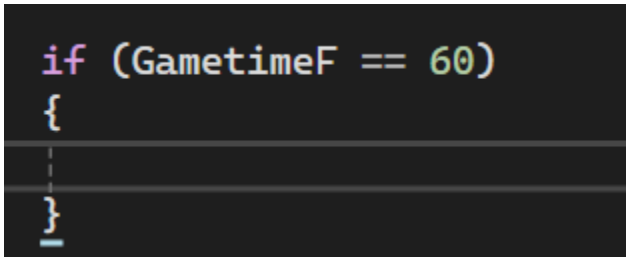
Time based events

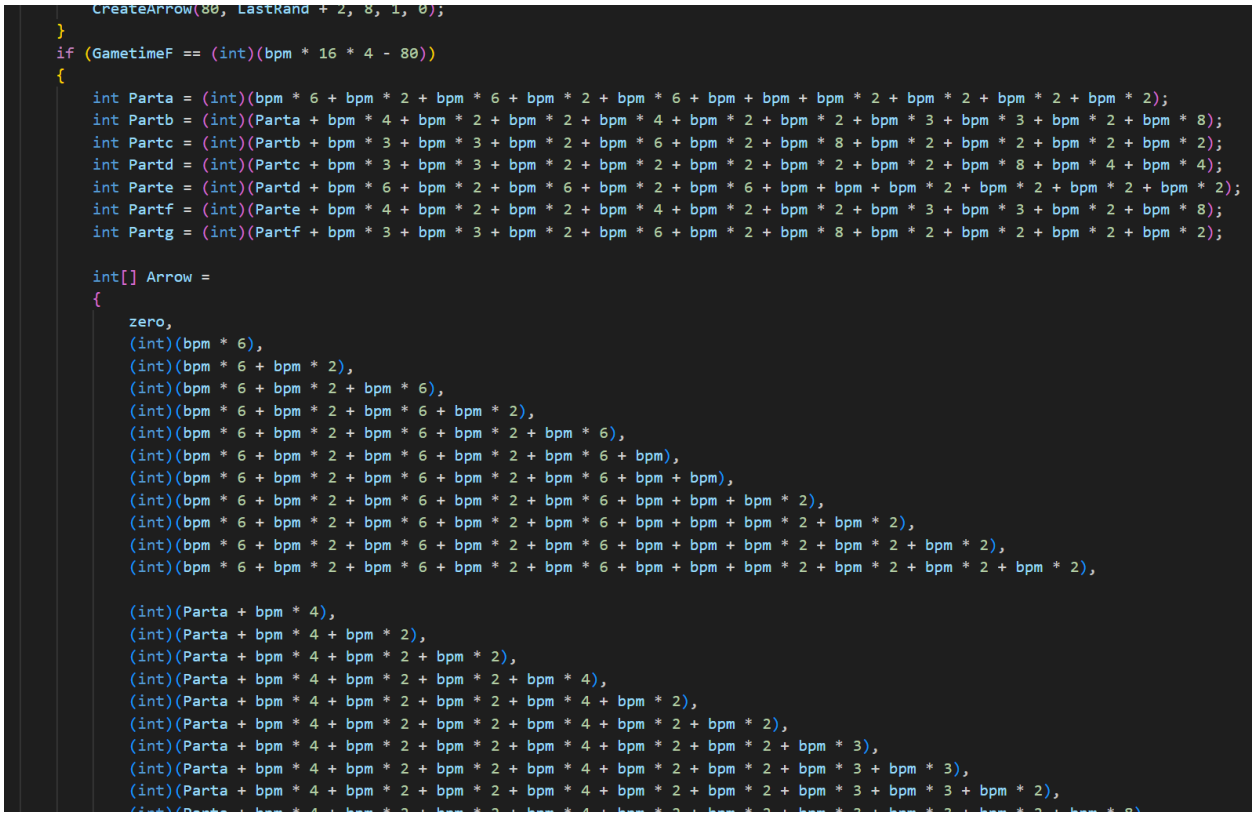
In the previous section, GametimeF (Frames elapsed in float) was introduced and it can be used as the basic form of time evaluation.



```
if (GametimeF == 60)
{

}
```

However the BPM of each chart is different, therefore this is not the best way to evaluate the current beat in the chart.
If you have defined a BPM of the chart, then you can use “SingleBeat” to determine the duration of 1 beat.
However, usage of this variable is strongly discouraged due to users may misuse this and cause the code to be unreadable and chunky, like this



```
    CreateArrow(80, LastRand + 2, 8, 1, 0);
}
if (GametimeF == (int)(bpm * 16 * 4 - 80))
{
    int Parta = (int)(bpm * 6 + bpm * 2 + bpm * 6 + bpm * 2 + bpm * 6 + bpm + bpm + bpm * 2 + bpm * 2 + bpm * 2 + bpm * 2);
    int Partb = (int)(Parta + bpm * 4 + bpm * 2 + bpm * 2 + bpm * 4 + bpm * 2 + bpm * 2 + bpm * 3 + bpm * 3 + bpm * 2 + bpm * 8);
    int Partc = (int)(Partb + bpm * 3 + bpm * 3 + bpm * 2 + bpm * 6 + bpm * 2 + bpm * 8 + bpm * 2 + bpm * 2 + bpm * 2 + bpm * 2);
    int Partd = (int)(Partc + bpm * 3 + bpm * 3 + bpm * 2 + bpm * 2 + bpm * 2 + bpm * 2 + bpm * 2 + bpm * 8 + bpm * 4 + bpm * 4);
    int Parte = (int)(Partd + bpm * 6 + bpm * 2 + bpm * 6 + bpm * 2 + bpm * 6 + bpm + bpm + bpm * 2 + bpm * 2 + bpm * 2 + bpm * 2);
    int Partf = (int)(Parte + bpm * 4 + bpm * 2 + bpm * 2 + bpm * 4 + bpm * 2 + bpm * 2 + bpm * 3 + bpm * 3 + bpm * 2 + bpm * 8);
    int Partg = (int)(Partf + bpm * 3 + bpm * 3 + bpm * 2 + bpm * 6 + bpm * 2 + bpm * 8 + bpm * 2 + bpm * 2 + bpm * 2 + bpm * 2);

    int[] Arrow =
    {
        zero,
        (int)(bpm * 6),
        (int)(bpm * 6 + bpm * 2),
        (int)(bpm * 6 + bpm * 2 + bpm * 6),
        (int)(bpm * 6 + bpm * 2 + bpm * 6 + bpm * 2),
        (int)(bpm * 6 + bpm * 2 + bpm * 6 + bpm * 2 + bpm * 6),
        (int)(bpm * 6 + bpm * 2 + bpm * 6 + bpm * 2 + bpm * 6 + bpm),
        (int)(bpm * 6 + bpm * 2 + bpm * 6 + bpm * 2 + bpm * 6 + bpm + bpm),
        (int)(bpm * 6 + bpm * 2 + bpm * 6 + bpm + bpm + bpm * 2),
        (int)(bpm * 6 + bpm * 2 + bpm * 6 + bpm * 2 + bpm * 6 + bpm + bpm + bpm * 2 + bpm * 2),
        (int)(bpm * 6 + bpm * 2 + bpm * 6 + bpm * 2 + bpm * 6 + bpm + bpm + bpm * 2 + bpm * 2 + bpm * 2),
        (int)(bpm * 6 + bpm * 2 + bpm * 6 + bpm * 2 + bpm * 6 + bpm + bpm + bpm * 2 + bpm * 2 + bpm * 2 + bpm * 2),

        (int)(Parta + bpm * 4),
        (int)(Parta + bpm * 4 + bpm * 2),
        (int)(Parta + bpm * 4 + bpm * 2 + bpm * 2),
        (int)(Parta + bpm * 4 + bpm * 2 + bpm * 2 + bpm * 4),
        (int)(Parta + bpm * 4 + bpm * 2 + bpm * 2 + bpm * 4 + bpm * 2),
        (int)(Parta + bpm * 4 + bpm * 2 + bpm * 2 + bpm * 4 + bpm * 2 + bpm * 2),
        (int)(Parta + bpm * 4 + bpm * 2 + bpm * 2 + bpm * 4 + bpm * 2 + bpm * 2 + bpm * 3),
        (int)(Parta + bpm * 4 + bpm * 2 + bpm * 2 + bpm * 4 + bpm * 2 + bpm * 2 + bpm * 3 + bpm * 3),
        (int)(Parta + bpm * 4 + bpm * 2 + bpm * 2 + bpm * 4 + bpm * 2 + bpm * 2 + bpm * 3 + bpm * 3 + bpm * 2),
```

(Universal Collapse Codes - Tlottgodinf)

It is obvious that this is not the ideal way to determine the current beat elapsed in the chart, the better way to approach this is to use the following functions.

Beat Functions

BeatTime(float x) -> float

Duration of the given beat time in frames
X: Amount of beats
Returns: Amount of frames of the X amount of beats

InBeat(float beat) -> bool

Whether the chart is at the given beat
Beat: The beat to check
Returns: Whether the chart is currently at the given beat

InBeat(float leftbeat, float rightbeat) -> bool

Whether the chart is currently in the range of the given beats
Leftbeat: The starting beat
Rightbeat: The ending beat
Returns: Whether the chart is currently between the given beat

At0thBeat(float beatCount) -> bool

Check whether the chart is currently at a multiple of the given beat
(i.e. At0thBeat(2) will return true when it is at the 0th beat, 2nd beat, 4th beat, 6th beat, etc)
Beatcount: The beat to check
Returns: Whether the chart is at a multiple of the Xth beat

AtKthBeat(float beatCount, float K) -> bool

Check whether the chart is currently at a multiple of the given beat plus the frames given
(i.e. AtKthBeat(2, BeatTime(1)) will return true when it is at the 1st beat, 3rd beat, 5th beat, 7th beat, etc)
Beatcount: The beat to check
K: The frame remainder to check
Returns: Whether the chart is at a multiple of the Xth beat plus the frames given

Delay(float delay, Action action)

Invokes an action after the given frames
Delay: The amount of frames to delay
Action: The action to invoke

DelayBeat(float delayBeat, Action action)

Invokes an action after the given beats
Delay: The amount of beats to delay
Action: The action to invoke

ForBeat([float delayBeat], float durationBeat, Action action)

Invokes an action for the next given beats (Using int calculation, recommended not to use)
(Optional) delayBeat: The amount of beats to delay before invoking the action
durationBeat: The duration of the action
Action: The action to invoke

ForBeat120([float delayBeat], float durationBeat, Action action)

Invokes an action for the next given beats (Using float calculation, recommended to use)
(Optional) delayBeat: The amount of beats to delay before invoking the action
durationBeat: The duration of the action
Action: The action to invoke

Charting functions

After learning the time calculation functions, it is time for basic charting.
There are some functions you can use to create charts in a rather visual way (You still need some imagination)

ArrowAllocate(int slot, int direction)

Allocates a direction for arrows
Slot: The slot to allocate in (Range is [0, 9])
Direction: The direction to allocate

CreateChart(float Delay, float Beat, float arrowspeed, string[] Barrage)

String based chart creator, use an empty string for an empty beat.
Optional Args: "!": No Score, "^": Accelerate, "<": RotateL, ">": RotateR, "*": Tap, "~": Void Sprite, "_": Hold
Order of parsing: ~*_<>^!
Direction Args: "R": Random, "D": Different, "+/-x" Add/Sub x to the last dir. , "$x": Fixed on x direction, "Nx": Not x, "Ax": The xth allocated direction
Optional Color Args: 0-> Blue, 1-> Red, 2-> Green, 3-> Purple
Optional Rotation Args: 0-> None, 1-> Reverse, 2-> Diagonal
GB：#xx#yz, Where "xx" means the duration beat, "y" beats direction, "z" means color, replace '#' by '%' if you don't want arrows

Combinations: "(R)(+0)", NOT "R(+0)"
Misc: use ' to multiply the speed of the arrow, << or >> to adjust the current beat (>>0.5 will skip 0.5 beats)
Use RegisterFunction() or RegisterFunctionOnce() to declare functions to execute them inside here
For example RegisterFunctionOnce("func", ()=> {});
"(func)(R)", will invoke the action in "func" and creates an arrow
"!!X*/Y", the beats will be a 8 * X beat for the next Y beats (If Y is undefined then it will last for the rest of the function)
You can add arguments in the form of "<Arg1,Arg2…>Action"
You may use ‘Arguments’ inside the declared action in RegisterFunction() to access them.

Delay: The delay for the events to be executed, generally used for preventing spawning immediately within view
Beat: Duration of 8 beats, generally used with BeatTime()
Arrowspeed: The speed of the arrows</param>
Barrage: The array of strings that contains the barrage</param>

RegisterFunction(string name, Action action)

Registers a function for CreateChart() to execute
Name: The name of the function
Action: The action to invoke when executed

RegisterFunctionOnce(string name, Action action)

Registers a one time function for CreateChart() to execute (Will be removed from memory after CreateChart() is executed)
Name: The name of the function
Action: The action to invoke when executed

Charting variables

float[] Arguments

Arguments supplied to the function in the strings in CreateChart()

float CurrentTime

The current time calculated in CreateChart()

Arrow LastArrow

The last arrow created from CreateChart()

object[] Temps

Temporary variable slot you can use, has a size of 100

Image explanation

I know there is a lot of text up there, so I'll try to use some images to explain.

```
CreateChart(0, BeatTime(2), 7, new string[]
{
    "$0", "", "$0", "", "$0", "", "$0", "",
    "R", "", "R", "", "R", "", "R", "",
    "R", "", "R", "+01", "", "", "R", "",
    "", "", "R", "+01", "R", "", "R", "",
    "R1", "", "R1", "", "R1", "", "R1", "",
    "R1", "", "R1", "", "R1", "", "R1", "",
    "R1", "", "R1", "+0", "", "", "R1", "",
    "", "", "R1", "+0", "R1", "", "R1", "",
    "$0", "", "$0", "", "$0", "", "$0", "",
    "R", "", "R", "", "R", "", "R", "",
    "R", "", "R", "+01", "", "", "R", "",
    "", "", "R", "+01", "R", "", "R", "",
    "R1", "", "R1", "", "R1", "", "R1", "",
    "R1", "", "R1", "", "R1", "", "R1", "",
    "R1", "", "R1", "", "R1", "", "R1", "",
    "D", "D1", "D", "D1", "D", "D1", "D", "D1",
});
```

This is an example on how to use the basic functions of CreateChart
The first argument, 0, means that the arrows are instantly created.
The second argument, BeatTime(2), means that every 8 string entries, 2 beats will pass.
The third argument, 7, means that the speed of the arrows are 7.
The fourth argument, being string[], is the list of strings that will create the chart.
In the strings:
"$0": A blue arrow will come from the right side.
"R": A blue arrow will come from a random direction.
"+01": A red arrow will be created in the last direction.
"R1": A red arrow will come from a random direction.
"D": A blue arrow will come from a different direction.
"D1": A red arrow will come from a different direction.

Can you guess which chart it is? (It is the intro part of the chart)

Here is another example of CreateChart

```
RegisterFunctionOnce("WaveR", () =>...);
RegisterFunctionOnce("wLineL", () =>...);
RegisterFunctionOnce("WaveL", () =>...);
RegisterFunctionOnce("wLineR", () =>...);
```

```
    CreateChart(BeatTime(4), BeatTime(1), 6f, new string[]
    {
"(#3.5#$3)(<+0'0.8)(>+0'0.8)","","","",   "+2","","","",
"+0","","","",    "+0","","","",
"","","","",    "","","","",
"","","","",    "","","","",

"(#3.5#N0)(WaveR)(wLineR)","","","",   "+2","","","",
"+0","","","",    "+0","","","",
"","","","",    "","","","",
"","","","",    "","","","",

"(#3.5#N3)(WaveR)(wLineR)","","","",   "+2","","","",
"+0","","","",    "+0","","","",
"","","","",    "","","","",
"","","","",    "","","","",

"(#3.5#N2)(WaveL)(wLineL)","","","",   "+2","","","",
"+0","","","",    "+0","","","",
"","","","",    "","","","",
"","","","",    "","","","",
//
```

(The tabbing of the code is poor)

You may have noticed that the first entry of the string[] is rather long, if you look closely, there are 3 pairs of brackets in it.

(#3.5$3), (<+0'0.8), and (>+0'0.8)

The () separates each action to execute, that means that 3 actions will be executed in the first beat.

#3.5$3: Creates a blue green soul blaster that lasts 3.5 beats, with the direction of up.

<+0'0.8: Creates a blue arrow that rotates to the left side, that comes from the same direction as the blaster, with a speed multiplier of 0.7 (Which being 6 * 0.8 = 4.8)

>+0'0.8: Creates a blue arrow that rotates to the left side, that comes from the same direction as the blaster, with a speed multiplier of 0.7 (Which being 6 * 0.8 = 4.8)

"+2": A blue arrow will be created at the opposite side of the box.

In the first entry of the second chunk of the string[], there is this "(#3.5#N0)(WaveR)(wLineR)"

#3.5#N0: Creates a blue green soul blaster that lasts 3.5 beats, with a direction of not right.

WaveR, wLineR: Execute functions that are declared

Can you guess which chart it is? (It is the intro part of the chart)

Here is another example of CreateChart()

```
RegisterFunctionOnce("Scale", () => RunEase(s => ScreenDrawing.ScreenScale = s, EaseOut(BeatTime(Arguments[1]),
  ScreenDrawing.ScreenScale, Arguments[0], EaseState.Expo)));
RegisterFunctionOnce("Angle", () => RunEase(s => ScreenDrawing.ScreenAngle = s,
        EaseOut(BeatTime(Arguments[1]), ScreenDrawing.ScreenAngle, Arguments[0], (EaseState)Arguments[2])));
```

```
CreateChart(BeatTime(4), BeatTime(4), 5.6f, new string[]
{
    //pre
    "", "", "", "",     "", "", "", "",

    //1
    "#3#d0(<1.3,3>Scale)(<12,3,6>Angle)", "", "", "",     "+20@A", "", "", "",
    "#3#d1(<1.12,3>Scale)(<-8,3,6>Angle)", "", "", "",     "+21@B", "", "", "",
    //2
    "#3#d0(<1.4,3>Scale)(<8,3,6>Angle)", "", "", "",     "+20@A", "", "", "",
    "#3#d1(<1,3>Scale)(<0,3,6>Angle)", "", "", "",     "+21@B", "", "", "",
    //3
    "#7#d0(<1.4,6>Scale)(<12,4,8>Angle)", "", "", "",     "+20@A", "", "", "",
    "+00@B", "", "", "",     "+00@A", "", "", "",
    //4
    "#7#d1(<1,6>Scale)(<-20,6,8>Angle)", "", "", "",     "+21@B", "", "", "",
    "+01@A", "", "", "+01@B",     "+01@A", "", "", "",
    //5
    "#3#d0(<1.3,3>Scale)(<12,3,6>Angle)", "", "", "",     "+20@A", "", "", "",
    "#3#d1(<1.15,3>Scale)(<0,3,6>Angle)", "", "", "",     "+21@B", "", "", "",
    //6
    "#3#d0(<1.24,3>Scale)(<-15,3,6>Angle)", "", "", "",     "+20@A", "", "", "",
    "#3#d1(<1.1,3>Scale)(<0,3,6>Angle)", "", "", "",     "+21@B", "", "", "",
```

It's getting more and more complicated, isn't it?
In the first image,"Scale" and "Angle" are defined actions, reading the code would tell you that they create an easing of the screen scale and the screen angle.
In the string[], you would see: "#3#d0(<1.3,3>Scale)(<12,3,6>Angle)"
#3#d0: Creates a blue green soul blaster that lasts 3 beats in a different direction.
<1.3,3>Scale: Executes the "Scale" function with '1.3' and '3' as its arguments, which means that this function will execute
RunEase(s => ScreenDrawing.ScreenScale = s, EaseOut(BeatTime(3), ScreenDrawing.ScreenScale, 1.3f, EaseState.Expo));
<12,3,6>Angle: Executes the "Angle" function with '12', '3', and '6' as its arguments, which means that this function will execute
RunEase(s => ScreenDrawing.ScreenAngle = s, EaseOut(BeatTime(3), ScreenDrawing.ScreenAngle, 12, (EaseState)6))

You may have noticed the "@A" and "@B" in the code, but that is for later.
I believe you have basic understanding on how to use CreateChart() by now, if not, you can just ask in the Discord server.

Non-green soul

Although the above functions are mainly for green soul attacks, CreateChart can also invoke functions for non-green soul functions, like creating bones.

Bone