsGameMode();
bool IsGameOverMode();
bool IsGameOver();
```

Sound

declare as static objects:

```
CSoundPlayer playerMusic;
CSoundPlayer playerEffect;
```

use several players is sounds to be played simultanously

Use the player

```
pl.Play("mus.wav");          // play once
pl.Play("mus.wav", 2);       // repeat sound twice
pl.Play("mus.wav", -1);      // play continuously
pl.Play("mus.wav", 0, 1000); // fade-in for 1000 ms
```

```
pl.Pause();
pl.Resume();
pl.Volume(vol);
pl.Stop();
pl.FadeOut(1000);
```

Text Output

```
*g << "by default, this text will is displayed in top left corner of the window" << endl << "and this is a new line";
*g << center << "this text is centered" << right << "and this right-aligned";
*g << bottom << "this line is in the bottom of the window" << top << left << "and this back in the top-left corner";
// the following shows big red letters in the centre of the screen if game is over; otherwise the timer in the bottom
if (IsGameOverMode()) *g << font(40) << color(CColor::Red()) << vcenter << center << "GAME OVER";
else                  *g << font(18) << color(CColor::Blue()) << bottom << timetext(GetTime());
```

modifiers

horizontal alignment	`left, right, center, centre`
vertical alignment	`top, bottom, vcenter, vcentre, down, up`
absolute positioning	`row(r), col(c), rowcol(r, c), xy(x, y)`
decoration time formatting	`font(name), font(size), font(name, size), color(c), color(r,g,b,a), margins(l,r,u,b)`
formatting	`timetext(ms)`
standard C++ modifiers	`hex, dec, oct, setw(w), setprecision(prec) and more`

Contact me at jarek@kingston.ac.uk

available in a larger font format

Version 4.7 November 2022