|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Member Object Sprites** | | | | | | | | | | | | | | | | | | | | | | | | | **GFC** **Games Fundamental Classes** Cheat Sheet | |
| Variable definition *is typically placed within CMyGame class in your MyGame.h file* | | | | | | | | 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* | | | | | | | | | | | | | | | | |
| class CMyGame : public CGame  {  **CSprite mySprite;** | | | | | | | | 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 | | | | Adding a new sprite to a list | | | | | | | | | | | | | | | | | | | | | | Deleting all sprites |
| CSpriteList myList;  CSpriteVector myVec; | | | | CSprite \*p = new CSprite(50, 300, "rocket.bmp", CColor::Blue(), GetTime()); myList.push\_back(p); | | | | | | | | | | | | | | | | | | | | | | myList.delete\_all(); |
| **Updating all Sprites in a List/Vector** | | | | | | | | | | | | **Drawing all Sprites in a List/Vector** | | | | | | | | | **Deleting all Sprites in a List/Vector** | | | | | |
| for (CSprite \*pSprite : myList)  pSprite->Update(GetTime()); | | | | | | | | | | | | for (CSprite \*pSprite: myList)  pSprite->Draw(g); | | | | | | | | | 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:* | | | | | | | | | | | | | | | | | | | | | | | | | | |
| // use sprite with no image:  mySprite(50, 300, 150, 100, 0) | | | | | | | | | | | | | | | | | | | | | | | | | | |
| *C:\Users\Jarek\Desktop\Tunic.bmp*// 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::Grid(4, 3).Tile(3, 2), CColor::Green()); | | | | | | | | | | | | | | | | | | | | | | | | | | |
| // load an animated sequence from a sprite sheet  mySprite.LoadAnimation("sheet.bmp", "walk", CSprite::Grid(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" | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **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 SetLeft(float x);  void SetRight(float x);  void SetTop(float y);  void SetBottom(float y); | | | | | | | | | *bounding rectangle – position and size of the sprite* | | | | | | | |
| void SetPosition(CVector v);  void SetPosition(float x, float y);  void SetPos(CVector v);  void SetPos(float x, float y); | | | | | | | | | | 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.* | | | | | | | |
| *move centre relative to the current position* | | | | | | | | | | *convert between* ***global*** *and* ***local*** *coords* | | | | | | | | |
| void Move(CVector v);  void Move(float dx, float dy); | | | | | | | | | | void GtoL(CVector &p);  void LtoG(CVector &p); | | | | | | | | | void SetPivot(CVector v);  void SetPivot(float x, float y); | | | | | | | |
| **Basic Motion** | | | | | | | | | | | | | | | | | | | | | | | | | | |
| *Speed (scalar only) [pixels per sec]* | | | | | | | | | *Direction (angle only) [degrees]* | | | | | | | | | *Velocity (vector of motion = speed \* direction vector)* | | | | | | | | |
| float GetSpeed();  void SetSpeed(float newV); | | | | | | | | | float GetDirection();  void SetDirection(float dir); | | | | | | | | | CVector GetVelocity();  void SetVelocity(CVector v);  void SetVelocity(float vx, float vy); | | | | | | | | |
| **Rotation and Rotational (Angular) Velocity** | | | | | | | | | | | | | | | | | | *Normalised Velocity (direction of movement)* | | | | | | | | |
| *Rotation [degrees]* | | | | | | | | | *Angular Velocity [degrees per sec.]* | | | | | | | | | CVector GetNormalisedVelocity();  void SetNormalisedVelocity(CVector v);  void SetNormalisedVelocity(float vx, float vy); | | | | | | | | |
| float GetRotation();  void SetRotation(float newRot);  void SetRotation(float a, float b);  void Rotate(float rot); | | | | | | | | | float GetOmega();  void SetOmega(float newOmega); | | | | | | | | |
| **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** | | | | | | | | | | | | | | | | | **Bullets & Enemies** | | | | | | | | | |
| // Game is over if pMyRocket hits any spaceship  for (CSprite \*pShip : spaceships)  if (pShip->HitTest(pPlayer))  GameOver(); | | | | | | | | | | | | | | | | | // 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*** | | | | | | | | | | | ***Time Information***  *Milliseconds from the start of the game (or entering the GameOver mode)* | | | | | | | | | ***Game Level Id***  *Game level is automatically incremented with each new game* | | | | | | |
| CVectorI GetSize();  Sint16 GetWidth();  Sint16 GetHeight(); | | | | | | | | | | | long GetTime();  long GetDeltaTime();  long GetTimeGameOver(); | | | | | | | | | int GetId();  void IncrementId();  void ResetId(); | | | | | | |
| *Game Life Cycle* | Initialisation  Menu Mode  Game Mode  Game Over Mode  Termination  StartGame()  GameOver()  StopGame()   |  | | --- | | NewGame() |   🡪 OnInitialize()  🡪 OnMenuMode()  🡪 OnStartGame()  🡪 OnGameOver()  🡪 OnTerminate() | | | | | | | | | | | | *Controlling the game life cycle* | | | | | | | | | | | | *Testing the game mode* | |
| 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 | | | | | | | | | | | | bool IsRunning();  bool IsPaused();  bool IsMenuMode();  bool IsGameMode();  bool IsGameOverMode();  bool IsGameOver(); | |
| **Sound** | | | | | | | | | | | | | | | | | | | | | | | | | | |
| *declare as static objects:* | | | | | | | *Use the player* | | | | | | | | | | | | | | | | | | | |
| CSoundPlayer playerMusic;  CSoundPlayer playerEffect; | | | | | | | 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); | | |
| *use several players is sounds to be played simultanously* | | | | | | |
| **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 | | | | | | | | | | | | | | | | | | |

Written by Jarek Francik, Kingston University, under Creative Common License: you are free to share (copy, distribute and transmit the work) and remix (adapt the work) but you must attribute the work to its original author. For non-commercial or educational use only. Tiled sprite image © by Courtney Senior.  
Contact me at jarek@kingston.ac.uk available in a larger font format Version 4.60 November 2018