Computer Graphics and Animation

Programming Assignment 15

Sprite Animation - Blast Hornet

Created by:

Natasya Oktavioni Susanto (001201900024)

Renaldo Fareza Tambunan (001201900084)

Samuel (001201900005)

Table of Contents

I. Introduction

This is a program that displays an animation and is provided with user’s control. The user’s control includes some actions such as moving the character’s position and attacking the enemy. Each of the character’s position and its implementation will be thoroughly explained in the design section.

The program is built using VB.NET programming language as it is easier to implement some elements of the animation into the program. We are using several elements such as a picture box to display the character, a timer to control the time frame, and the background image will be displayed on the window screen.

To create this program, we need to consider several elements. The background, sprite animation, the state and position of the character, the collision detection, and any other functions on the program, every element needs the right method and algorithm for it to run smoothly and efficiently. To fulfil them, we compare various methods to see which method can satisfy our expectation for the program. The basic theory of these elements will be explained later in the basic theory section and for its implementation will be discussed in the implementation section.

II. Basic Theory

Sprite Animation

Sprite animations are animation clips that are created from sprite. Sprite is a single graphic 2D (dimensional) image that is incorporated to a larger scene. The sprite will be compiled in order to create the animation.

Putting Sprite on a Background

To put a sprite on a background, we need to create a mask of the sprite. Simply, we convert every colour that is not black into black, and the colour is black becomes white. To be noted, we have to change the sprite background colour becoming black before performing masking. After performing masking, the black contour of the sprite will be placed on a white background. We have to perform AND operation between the background and the sprite(s). In this AND operation, the colour will be as shown below:

1. Any Colour AND Black (RGB(0, 0, 0) or #000000) will be Black (RGB(0, 0, 0) or #000000). Example for this operation:
    - RGB(153, 51, 102) AND RGB(0, 0, 0) will result in RGB(0, 0, 0).
2. Any Colour AND White (RGB(255, 255, 255) or #FFFFFF) will be “that” Any Colour. Example for this operation:
    - RGB(204, 255, 204) AND RGB(255, 255, 255) will result in RGB(204, 255, 204).

After doing the AND operation, we will get our sprite in black contour, but the background is no longer white, but becoming the background we expect. Of course, we want to see our sprite moving with each real colour, not in black colour. Then, we have to perform an OR operation. Performing OR operation for the existing background (this already contains the black contour sprite) and the sprite(s), the colour will be shown below:

1. Any Colour OR Black (RGB(0, 0, 0) or #000000) will be “that” Any Colour. Example for this operation:
    - RGB(102, 0, 102) OR RGB(0, 0, 0) will result in RGB(102, 0, 102).

By this OR operation, we get back our original colour of the sprite(s) as expected.

State Diagram and Transition Table

To perform every character working as our expectation, we need a character state diagram. Character state diagram is a finite state representation to model transitions between character action. Each state indicates character actions. While creating a character state diagram, we try to minimize explanation in every arrow, therefore we use several colour codes to indicate the state, while the details will be explained further in the character state transition table. Several colour code arrow use in this course are as follows:

1. Black arrow indicates inputs that are based on decision,
2. Blue arrow indicates inputs based on finished animation,
3. Red arrow indicates inputs based on interruption of other character or the character is threatened by other character,
4. Green arrow indicates inputs based on the trigger of some events, but not from other characters.

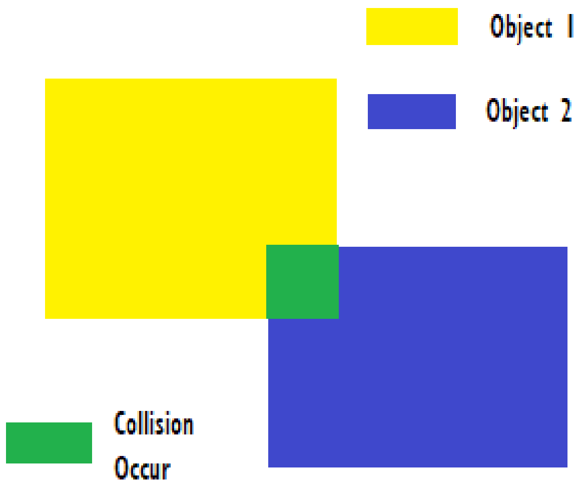
Using these colour code arrows, we minimize explanation on our character state diagram, however every detail of the trigger will be explained further on the character state transition table.

Collision Detection

Collision detection is a method that attempts to detect a collision between two or more different objects. In animation, a collision occurs when the shapes of two objects intersect or when the distance between these shapes falls below a certain tolerance.

There are several types of collision detection, such as Rectangle Collision Detection, Circle Collision Detection, Polygon Collision Detection, and Per Pixel Collision Detection.

1. **Rectangle Collision**

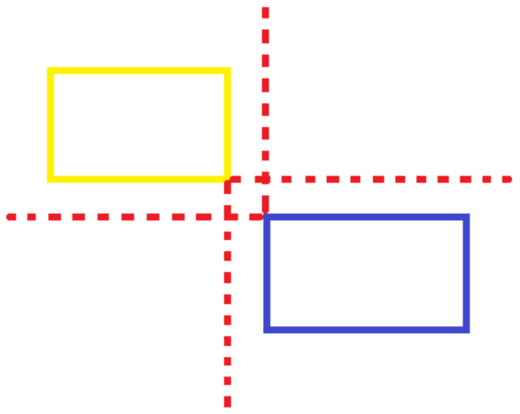


As the name suggests, the object is defined as a rectangle or box with its four sides and corners. For objects that don't need precise detection, rectangles are often used to identify collisions. A collision happens when both sides of each rectangle overlap or inside the other rectangle.

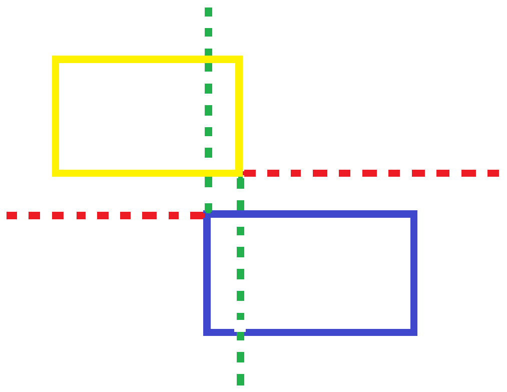
The easiest method to check whether the objects are colliding or not is by creating an anti-collision-detection algorithm — later will be used in our program and be explained more in the design section. Every edge will be evaluated using the if condition and will return a true value if the x axis and y axis overlaps.

The following images will illustrate the rectangle collision detection.

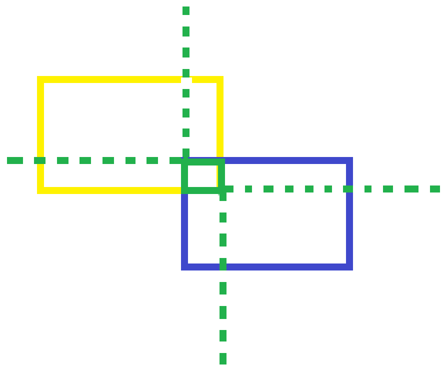
1. The rectangles have not collided yet. The red lines indicate that neither the right bottom of the yellow rectangle nor the left top edge of the blue rectangle fall inside one another.



2. The green lines mean one side of each rectangle has collided. Yet, it doesn't mean these two rectangles completely collided.



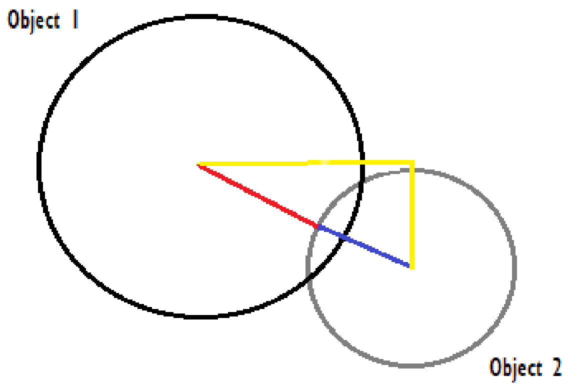
3. A collision has happened and both sides of each rectangle are inside the other rectangle.



In detail, the rules that need to be followed for two rectangles to overlap is as follows:

$$R_1 Left <= R_2 Right$$
$$R_2 Left <= R_1 Right$$
$$R_1 Bottom <= R_2 Top$$
$$R_2 Bottom <= R_1 Top$$

2. Circle Collision



As the figure above shown, each object has its own radius. Red lines represent the radius of Object 1 and the blue line represents the radius of Object 2. To detect a collision, a Pythagoras theorem must be performed in order to get the distance between the center points. A collision happens when the distance between center points is less than or equal the sum of radius of Object 1 and Object 2.

We can calculate the distance between two points with the following formula:

$$c = \sqrt{(x_1 - x_2)^2 + ((y_1 - y_2)^2}$$

$$Radius = (r1 + r2)^2$$

$$Return\, c \;<=\; Radius$$

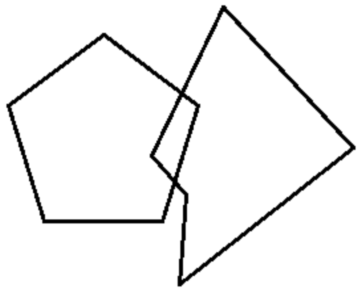
3. **Polygon Collision**

Polygons have various shapes, it doesn’t have a set radius or a set width and height. As a result, it is impossible to use the same method as the rectangle collision detection method. A polygon has corners, we can decide the type of a polygon by its corner whether it is a concave or a convex polygon.

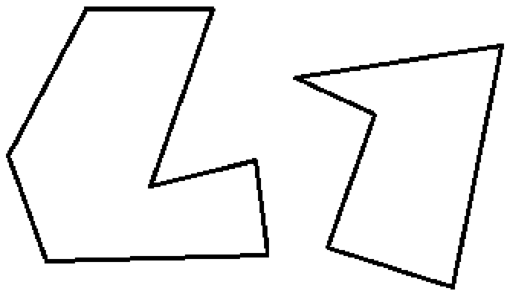
To detect if two polygons are intersecting, we use the Separating Axis Theorem. This theorem's implementation is straightforward, and it can be summarized as follows:

For each edge of both polygons:

1. Find the axis perpendicular to the current edge.
2. Project both polygons on that axis.
3. If these projections don't overlap, the polygons don't intersect (then exit the loop).



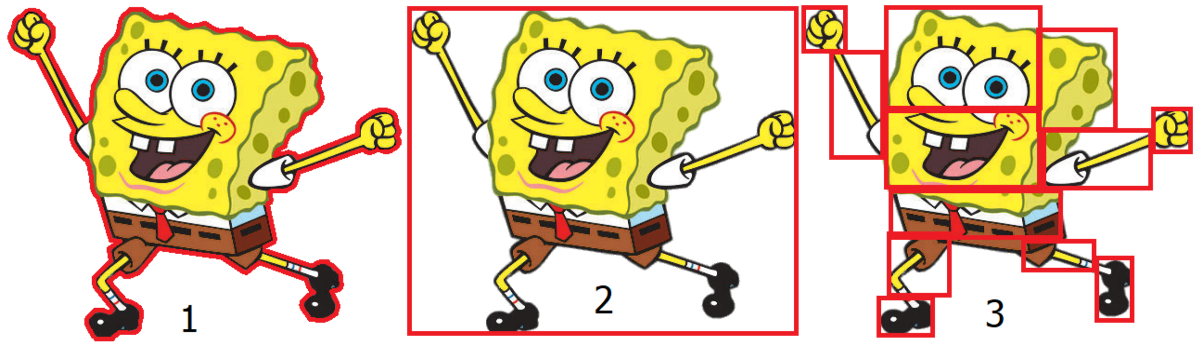
Objects overlap, collision happens.



Objects do not overlap.

4. Per Pixel Collision

For images, we can apply pixel perfect collision detection. But, it requires a heavy CPU processing since it needs to check every pixel to one another. To make things easier, the hitboxes are used. Hitboxes are imaginary geometric shapes around game objects that are used to determine collision detection. It's a method for making collision detection simpler that relies solely on basic geometric shapes. We can simply use the function for rectangle collision detection to check the hitboxes for collision.



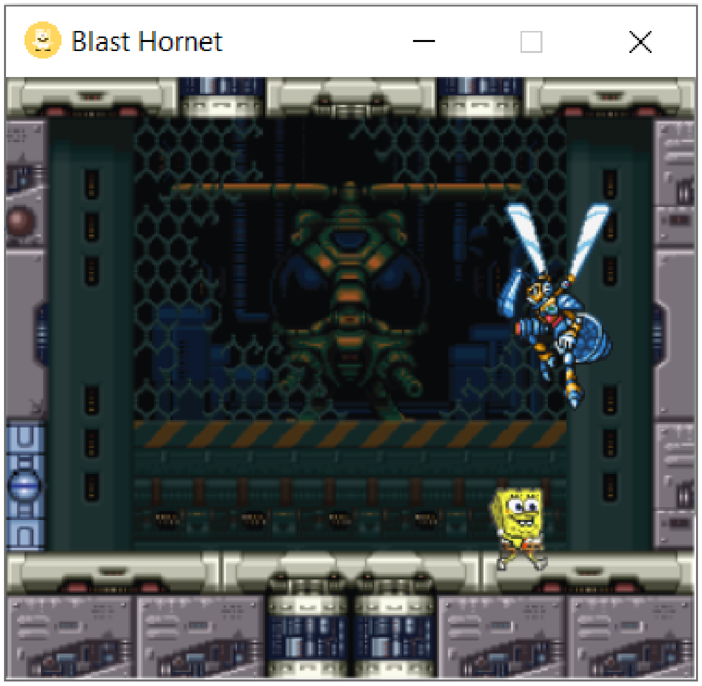
The image above illustrates the different types of collision detection. They each have their own advantages and disadvantages:

1. **Pixel perfect** - Extremely precise collision detection, but it requires a significant amount of system resources. In most cases this is excessive.
2. **Hitbox** - Much better performance, but the collision detection can be pretty imprecise. In many game scenarios though, this doesn't really matter. Each object has a hit area consisting of a hit box and hurt box. If a hit box of any object overlaps with another object's hurt box, that is when collisions happen.
3. **Multiple hitboxes** - Even though it is less powerful than a single hitbox, it outperforms the pixel perfect. And this can also support complex shapes. This is a good choice to use for important game objects that require some extra precision.

In the game, the pixel method is used when precise collision detection is important and not too many other game tasks are running. As a result, in this case, it is a good choice to use multiple hitboxes for more precise results.

In order to implement the requirements given, we need to implement a suitable method to display better graphics. We will use the rectangle collision detection for efficiency and its simplicity. By implementing the rectangle collision detection, the program will check the sides of each of the characters and will return a boolean value.

III. Implementation



The image above is the interface of our program. The interface displays the background of the animation and the characters — Blast Hornet and Spongebob dummy. The Spongebob dummy will be passive, it will produce projectiles and both of them will move randomly by default. On the other side, the Blast Hornet can be controlled by user input in the form of mouse click and keyboard keydown. All the features will be listed below:

- **Left Button Mouse** to Fly

  The program requires the Blast Hornet to fly around. In order to satisfy that, we implement it through the mouse click. If the user clicks on the left side of the mouse, then the Blast Hornet will move and fly to the coordinate of the mouse click. The Blast Hornet can move freely to any location inside the program, but for the facing direction of the Blast Hornet, it will follow wherever the spongebob goes.

- **Right Button Mouse** to released Parasitic Bomb

  Parasitic bomb is a projectile consisting of three tiny parasites, it comes from the Blast Hornet and will act as a weapon. To attack the dummy with a Parasitic Bomb, the user needs to click on the right button of the mouse. The parasites then will start to fly to the direction of the coordinate of the mouse click. If the dummy happens to be on the side of the projectile's direction, it will result in the death of the dummy.

- **Key Z** to released Sting Attack

  To perform the hit with stinger action, the user needs to press down the key Z of the keyboard. The animation will start not long after that. The Blast Hornet will aim at the current location of the spongebob and will fly down to the exact location. If the Blast Hornet successfully hit the target, which is the spongebob dummy, the target will be dead. Otherwise, if the hit misses, the Blast Hornet will fly back to its previous location.

- **Key X** to Shoot Crosshair

  A crosshair is an object with its main purpose to aim and lock the target. For the Blast Hornet to do this action, the user must press down the X key. Then the crosshair will come out and slowly home in on the target. Once it reaches the target, the crosshair will become large and will go along with the target. Since it successfully locked the spongebob dummy, all the parasite projectiles of the Blast Hornet will go after the spongebob dummy and attack it. The crosshair will disappear after 15 seconds.

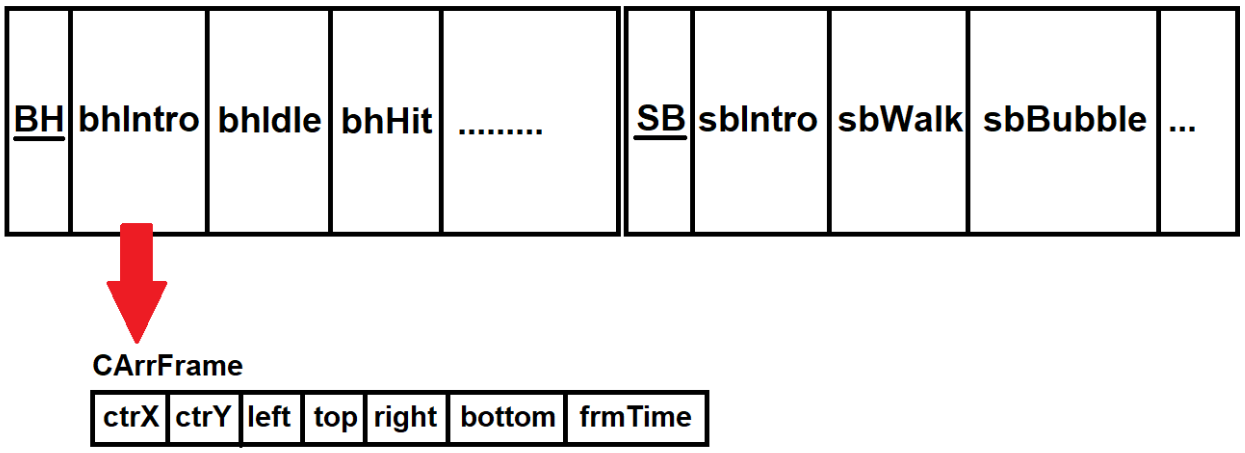
IV. Design

Data Structures

Array and List are used for data structure in this program that respectively apply for collection of frames in every state and collection of sprites of every which are represented as arrays and all characters are stored inside list. The implementation of each data structure explained below:

1. Array
   Array is used to store frames of the sprite animation and its position. For our program we use the dynamic array. To create a dynamic array, we must initialize a string array first by using the Dim statement, because we didn't know the array's actual size. We ReDim the statement to resize the existing array by specifying the subscript. After that, we insert the value into the array. The array index starts with 0 so we have to make sure that the dimension of our array is correct before using it. To access the element inside the array, we use the index.
2. List
   List is used to put every character that has to be displayed on the screen. We use list to store the characters that have to be shown because list is a resizable array. Using list, it will change the size directly when a data is inserted into a list.

Representation of background and sprite generally created as `CImage()`, it is needed to generate picture `.bmp` into computable format which is used for collecting, masking, and displaying. For sprite, it is detailed with a dynamic array `CArryFrame()` that is used to determine collection of sprites per state. It consists of an index to track the element and `CElementFrame()` that consist of center point, top/bottom/left/right point, index, max frame time.



Global variables

In this program, global variables generally are used for representing the classes which purpose the object of character itself and carry the results for calculation of each character motion. Any details will spell out below:

```
Dim bmp As Bitmap
Dim bg, bg1, img As CImage
Dim SpriteMap As CImage
Dim SpriteMask As CImage
Dim bhIntro, bhIdle, bhHit, bhHitWall,
    bhintroBomb, bhBomb, bhAfterBomb,
    bhShield, bhAfterShield, bhDeath,
    bhAttackStinger As CArrFrame
Dim parasiteFly, crossHairFind,
    crossHairTransit, crossHairLock As CArrFrame
Dim sbIntro, sbWalk, sbBubble, sbDeath As CArrFrame
Dim bubbleAttack As CArrFrame
Dim ListChar As New List(Of CCharacter)
Dim BH As CCharBlastHornet
Dim PP As CCharParasiteProjectile
Dim SB As CCharSpongeBob
Dim BB As CCharBubbleProjectile
Dim CH As CrossHairProjectile

Dim tr As Double = 2 * Math.PI / 180
Dim V As Double = 5
```

`Dim bmp As Bitmap`

- ➔ Used for creating a canvas that build-in system that commonly works with images defined by pixel.

```
bmp = New Bitmap(img.Width, img.Height)
```

`Dim bg, bg1, img As CImage`

- ➔ Representation of objects from CImage classes due to the background image of application.

```
bg = New CImage
bg.OpenImage("./24bit/bg_m.bmp")
bg.CopyImg(img)
bg.CopyImg(bg1)
```

`Dim SpriteMap As CImage`

- ➔ Representation of objects from CImage classes regard to the collection of sprite animation with color based.

```
SpriteMap = New CImage
SpriteMap.OpenImage("./24bit/spt.bmp")
SpriteMap.CreateMask(SpriteMask)
```

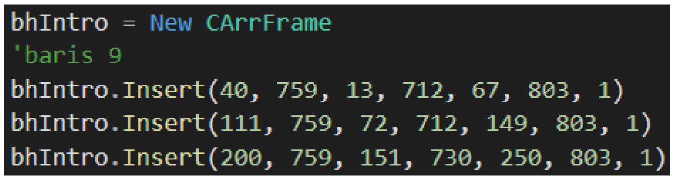
`Dim SpriteMask As CImage`

- ➔ Representation of objects from CImage classes used for the collection of sprite animation with mask (black) based.

```
SpriteMask.Elmt(EF.Left + i, EF.Top + j)
```

```
Dim bhIntro, bhIdle, bhHit, bhHitWall,
     bhintroBomb, bhBomb, bhAfterBomb,
     bhShield, bhAfterShield, bhDeath,
     bhAttackStinger As CArrFrame
```

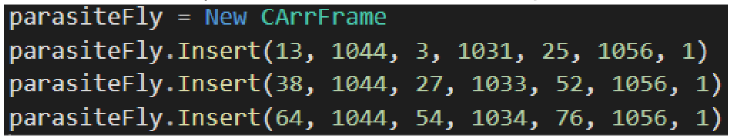
➔ Used for storing all elements of the sprite animation for each state especially Blast Hornet.



```
bhIntro = New CArrFrame
'baris 9
bhIntro.Insert(40, 759, 13, 712, 67, 803, 1)
bhIntro.Insert(111, 759, 72, 712, 149, 803, 1)
bhIntro.Insert(200, 759, 151, 730, 250, 803, 1)
```

```
Dim parasiteFly, crossHairFind,
     crossHairTransit, crossHairLock As CArrFrame
```

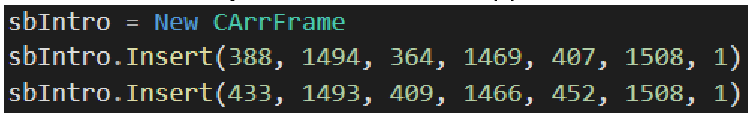
➔ Used for storing all elements of the sprite animation for object projectiles (parasite and crosshair) from Blast Hornet while parasitic bomb and crosshair attack.



```
parasiteFly = New CArrFrame
parasiteFly.Insert(13, 1044, 3, 1031, 25, 1056, 1)
parasiteFly.Insert(38, 1044, 27, 1033, 52, 1056, 1)
parasiteFly.Insert(64, 1044, 54, 1034, 76, 1056, 1)
```

```
Dim sbIntro, sbWalk, sbBubble, sbDeath As CArrFrame
```

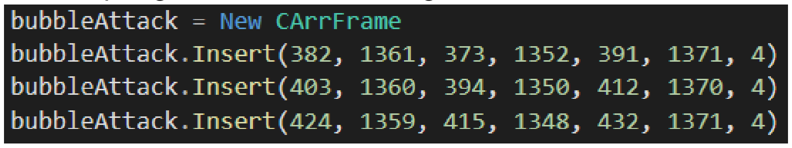
➔ Used for storing all elements of the sprite animation for each state especially SpongeBob as a dummy character on this application.



```
sbIntro = New CArrFrame
sbIntro.Insert(388, 1494, 364, 1469, 407, 1508, 1)
sbIntro.Insert(433, 1493, 409, 1466, 452, 1508, 1)
```

```
Dim bubbleAttack As CArrFrame
```

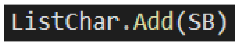
➔ Used for storing all elements of the sprite animation for object projectiles (bubble) from SpongeBob while attacking.



```
bubbleAttack = New CArrFrame
bubbleAttack.Insert(382, 1361, 373, 1352, 391, 1371, 4)
bubbleAttack.Insert(403, 1360, 394, 1350, 412, 1370, 4)
bubbleAttack.Insert(424, 1359, 415, 1348, 432, 1371, 4)
```

```
Dim ListChar As New List(Of CCharacter)
```

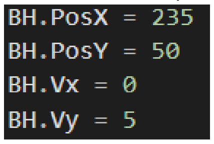
➔ Used for storing all collections of characters that perform in our application. In this case,two characters are going to be used (Blast Hornet and SpongeBob).



```
ListChar.Add(SB)
```

```
Dim BH As CCharBlastHornet
```

➔ Representation of objects from each character that have their own speciality for doing action (state) and controlling the character.



```
BH.PosX = 235
BH.PosY = 50
BH.Vx = 0
BH.Vy = 5
```

`Dim PP As CCharParasiteProjectile`

➔ Representation of objects from each character projectile that have their own speciality for doing and controlling the object itself in Blast Hornet character.

```
PP.PosX = BH.PosX - 24
PP.Vinitx = -V0X
PP.FDir = FaceDir.Right
```

`Dim SB As CCharSpongeBob`

➔ Representation of objects from each character that have their own speciality for doing action (state) and controlling the character.

```
SB.PosY = 50
SB.Vx = 0
SB.Vy = 5
```

`Dim BB As CCharBubbleProjectile`

➔ Representation of objects from each character projectile that have their own speciality for doing and controlling the object itself in SpongeBob character.

```
BB = New CCharBubbleProjectile
```

`Dim CH As CrossHairProjectile`

➔ Representation of objects from each character projectile that have their own speciality for doing and controlling the object itself in Blast Hornet character.

```
CH = New CrossHairProjectile
```

`Dim tr As Double = 2 * Math.PI / 180`

➔ Variable to calculate the turn rate of homing motion of CrossHair projectile from Blast Hornet character.

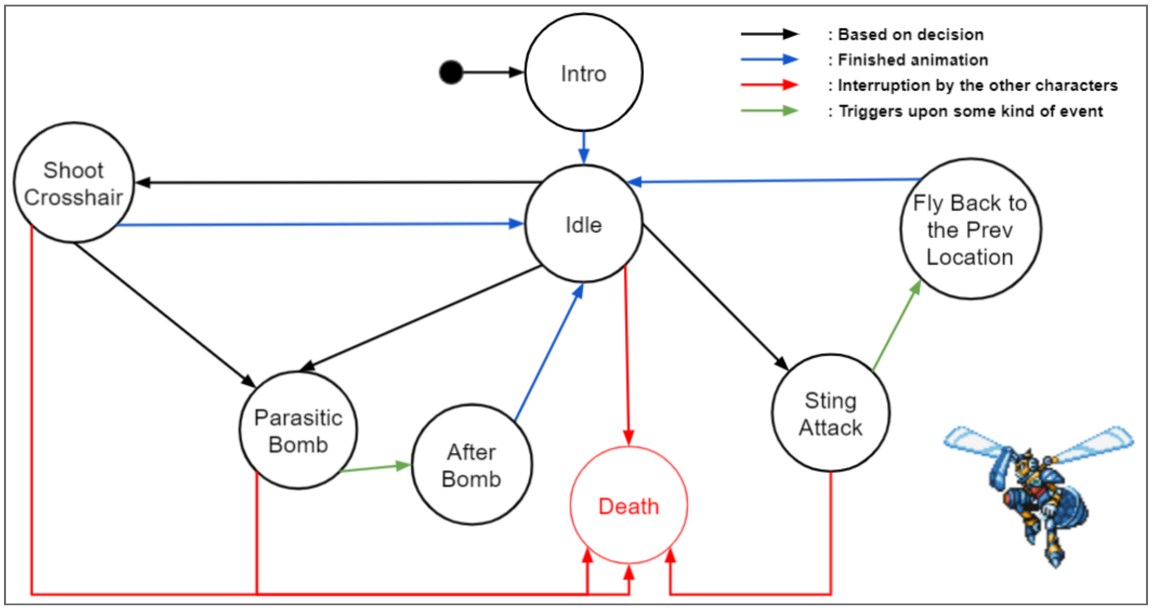
```
parasite.dir = parasite.dir + 30 * tr
```

`Dim V As Double = 5`

➔ Variable for containing the value of velocity of an CrossHair projectile from Blast Hornet character.

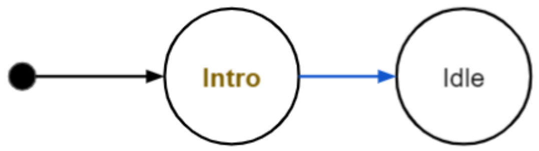
```
vx = V * Math.Cos(parasite.dir)
```

State Diagram and Transition Table of Blast Hornet



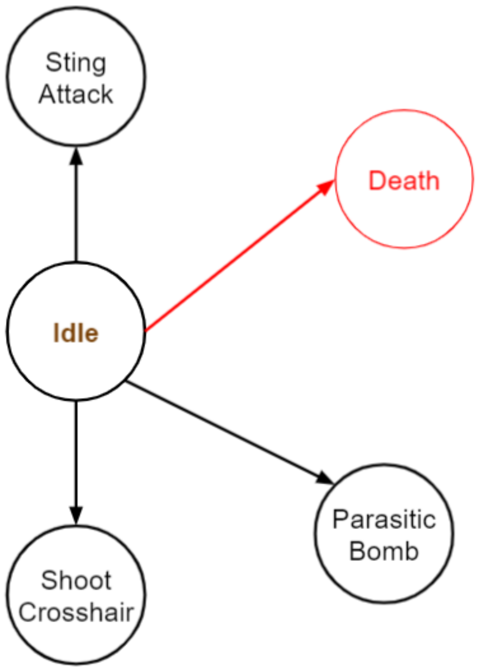
Based on the image aforementioned, This is the description of each state diagram connected into each other by transition table:

1. **Intro (Starting State) - Idle**



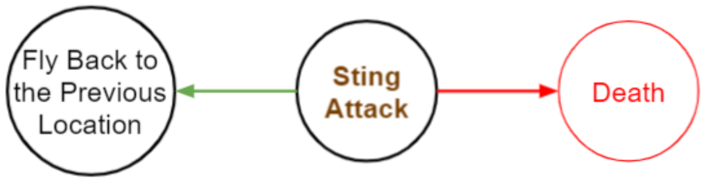
Previous State	New State	Trigger
Intro	Idle	Intro animation is finished

2. **Idle - Sting Attack/Parasitic Bomb/Shoot Crosshair/Death**



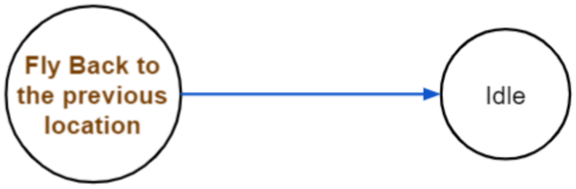
Previous State	New State	Trigger
Idle	Sting Attack	Blast Hornet decides to hit with stinger
Idle	Parasitic Bomb	Blast Hornet shoots parasitic bomb
Idle	Shoot Crosshair	Blast Hornet attempts to aim and lock the dummy
Idle	Death	Blast Hornet gets hit and HP = 0

3. **Sting Attack - Fly Back to The Prev Location/Death**



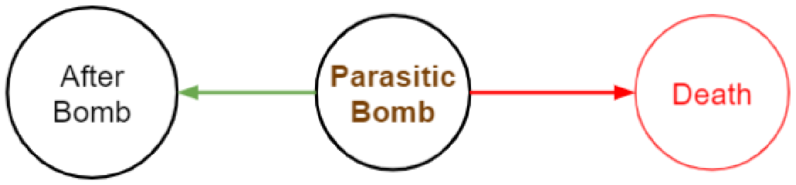
Previous State	New State	Trigger
Sting Attack	Fly Back to the Previous Location	Blast Hornet flies back to the previous location
Sting Attack	Death	Blast Hornet gets hit and HP = 0

4. **Fly Back to The Prev Location - Idle**



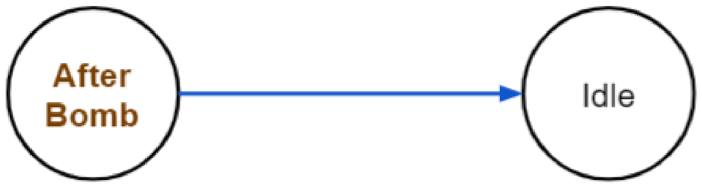
Previous State	New State	Trigger
Hit The Wall/Ground	Idle	Hit The Wall/Ground animation is finished

5. **Parasitic Bomb - After Bomb/Death**



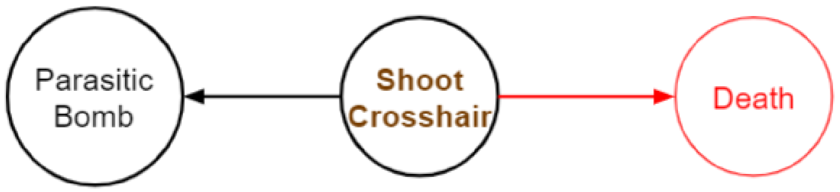
Previous State	New State	Trigger
Parasitic Bomb	After Bomb	Blast Hornet released parasite
Parasitic Bomb	Death	Blast Hornet gets hit and HP = 0

6. **After Bomb - Idle**



Previous State	New State	Trigger
After Bomb	Idle	After Bomb animation is finished

7. **Shoot Crosshair - Parasitic Bomb/Idle**



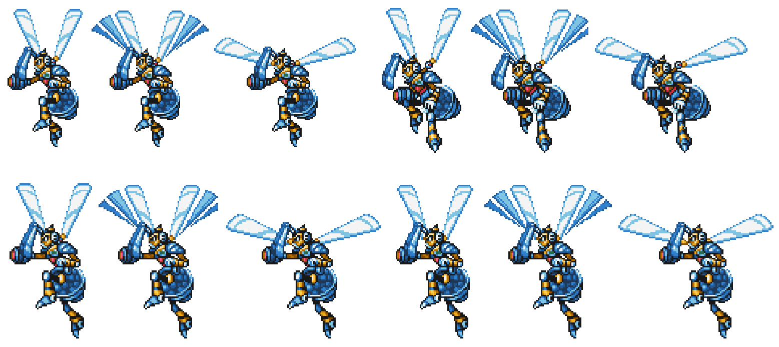
Previous State	New State	Trigger
Shoot Crosshair	Parasitic Bomb	Blast Hornet aims to dummy and attempt to lock it
Shoot Crosshair	Death	Blast Hornet gets hit and HP = 0

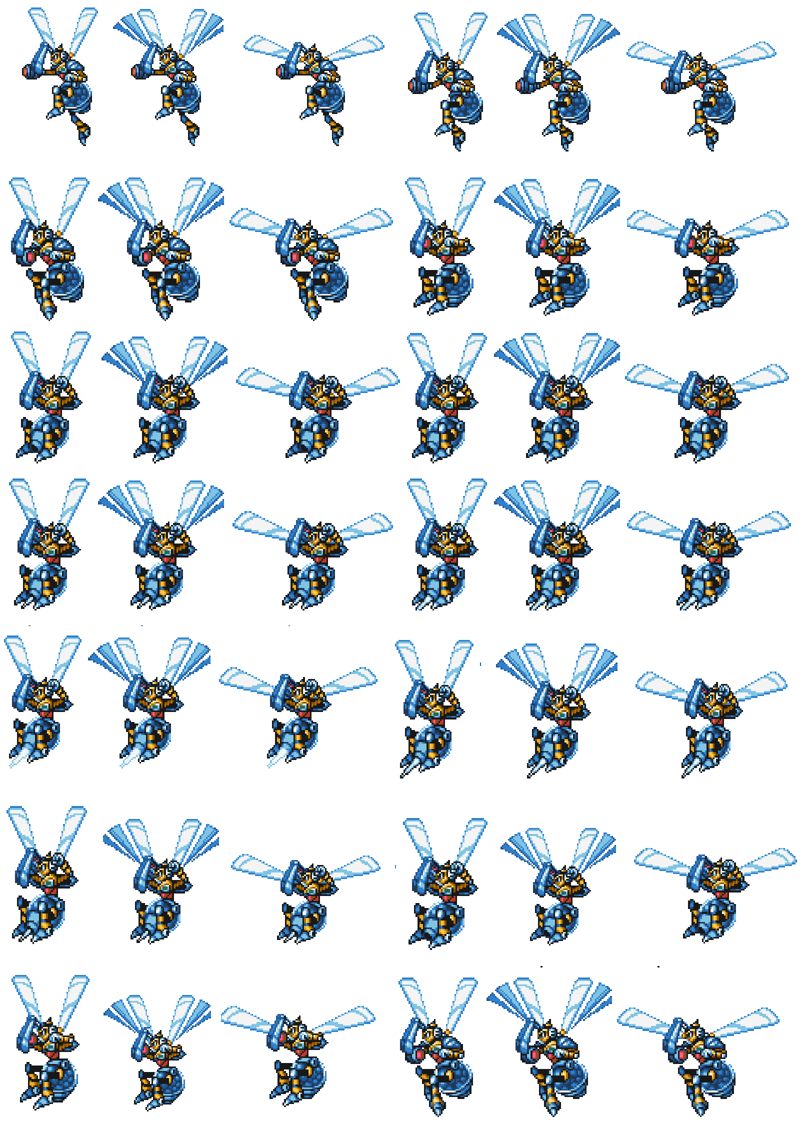
The Sprite Animation

Basically, the Blast hornet has three different types of wings for each element of movement. Also, it will make the character look flying in the air. For this following explanation, the sprite animation of Blast Hornet:

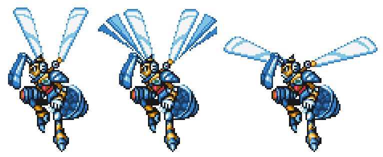
1. **Intro Animation**

   This animation is going to show at the beginning of the application run, this shows almost all possible movements that Blast Hornet have. After, the animation finished will go back into the idle position. In total, this animation has a duration of 1 tick in which repetition of `update()` and 54 frames (18 movements).

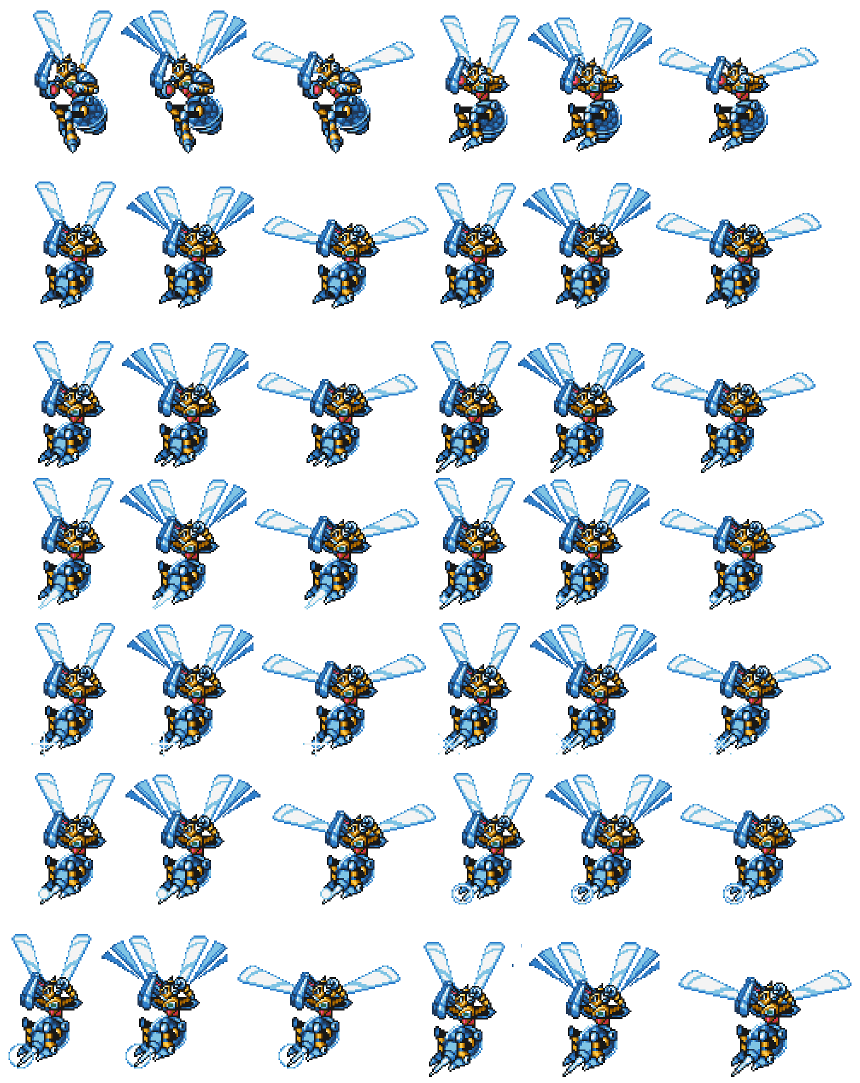




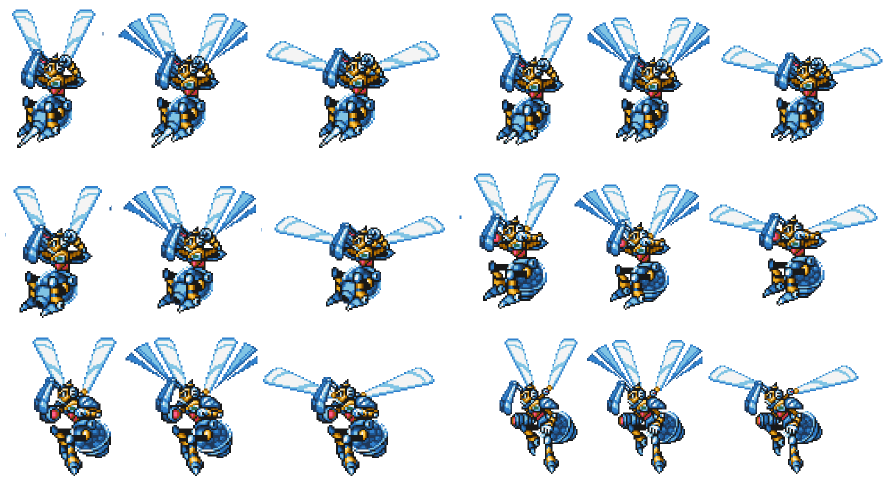
2. **Idle Position**
   This is a general state for Blast Hornet while not doing anything in action, so basically will have 3 frames, one and only one position of the Blast Hornet at steady position and move around without being triggered.



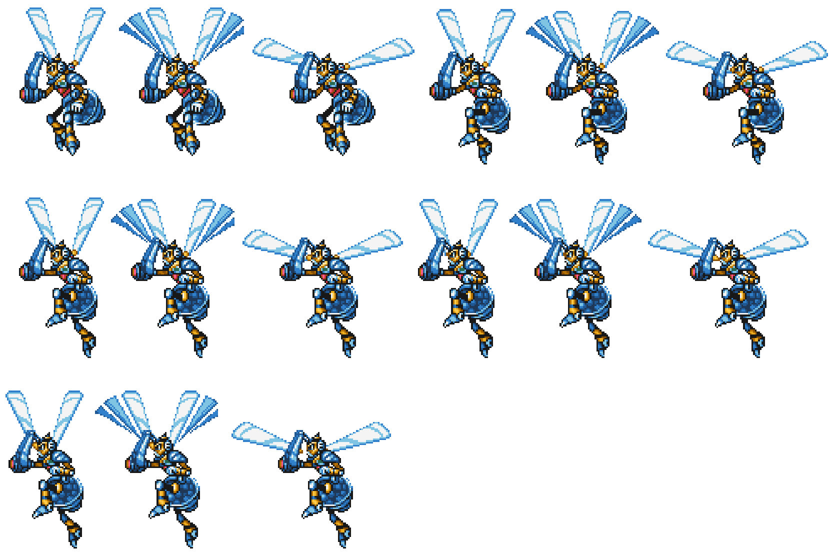
3. **Sting Attack**
   This animation will be triggered by keyboard events which show the stinger and go down (attack) to the dummy character that has been placed. In total 42 frames with 14 different forms.



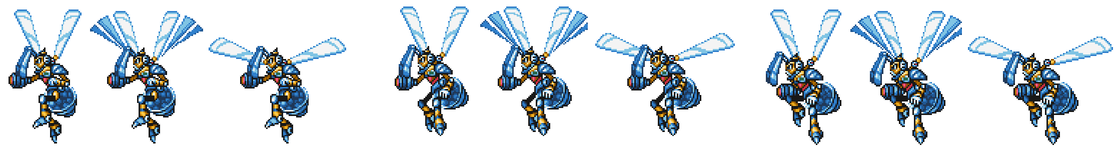
4. **Fly Back to The Previous Location**
   This will be implemented after the Blast Hornet attacks a dummy character. In total, 18 frames with 6 different forms.



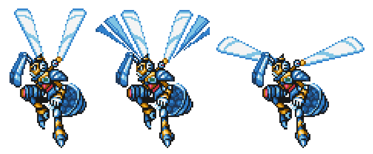
5. **Parasitic Bomb**
   This animation will be triggered by mouse events due to pre-attack which show the hand and spread paradise projectile to the dummy character that has been placed. In total 15 frames with 5 different forms.



6. **After Bomb**
   This will be implemented after the Blast Hornet attacks with parasite to a dummy character. In total, 18 frames with 6 different forms.

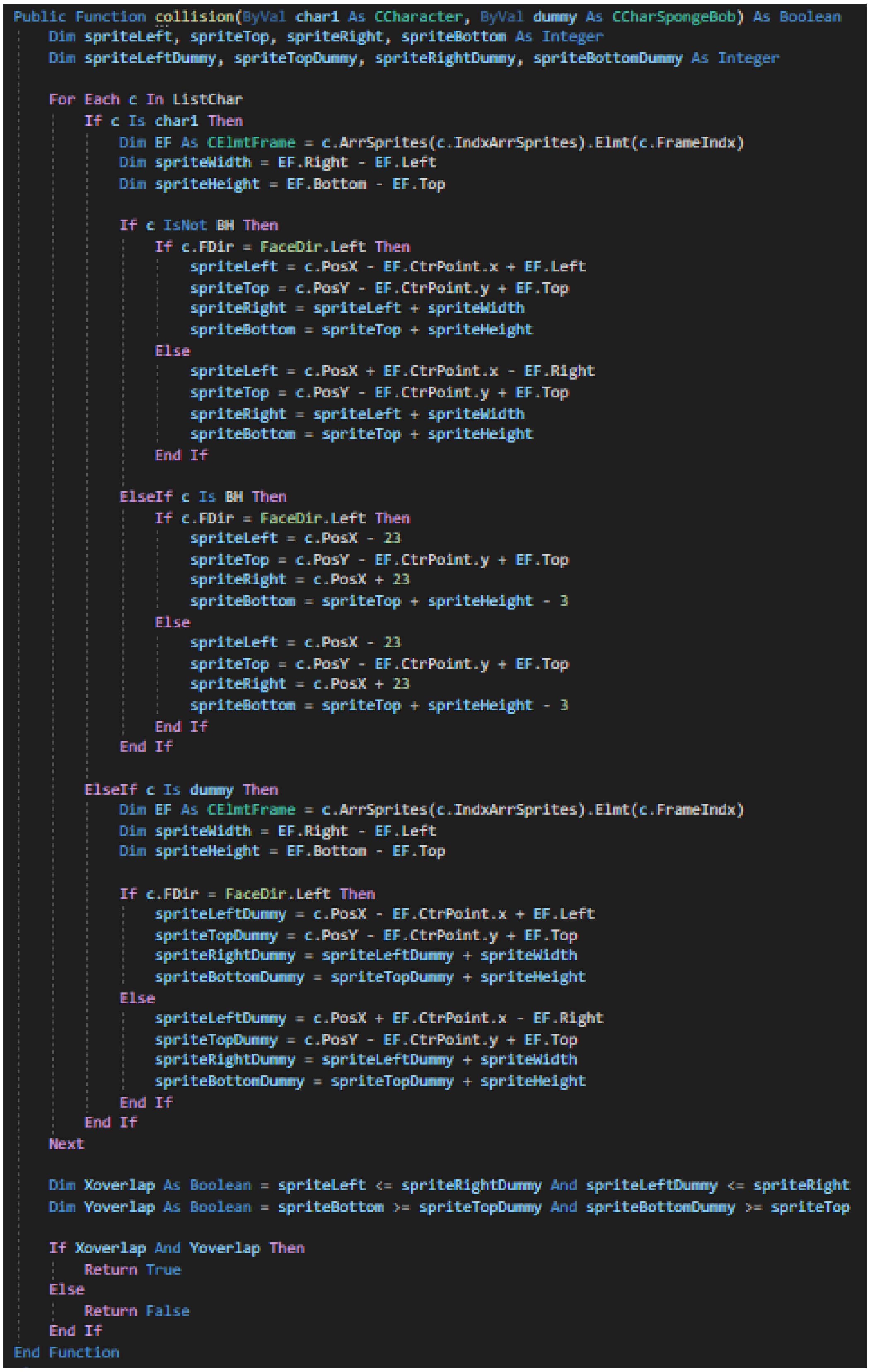


7. **Shoot Crosshair**
   This animation will be triggered by keyboard events by ejecting the crosshair and attaching to a dummy character. All action in this state takes place into idle position without changing form.



8. **Death**
   This state will be triggered by the HP = 0, in this application still not provided the HP system for the Blast Hornet.

Collision Detection



```
Public Function collision(ByVal char1 As CCharacter, ByVal dummy As CCharSpongeBob) As Boolean
    Dim spriteLeft, spriteTop, spriteRight, spriteBottom As Integer
    Dim spriteLeftDummy, spriteTopDummy, spriteRightDummy, spriteBottomDummy As Integer

    For Each c In ListChar
        If c Is char1 Then
            Dim EF As CElmtFrame = c.ArrSprites(c.IndxArrSprites).Elmt(c.FrameIndx)
            Dim spriteWidth = EF.Right - EF.Left
            Dim spriteHeight = EF.Bottom - EF.Top

            If c IsNot BH Then
                If c.FDir = FaceDir.Left Then
                    spriteLeft = c.PosX - EF.CtrPoint.x + EF.Left
                    spriteTop = c.PosY - EF.CtrPoint.y + EF.Top
                    spriteRight = spriteLeft + spriteWidth
                    spriteBottom = spriteTop + spriteHeight
                Else
                    spriteLeft = c.PosX + EF.CtrPoint.x - EF.Right
                    spriteTop = c.PosY - EF.CtrPoint.y + EF.Top
                    spriteRight = spriteLeft + spriteWidth
                    spriteBottom = spriteTop + spriteHeight
                End If

            ElseIf c Is BH Then
                If c.FDir = FaceDir.Left Then
                    spriteLeft = c.PosX - 23
                    spriteTop = c.PosY - EF.CtrPoint.y + EF.Top
                    spriteRight = c.PosX + 23
                    spriteBottom = spriteTop + spriteHeight - 3
                Else
                    spriteLeft = c.PosX - 23
                    spriteTop = c.PosY - EF.CtrPoint.y + EF.Top
                    spriteRight = c.PosX + 23
                    spriteBottom = spriteTop + spriteHeight - 3
                End If
            End If

        ElseIf c Is dummy Then
            Dim EF As CElmtFrame = c.ArrSprites(c.IndxArrSprites).Elmt(c.FrameIndx)
            Dim spriteWidth = EF.Right - EF.Left
            Dim spriteHeight = EF.Bottom - EF.Top

            If c.FDir = FaceDir.Left Then
                spriteLeftDummy = c.PosX - EF.CtrPoint.x + EF.Left
                spriteTopDummy = c.PosY - EF.CtrPoint.y + EF.Top
                spriteRightDummy = spriteLeftDummy + spriteWidth
                spriteBottomDummy = spriteTopDummy + spriteHeight
            Else
                spriteLeftDummy = c.PosX + EF.CtrPoint.x - EF.Right
                spriteTopDummy = c.PosY - EF.CtrPoint.y + EF.Top
                spriteRightDummy = spriteLeftDummy + spriteWidth
                spriteBottomDummy = spriteTopDummy + spriteHeight
            End If
        End If
    Next

    Dim Xoverlap As Boolean = spriteLeft <= spriteRightDummy And spriteLeftDummy <= spriteRight
    Dim Yoverlap As Boolean = spriteBottom >= spriteTopDummy And spriteBottomDummy >= spriteTop

    If Xoverlap And Yoverlap Then
        Return True
    Else
        Return False
    End If
End Function
```

As the code shown above, the collision detection method is using the rectangle collision method. By passing two parameters, attacker and target, we will obtain every location of most top, most left, most right, and most bottom of the sprite. There is special treatment for Blast Hornet, due to the long wing size, the most left and most right of the sprite is obtained by getting the most left and most right of the body of Blast Hornet only. After every data obtained, it will check Xoverlap and Yoverlap.

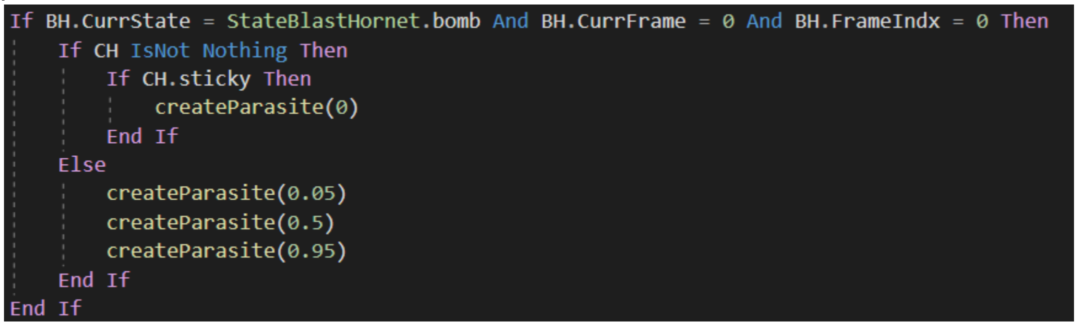
Xoverlap will check if the sprite left of the attacker has to be less than or equal to the sprite right of the target and the sprite right of the attacker has to be greater than or equal to the sprite left of the target. While in Yoverlap, it will check if the sprite bottom of the attacker has to be greater than or equal to the sprite top of the target and the sprite top of the attacker has to be less than or equal to the sprite bottom of the target. In Yoverlap, we are checking if it is greater than or equal to because the VB.Net programming language is giving Y-Axis positive numbers down. After both Xoverlap and Yoverlap obtain, if both resulting True, then it is true that both objects collide.

Pseudocode for each Time Tick

```
Update picturebox
Update all character inside List of Characters
If BH current state is bomb state And BH current frame and frame
index is 0, Then
→ If crosshair exist, Then
      → If crosshair is sticky to the dummy, Then
            → create a parasite
      → Else
            → Create three parasites with different $V_{0x}$
      → End if
→ Else if crosshair is not exist, Then
      → Create three parasites with different $V_{0x}$
→ End if
End if

If dummy X position is less than BH And BH current state is idle
state
→ BH facing direction to the left
Else
→ BH facing direction to the right
End if

If crosshair exist, Then
→ Call a procedure to calculate crosshair location
→ If crosshair and dummy collide, Then
      → Set crosshair sticky to True
      → Start timer for crosshair
→ End if
→ If crosshair is sticky, Then
     → Set crosshair X position equal to dummy X position
     → Set crosshair Y position equal to dummy Y position
     → If parasite exist, Then
          → Call a procedure to calculate parasite location
     → End if
→ End if
End if

If dummy current state is shoot, Then
→ Call a procedure to create projectile
End if

If BH and dummy are collide And dummy current state is not death,
Then
→ Set dummy state to death
→ Start timer for dummy
→ If crosshair exist, Then
     → Call a procedure to reset crosshair
→ End if
Else if parasite exist And parasite and dummy are collide And dummy
current state is not death, Then
→ Set dummy state to death
→ Start timer for dummy
→ Set parasite destroy to True
→ If crosshair exist, Then
     → Call a procedure to reset crosshair
→ End if
End if

Update all character inside List of Characters that not destroy yet
Call a procedure to display image
```

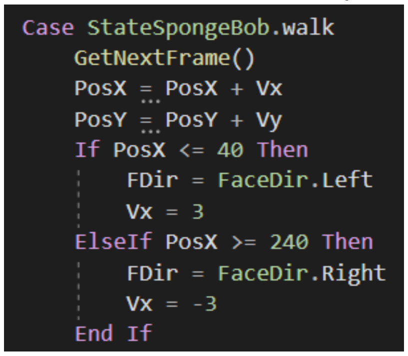
Bonuses Implemented

1. **Blast Hornet can shoot multiple parasitic bombs simultaneously in different directions.**
   In this implementation of multiple shoots, there is a procedure to create parasite projectiles that need some parameter to define and difference the velocity of each parasite.



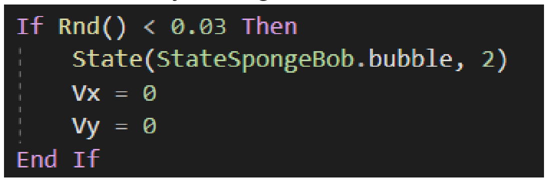
```
If BH.CurrState = StateBlastHornet.bomb And BH.CurrFrame = 0 And BH.FrameIndx = 0 Then
    If CH IsNot Nothing Then
        If CH.sticky Then
            createParasite(0)
        End If
    Else
        createParasite(0.05)
        createParasite(0.5)
        createParasite(0.95)
    End If
End If
```

2. **The dummy is able to move around. You decide how the dummy moves around.**
   Dummy can move around that is determined by behaviour of SpongeBob character class. The velocity of the X-axis is set into 3 to get characters move to left or even right with limitation of minimum position 40 and maximum 240.



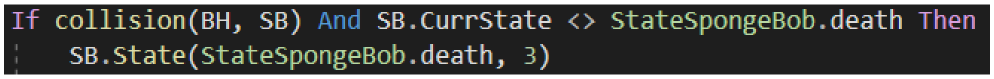
```
Case StateSpongeBob.walk
    GetNextFrame()
    PosX = PosX + Vx
    PosY = PosY + Vy
    If PosX <= 40 Then
        FDir = FaceDir.Left
        Vx = 3
    ElseIf PosX >= 240 Then
        FDir = FaceDir.Right
        Vx = -3
    End If
```

3. **The dummy can shoot projectiles. You decide how the projectiles move.**
   SpongeBob character is able to projectile with bubbles, the implementation automatically determined by configuration of state on behaviour of classes update().



```
If Rnd() < 0.03 Then
    State(StateSpongeBob.bubble, 2)
    Vx = 0
    Vy = 0
End If
```

4. **Display an animation of the dummy getting destroyed.**
   The dummy can be destroyed by attacking from another character that can be detected by collision detection with sting attack, parasite projectile, or any collision between SpongeBob and Blast Hornet.

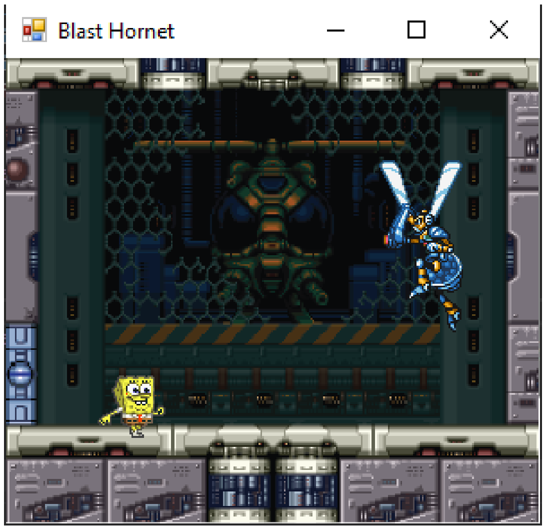


```
If collision(BH, SB) And SB.CurrState <> StateSpongeBob.death Then
    SB.State(StateSpongeBob.death, 3)
```

V. Evaluation

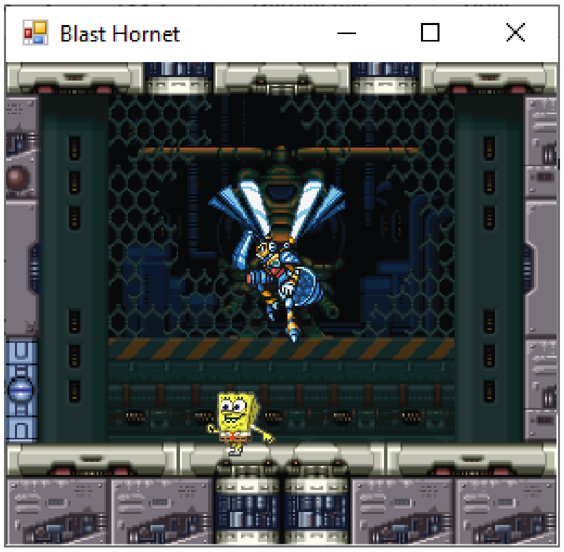
Test Case 1: Starting the program (the intro animation).

As the figure attached below, shows that this application successfully shows the intro animation.



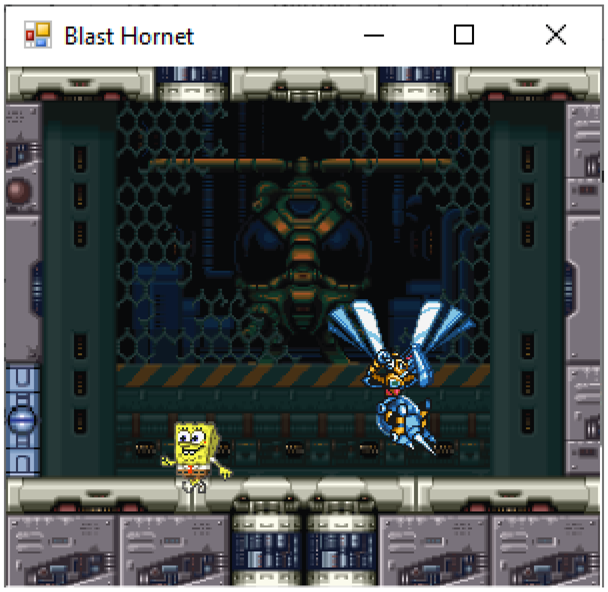
Test Case 2: Blast Hornet flying.

As the figure attached below, shows that this application successfully shows the Blast Hornet flying animation.



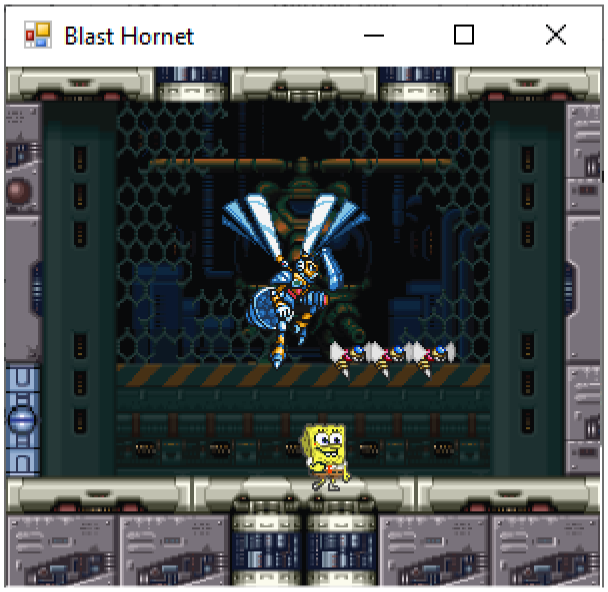
Test Case 3: Blast Hornet doing a sting attack.

As the figure attached below, shows that this application successfully shows the sting attack animation.



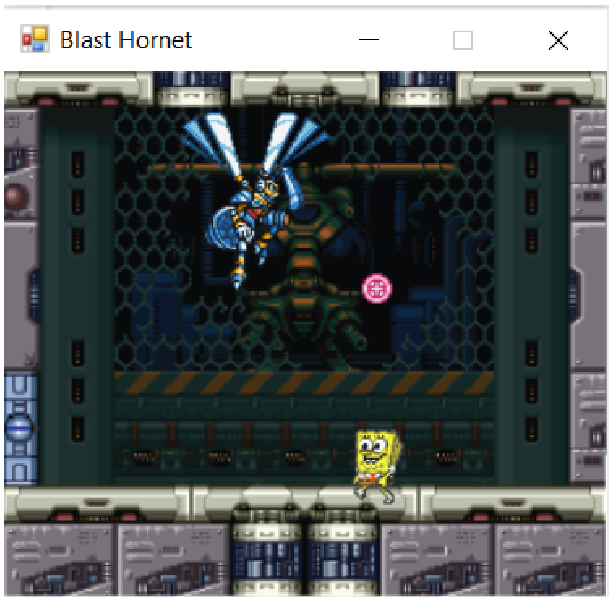
Test Case 4: Blast Hornet shooting multiple parasitic bombs simultaneously in different directions.

As the figure attached below, shows that this application successfully shows the multiple parasitic bombs animation.



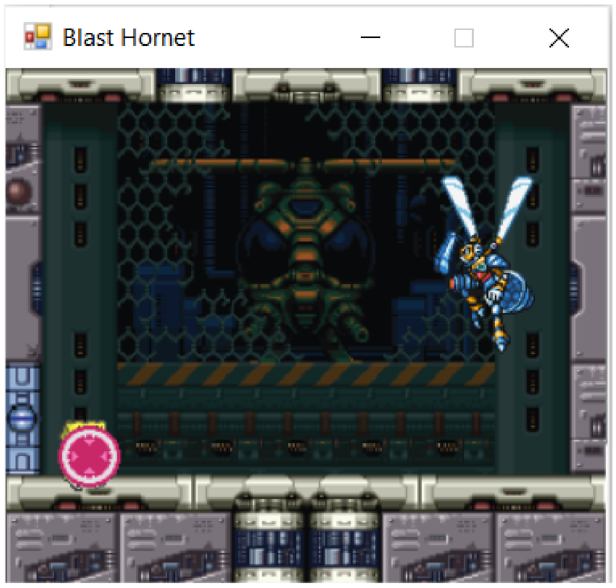
Test Case 5: Blast Hornet shooting a crosshair.

As the figure attached below, shows that this application successfully shows the shooting crosshair animation.



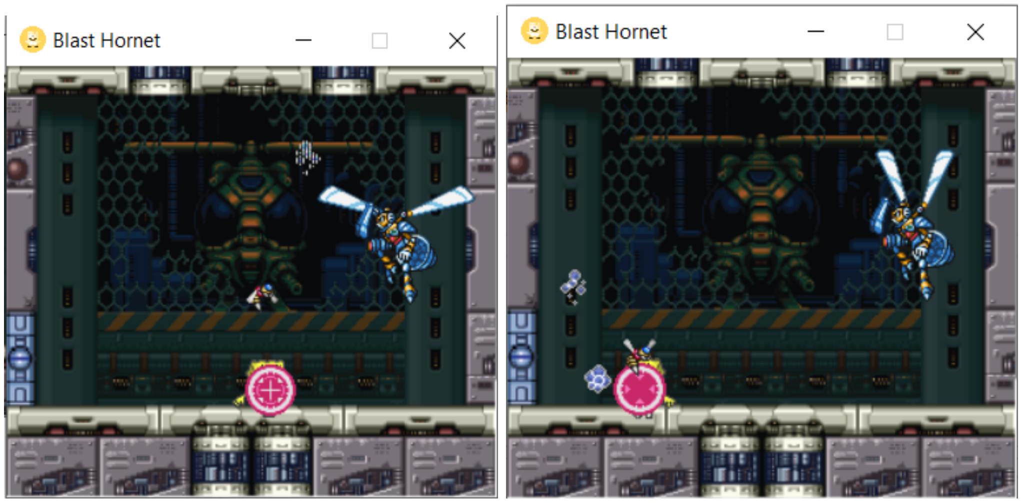
Test Case 6: The crosshair homing in on the Megaman dummy.

As the figure attached below, shows that this application successfully shows the crosshair homing on the dummy animation.



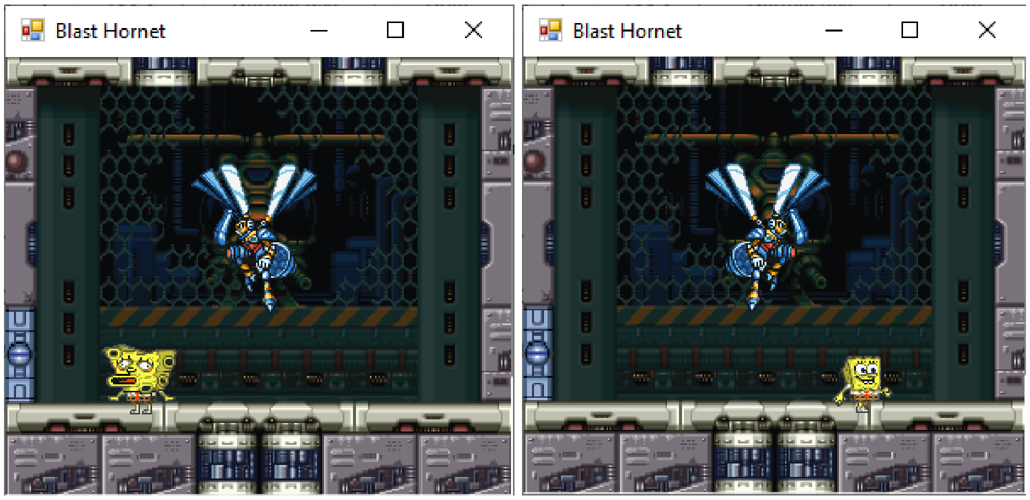
Test Case 7: Parasitic bombs homing in on the Megaman dummy whenever a large crosshair is present.

As the figure attached below, shows that this application successfully shows the parasitic bombs homing when the crosshair is sticky animation.



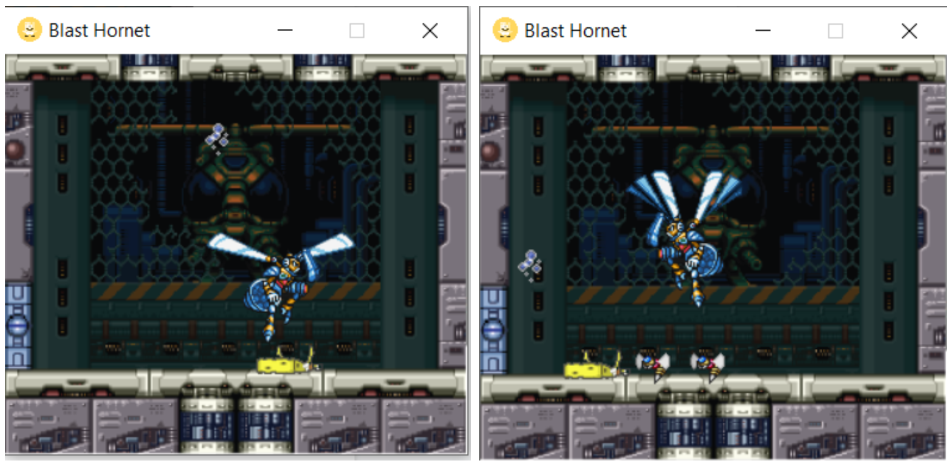
Test Case 8: Blast Hornet changing facing direction.

As the figure attached below, shows that this application successfully shows the Blast Hornet changing direction whenever the dummy is behind animation.



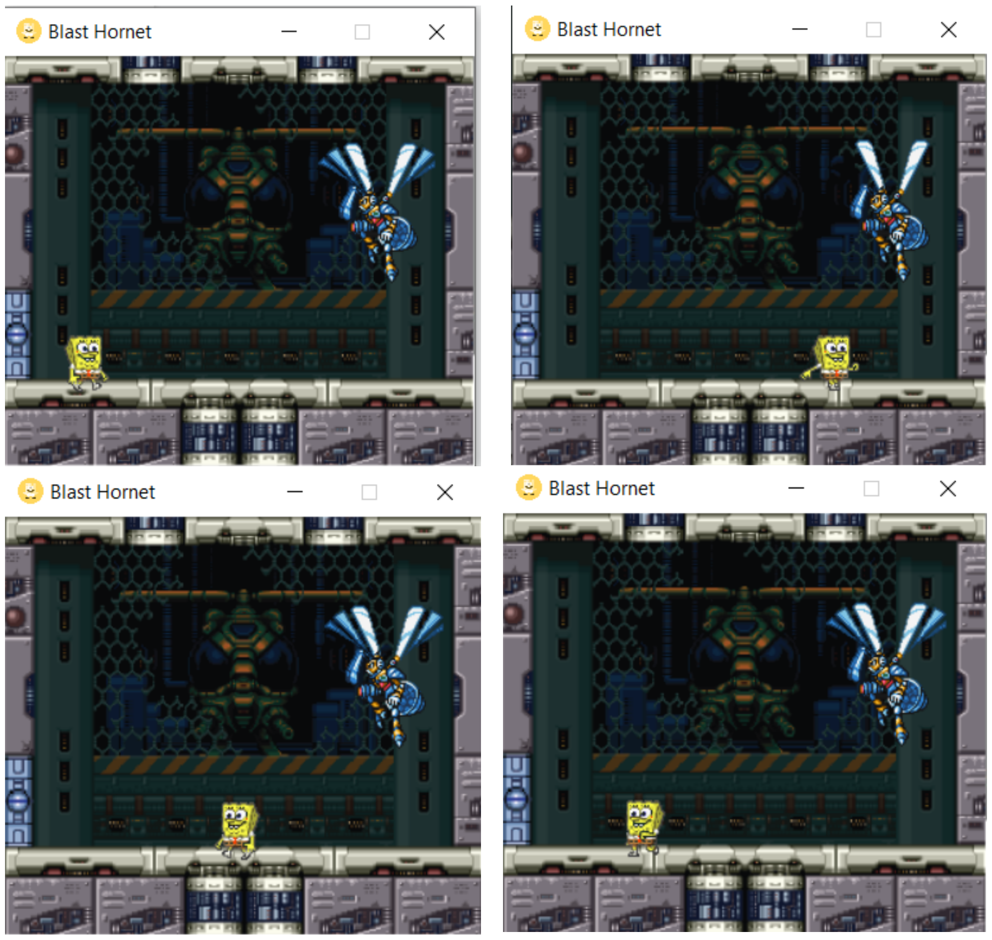
Test Case 9: The dummy getting hit and destroyed by Blast Hornet's attacks.

As the figure attached below, shows that this application successfully shows the dummy getting hit and destroyed animation.

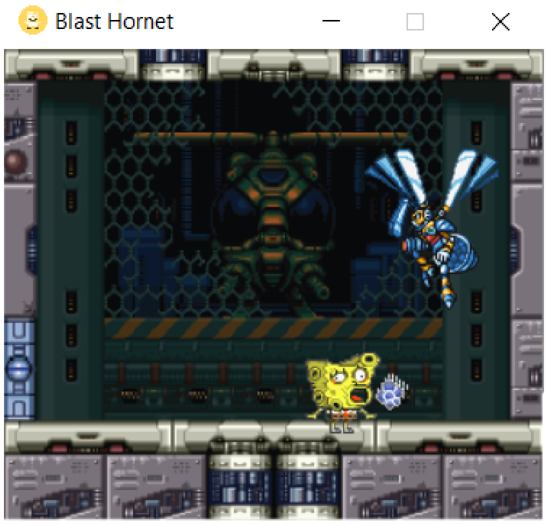


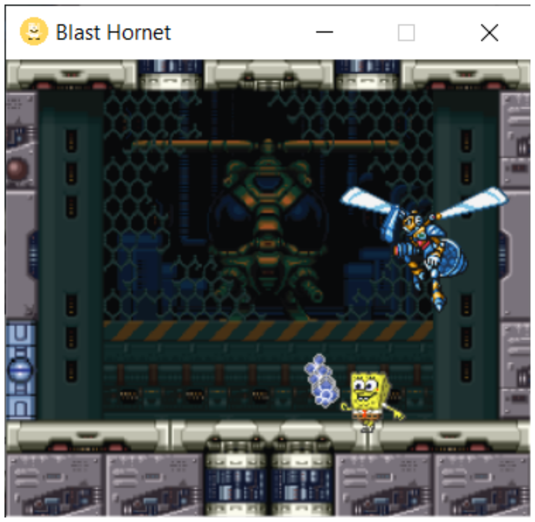
Bonus Test Case:

- The dummy is able to move around.



- The dummy can shoot projectiles.





VI. Work Log

Date	Activity / Progress	Personnel Involved
Friday, April 23rd, 2021	1. Briefing 2. Creating Base: Background and Sprite 3. Creating Sprite Sheet	Natasya, Renaldo F., Samuel
Saturday, April 24th, 2021	1. Insert Sprite Sheet for Each State 2. Creating State 3. Creating Intro Animation	Natasya, Renaldo F., Samuel
Monday, April 26th, 2021	1. Improve Intro Animation 2. Briefing PA Specifications 3. Developing User Input Fly Position Using Mouse Click 4. Creating Report Base	Natasya, Renaldo F., Samuel
Tuesday, April 27th, 2021	1. Improve User Input to Fly 2. Creating Dummy Sprite Sheet 3. Developing Sting Attack 4. Arranging Report: Introduction and Basic Theory	Natasya, Renaldo F., Samuel
Wednesday, April 28th, 2021	1. Improve Sting Attack 2. Developing Parasitic Bomb Using Mouse Control 3. Bug Fixing	Natasya, Renaldo F., Samuel
Friday, April 30th, 2021	1. Insert Multiple Parasite Projectile 2. Improve Parasitic Bomb 3. Insert Dummy Character : Spongebob	Natasya, Renaldo F., Samuel
Saturday, May 1st, 2021	1. Improve Blast Hornet Face Direction Towards Dummy Face Direction 2. Developing Collision Detection 3. Developing Crosshair Shoot	Natasya, Renaldo F., Samuel
Thursday, May 6th, 2021	1. Implementing Collision Detection 2. Develop and Implementing Respawn for Dummy Character 3. Dummy Shoot Projectile in Random Positions 4. Arranging Report	Natasya, Renaldo F., Samuel
Monday, May 10th, 2021	1. Implementing Crosshair Using Homing Algorithm 2. Develop and Implementing Parasite Bomb using Homing Algorithm 3. Bug Fixing 4. Arranging Report	Natasya, Renaldo F., Samuel

Tuesday, May 11th, 2021	1. Bug Fixing 2. Review All Specifications 3. Finalize Application 4. Arranging Report	Natasya, Renaldo F., Samuel
Wednesday, May 12th, 2021	1. Finalizing Report	Natasya, Renaldo F., Samuel

- Based on the requirements given for Programming Assignment 15 - Blast Hornet - Sprite Animation, below is the details of whether the requirement is fully implemented, partially implemented, or not implemented:
    1. Provide a background is **fully implemented**,
    2. Start with Blast Hornet doing his intro animation is **fully implemented**,
    3. Allow the user to control Blast Hornet. Blast Hornet can do the following actions:
        a. Fly.
            i. The user clicks at a certain location and Blast Hornet flies to that location is **fully implemented**,
            ii. Blast Hornet flies in a figure-eight path is **not implemented** due to lack of idea.
        b. Sting attack and the following requirements are **fully implemented**,
            i. Blast Hornet protrudes his sting and flies towards the location of the Megaman dummy (which will be explained later).
            ii. Blast Hornet then files back to where he was when he started to perform this move.
        c. Shoot parasitic bombs and the following requirements are **fully implemented**,
            i. Blast Hornet summons a small robotic hornet (a.k.a. “parasitic bomb”).
            ii. The user clicks in a certain direction and the parasitic bomb flies in that direction.
            iii. The parasitic bomb stops when reaching the ceiling, the floor, or a wall.
            iv. The parasitic bomb disappears after a few seconds.
            v. Blast Hornet can shoot multiple parasitic bombs simultaneously in different directions.

    d. Shoot crosshair and the following requirements are **fully implemented**,
        i. Blast Hornet shoots a small crosshair that slowly homes in on the Megaman dummy.
        ii. If the crosshair hits the Megaman dummy, the crosshair will become large and stay with the Megaman dummy wherever it goes.
        iii. If the Megaman dummy has a large crosshair following it, any parasitic bombs shot by Blast Hornet will home in on the Megaman dummy instead of simply moving in a straight line.
        iv. The crosshair disappears after 15 seconds.
    e. Change facing direction and the following requirement are **fully implemented**,
        i. Blast Hornet always faces towards the Megaman dummy and will automatically change facing direction whenever the Megaman dummy is behind him.
4. Place a dummy, doll, or puppet of Megaman (or anything else that you like) as a target for Blast Hornet.
    a. The dummy starts at a fixed location on the ground is **fully implemented**,
    b. The dummy is by default passive. It does not perform any action and the following extra requirements are **fully implemented**,
        i. The dummy is able to move around. You decide how the dummy moves around.
        ii. The dummy can shoot projectiles. You decide how the projectiles move.
    c. If the dummy gets hit by Blast Hornet or his parasitic bombs, it will disappear and the following extra requirements, except the last one, are **fully implemented**.
        i. After a few seconds, it will respawn at a random location on the ground.
        ii. The dummy must not collide with Blast Hornet when it respawns.
        iii. Display an animation of the dummy getting destroyed.
        iv. Display an animation of the dummy getting sucked into the gravity well if it reaches the center of the well is **not implemented** because we never seen this before inside the Blast Hornet stage.

5. Providing the proper animations for each action is **fully implemented**.

To sum up, all of the important requirements (red and blue color) are fully implemented. Several bonus requirements (green color) are also fully implemented. There are two bonus requirements that are not implemented at all.

VII. Conclusion and Remarks

The application has been created according to the requirements. It works as our expectation, so with several bonus requirements. We have several shortcomings to finish two bonus requirements. For Blast Hornet flies in figure-eight path, we are unable to figure out how the equation will help to finish this part. Also with the dummy getting sucked into the gravity well, we have never seen before how this part is going on. Overall, this program still works as our expectation. We also try our best to give the best user experience to make the application easy to use.

After around 3 weeks working on this assignment, it is fun to be able creating animation using a programming language. In our mind, creating animation is part of some animation maker program. It is a totally new insight for us. Every step to finish our work is challenging and still giving us more and more insight. We are really happy that we are able to work on this assignment and finish this programming assignment with new experiences that have never been experienced before. We are looking forward to the next step of this implementation.

References

1. “Game Development.” *Game Development | MDN*, developer.mozilla.org/en-US/docs/Games/Techniques/2D_collision_detection.
2. Day, John. “Collision Detection - Circles, Rectangles and Polygons.” *Game Development*, 23 July 2018, www.gamedevelopment.blog/collision-detection-circles-rectangles-and-polygons/.