INSIDE THE ULTIMA ONLINE GOLD DEMO
- The Fatigue Algorithm

GOAL

It's our goal to get a deep understanding of how the Ultima Online Gold Demo works. This demo is a representation of the rule set from the Ultima Online Second Age Era.

There is proof that some people have already reversed this demo partially or as a whole, however so far no tools or knowledge has been published. This project is to overcome those shortcomings.

URL's with some proof for this:
http://www.runuo.com/forums/general-discussion/94767-help-m-files.html
http://azaroth.org/2008/12/31/your-topic/ (posting by Faust)

If we understand the demo there is a big chance we can alter the demo and even create our own demo. By default mounting horses is not possible in the demo, but what if we can alter the demo and unlock horses; can we then see how horses behaved during T2A?

This demo is 10 years old and I do not understand no one published his/her work. Maybe that DMCA thing is in the way?

UTILITIES USED

IDA Pro, a very professional utility, definitely worth buying, Standard version is affordable.

ABOUT ME

I love computers, I love computer internals, I love hacking, but above all, I love my wife.

INTRODUCTION

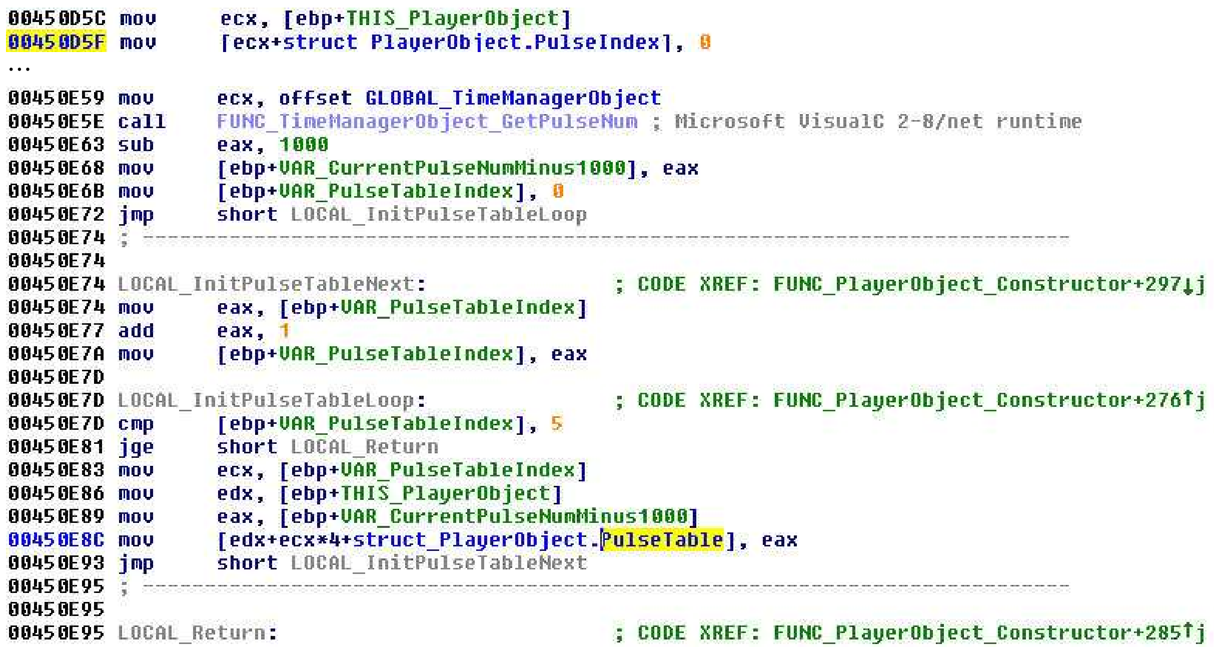
When I started hacking and reverse engineering the demo, now more than a year ago, there wasn't much known about the internals. But with the help of IDA Pro and lots of time, a lot of the core OSI functions/algorithms have been documented by me. In this document I will show you the code required to implement fatigue loss when you move in the game, either by walking or running.

Know that the move packet (02) is a two-byte packet in the demo and that anti-fastwalk code was only added later on by OSI.

THE PULSE TABLE

OSI's fatigue loss algorithm uses an array of 5 integers which initially contains the Server-Side pulse number minus 1000 pulses. Pulse Number is the official OSI term inside their code. On the internet OSI referred to a pulse as a tick. A tick is 250ms but if the server is too busy a tick (or pulse) could take longer. There is also an index into this table which will be increased per move packet.

The index and table are initialized when the Player Object is created:



```
00450D5C mov     ecx, [ebp+THIS_PlayerObject]
00450D5F mov     [ecx+struct PlayerObject.PulseIndex], 0
...
00450E59 mov     ecx, offset GLOBAL_TimeManagerObject
00450E5E call    FUNC_TimeManagerObject_GetPulseNum ; Microsoft VisualC 2-8/net runtime
00450E63 sub     eax, 1000
00450E68 mov     [ebp+VAR_CurrentPulseNumMinus1000], eax
00450E6B mov     [ebp+VAR_PulseTableIndex], 0
00450E72 jmp     short LOCAL_InitPulseTableLoop
00450E74 ; ---------------------------------------------------------------------------
00450E74
00450E74 LOCAL_InitPulseTableNext:                ; CODE XREF: FUNC_PlayerObject_Constructor+297↓j
00450E74 mov     eax, [ebp+VAR_PulseTableIndex]
00450E77 add     eax, 1
00450E7A mov     [ebp+VAR_PulseTableIndex], eax
00450E7D
00450E7D LOCAL_InitPulseTableLoop:                ; CODE XREF: FUNC_PlayerObject_Constructor+276↑j
00450E7D cmp     [ebp+VAR_PulseTableIndex], 5
00450E81 jge     short LOCAL_Return
00450E83 mov     ecx, [ebp+VAR_PulseTableIndex]
00450E86 mov     edx, [ebp+THIS_PlayerObject]
00450E89 mov     eax, [ebp+VAR_CurrentPulseNumMinus1000]
00450E8C mov     [edx+ecx*4+struct_PlayerObject.PulseTable], eax
00450E93 jmp     short LOCAL_InitPulseTableNext
00450E95 ; ---------------------------------------------------------------------------
00450E95
00450E95 LOCAL_Return:                            ; CODE XREF: FUNC_PlayerObject_Constructor+285↑j
```

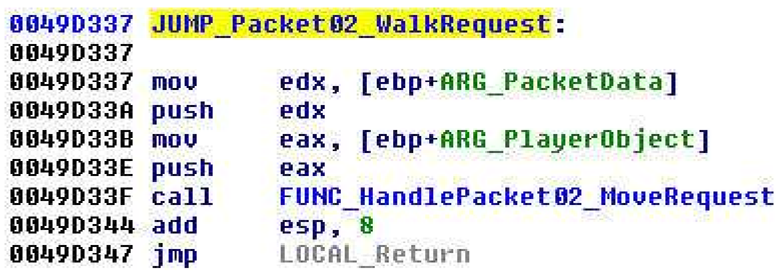
```
⇨ this->PulseIndex = 0;
  …
  int CurrentPulseNumMinus1000 = TimeManager.GetPulseNum() - 1000;
  for(int i = 0; i < 5; i ++)
    this->PulseTable[i] = CurrentPulseNumMinus1000;
```

As shown later, this Pulse Table is used to detect the moving speed of the player.

MOVE PACKET

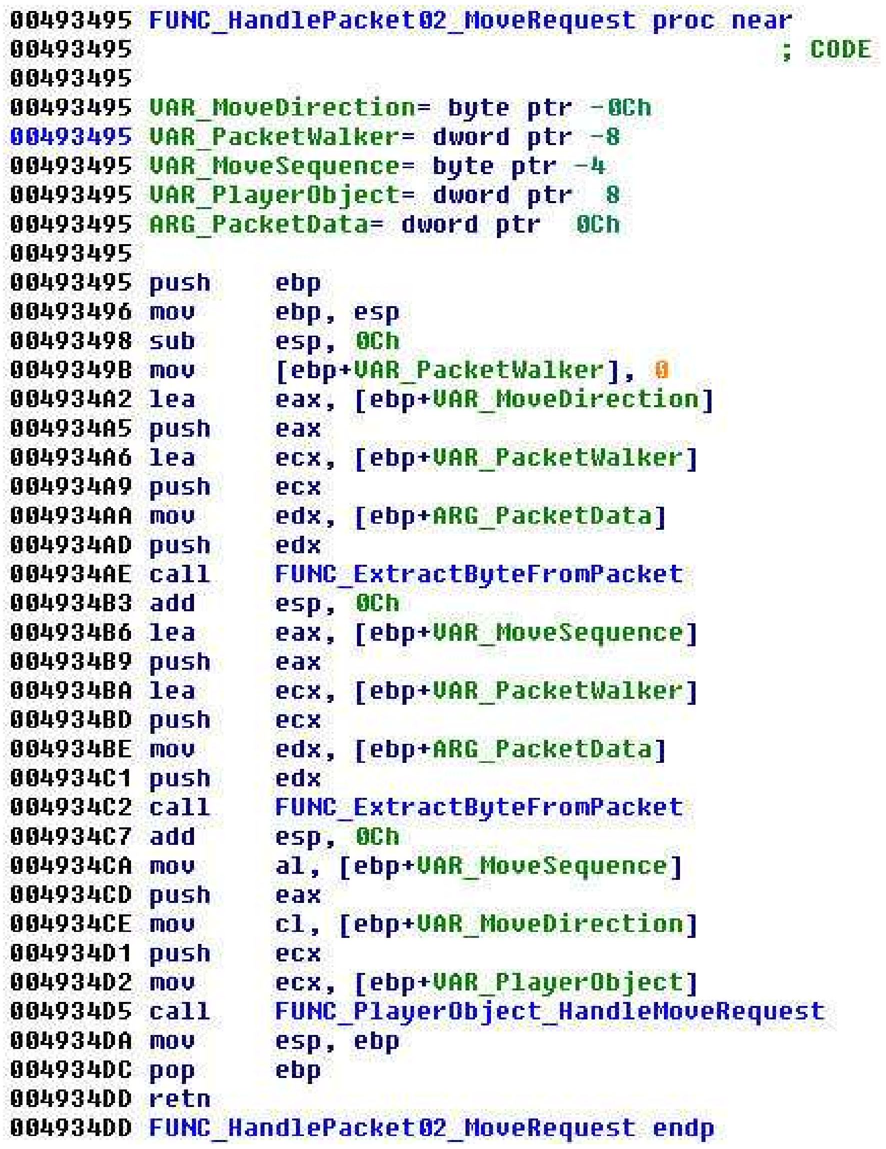
As mentioned earlier, the move packet will trigger the fatigue loss algorithm. Let's take a look at how this goes:

1) First the packet itself is received and using a switch/case statement, packet 02 will be handled, this screenshot is part of the switch/case statement:



```
0049D337 JUMP_Packet02_WalkRequest:
0049D337
0049D337 mov     edx, [ebp+ARG_PacketData]
0049D33A push    edx
0049D33B mov     eax, [ebp+ARG_PlayerObject]
0049D33E push    eax
0049D33F call    FUNC_HandlePacket02_MoveRequest
0049D344 add     esp, 8
0049D347 jmp     LOCAL_Return
```

2) This is the Move Packet (or Move Request). This function will extract the 2 packet variables (Direction & Sequence) and then call a member function of the Player class to do the actual handling:

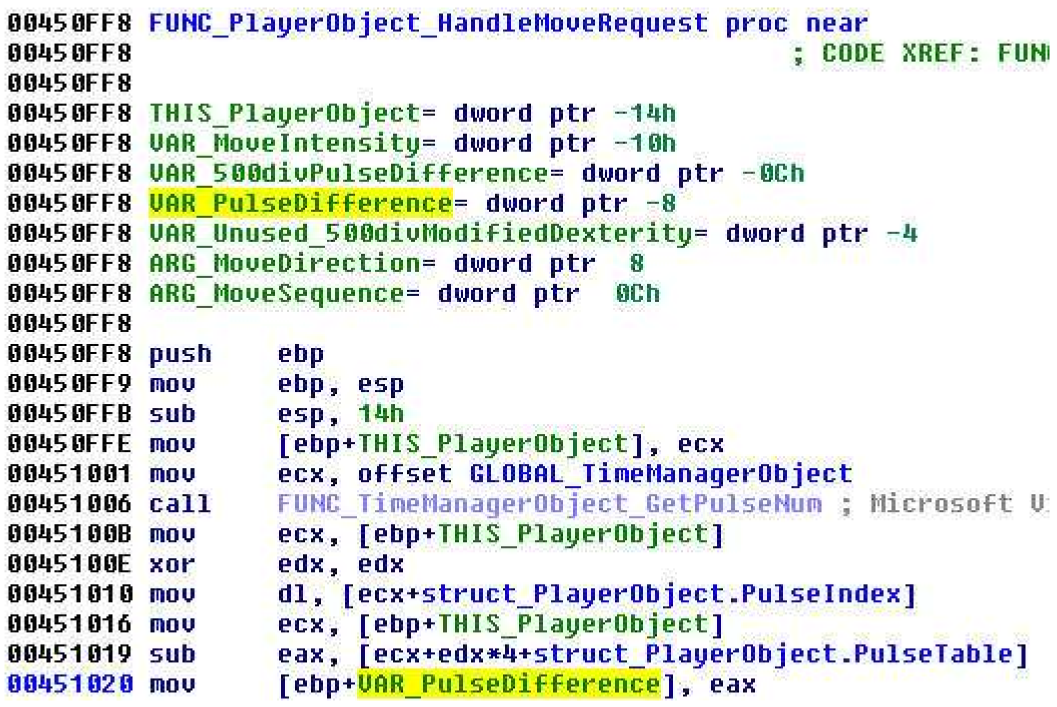


```
00493495 FUNC_HandlePacket02_MoveRequest proc near
00493495                                         ; CODE
00493495
00493495 VAR_MoveDirection= byte ptr -0Ch
00493495 VAR_PacketWalker= dword ptr -8
00493495 VAR_MoveSequence= byte ptr -4
00493495 VAR_PlayerObject= dword ptr  8
00493495 ARG_PacketData= dword ptr  0Ch
00493495
00493495 push    ebp
00493496 mov     ebp, esp
00493498 sub     esp, 0Ch
0049349B mov     [ebp+VAR_PacketWalker], 0
004934A2 lea     eax, [ebp+VAR_MoveDirection]
004934A5 push    eax
004934A6 lea     ecx, [ebp+VAR_PacketWalker]
004934A9 push    ecx
004934AA mov     edx, [ebp+ARG_PacketData]
004934AD push    edx
004934AE call    FUNC_ExtractByteFromPacket
004934B3 add     esp, 0Ch
004934B6 lea     eax, [ebp+VAR_MoveSequence]
004934B9 push    eax
004934BA lea     ecx, [ebp+VAR_PacketWalker]
004934BD push    ecx
004934BE mov     edx, [ebp+ARG_PacketData]
004934C1 push    edx
004934C2 call    FUNC_ExtractByteFromPacket
004934C7 add     esp, 0Ch
004934CA mov     al, [ebp+VAR_MoveSequence]
004934CD push    eax
004934CE mov     cl, [ebp+VAR_MoveDirection]
004934D1 push    ecx
004934D2 mov     ecx, [ebp+VAR_PlayerObject]
004934D5 call    FUNC_PlayerObject_HandleMoveRequest
004934DA mov     esp, ebp
004934DC pop     ebp
004934DD retn
004934DD FUNC_HandlePacket02_MoveRequest endp
```

MOVE PACKET

As mentioned earlier, the move packet will trigger the fatigue loss algorithm. Let's take a look at how this goes.

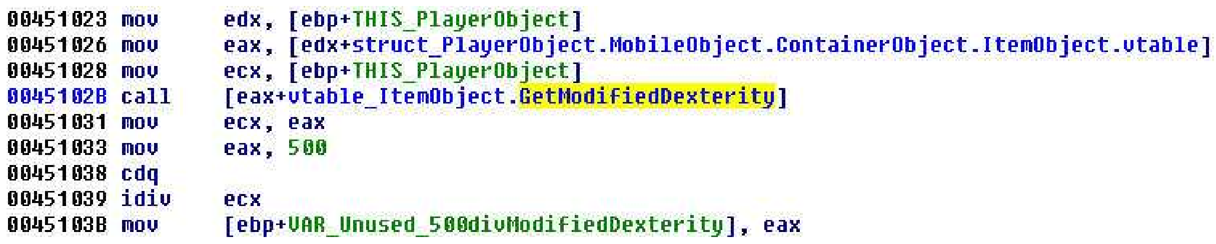
1) The first step of this member function will use the Pulse Table to calculate the difference with the previous move:



```
00450FF8 FUNC_PlayerObject_HandleMoveRequest proc near
00450FF8                                               ; CODE XREF: FUN
00450FF8
00450FF8 THIS_PlayerObject= dword ptr -14h
00450FF8 VAR_MoveIntensity= dword ptr -10h
00450FF8 VAR_500divPulseDifference= dword ptr -0Ch
00450FF8 VAR_PulseDifference= dword ptr -8
00450FF8 VAR_Unused_500divModifiedDexterity= dword ptr -4
00450FF8 ARG_MoveDirection= dword ptr  8
00450FF8 ARG_MoveSequence= dword ptr  0Ch
00450FF8
00450FF8 push    ebp
00450FF9 mov     ebp, esp
00450FFB sub     esp, 14h
00450FFE mov     [ebp+THIS_PlayerObject], ecx
00451001 mov     ecx, offset GLOBAL_TimeManagerObject
00451006 call    FUNC_TimeManagerObject_GetPulseNum ; Microsoft V
0045100B mov     ecx, [ebp+THIS_PlayerObject]
0045100E xor     edx, edx
00451010 mov     dl, [ecx+struct_PlayerObject.PulseIndex]
00451016 mov     ecx, [ebp+THIS_PlayerObject]
00451019 sub     eax, [ecx+edx*4+struct_PlayerObject.PulseTable]
00451020 mov     [ebp+VAR_PulseDifference], eax
```

```
⇨  int PulseDifference  = TimeManager.GetPulseNum()
                        - this->PulseTable[this->PulseIndex];
```

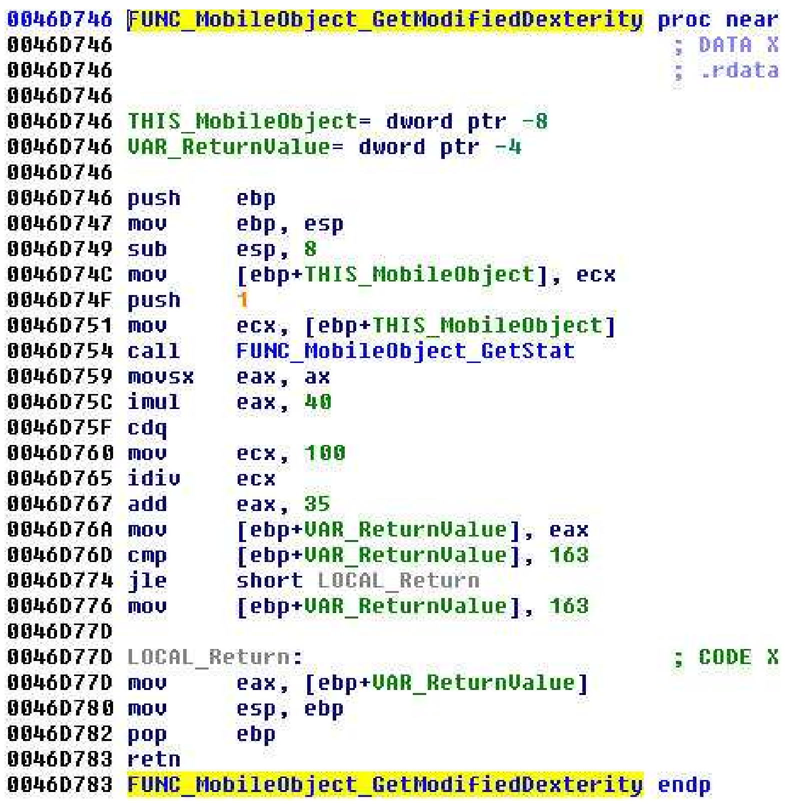
2) The second step is code that divides 500 by the player's 'modified' dexterity. This step is unneeded since the calculated value is used nowhere else. The code nonetheless:



```
00451023 mov     edx, [ebp+THIS_PlayerObject]
00451026 mov     eax, [edx+struct_PlayerObject.MobileObject.ContainerObject.ItemObject.vtable]
00451028 mov     ecx, [ebp+THIS_PlayerObject]
0045102B call    [eax+vtable_ItemObject.GetModifiedDexterity]
00451031 mov     ecx, eax
00451033 mov     eax, 500
00451038 cdq
00451039 idiv    ecx
0045103B mov     [ebp+VAR_Unused_500divModifiedDexterity], eax
```

```
⇨ int Unused_500divModifiedDexterity = 500 / GetModifiedDexterity();
```

This is the actual GetModifiedDexterity member function:

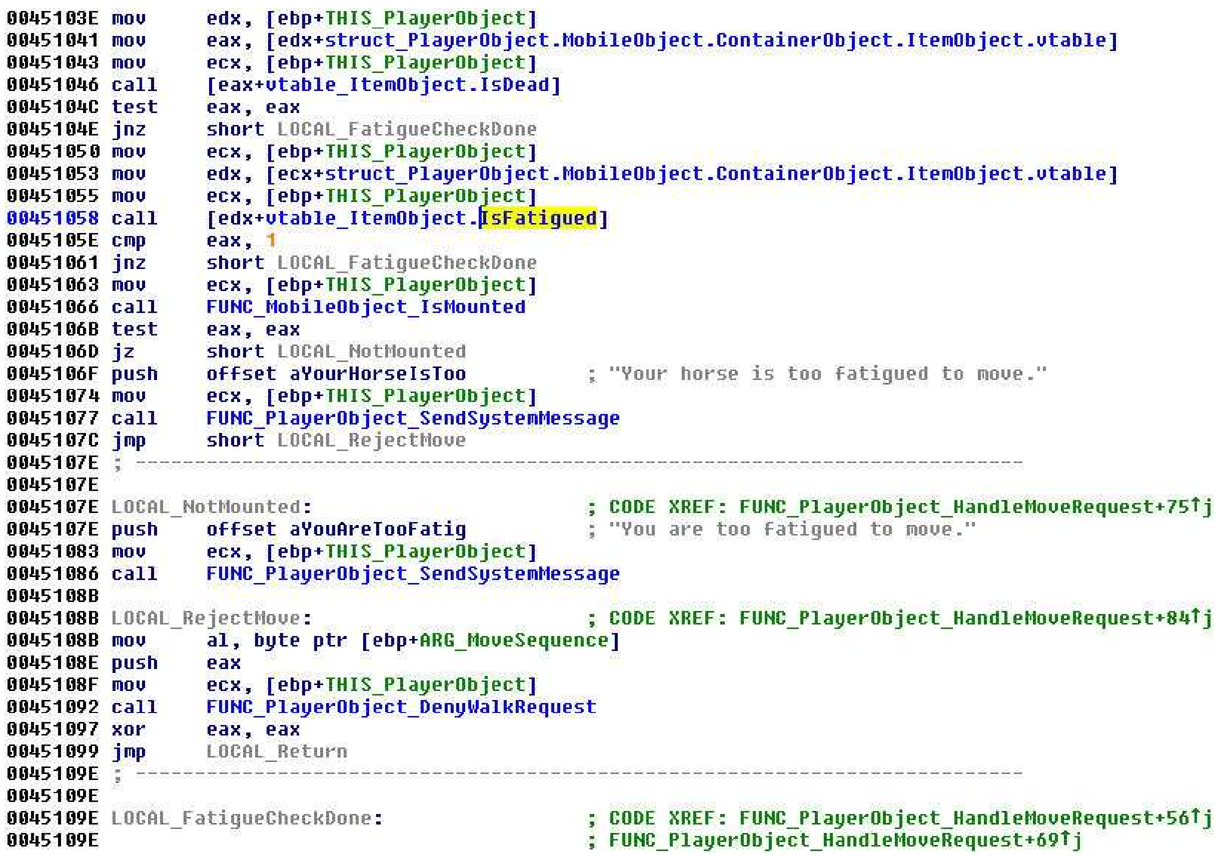


```
0046D746 FUNC_MobileObject_GetModifiedDexterity proc near
0046D746                                         ; DATA X
0046D746                                         ; .rdata
0046D746
0046D746 THIS_MobileObject= dword ptr -8
0046D746 VAR_ReturnValue= dword ptr -4
0046D746
0046D746 push    ebp
0046D747 mov     ebp, esp
0046D749 sub     esp, 8
0046D74C mov     [ebp+THIS_MobileObject], ecx
0046D74F push    1
0046D751 mov     ecx, [ebp+THIS_MobileObject]
0046D754 call    FUNC_MobileObject_GetStat
0046D759 movsx   eax, ax
0046D75C imul    eax, 40
0046D75F cdq
0046D760 mov     ecx, 100
0046D765 idiv    ecx
0046D767 add     eax, 35
0046D76A mov     [ebp+VAR_ReturnValue], eax
0046D76D cmp     [ebp+VAR_ReturnValue], 163
0046D774 jle     short LOCAL_Return
0046D776 mov     [ebp+VAR_ReturnValue], 163
0046D77D
0046D77D LOCAL_Return:                           ; CODE X
0046D77D mov     eax, [ebp+VAR_ReturnValue]
0046D780 mov     esp, ebp
0046D782 pop     ebp
0046D783 retn
0046D783 FUNC_MobileObject_GetModifiedDexterity endp
```

```
⇨ int ReturnValue = int(int(this->GetStat(1)) * 40) / 100 + 35;
  if(ReturnValue > 163)
    ReturnValue = 163;
  return ReturnValue;
```

NOTE: GetStat(1) will return the dexterity + its modifiers (like spell effects)

3) The third step inside the Move Packet handler (Move Request) is checking the player's fatigue level. If the player (or his mount) is too fatigued, then the move will be denied! The code:

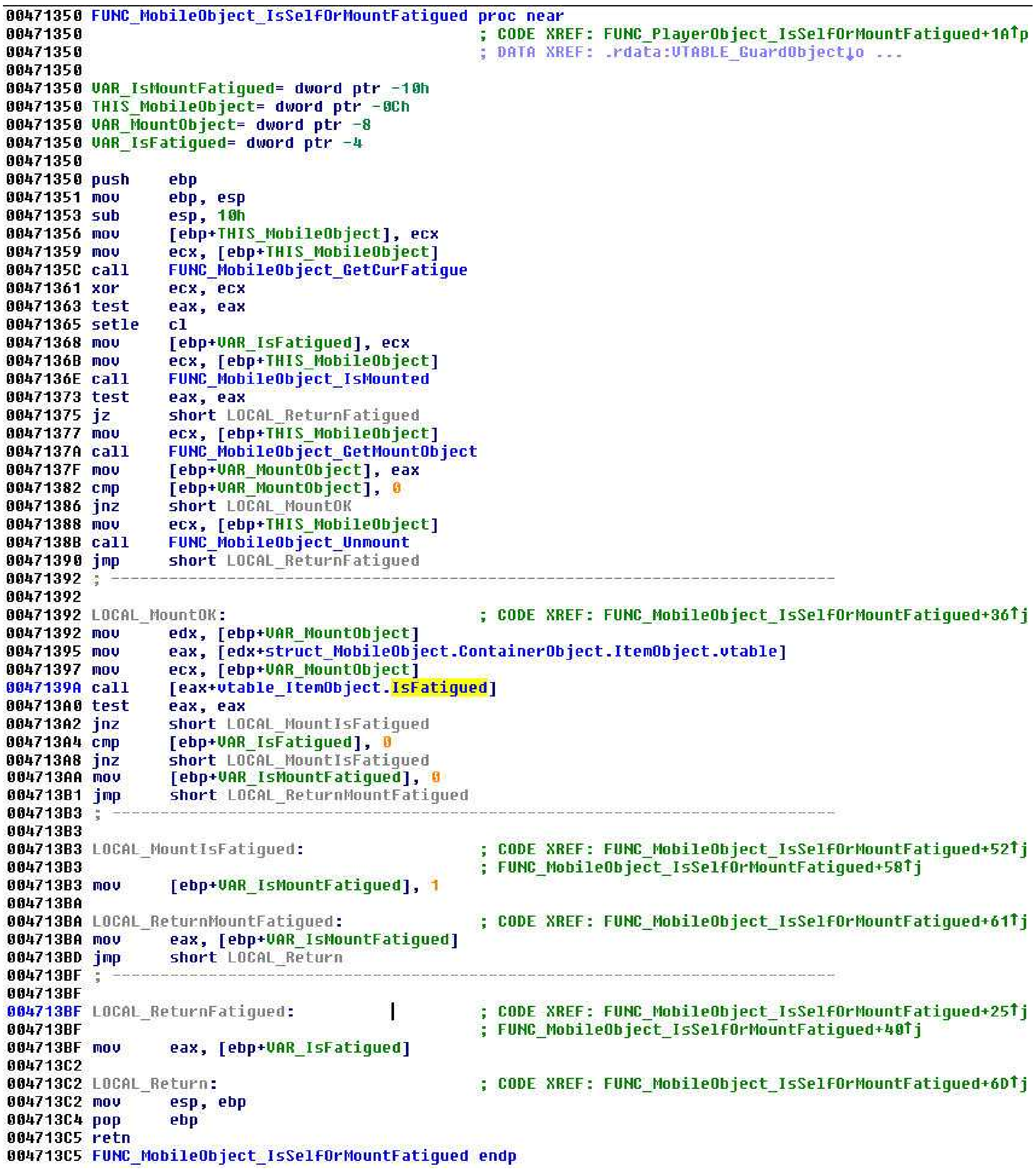


```
0045103E mov     edx, [ebp+THIS_PlayerObject]
00451041 mov     eax, [edx+struct_PlayerObject.MobileObject.ContainerObject.ItemObject.vtable]
00451043 mov     ecx, [ebp+THIS_PlayerObject]
00451046 call    [eax+vtable_ItemObject.IsDead]
0045104C test    eax, eax
0045104E jnz     short LOCAL_FatigueCheckDone
00451050 mov     ecx, [ebp+THIS_PlayerObject]
00451053 mov     edx, [ecx+struct_PlayerObject.MobileObject.ContainerObject.ItemObject.vtable]
00451055 mov     ecx, [ebp+THIS_PlayerObject]
00451058 call    [edx+vtable_ItemObject.IsFatigued]
0045105E cmp     eax, 1
00451061 jnz     short LOCAL_FatigueCheckDone
00451063 mov     ecx, [ebp+THIS_PlayerObject]
00451066 call    FUNC_MobileObject_IsMounted
0045106B test    eax, eax
0045106D jz      short LOCAL_NotMounted
0045106F push    offset aYourHorseIsToo         ; "Your horse is too fatigued to move."
00451074 mov     ecx, [ebp+THIS_PlayerObject]
00451077 call    FUNC_PlayerObject_SendSystemMessage
0045107C jmp     short LOCAL_RejectMove
0045107E ; ---------------------------------------------------------------------------
0045107E
0045107E LOCAL_NotMounted:                       ; CODE XREF: FUNC_PlayerObject_HandleMoveRequest+75↑j
0045107E push    offset aYouAreTooFatig         ; "You are too fatigued to move."
00451083 mov     ecx, [ebp+THIS_PlayerObject]
00451086 call    FUNC_PlayerObject_SendSystemMessage
0045108B
0045108B LOCAL_RejectMove:                       ; CODE XREF: FUNC_PlayerObject_HandleMoveRequest+84↑j
0045108B mov     al, byte ptr [ebp+ARG_MoveSequence]
0045108E push    eax
0045108F mov     ecx, [ebp+THIS_PlayerObject]
00451092 call    FUNC_PlayerObject_DenyWalkRequest
00451097 xor     eax, eax
00451099 jmp     LOCAL_Return
0045109E ; ---------------------------------------------------------------------------
0045109E
0045109E LOCAL_FatigueCheckDone:                 ; CODE XREF: FUNC_PlayerObject_HandleMoveRequest+56↑j
0045109E                                         ; FUNC_PlayerObject_HandleMoveRequest+69↑j
```

```
⇨ if( ! this->IsDead() )
  {
    if( this->IsFatigued() )
    {
      if( this->IsMounted() )
        this->SendSystemMessage
             ("Your horse is too fatigued to move.");
      else
        this->SendSystemMessage
             ("You are too fatigued to move.");
      this->DenyWalkRequest(); // or DenyMoveRequest, whatever...
    }
  }
```

NOTE: this code also includes a bug, if the player is fatigued but the horse (mount) is not then he/she'll still see the message "Your horse is too fatigued to move."!

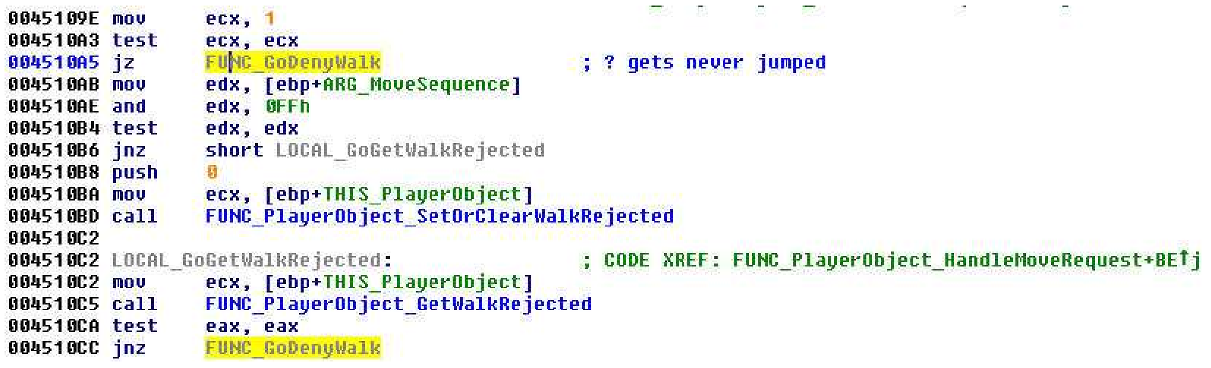
To make the picture complete you will also need the IsFatigued virtual member function. The IsFatigued function actually checks if the mobile itself is fatigued or its mount. Mobiles include players and NPCs.



```
00471350 FUNC_MobileObject_IsSelfOrMountFatigued proc near
00471350                                         ; CODE XREF: FUNC_PlayerObject_IsSelfOrMountFatigued+1A↑p
00471350                                         ; DATA XREF: .rdata:VTABLE_GuardObject↓o ...
00471350
00471350 VAR_IsMountFatigued= dword ptr -10h
00471350 THIS_MobileObject= dword ptr -0Ch
00471350 VAR_MountObject= dword ptr -8
00471350 VAR_IsFatigued= dword ptr -4
00471350
00471350 push    ebp
00471351 mov     ebp, esp
00471353 sub     esp, 10h
00471356 mov     [ebp+THIS_MobileObject], ecx
00471359 mov     ecx, [ebp+THIS_MobileObject]
0047135C call    FUNC_MobileObject_GetCurFatigue
00471361 xor     ecx, ecx
00471363 test    eax, eax
00471365 setle   cl
00471368 mov     [ebp+VAR_IsFatigued], ecx
0047136B mov     ecx, [ebp+THIS_MobileObject]
0047136E call    FUNC_MobileObject_IsMounted
00471373 test    eax, eax
00471375 jz      short LOCAL_ReturnFatigued
00471377 mov     ecx, [ebp+THIS_MobileObject]
0047137A call    FUNC_MobileObject_GetMountObject
0047137F mov     [ebp+VAR_MountObject], eax
00471382 cmp     [ebp+VAR_MountObject], 0
00471386 jnz     short LOCAL_MountOK
00471388 mov     ecx, [ebp+THIS_MobileObject]
0047138B call    FUNC_MobileObject_Unmount
00471390 jmp     short LOCAL_ReturnFatigued
00471392 ; ---------------------------------------------------------------------------
00471392
00471392 LOCAL_MountOK:                          ; CODE XREF: FUNC_MobileObject_IsSelfOrMountFatigued+36↑j
00471392 mov     edx, [ebp+VAR_MountObject]
00471395 mov     eax, [edx+struct_MobileObject.ContainerObject.ItemObject.vtable]
00471397 mov     ecx, [ebp+VAR_MountObject]
0047139A call    [eax+vtable_ItemObject.IsFatigued]
004713A0 test    eax, eax
004713A2 jnz     short LOCAL_MountIsFatigued
004713A4 cmp     [ebp+VAR_IsFatigued], 0
004713A8 jnz     short LOCAL_MountIsFatigued
004713AA mov     [ebp+VAR_IsMountFatigued], 0
004713B1 jmp     short LOCAL_ReturnMountFatigued
004713B3 ; ---------------------------------------------------------------------------
004713B3
004713B3 LOCAL_MountIsFatigued:                  ; CODE XREF: FUNC_MobileObject_IsSelfOrMountFatigued+52↑j
004713B3                                         ; FUNC_MobileObject_IsSelfOrMountFatigued+58↑j
004713B3 mov     [ebp+VAR_IsMountFatigued], 1
004713BA
004713BA LOCAL_ReturnMountFatigued:              ; CODE XREF: FUNC_MobileObject_IsSelfOrMountFatigued+61↑j
004713BA mov     eax, [ebp+VAR_IsMountFatigued]
004713BD jmp     short LOCAL_Return
004713BF ; ---------------------------------------------------------------------------
004713BF
004713BF LOCAL_ReturnFatigued:                   ; CODE XREF: FUNC_MobileObject_IsSelfOrMountFatigued+25↑j
004713BF                                         ; FUNC_MobileObject_IsSelfOrMountFatigued+40↑j
004713BF mov     eax, [ebp+VAR_IsFatigued]
004713C2
004713C2 LOCAL_Return:                           ; CODE XREF: FUNC_MobileObject_IsSelfOrMountFatigued+6D↑j
004713C2 mov     esp, ebp
004713C4 pop     ebp
004713C5 retn
004713C5 FUNC_MobileObject_IsSelfOrMountFatigued endp
```

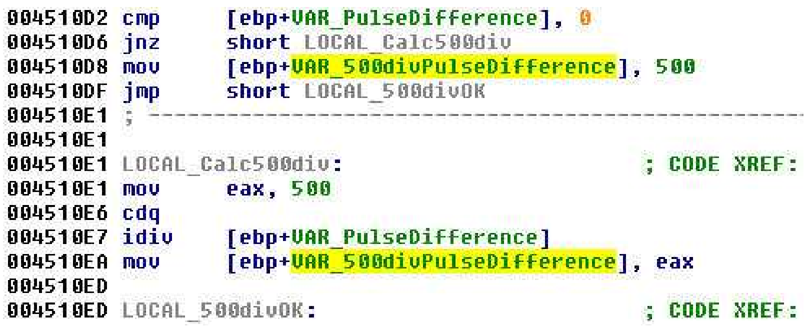
⇨
```
int IsFatigued = this->GetCurFatigue() <= 0;
if( this->IsMounted() )
{
  MOBILE *Mount = this->GetMountObject();
  if(Mount != NULL)
  {
    int IsMountFatigued;
    if( Mount->IsFatigued() || IsFatigued  )
      IsMountFatigued = 1;
    else
      IsMountFatigued = 0;
    return IsMountFatigued;
  }
  this->Unmount(); // Should never occur!
}
return IsFatigued;
```

4) The fourth part I'm not going to pay much attention into. It is provided 'as is' for now:



```
0045109E mov     ecx, 1
004510A3 test    ecx, ecx
004510A5 jz      FUNC_GoDenyWalk                 ; ? gets never jumped
004510AB mov     edx, [ebp+ARG_MoveSequence]
004510AE and     edx, 0FFh
004510B4 test    edx, edx
004510B6 jnz     short LOCAL_GoGetWalkRejected
004510B8 push    0
004510BA mov     ecx, [ebp+THIS_PlayerObject]
004510BD call    FUNC_PlayerObject_SetOrClearWalkRejected
004510C2
004510C2 LOCAL_GoGetWalkRejected:                ; CODE XREF: FUNC_PlayerObject_HandleMoveRequest+BE↑j
004510C2 mov     ecx, [ebp+THIS_PlayerObject]
004510C5 call    FUNC_PlayerObject_GetWalkRejected
004510CA test    eax, eax
004510CC jnz     FUNC_GoDenyWalk
```

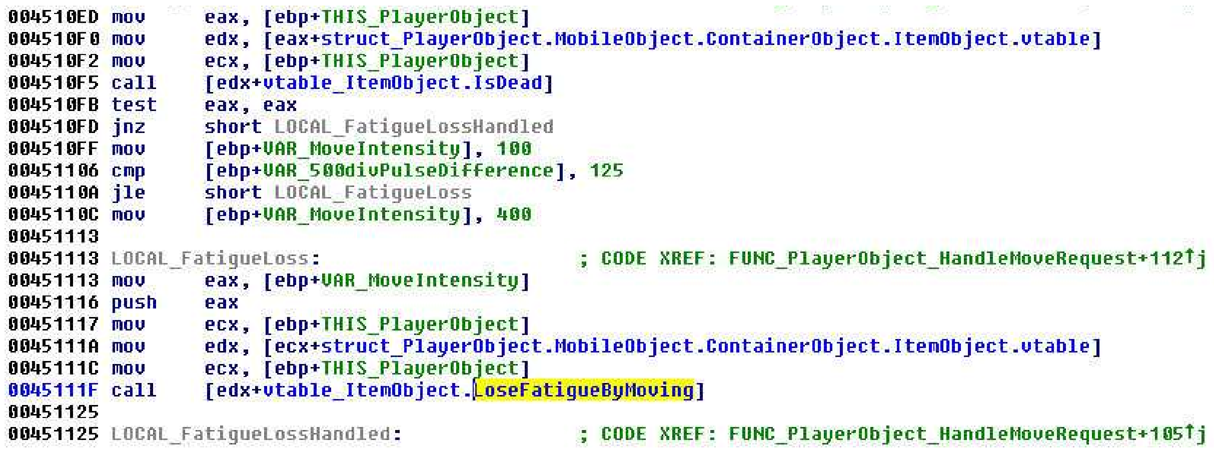
5) The fifth part is much more important for the fatigue loss algorithm:



```
004510D2 cmp     [ebp+VAR_PulseDifference], 0
004510D6 jnz     short LOCAL_Calc500div
004510D8 mov     [ebp+VAR_500divPulseDifference], 500
004510DF jmp     short LOCAL_500divOK
004510E1 ; ---------------------------------------------------------
004510E1
004510E1 LOCAL_Calc500div:                       ; CODE XREF:
004510E1 mov     eax, 500
004510E6 cdq
004510E7 idiv    [ebp+VAR_PulseDifference]
004510EA mov     [ebp+VAR_500divPulseDifference], eax
004510ED
004510ED LOCAL_500divOK:                         ; CODE XREF:
```

```
⇨ int _500divPulseDifference = PulseDifference == 0
                                ? 500
                                : 500 / PulseDifference;
```

6) The sixth part of the algorithm will call the LoseFatigueByMoving member function of the Mobile class (which also applies to players).

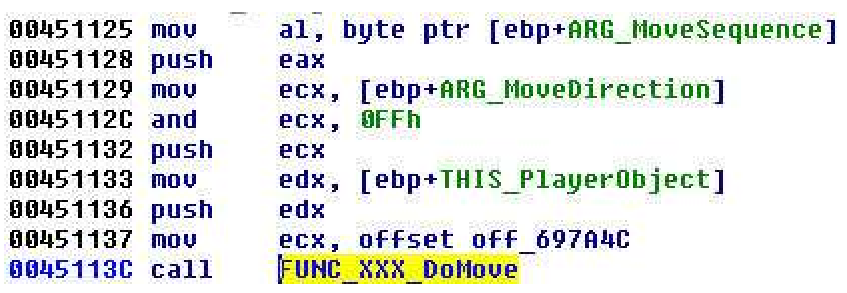


```
004510ED mov     eax, [ebp+THIS_PlayerObject]
004510F0 mov     edx, [eax+struct_PlayerObject.MobileObject.ContainerObject.ItemObject.vtable]
004510F2 mov     ecx, [ebp+THIS_PlayerObject]
004510F5 call    [edx+vtable_ItemObject.IsDead]
004510FB test    eax, eax
004510FD jnz     short LOCAL_FatigueLossHandled
004510FF mov     [ebp+VAR_MoveIntensity], 100
00451106 cmp     [ebp+VAR_500divPulseDifference], 125
0045110A jle     short LOCAL_FatigueLoss
0045110C mov     [ebp+VAR_MoveIntensity], 400
00451113
00451113 LOCAL_FatigueLoss:                      ; CODE XREF: FUNC_PlayerObject_HandleMoveRequest+112↑j
00451113 mov     eax, [ebp+VAR_MoveIntensity]
00451116 push    eax
00451117 mov     ecx, [ebp+THIS_PlayerObject]
0045111A mov     edx, [ecx+struct_PlayerObject.MobileObject.ContainerObject.ItemObject.vtable]
0045111C mov     ecx, [ebp+THIS_PlayerObject]
0045111F call    [edx+vtable_ItemObject.LoseFatigueByMoving]
00451125
00451125 LOCAL_FatigueLossHandled:               ; CODE XREF: FUNC_PlayerObject_HandleMoveRequest+105↑j
```

```
⇨ if( ! this->IsDead() )
  {
    int MoveIntensity = _500divPulseDifference <= 125 ? 100 : 400;
    this->LoseFatigueByMoving(MoveIntensity);
  }
```

NOTE: The Pulse Difference is influenced by the number of packets received.
Therefor: Running: MoveIntensity = 400, Walking : MoveIntensity = 100.

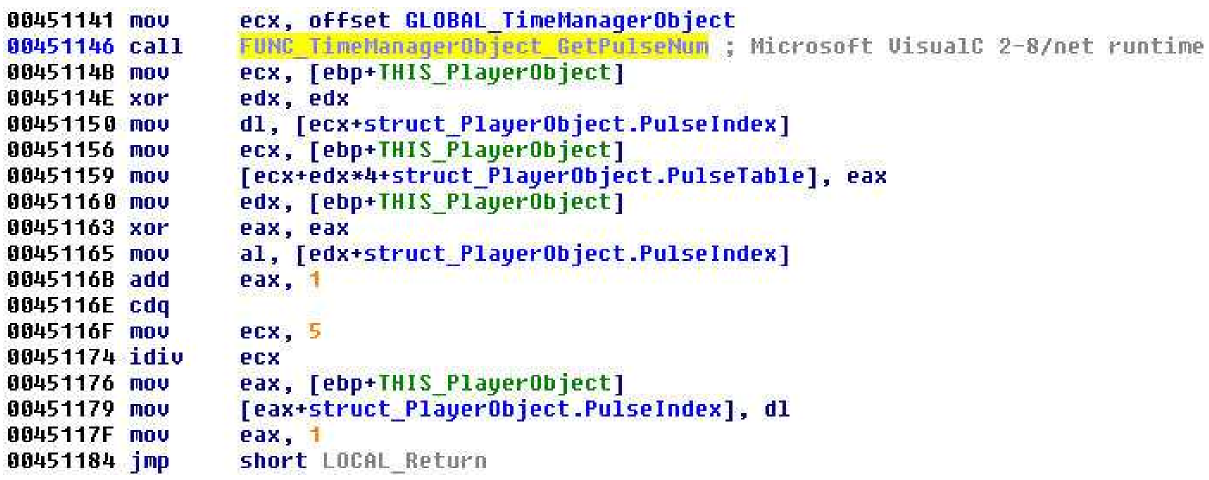
7) The seventh step is the actual moving. If a player bounces against another player or other objects, this goes on here. Changing direction also happens here; this means that turning has great impact on the fatigue of a player, since the fatigue handling is executed BEFORE the direction-changing code:



```
00451125 mov     al, byte ptr [ebp+ARG_MoveSequence]
00451128 push    eax
00451129 mov     ecx, [ebp+ARG_MoveDirection]
0045112C and     ecx, 0FFh
00451132 push    ecx
00451133 mov     edx, [ebp+THIS_PlayerObject]
00451136 push    edx
00451137 mov     ecx, offset off_697A4C
0045113C call    FUNC_XXX_DoMove
```

⇨ `ObjectAt6974AC.DoMove(this, (int) MoveDirection, MoveSequence);`

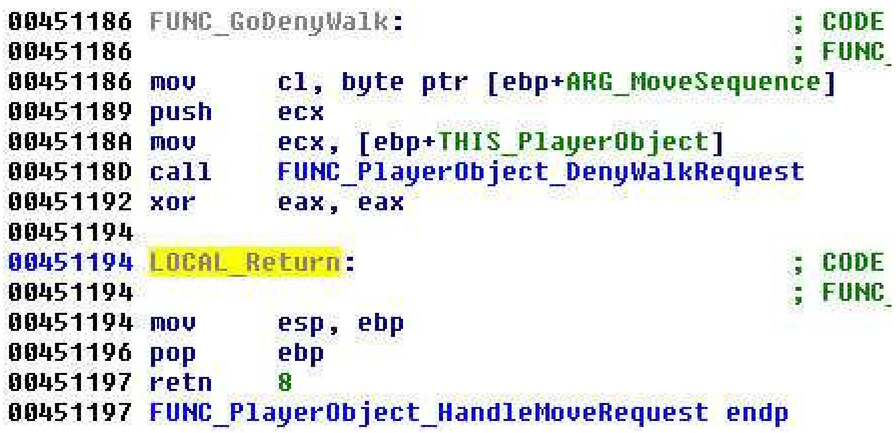
8) The eight step is updating the Pulse Table and increasing/rotating the Pulse Index:



```
00451141 mov     ecx, offset GLOBAL_TimeManagerObject
00451146 call    FUNC_TimeManagerObject_GetPulseNum ; Microsoft VisualC 2-8/net runtime
0045114B mov     ecx, [ebp+THIS_PlayerObject]
0045114E xor     edx, edx
00451150 mov     dl, [ecx+struct_PlayerObject.PulseIndex]
00451156 mov     ecx, [ebp+THIS_PlayerObject]
00451159 mov     [ecx+edx*4+struct_PlayerObject.PulseTable], eax
00451160 mov     edx, [ebp+THIS_PlayerObject]
00451163 xor     eax, eax
00451165 mov     al, [edx+struct_PlayerObject.PulseIndex]
0045116B add     eax, 1
0045116E cdq
0045116F mov     ecx, 5
00451174 idiv    ecx
00451176 mov     eax, [ebp+THIS_PlayerObject]
00451179 mov     [eax+struct_PlayerObject.PulseIndex], dl
0045117F mov     eax, 1
00451184 jmp     short LOCAL_Return
```

⇨
```
this->PulseTable[this->PulseIndex] = TimeManager.GetPulseNum();
this->PulseIndex = (this->PulseIndex + 1) % 5;
```

9) The ninth and last step is sending a Deny Request packet if the move was denied earlier on (see step 3 and 4).

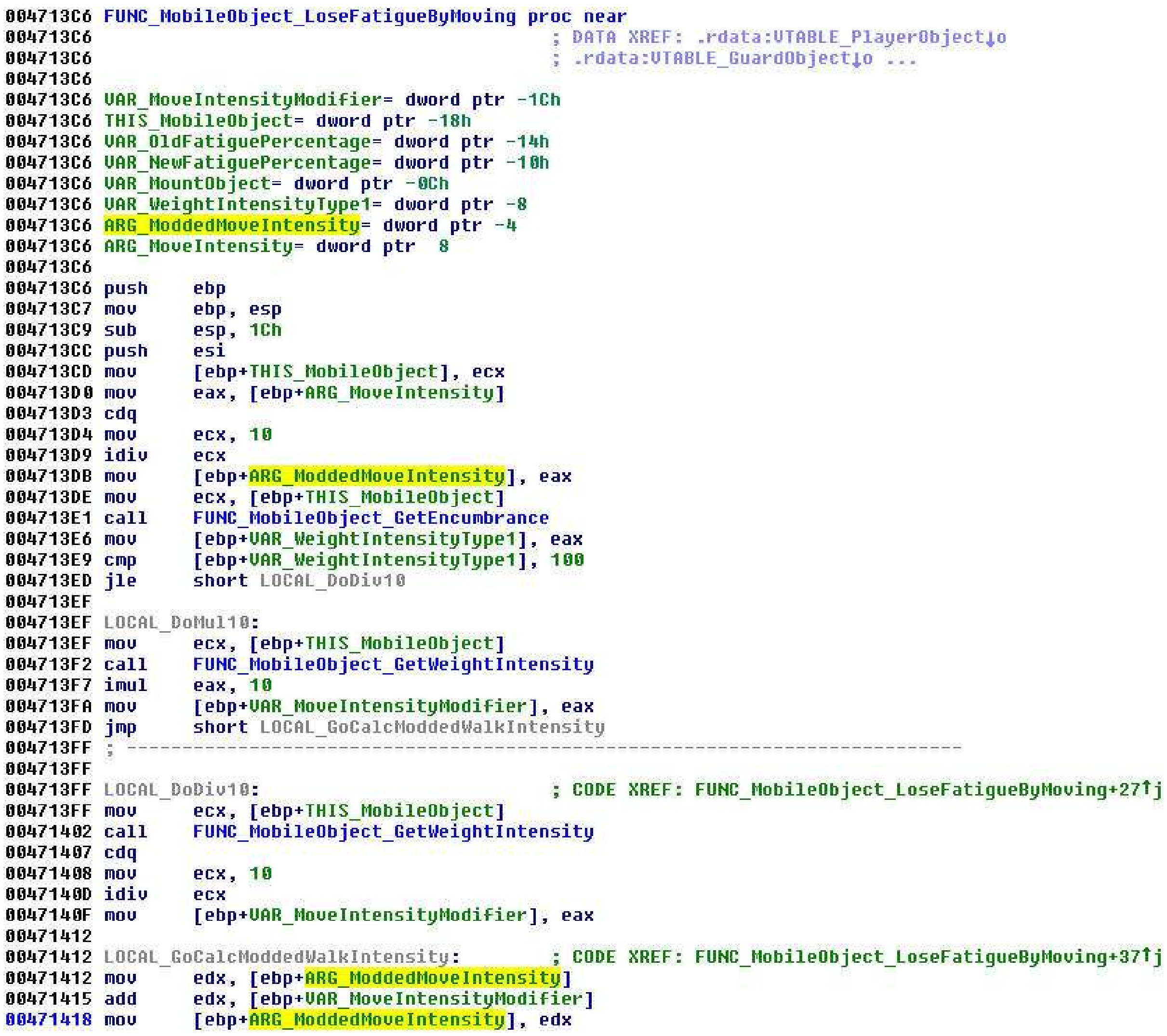


```
00451186 FUNC_GoDenyWalk:                             ; CODE
00451186                                              ; FUNC_
00451186 mov     cl, byte ptr [ebp+ARG_MoveSequence]
00451189 push    ecx
0045118A mov     ecx, [ebp+THIS_PlayerObject]
0045118D call    FUNC_PlayerObject_DenyWalkRequest
00451192 xor     eax, eax
00451194
00451194 LOCAL_Return:                                ; CODE
00451194                                              ; FUNC_
00451194 mov     esp, ebp
00451196 pop     ebp
00451197 retn    8
00451197 FUNC_PlayerObject_HandleMoveRequest endp
```

LOSE FATIGUE BY MOVING

Now let's see what goes on when you lose fatigue by moving.

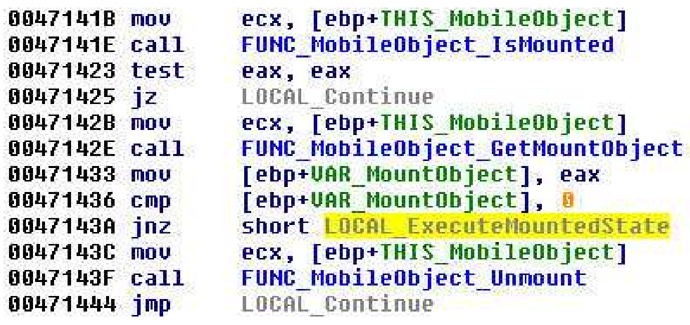
First the MoveIntensity (either 100 or 400) is modified based on the mobile's encumbrance (load percentage). If you are overloaded, the MoveIntensity is modified with the WeightIntensity multiplied by 10, otherwise the MoveIntensity is modified with the WeightIntensity divided by 10. This shows that being overloaded is punished severely.



```
004713C6 FUNC_MobileObject_LoseFatigueByMoving proc near
004713C6                                         ; DATA XREF: .rdata:VTABLE_PlayerObject↓o
004713C6                                         ; .rdata:VTABLE_GuardObject↓o ...
004713C6
004713C6 VAR_MoveIntensityModifier= dword ptr -1Ch
004713C6 THIS_MobileObject= dword ptr -18h
004713C6 VAR_OldFatiguePercentage= dword ptr -14h
004713C6 VAR_NewFatiguePercentage= dword ptr -10h
004713C6 VAR_MountObject= dword ptr -0Ch
004713C6 VAR_WeightIntensityType1= dword ptr -8
004713C6 ARG_ModdedMoveIntensity= dword ptr -4
004713C6 ARG_MoveIntensity= dword ptr  8
004713C6
004713C6 push    ebp
004713C7 mov     ebp, esp
004713C9 sub     esp, 1Ch
004713CC push    esi
004713CD mov     [ebp+THIS_MobileObject], ecx
004713D0 mov     eax, [ebp+ARG_MoveIntensity]
004713D3 cdq
004713D4 mov     ecx, 10
004713D9 idiv    ecx
004713DB mov     [ebp+ARG_ModdedMoveIntensity], eax
004713DE mov     ecx, [ebp+THIS_MobileObject]
004713E1 call    FUNC_MobileObject_GetEncumbrance
004713E6 mov     [ebp+VAR_WeightIntensityType1], eax
004713E9 cmp     [ebp+VAR_WeightIntensityType1], 100
004713ED jle     short LOCAL_DoDiv10
004713EF
004713EF LOCAL_DoMul10:
004713EF mov     ecx, [ebp+THIS_MobileObject]
004713F2 call    FUNC_MobileObject_GetWeightIntensity
004713F7 imul    eax, 10
004713FA mov     [ebp+VAR_MoveIntensityModifier], eax
004713FD jmp     short LOCAL_GoCalcModdedWalkIntensity
004713FF ; ---------------------------------------------------------------------------
004713FF
004713FF LOCAL_DoDiv10:                          ; CODE XREF: FUNC_MobileObject_LoseFatigueByMoving+27↑j
004713FF mov     ecx, [ebp+THIS_MobileObject]
00471402 call    FUNC_MobileObject_GetWeightIntensity
00471407 cdq
00471408 mov     ecx, 10
0047140D idiv    ecx
0047140F mov     [ebp+VAR_MoveIntensityModifier], eax
00471412
00471412 LOCAL_GoCalcModdedWalkIntensity:        ; CODE XREF: FUNC_MobileObject_LoseFatigueByMoving+37↑j
00471412 mov     edx, [ebp+ARG_ModdedMoveIntensity]
00471415 add     edx, [ebp+VAR_MoveIntensityModifier]
00471418 mov     [ebp+ARG_ModdedMoveIntensity], edx
```

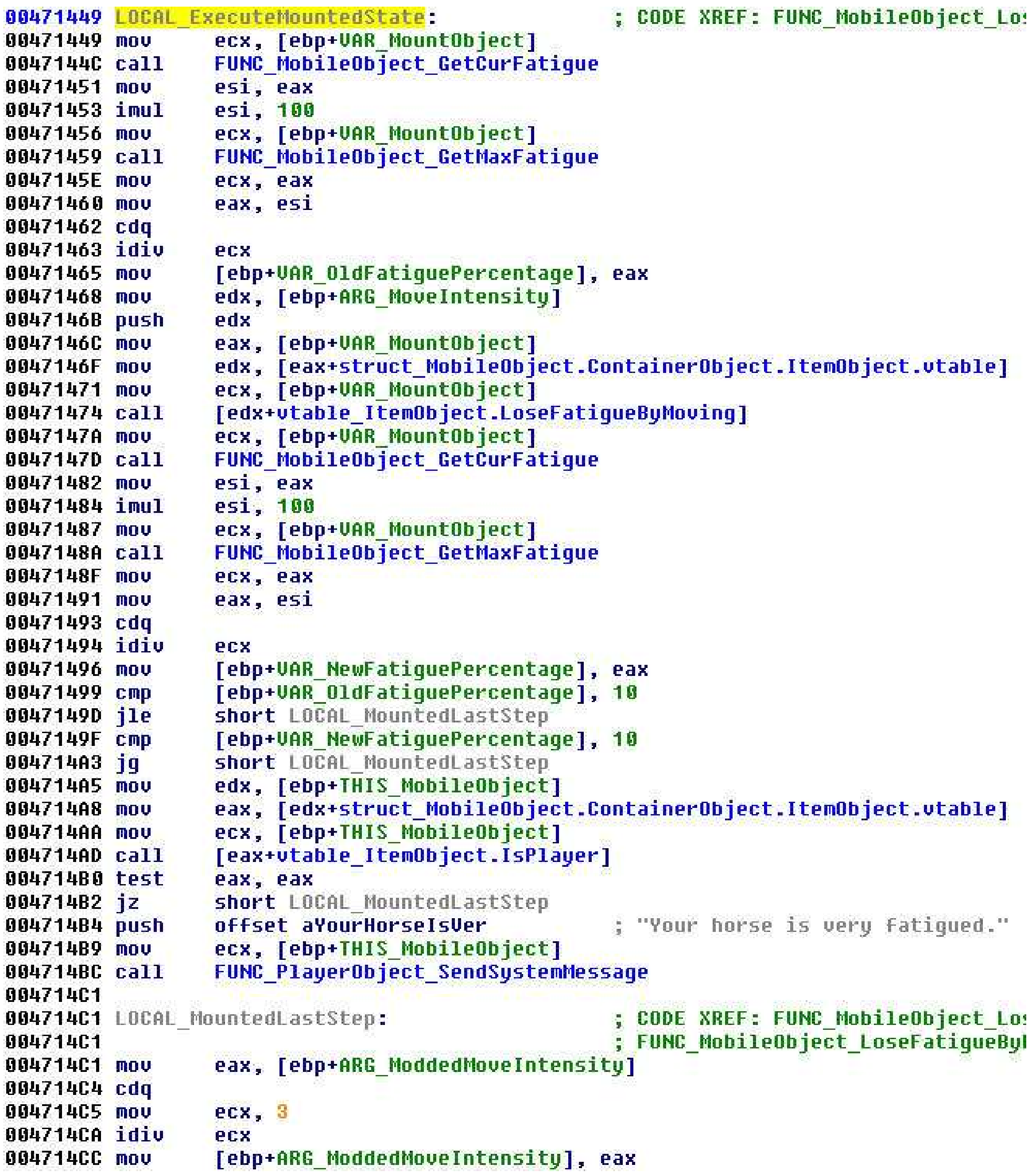
```
⇨ int ModdedMoveIntensity = MoveIntensity / 10;
  int MoveIntensityModifier;
  if( this->GetEncumbrance() <= 100)
    MoveIntensityModifier = this->GetWeightIntensity() * 10;
  Else
    MoveIntensityModifier = this->GetWeightIntensity() / 10;
  ModdedMoveIntensity = ModdedMoveIntensity + MoveIntensityModifier;
```

The code executed next is deciding whether or not you are mounted:



```
0047141B mov     ecx, [ebp+THIS_MobileObject]
0047141E call    FUNC_MobileObject_IsMounted
00471423 test    eax, eax
00471425 jz      LOCAL_Continue
0047142B mov     ecx, [ebp+THIS_MobileObject]
0047142E call    FUNC_MobileObject_GetMountObject
00471433 mov     [ebp+VAR_MountObject], eax
00471436 cmp     [ebp+VAR_MountObject], 0
0047143A jnz     short LOCAL_ExecuteMountedState
0047143C mov     ecx, [ebp+THIS_MobileObject]
0047143F call    FUNC_MobileObject_Unmount
00471444 jmp     LOCAL_Continue
```

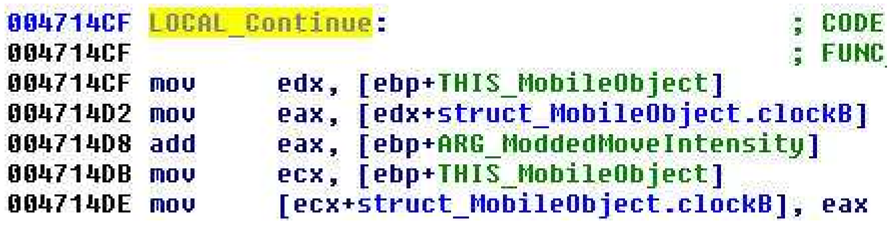
If mounted, the mount's fatigue is handled similar to that of the owner and a warning is given if the level of fatigue (in percentages) dropped below 10. Also notice that the player will still lose fatigue but only based on one third of the original ModdedMoveIntensity:



```
00471449 LOCAL_ExecuteMountedState:                ; CODE XREF: FUNC_MobileObject_Lo
00471449 mov     ecx, [ebp+VAR_MountObject]
0047144C call    FUNC_MobileObject_GetCurFatigue
00471451 mov     esi, eax
00471453 imul    esi, 100
00471456 mov     ecx, [ebp+VAR_MountObject]
00471459 call    FUNC_MobileObject_GetMaxFatigue
0047145E mov     ecx, eax
00471460 mov     eax, esi
00471462 cdq
00471463 idiv    ecx
00471465 mov     [ebp+VAR_OldFatiguePercentage], eax
00471468 mov     edx, [ebp+ARG_MoveIntensity]
0047146B push    edx
0047146C mov     eax, [ebp+VAR_MountObject]
0047146F mov     edx, [eax+struct_MobileObject.ContainerObject.ItemObject.vtable]
00471471 mov     ecx, [ebp+VAR_MountObject]
00471474 call    [edx+vtable_ItemObject.LoseFatigueByMoving]
0047147A mov     ecx, [ebp+VAR_MountObject]
0047147D call    FUNC_MobileObject_GetCurFatigue
00471482 mov     esi, eax
00471484 imul    esi, 100
00471487 mov     ecx, [ebp+VAR_MountObject]
0047148A call    FUNC_MobileObject_GetMaxFatigue
0047148F mov     ecx, eax
00471491 mov     eax, esi
00471493 cdq
00471494 idiv    ecx
00471496 mov     [ebp+VAR_NewFatiguePercentage], eax
00471499 cmp     [ebp+VAR_OldFatiguePercentage], 10
0047149D jle     short LOCAL_MountedLastStep
0047149F cmp     [ebp+VAR_NewFatiguePercentage], 10
004714A3 jg      short LOCAL_MountedLastStep
004714A5 mov     edx, [ebp+THIS_MobileObject]
004714A8 mov     eax, [edx+struct_MobileObject.ContainerObject.ItemObject.vtable]
004714AA mov     ecx, [ebp+THIS_MobileObject]
004714AD call    [eax+vtable_ItemObject.IsPlayer]
004714B0 test    eax, eax
004714B2 jz      short LOCAL_MountedLastStep
004714B4 push    offset aYourHorseIsVer        ; "Your horse is very fatigued."
004714B9 mov     ecx, [ebp+THIS_MobileObject]
004714BC call    FUNC_PlayerObject_SendSystemMessage
004714C1
004714C1 LOCAL_MountedLastStep:                    ; CODE XREF: FUNC_MobileObject_Lo
004714C1                                           ; FUNC_MobileObject_LoseFatigueBy
004714C1 mov     eax, [ebp+ARG_ModdedMoveIntensity]
004714C4 cdq
004714C5 mov     ecx, 3
004714CA idiv    ecx
004714CC mov     [ebp+ARG_ModdedMoveIntensity], eax
```

```
⇨  if( this->IsMounted() )
   {
     MOBILE *Mount = this->GetMountObject();
     if(Mount == NULL)
     {
       this->Unmount();
     }
     else
     {
       int OldFatiguePercentage = int(this->GetCurFatigue() * 100)
                                  / this->GetMaxFatigue();
       Mount->LoseFatigueByMoving(MoveIntensity);
       int NewFatiguePercentage = int(this->GetCurFatigue() * 100)
                                  / this->GetMaxFatigue();
       if( OldFatiguePercentage > 10 && NewFatiguePercentage <= 10 )
         this->SendSystemMessage("Your horse is very fatigued.");
       ModdedMoveIntensity = ModdedMoveIntensity / 3;
     }
   }
```

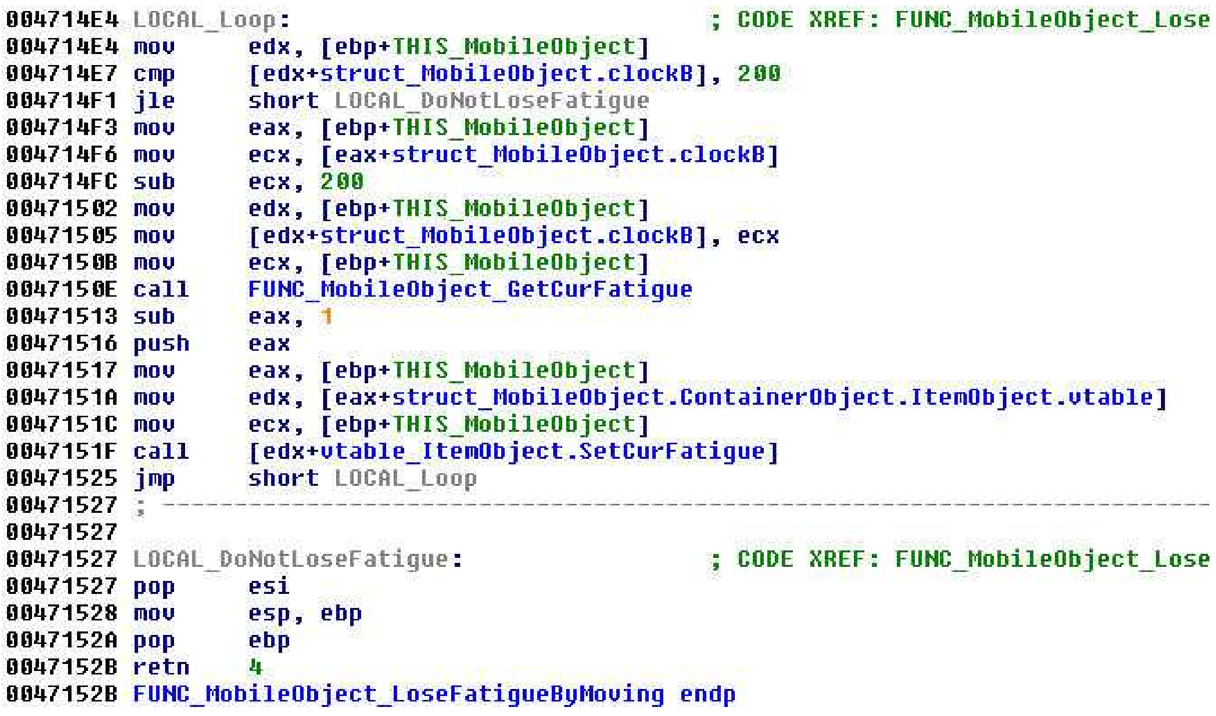
The next step, executed for both mounts and mobiles (players and NPC's), is modifying one of its clocks (namely clockB):



```
004714CF LOCAL_Continue:                          ; CODE
004714CF                                          ; FUNC
004714CF mov     edx, [ebp+THIS_MobileObject]
004714D2 mov     eax, [edx+struct_MobileObject.clockB]
004714D8 add     eax, [ebp+ARG_ModdedMoveIntensity]
004714DB mov     ecx, [ebp+THIS_MobileObject]
004714DE mov     [ecx+struct_MobileObject.clockB], eax
```

```
⇨  this->clockB = this->clockB + ModdedMoveIntensity;
```

Last but not least, the actual lowering of the fatigue based on this clock:



```
004714E4 LOCAL_Loop:                              ; CODE XREF: FUNC_MobileObject_Lose
004714E4 mov     edx, [ebp+THIS_MobileObject]
004714E7 cmp     [edx+struct_MobileObject.clockB], 200
004714F1 jle     short LOCAL_DoNotLoseFatigue
004714F3 mov     eax, [ebp+THIS_MobileObject]
004714F6 mov     ecx, [eax+struct_MobileObject.clockB]
004714FC sub     ecx, 200
00471502 mov     edx, [ebp+THIS_MobileObject]
00471505 mov     [edx+struct_MobileObject.clockB], ecx
0047150B mov     ecx, [ebp+THIS_MobileObject]
0047150E call    FUNC_MobileObject_GetCurFatigue
00471513 sub     eax, 1
00471516 push    eax
00471517 mov     eax, [ebp+THIS_MobileObject]
0047151A mov     edx, [eax+struct_MobileObject.ContainerObject.ItemObject.vtable]
0047151C mov     ecx, [ebp+THIS_MobileObject]
0047151F call    [edx+vtable_ItemObject.SetCurFatigue]
00471525 jmp     short LOCAL_Loop
00471527 ; ---------------------------------------------------------------------------
00471527
00471527 LOCAL_DoNotLoseFatigue:                  ; CODE XREF: FUNC_MobileObject_Lose
00471527 pop     esi
00471528 mov     esp, ebp
0047152A pop     ebp
0047152B retn    4
0047152B FUNC_MobileObject_LoseFatigueByMoving endp
```

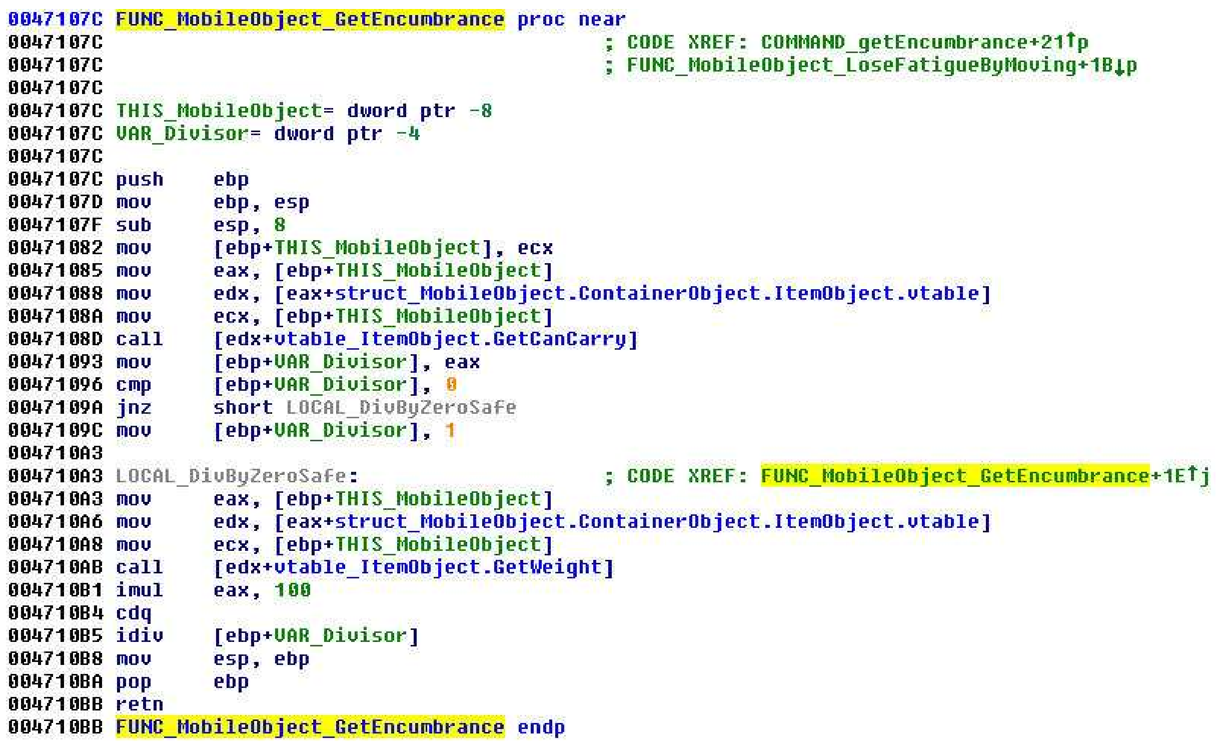
```
⇨ while( this->clockB > 200)
  {
    this->clockB = this->clockB – 200;
    this->SetCurFatigue( this->GetCurFatigue() – 1 );
  }
```

NOTE: SetCurFatigue will correct negative values to zero.

EXTRA FUNCTIONS

Let's look at some other member functions used by the LoseFatigueByMoving member function.

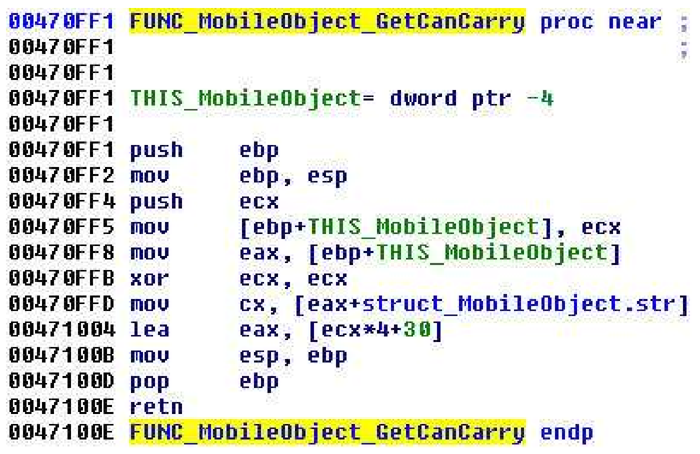
GetEncumbrance:



```
0047107C FUNC_MobileObject_GetEncumbrance proc near
0047107C                                         ; CODE XREF: COMMAND_getEncumbrance+21↑p
0047107C                                         ; FUNC_MobileObject_LoseFatigueByMoving+1B↓p
0047107C
0047107C THIS_MobileObject= dword ptr -8
0047107C VAR_Divisor= dword ptr -4
0047107C
0047107C push    ebp
0047107D mov     ebp, esp
0047107F sub     esp, 8
00471082 mov     [ebp+THIS_MobileObject], ecx
00471085 mov     eax, [ebp+THIS_MobileObject]
00471088 mov     edx, [eax+struct_MobileObject.ContainerObject.ItemObject.vtable]
0047108A mov     ecx, [ebp+THIS_MobileObject]
0047108D call    [edx+vtable_ItemObject.GetCanCarry]
00471093 mov     [ebp+VAR_Divisor], eax
00471096 cmp     [ebp+VAR_Divisor], 0
0047109A jnz     short LOCAL_DivByZeroSafe
0047109C mov     [ebp+VAR_Divisor], 1
004710A3
004710A3 LOCAL_DivByZeroSafe:                    ; CODE XREF: FUNC_MobileObject_GetEncumbrance+1E↑j
004710A3 mov     eax, [ebp+THIS_MobileObject]
004710A6 mov     edx, [eax+struct_MobileObject.ContainerObject.ItemObject.vtable]
004710A8 mov     ecx, [ebp+THIS_MobileObject]
004710AB call    [edx+vtable_ItemObject.GetWeight]
004710B1 imul    eax, 100
004710B4 cdq
004710B5 idiv    [ebp+VAR_Divisor]
004710B8 mov     esp, ebp
004710BA pop     ebp
004710BB retn
004710BB FUNC_MobileObject_GetEncumbrance endp
```

```
⇨ int Divisor = this->GetCanCarry();
  if( Divisor == 0) Divisor = 1;
  return int(this->GetWeight() * 100) / Divisor;
```

GetCanCarry:

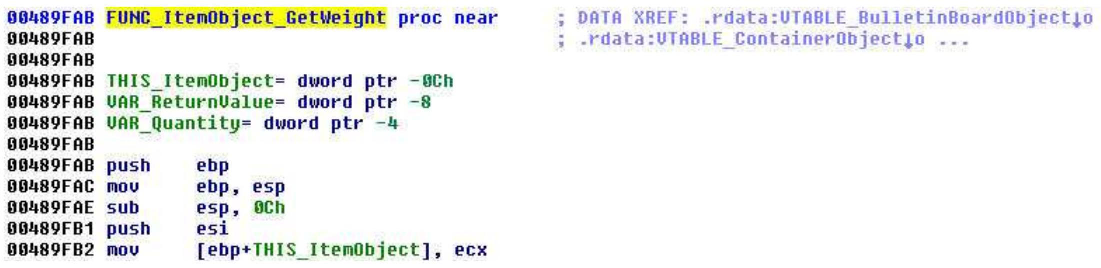


```
00470FF1 FUNC_MobileObject_GetCanCarry proc near ;
00470FF1                                         ;
00470FF1
00470FF1 THIS_MobileObject= dword ptr -4
00470FF1
00470FF1 push    ebp
00470FF2 mov     ebp, esp
00470FF4 push    ecx
00470FF5 mov     [ebp+THIS_MobileObject], ecx
00470FF8 mov     eax, [ebp+THIS_MobileObject]
00470FFB xor     ecx, ecx
00470FFD mov     cx, [eax+struct_MobileObject.str]
00471004 lea     eax, [ecx*4+30]
0047100B mov     esp, ebp
0047100D pop     ebp
0047100E retn
0047100E FUNC_MobileObject_GetCanCarry endp
```

```
⇨ return this->str * 4 + 30;
```

NOTE: the GetCanCarry function changed somewhere early on in the T2A era

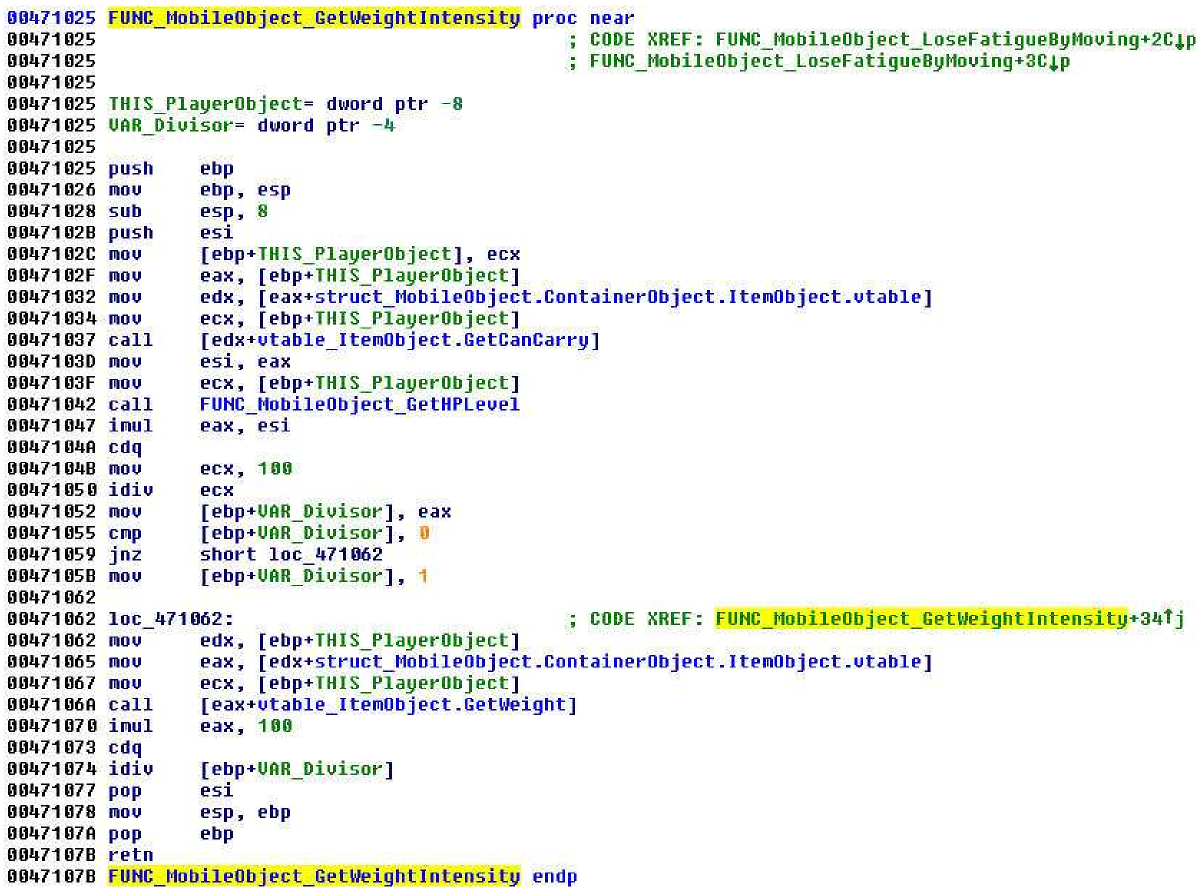
GetWeight:



```
00489FAB FUNC_ItemObject_GetWeight proc near     ; DATA XREF: .rdata:VTABLE_BulletinBoardObject↓o
00489FAB                                         ; .rdata:VTABLE_ContainerObject↓o ...
00489FAB
00489FAB THIS_ItemObject= dword ptr -0Ch
00489FAB VAR_ReturnValue= dword ptr -8
00489FAB VAR_Quantity= dword ptr -4
00489FAB
00489FAB push    ebp
00489FAC mov     ebp, esp
00489FAE sub     esp, 0Ch
00489FB1 push    esi
00489FB2 mov     [ebp+THIS_ItemObject], ecx
```

NOTE: I've only provided a screenshot of the beginning of this function which is inherited by all in-game objects. What's important is: all mobiles have a basic weight which is equal to their MaxHP divided by two. Since horses don't have a container this will be the value returned for them. For other mobiles the weight of all items he/she's carrying will be added.

GetWeightIntensity:



```
00471025 FUNC_MobileObject_GetWeightIntensity proc near
00471025                                         ; CODE XREF: FUNC_MobileObject_LoseFatigueByMoving+2C↓p
00471025                                         ; FUNC_MobileObject_LoseFatigueByMoving+3C↓p
00471025
00471025 THIS_PlayerObject= dword ptr -8
00471025 VAR_Divisor= dword ptr -4
00471025
00471025 push    ebp
00471026 mov     ebp, esp
00471028 sub     esp, 8
0047102B push    esi
0047102C mov     [ebp+THIS_PlayerObject], ecx
0047102F mov     eax, [ebp+THIS_PlayerObject]
00471032 mov     edx, [eax+struct_MobileObject.ContainerObject.ItemObject.vtable]
00471034 mov     ecx, [ebp+THIS_PlayerObject]
00471037 call    [edx+vtable_ItemObject.GetCanCarry]
0047103D mov     esi, eax
0047103F mov     ecx, [ebp+THIS_PlayerObject]
00471042 call    FUNC_MobileObject_GetHPLevel
00471047 imul    eax, esi
0047104A cdq
0047104B mov     ecx, 100
00471050 idiv    ecx
00471052 mov     [ebp+VAR_Divisor], eax
00471055 cmp     [ebp+VAR_Divisor], 0
00471059 jnz     short loc_471062
0047105B mov     [ebp+VAR_Divisor], 1
00471062
00471062 loc_471062:                             ; CODE XREF: FUNC_MobileObject_GetWeightIntensity+34↑j
00471062 mov     edx, [ebp+THIS_PlayerObject]
00471065 mov     eax, [edx+struct_MobileObject.ContainerObject.ItemObject.vtable]
00471067 mov     ecx, [ebp+THIS_PlayerObject]
0047106A call    [eax+vtable_ItemObject.GetWeight]
00471070 imul    eax, 100
00471073 cdq
00471074 idiv    [ebp+VAR_Divisor]
00471077 pop     esi
00471078 mov     esp, ebp
0047107A pop     ebp
0047107B retn
0047107B FUNC_MobileObject_GetWeightIntensity endp
```

```
⇨ int Divisor = int( this->GetCanCarry() * this->GetHPLevel() )
                / 100;
  if( Divisor == 0 )Divisor = 1;
  return (this->GetWeight() * 100) / Divisor;
```

GetHPLevel:

```
⇨ if(this->GetMaxHP() == 0 ) return 0;
  return int(this->GetCurHP() * 100) / this->GetMaxHP();
```