* Một khi bạn biết cách kiểm tra va chạm giữa hai HCN, bạn có thể kiểm tra va chạm giữa hai hình ảnh vì tất cả hình ảnh đều có dạng HCN.
* Mọi thứ có thể tạo ra từ HCN trong video game. Vì ảnh được tạo từ các pixel nên tất cả những gì chúng ta phải làm là xem tập hợp các box colliders của mỗi vật thể có phần tử chung không?

|  |
| --- |
| //The dot that will move around on the screen  class Dot  {      public:          //The dimensions of the dot          static const int DOT\_WIDTH = 20;          static const int DOT\_HEIGHT = 20;          //Maximum axis velocity of the dot          static const int DOT\_VEL = 1;          //Initializes the variables          Dot( int x, int y );          //Takes key presses and adjusts the dot's velocity          void handleEvent( SDL\_Event& e );          //Moves the dot and checks collision          void move( std::vector<SDL\_Rect>& otherColliders );          //Shows the dot on the screen          void render();          //Gets the collision boxes          std::vector<SDL\_Rect>& getColliders();      private:          //The X and Y offsets of the dot          int mPosX, mPosY;          //The velocity of the dot          int mVelX, mVelY;            //Dot's collision boxes          std::vector<SDL\_Rect> mColliders;          //Moves the collision boxes relative to the dot's offset          void shiftColliders();  }; |

* Class dot phục vụ cho việc kiểm tra va chạm từng pixel. Vận tốc giảm còn 1 pixel/khung hình để kiểm tra va chạm dễ nhìn hơn. Hàm di chuyển bây giờ sẽ lấy một vector của collision box để chúng ta có thể kiểm tra hai tập với nhau. Vì chúng ta sẽ có hai điểm va chạm, chúng ta cần có khả năng lấy colliders, vì thế chúng ta có hàm cho việc đó.
* Thay vì có một collision box đơn, chúng ta có một vector các colliders. Chúng ta cũng có một hàm bên trong điểm dịch các colliders để khớp với vị trí của điểm.

|  |
| --- |
| Dot::Dot(int x, int y)  {      // Initialize the offsets      mPosX = x;      mPosY = y;      // Create the necessary SDL\_Rects      mColliders.resize(11);      // Initialize the velocity      mVelX = 0;      mVelY = 0;      // Initialize the collision boxes' width and height      mColliders[0].w = 6;      mColliders[0].h = 1;      mColliders[1].w = 10;      mColliders[1].h = 1;      mColliders[2].w = 14;      mColliders[2].h = 1;      mColliders[3].w = 16;      mColliders[3].h = 2;      mColliders[4].w = 18;      mColliders[4].h = 2;      mColliders[5].w = 20;      mColliders[5].h = 6;      mColliders[6].w = 18;      mColliders[6].h = 2;      mColliders[7].w = 16;      mColliders[7].h = 2;      mColliders[8].w = 14;      mColliders[8].h = 1;      mColliders[9].w = 10;      mColliders[9].h = 1;      mColliders[10].w = 6;      mColliders[10].h = 1;      // Initialize colliders relative to position      shiftColliders();  } |

* Vẫn như trước, chúng ta thiết lập kích thước của colliders trong constructor. Chỉ khác là chúng ta có nhiều collision box để thiết lập.

|  |
| --- |
| void Dot::move(std::vector<SDL\_Rect> &otherColliders)  {      // Move the dot left or right      mPosX += mVelX;      shiftColliders();      // If the dot collided or went too far to the left or right      if ((mPosX < 0) || (mPosX + DOT\_WIDTH > SCREEN\_WIDTH) || checkCollision(mColliders, otherColliders))      {          // Move back          mPosX -= mVelX;          shiftColliders();      }      // Move the dot up or down      mPosY += mVelY;      shiftColliders();      // If the dot collided or went too far up or down      if ((mPosY < 0) || (mPosY + DOT\_HEIGHT > SCREEN\_HEIGHT) || checkCollision(mColliders, otherColliders))      {          // Move back          mPosY -= mVelY;          shiftColliders();      }  } |

* Hàm này khá giống trước. Mỗi khi chúng ta di chuyển điểm, ta di chuyển collider với nó. Sau khi chúng ta di chuyển điểm, chúng ta kiểm tra xem nó ra ngoài màn hình hay chạm gì chưa. Nếu rồi thì di điểm trở lại cũng như di colliders trở lại.

|  |
| --- |
| void Dot::shiftColliders()  {      // The row offset      int r = 0;      // Go through the dot's collision boxes      for (int set = 0; set < mColliders.size(); ++set)      {          // Center the collision box          mColliders[set].x = mPosX + (DOT\_WIDTH - mColliders[set].w) / 2;          // Set the collision box at its row offset          mColliders[set].y = mPosY + r;          // Move the row offset down the height of the collision box          r += mColliders[set].h;      }  }  std::vector<SDL\_Rect> &Dot::getColliders()  {      return mColliders;  } |

* Đừng quá quan tâm shiftColliders hoạt động ntn. Nó chỉ là một cách viết ngắn của việc gán mColliders[0].x = …, mColliders[1].x = …, … và nó chỉ hoạt động trong trường hợp này. Với từng vật thể thì sẽ có hàm khác nhau.
* Sau khi shiftColliders, thì có một hàm accessor để lấy colliders.

|  |
| --- |
| bool checkCollision(std::vector<SDL\_Rect> &a, std::vector<SDL\_Rect> &b)  {      // The sides of the rectangles      int leftA, leftB;      int rightA, rightB;      int topA, topB;      int bottomA, bottomB;      // Go through the A boxes      for (int Abox = 0; Abox < a.size(); Abox++)      {          // Calculate the sides of rect A          leftA = a[Abox].x;          rightA = a[Abox].x + a[Abox].w;          topA = a[Abox].y;          bottomA = a[Abox].y + a[Abox].h; |

* Đây là hàm phát hiện va chạm, chúng ta có một vòng lặp mà tính top/bottom/left/right của mỗi collision box của vật A.

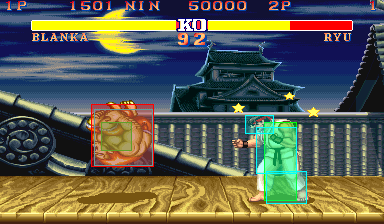
|  |
| --- |
| // Go through the B boxes          for (int Bbox = 0; Bbox < b.size(); Bbox++)          {              // Calculate the sides of rect B              leftB = b[Bbox].x;              rightB = b[Bbox].x + b[Bbox].w;              topB = b[Bbox].y;              bottomB = b[Bbox].y + b[Bbox].h;              // If no sides from A are outside of B              if (((bottomA <= topB) || (topA >= bottomB) || (rightA <= leftB) || (leftA >= rightB)) == false)              {                  // A collision is detected                  return true;              }          }      }      // If neither set of collision boxes touched      return false;  } |

* Sau đó chúng ta tính top/bottom/left/right của từng collision box của vật B. Sau rồi kiểm tra có tách nhau trên trục không. Nếu tác trên trục thì return true. Sau khi xét hết tập thì return false.

|  |
| --- |
| // Main loop flag              bool quit = false;              // Event handler              SDL\_Event e;              // The dot that will be moving around on the screen              Dot dot(0, 0);              // The dot that will be collided against              Dot otherDot(SCREEN\_WIDTH / 4, SCREEN\_HEIGHT / 4); |

* Trước khi chúng ta đi vào trong main loop, Chúng ta khai báo điểm của chúng ta và điểm cần kiểm tra va chạm.

|  |
| --- |
| // While application is running              while (!quit)              {                  // Handle events on queue                  while (SDL\_PollEvent(&e) != 0)                  {                      // User requests quit                      if (e.type == SDL\_QUIT)                      {                          quit = true;                      }                      // Handle input for the dot                      dot.handleEvent(e);                  }                  // Move the dot and check collision                  dot.move(otherDot.getColliders());                  // Clear screen                  SDL\_SetRenderDrawColor(gRenderer, 0xFF, 0xFF, 0xFF, 0xFF);                  SDL\_RenderClear(gRenderer);                  // Render dots                  dot.render();                  otherDot.render();                  // Update screen                  SDL\_RenderPresent(gRenderer);              } |

* Một lần nữa trong main loop, chúng ta xử lý events cho điểm, di chuyển với kiểm tra collision cho điểm, và cuối cùng render vật.
* Một câu hỏi tôi gặp khá nhiều là làm thế nào để làm một hàm mà load ảnh và tự động generate tập hợp các collision box cho mỗi lần phát hiện va chạm từng pixel. Câu trả lời đơn giản là: Không.
* Trong hầu hết game, bạn không cần độ chính xác 100%. Càng nhiều collision box, càng nhiều va chạm cần xét và game càng chậm. Điều mà các trò chơi hướng đến là gần đúng, ví dụ game Street Fighter: 
* Không hoàn hảo đến từng pixel nhưng khá đủ.
* Và cũng có một thứ ta có thể tối ưu. Ta có thể có một bounding box là các điểm bao quanh tất cả các collision box và sau đó kiểm tra xem cái đầu tiên trước khi đi đến per-pixel collision box. Điều này sẽ thêm một lần phát hiện va chạm, nhưng vì nhiều khả năng là hai đối tượng không va chạm nên sẽ tiết kiệm rất nhiều thời gian. Trong game, điều này có thể được làm với CTDL cây, cái mà có các level chi tiết khác nhau để cho phép xuất hiện sớm nhằm tránh các lần kiểm tra không cần thiết ở cấp độ pixel. Giống như trong các bài trước, cấu trúc cây nằm ngoài phạm vi của chúng tôi.