# SmartCTF

The_Cowboy

April 15, 2019

## 1 Introduction

SmartCTF is a Mutator/ServerActor which reinforces and encourages the collaborative "Teamplay" by rewarding the players "Smartly". Furthermore, the aim is to display the complete information required for a decent "Multiplayer" team game which includes TickRate, Current/Elapsed game time and "Collective Ping/Netspeed" of the teams.

SmarCTF (for UnrealTournament) has been developed for over a decade by several "Coder Players" to meet the dynamic needs of the manifold of modern "PlayerBase". For the accurate chronological timeline click here.

This document concerns with the incarnation of SmartCTF (version 1A) for UT2004.

## 2 Installation

- place the SmartCTF1A.u and SmartCTF1A.ucl files in the `System` directory.
- place the CountryFlags2.utx in the `Textures` directory. For Client install, you are done. Load the SmartCTF1A from mutator list in the game.
- for Server install, in the UT2004.ini or Server.ini, add the line

```
ServerPackages = SmartCTF1A
ServerPackages = CountryFlags
```

Note: SmartCTF1A should be loaded as a mutator via appropriate command line

```
?mutator=SmartCTF1A.SmartCTF
```

Furthermore, admin needs to set the "Red(Blue)FlagZone" for each CTF map individially (via Mapvote, see Section 3).

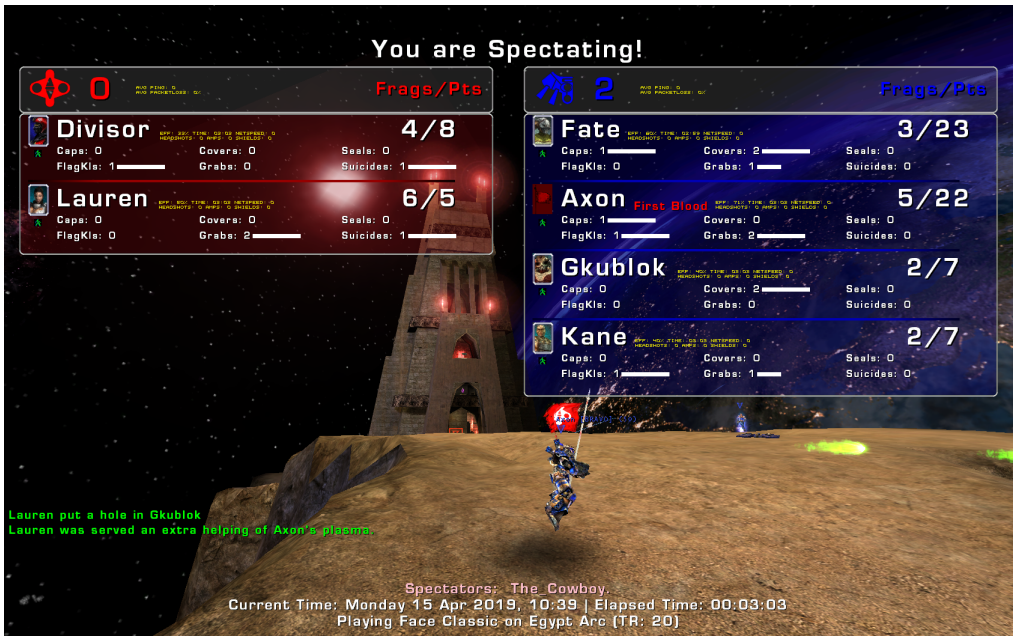


Figure 1: A screenshot of the SmartCTF1A Scoreboard.

- If you don't "trust" the SmartCTF's default QueryServerHosts (located in IpToNation.ini) you can upload the scripts in `MasterServerv1.6` on the PHP webserver as per the readme there. The original PHP scripts should work so I haven't renamed/ported them.

# 3 Configuration

There are three configurable settings for SmartCTF1A.

## 3.1 SmartCTF.ini

The SmartCTF.ini file is included with the package and contains the following section

```
[SmartCTF1A.SmartCTF]
bShowLogo=True
CoverReward=2
CoverAdrenalineUnits=5
SealAward=2
SealAdrenalineUnits=5
RedFlagZone=RED BASE LOWER LEVEL
BlueFlagZone=BLUE BASE LOWER LEVEL
ScoreBoardType=SmartCTF1A.SmartCTFScoreBoard
bShowFCLocation=True
bBroadcastMonsterKillAndAbove=True
```

Most configurable variables are self-explanatory. To identify a "Seal", the admin must recognize the mapzones with the flags. In order to do that, start the game with the CTF map and stand near the flag. Then give some order ("Defend the Flag" etc) which will reveal the zone you are standing in. For example in CTF-FaceClassic, the RedFlagZone is "RED BASE LOWER LEVEL" and so on. Different maps may have different Flag zone names (if the author has been decent enough to define zones).

bShowFCLocation if set true will display the team FC location at the top of the HUD to all the team players.

bBroadcastMonsterKillAndAbove is set to true will broaccast the Monsterkill message to all the players when someone achieves it.

## 3.2 IpToNation.ini

SmartCTF has built in IpToNation mutator which is based on IpToCountry mutator developed by [es]Rush and Matthew 'MSuLL' Sullivan for UT99. An excerpt from the IpToCountry readme

> "To your knowledge, resolving a country from an IP requires quite a big database and implementing it in UnrealScript would be very hard, thus IpToCountry connects to a PHP script which is located on a web server. This PHP script uses a database kindly distributed by www.maxmind.com If you want to know how the querying stuff exactly works just "Use the source Luke!"
>
> AOL is quite troublesome. AOL has all its IP ranges registered in the USA and thus it doesn't make identification easy. Fortunately Rush found a way to go around the problem. Three major countries where AOL is very popular are: USA, Great Britain and Germany. All those three have different timezones, so Rush just had to get a player's time and compare it to GMT. The idea maybe was Rush's but Cratos was the first to implement it in LeagueAS, Rush got some code from him so big thanks, mate."

Now to configure, open IpToNation.ini (a fresh copy is always generated on the first run, if not there in the directory)

```
[SmartCTF1A.LinkActor]
QueryServerHost[0]=iptocountry.ut-files.com
QueryServerHost[1]=www.ut-slv.com
QueryServerHost[2]=utgl.unrealadmin.org
QueryServerHost[3]=
QueryServerFilePath[0]=/iptocountry16.php
QueryServerFilePath[1]=/iptocountry/iptocountry16.php
QueryServerFilePath[2]=/iptocountry16.php
QueryServerFilePath[3]=
QueryServerPort[0]=80
QueryServerPort[1]=80
QueryServerPort[2]=80
QueryServerPort[3]=80
MaxTimeout=10
ErrorLimit=5
IPData[0]=27.7.228.155:27.7.228.155:INDIA:IND:in
IPData[1]=2.20.152.0:a2-20-152-0.deploy.static.akamaitechnologies.com:::eu
...
ResolvedAddress[0]=192.111.155.210
ResolvedAddress[1]=45.250.173.52
ResolvedAddress[2]=216.218.207.107
ResolvedAddress[3]=
bNeverPurgeAddress=False
```

You can learn from the example given above on configuring QueryServerHosts (right now 4 are supported). The array IPData is filled automagically with the resolved IPs.

MaxTimeout is the time in seconds after which IpToNation gives up querying the serverhosts. ErrorLimit is the limiting number of times before IpToNation starts querying another serverhost after restart.

## 3.3 Server.ini

SmartCTF logo splash can be highly configured in the section SmartCTF1A.SmartCTFLogo generated in the UT2004.ini or Server.ini.

```
[SmartCTF1A.SmartCTFLogo]
LogoColor=(B=255,G=255,R=255,A=255)
LogoTexCoords=(X=0,Y=0,W=0,H=0)
StartLogoRotationRate=0
LogoRotationRate=0
EndLogoRotationRate=0
StartPos=(X=0.990000,Y=0.100000)
pos=(X=0.990000,Y=0.100000)
EndPos=(X=0.990000,Y=0.100000)
DrawPivot=DP_MiddleRight
StartScale=(X=1.000000,Y=1.000000)
Scale=(X=1.000000,Y=1.000000)
EndScale=(X=1.000000,Y=1.000000)
FadeInSound=
DisplaySound=WeaponSounds.BLockOn1
FadeOutSound=
AnnouncerSounds=False
FadeInDuration=0.500000
DisplayDuration=3.000000
FadeOutDuration=1.500000
InitialDelay=0.000000
FadeInRotationTransition=FT_None
FadeOutRotationTransition=FT_None
FadeInAlphaTransition=FT_Linear
FadeOutAlphaTransition=FT_Linear
FadeInScaleTransition=FT_None
FadeOutScaleTransition=FT_None
FadeInPosXTransition=FT_None
FadeInPosYTransition=FT_None
FadeOutPosXTransition=FT_None
FadeOutPosYTransition=FT_None
```

The logo code is based on Wormbo's Server Logo. That package has detailed information on configuring the logo (including animations such as rotation etc). Don't play with the above settings if you don't know what you are doing!

# 4 Features

SmartCTF has a list of cool features which makes the CTF more fun.


- Logo: The traditional SmartCTF "Powered" logo splashes on the top right corner of the screen on each client's box which marks the manifestation of the "Smart" gameplay.

- More Statistics: The SmartCTF interface registers a lot more CTF game relevant events and generates the "appropriate" rewards (which can be configured in SmartCTF.ini). This list includes
  - Covers: Rewards score and adrenaline[1]
  - Seals: Rewards score and adrenaline.
  - FlagKills
  - Captures
  - (Flag) Grabs
  - First Blood
  - Frags
  - Efficiency: Defined by the equation

$$\text{Eff} = \frac{\text{Frags}}{\text{Frags} + \text{Deaths}} \tag{1}$$

  - Points
  - Amps
  - Suicides
  - HeadShots
  - Nation: The CountryFlag 
- More MultiPlayer, Team and Server information
  - Average/individual pings, packetlosses and netspeeds are computed and shown.
  - The Server TickRate is displayed in the Title component. For more information on TickRates visit Netspeed Tutorial.
  - Current Date (Day-Month-Year)/Time and Elapsed Time are obtained and shown in the Title component of ScoreBoard.
  - The Server name and Map name are shown in the Scoreboard.
- Messaging System: SmartCTF has a unique way of broadcasting the Cover/Seal message to all the players. The hope is to encourage players more for Covering the FlagCarrier or Sealing the Base over the "Narcissist" DeathMatch gameplay mentality. Furthermore, SmarCTF can broadcast global messages related to "MonsterKill" and above.
- Scoreboard: The most admirable component of SmartCTF1A is the Scoreboard (Figure 1). It consists of the title and the body. The title of the Scoreboard is divided in two components

[1] The methodology is written in Section 5.2.


Figure 2: A screenshot of FC location.

  - Header: shows the current state of the game.
  - Footer: shows the spectator list (with Epic's bugfix!), current date/-time and elapsed time and server/map name with the "Tickrate".

  The body of the scoreboards shows more player statistics.

- Statistics Restoration: SmartCTF1A has the capability to restore the statistics of the player who disconnected due to unfavorable circumstances (poor net connection, power fluctuations etc). The resoration is done on the name basis. Note that the total player score is *not* restored by SmartCTF (to avoid the conflict with other stats restoring mutators).
- Flag Carrier Location: SmartCTF can render the current location of the Flag Carrier at the top of the screen (below the team scores).

# 5 Working

In this section, we will explore the actual but brief working of SmartCTF by directly referencing the "UnrealScript Code". Once we understand the code, we can perform further tweaks to enhance the gameplay.

## 5.1 Witness

The class SmartCTF.uc spawns an instance named "Witness" to enforce actual counting of the player statistics. The code is

```
Witness = Level.Game.Spawn(class'UTServerAdminSpectator');
Witness.PlayerReplicationInfo.PlayerName = "Witness";
if(Witness != none)
    Log("### Successfully Spawned the Witness"@Witness, 'SmartCTF'
    );
else
    Log("ERROR! Couldn't Spawn the Witness", 'SmartCTF');
...
```

The role of the "Witness" is to silently spectate the game and receive the game messages. Then in the function "EvaluateMessageEvent", as follows,

```
1   switch(Switch){
            // CAPTURE
3           // Sender: CTFGame, PRI: Scorer.PlayerReplicationInfo,
      OptObj: TheFlag.Team
            case 0:
5               if(MessagingSpectator(Receiver) == Witness){
                    ReceiverStats = SCTFGame.GetStats(Controller(
      RelatedPRI_1.Owner));
7                   if(ReceiverStats != none) ReceiverStats.Captures++;
                    ResetSprees(0);
9                   ResetSprees(1);
                    FCs[0] = none;
11                  FCs[1] = none;
                }
13              break;
...
```

line 5 essentially makes sure that line 7 gets executed only once. Otherwise the "Capture" counter will increase the number of times the function "EvlauateMessageEvent" is executed which is equal to the number of Human players in the game.

Note that the "Witness" is never shown in the spectator list of the "SmartScoreboard". This can be easily seen from the line 9 of the code (in SmartCTFScoreBoard.uc)

```
  function string GetSpectatorString(){
2
    local string ReturnString;
4   local int i;

6   if(GRI == none || !GRI.bMatchHasBegun)// No spectators are
    shown during the Match Survey
        return "";
8   for(i = 0; i < GRI.PRIArray.Length; i++){
        if(GRI.PRIArray[i].bIsSpectator && GRI.PRIArray[i].
    PlayerName != "Witness"){
10          if(ReturnString == "")
                ReturnString = ReturnString@GRI.PRIArray[i].PlayerName
    ;
12          else
                ReturnString = ReturnString$","@GRI.PRIArray[i].
    PlayerName;
14      }
    }
16  return ReturnString;
}
```

## 5.2 Covers/Seals

Consider the situation in which Red players PlayerR1 and PlayerR2 (with the Blue flag) and Blue player PlayerB1 (with or without Red flag) are involved. PlayerR1 kills PlayerB1. In code[2]:

```
if(KillerPRI.HasFlag == none && FCs[KillerPRI.Team.TeamIndex] !=
  none && FCs[KillerPRI.Team.TeamIndex].PlayerReplicationInfo.
  HasFlag != none)
```

We define "Cover" when:

- PlayerB1 was killed within 576 uu (Unreal Units[3]) of PlayerR2
- or PlayerR1 was withing 576 uu of PlayerR2
- or PlayerB1 could see PlayerR2 and was killed within 1728 uu of PlayerR2
- or PlayerB1 had direct line-of-sight to PlayerR2 and was killed within 864 uu of PlayerR2.

Note that these figures are obtained from experience and are open for further modifications based on appropriate discussions!

Code is:

```
if((VSize(Killed.Location - FCs[KillerPRI.Team.TeamIndex].Pawn.
    Location) < 512*1.125)
       || (VSize(Killer.Pawn.Location - FCs[KillerPRI.Team.
    TeamIndex].Pawn.Location) < 512*1.125)
       || (VSize(Killed.Location - FCs[KillerPRI.Team.TeamIndex].
    Pawn.Location) < 1536*1.125 && Killed.Controller.CanSee(FCs[
    KillerPRI.Team.TeamIndex].Pawn))
       || (VSize(Killed.Location - FCs[KillerPRI.Team.TeamIndex].
    Pawn.Location) < 1024*1.125 && Killer.CanSee(FCs[KillerPRI.Team
    .TeamIndex].Pawn))
       || (VSize(Killed.Location - FCs[KillerPRI.Team.TeamIndex].
    Pawn.Location) < 768*1.125 && Killed.Controller.LineOfSightTo(
    FCs[KillerPRI.Team.TeamIndex].Pawn))){
       // Killer DEFENDED THE Flag CARRIER
           if(KillerStats != none){
               KillerStats.Covers++;
               KillerStats.CoverSpree++;// Increment Cover spree
               if(KillerStats.CoverSpree == 3){// Cover x 3
                   BroadcastLocalizedMessage(class'
    SmartCTFMoreMessages', 2, KillerPRI);
               }
               else if(KillerStats.CoverSpree == 4){// Cover x 4
                   BroadcastLocalizedMessage(class'
    SmartCTFMoreMessages', 1, KillerPRI);
               }
```

[2]The code written in this section can be located in SmartCTF.uc

[3]These are the units for UT2004. For more information on units, consult UnrealWiki.

```
            else{// Cover
17              BroadcastLocalizedMessage(class'
    SmartCTFMoreMessages', 0, KillerPRI);
            }
19      }
        KillerPRI.Score += CoverReward;// Cover Bonus
21      Killer.AwardAdrenaline(CoverAdrenalineUnits);
    }
```

Lines 8 increases the Cover counter (which is displayed in the Scoreboard), lines 11, 13 and 17 broadcast the message to all the players on HUD and lines 20 and 21 provide the appropriate rewards to the player.

We define "Seal" when:

- PlayerR2 is in "Red FlagZone"[4] and Red flag is *not* with any Blue player.
- PlayerB1 is killed in "Red FlagZone".

The relevant code is

```
bKilledTeamHasFlag = true;
2   if(FCs[KilledPRI.Team.TeamIndex] == none) bKilledTeamHasFlag
    = false;
    if(FCs[KilledPRI.Team.TeamIndex] != none &&
4    FCs[KilledPRI.Team.TeamIndex].PlayerReplicationInfo.HasFlag
    == none) bKilledTeamHasFlag = false;// Safety check

6   // if Killed's FC has not been set / if Killed's FC doesn't
    have our Flag
    if(!bKilledTeamHasFlag){
8       // If Killed and Killer's FC are in Killer's Flag Zone
        if(IsInZone(KilledPRI, KillerPRI.Team.TeamIndex) ||
    IsInzone(FCs[KillerPRI.Team.TeamIndex].PlayerReplicationInfo,
    KillerPRI.Team.TeamIndex)){
10          // Killer SEALED THE BASE
            if(KillerStats != none)
12              KillerStats.Seals++;
            BroadcastLocalizedMessage(class'SmartCTFMoreMessages',
     3, KillerPRI);
14          KillerPRI.Score += SealAward;//Seal Bonus
            Killer.AwardAdrenaline(SealAdrenalineUnits);
16      }
    }
```

## 5.3 Replication

Replication is one of the very important aspects of Unreal Engine. So important that Tim Sweeney[5] wrote an article exclusively for it in 1999-07-21 (for the

[4] For definition of FlagZone we refer the reader to Section .

[5] If you don't know him, you should probably purge this document.

current avatar of the document visit Unreal Networking Architecture). In this section I will demonstrate the "Replication at Work" from "Live Example". One of the central features of SmartCTF is to collect and display lot more statistics (see Section 4). These statistics are generated on the server instance of the game. On the other hand statistics are displayed on the client instance of the game connected to server via internet or LAN (party?).

First we spawn an Actor named "SmartCTFPlayerReplicationInfo" which inherites is properties from the base class "ReplicationInfo". This is done by the code (in SmartCTF.uc)

```
1  function AssociateSmartReplication(Pawn Other){

3      local SmartCTFPlayerReplicationInfo NewSPRI;
       local int i;
5
       NewSPRI = Spawn(class'SmartCTFPlayerReplicationInfo', Other.
   PlayerReplicationInfo);
7      NewSPRI.SPlayerName = Other.PlayerReplicationInfo.PlayerName
   ;
       NewSPRI.SPlayerID = Other.PlayerReplicationInfo.PlayerID;
9      NewSPRI.IpToNation = IpToNation;
       NewSPRI.bIpToNation = true;
11     for(i = 0; i < GoneSmartPRI.Length; i++){
           if(GoneSmartPRI[i] != none && Other.PlayerReplicationInfo
   .PlayerName == GoneSmartPRI[i].SPlayerName){// Include IP check
   ?
13             NewSPRI.CopyStats(GoneSmartPRI[i]);
               GoneSmartPRI.Remove(i, 1);// Save the bandwidth :D
15         }
       }
17 }
...
```

This code is executed on both server and client machines. Line 6 essentially spawns the Actor on both the machines and sets the "Owner" of this instance as PlayerReplicationInfo (which itself is an Actor). But the owner is *not* set client side. Therfore we need a way to identify this instance from client side (for example while displaying the ScoreBoard). We can do this by assigning a unique "number" to a memberdata of the instance. Fortunately this number is generated by the engine and known as PlayerID memberdata of class "PlayerReplicationInfo".

The server instance always has the bleeding (updated) values of all the memberdata. Thus from line 8 we assign the ID to the Actor instance of class "SmartCTFPlayerReplicationInfo". But this is done serverside. The client (as usual) is sitting dumb ignorant of this information. So in order to make sure that the information "flows", we replicate the memberdata in the "SmartCTF-PlayerReplicationInfo" by using the code

```
replication{
```

```
...
    // Things the server should send to the client.
  reliable if (bNetDirty && (Role == Role_Authority))
    NetSpeed, NationPrefix, SPlayerName, SPlayerID;
...
}
```

Now any function, executed from client instance can easily access the member-data SPlayerID. The example can be easily seen by the code

```
simulated function SmartCTFPlayerReplicationInfo GetStats(
   PlayerReplicationInfo A){

 local int i;

 if(A == none) return none;
 ReloadBuffer();// Collect the swarm of SmartCTFPlayerReplication
   instances.
 for(i = 0; i < PRIBuffer.Length; i++){
    if(A.PlayerID == PRIBuffer[i].SPlayerID){
       return PRIBuffer[i];
    }
 }
 return none;
}
```

Here, when the code is being executed on the client instance, line 8 is useful only if SPlayerID has already replicated properly to the client.

## 5.4 SmartScoreboard

I have taken the help of Epic's default Scoreboard for the "Compression Scheme" and SmartCTF's years of experience in setting the layout. The end result is that it can show the statistics of 23 players with the resolution of $1680 \times 1050$ without cluttering the screen. Most of the code is self explanatory (see the associated comments to get a deep understanding). However there are few code snippets that require explainnation.

First, the scoreboard is drawn on the abstract instance Canvas. In order to work with the instance, there is a natural sequence of code is written

```
Canvas.Font = SomeFont;// Sets the Font Size
Canvas.Strlen("Test", XL, YL);
Canvas.SetPos(SomeX, SomeY);// Sets the Position on the Screen
Canvas.DrawColor = SomeColor;// Sets the Color
Canvas.SomeDrawCommand(Some Arguments);// Actual Draw Command
```

Line 2 essentially measures the width and height of the text 'Test' with the "said" fontsize. It is useful to estimate the position of "next" possible drawing item.

### 5.4.1 Compression Scheme

The compression of the scoreboard is done by reducing the fontsize of the text "PlayerName" in 4 stages. The rest of the structural components which include "PlayerBoxSizeY" (the height of individual PlayerBox) and "BoxSizeY" (the height of header box) are computed accordingly.

The code is

```
1  if(MaxPlayerCount > (C.ClipY - MessageFoot)/(PlayerBoxSizeY +
     BoxSpaceY)){
     // Compress the Scoreboard: Decrease the Header Box
3    BoxSpaceY = 0.125 * YL;
     if(MaxPlayerCount > (C.ClipY - MessageFoot)/(PlayerBoxSizeY +
   BoxSpaceY)){
5        // Compress the Scoreboard: Decrease individual
   PlayerBoxSize and Font
         FontReduction++;
7        C.Font = GetSmallerFontFor(C, FontReduction);
         C.StrLen("Test", XL, YL);
9        BoxSpaceY = 0.125 * YL;
         if (HaveHalfFont(C, FontReduction))
11           PlayerBoxSizeY = 2.125 * YL;
         else
13           PlayerBoxSizeY = 1.75 * YL;
         HeadFoot = 4*YL + IconSize;
15       if(MaxPlayerCount > (C.ClipY - MessageFoot)/(
   PlayerBoxSizeY + BoxSpaceY)){
           // Compress further by reducing Font
17         FontReduction++;
           C.Font = GetSmallerFontFor(C, FontReduction);
19       C.StrLen("Test", XL, YL);
         BoxSpaceY = 0.125 * YL;
21       if (HaveHalfFont(C, FontReduction))
             PlayerBoxSizeY = 2.125 * YL;
23       else
             PlayerBoxSizeY = 1.75 * YL;
25       HeadFoot = 4*YL + IconSize;
         if(C.ClipY >= 600 && (MaxPlayerCount > (C.ClipY -
   HeadFoot)/(PlayerBoxSizeY + BoxSpaceY))){
27           // Compress further if Resolution is High enough (Why
    HeadFoot?)
             FontReduction++;
29             C.Font = GetSmallerFontFor(C, FontReduction);
           C.StrLen("Test", XL, YL);
31         BoxSpaceY = 0.125 * YL;
           if (HaveHalfFont(C, FontReduction))
33             PlayerBoxSizeY = 2.125 * YL;
           else
35             PlayerBoxSizeY = 1.75 * YL;
           HeadFoot = 4*YL + IconSize;
37         if(MaxPlayerCount > (C.ClipY - 1.5 * HeadFoot)/(
   PlayerBoxSizeY + BoxSpaceY)){
```

```
                    // final Compression
39                  FontReduction++;
                    C.Font = GetSmallerFontFor(C, FontReduction);
41                  C.StrLen("Test", XL, YL);
                    BoxSpaceY = 0.125 * YL;
43                  if (HaveHalfFont(C, FontReduction))
                        PlayerBoxSizeY = 2.125 * YL;
45                  else
                        PlayerBoxSizeY = 1.75 * YL;
47                  HeadFoot = 4*YL + IconSize;
                }
49          }
        }
51      }
    }
```

# 6 Developing

## 6.1 ScoreBoard

The most appropriate layout of the "SmartScoreboard" is already set. What is required is an appropriate "Color Scheme". It means applying appropriate colors to various components of the the scoreboard in the most visually appealing manner. In order to apply the scheme, one can first consult the code of section 5.4.

UT99 community has developed a snowy version of the scoreboard with "animated" snow falling down. Similar techniques can be applied to the "SmartScoreboard" to render the snow!

# 7 TODO

As far as coding is concerned, I have done most of it. One feature might be adding the "SpawnKill" detection. I leave it for the future. Now it would be cool to add various resources to SmartCTF. The list includes

- Announcer Sounds for the "Cover" and "Seal". And I am talking about the good ol' Classic Announcer-style!
- Textures for the Scoreboard including background, FirstBlood, and partition generators. Animated textures would be cool. [es]Rush has already done awesome job with the CountryFlags textures. It would be nice to have better textures for FC in the socreboard (Figure 1).


Figure 3: A screenshot of the snowy scoreboard.

# 8 Credits

SmartCTF has a rich and animated histroy due to the contributions of several outstanding coders and their inspirations. The list (in no particular order) includes

- {PiN}Kev
- {DnF2}SiNiSTeR
- [es]Rush
- Adminthis
- Sp0ngeb0b

For this UT2k4 port, I would like to especially thank

- Wormbo (for his ServerLogo mod and endless contribution in the programming universe of Unreal Engine)
- [es]Rush and Matthew 'MSuLL' Sullivan for IpToCountry methods
- Epic Games for releasing new versions of the "Game we Love"

Finally the online resources

- https://ut-files.com/
- http://unrealtournament.99.free.fr/

A big thanks for keeping the “ancient goodwill” intact!