Subject: EncyclopediaMgrClass
Posted by Neijwiert on Wed, 01 Nov 2017 13:08:28 GMT
View Forum Message <> Reply to Message

While working on something else to set my mind of singleplayer code, I needed access to EncyclopediaMgrClass. I found out that you guys do not have the source for that since you are jumping to the function in game2.exe.

So here's the complete code to make EncyclopediaMgrClass work (Commented out some stuff I don't have source for, but you guys do):

vector.h
Toggle Spoiler

```
#ifndef SCRIPTS_INCLUDE__VECTOR_H
#define SCRIPTS_INCLUDE__VECTOR_H

#include "engine_vector.h"

#pragma warning(disable: 4521) // Multiple copy constructors
class BooleanVectorClass
{
 public:
  BooleanVectorClass(int bitCount = 0, const unsigned char *array = 0) : BitCount(bitCount),
LastBitSet(false), LastIndex(-1), Vector()
  {
   this->Vector.Resize((bitCount + 7) >> 3, array);
  }

  BooleanVectorClass(int bitCount, unsigned char *array) : BitCount(bitCount), LastBitSet(false),
LastIndex(-1), Vector()
  {
   this->Vector.Resize((bitCount + 7) >> 3, array);
  }

  BooleanVectorClass(const BooleanVectorClass &other) : Vector(), LastIndex(-1)
  {
   *this = other;
  }

  BooleanVectorClass(BooleanVectorClass &other) : Vector(), LastIndex(-1)
  {
   *this = other;
  }

  BooleanVectorClass &operator=(BooleanVectorClass const &other)
  {
   Fixup(-1);
```

```
 this->LastBitSet = other.LastBitSet;
 this->LastIndex = other.LastIndex;
 this->Vector = other.Vector;
 this->BitCount = other.BitCount;

 return *this;
}

bool operator==(BooleanVectorClass const &other) const
{
 Fixup(this->LastIndex);

 return (this->BitCount == other.BitCount && this->Vector == other.Vector);
}

bool Resize(int bitCount)
{
 bool result = false;

 Fixup(-1);

 if (bitCount)
 {
  int oldBitCount = this->BitCount;

  result = this->Vector.Resize((bitCount + 7) >> 3);

  this->BitCount = bitCount;
  if (result && oldBitCount < bitCount)
  {
   for (; oldBitCount < bitCount; bitCount++)
   {
    if (this->LastIndex != oldBitCount)
    {
     Fixup(oldBitCount);
    }

    this->LastBitSet = false;
   }
  }
 }
 else
 {
  Clear();

  result = true;
 }
```

```
 return result;
}

void Clear()
{
 Fixup(-1);

 this->BitCount = 0;
 this->Vector.Clear();
}

void Reset()
{
 this->LastIndex = -1;

 if (this->Vector.Length() > 0)
 {
  TT_ASSERT(0 < this->Vector.Length());

  unsigned char &vector = this->Vector[0];
  memset(&vector, 0, this->Vector.Length());
 }
}

void Set()
{
 this->LastIndex = -1;

 if (this->Vector.Length() > 0)
 {
  TT_ASSERT(0 < this->Vector.Length());

  unsigned char &vector = this->Vector[0];
  memset(&vector, -1, this->Vector.Length());
 }
}

void Fixup(int bitIndex) const // Yes, this is const
{
 if (bitIndex >= this->BitCount)
 {
  bitIndex = -1;
 }

 if (bitIndex != this->LastIndex)
 {
  if (this->LastIndex != -1)
  {
```

```
   TT_ASSERT(this->LastIndex < this->BitCount);
   TT_ASSERT(0 < this->Vector.Length());

   unsigned char &vector = const_cast<unsigned char &>(this->Vector[0]);
   Set_Bit(&vector, this->LastIndex, this->LastBitSet);
  }
 }

 if (bitIndex != -1)
 {
  TT_ASSERT(bitIndex < this->BitCount);
  TT_ASSERT(0 < this->Vector.Length());

  unsigned char &vector = const_cast<unsigned char &>(this->Vector[0]);
  const_cast<BooleanVectorClass *>(this)->LastBitSet = Get_Bit(&vector, bitIndex);
 }

 const_cast<BooleanVectorClass *>(this)->LastIndex = bitIndex;
}

bool Init(int bitCount, const unsigned char *array = 0)
{
 this->LastBitSet = false;
 this->LastIndex = -1;
 this->BitCount = bitCount;

 return this->Vector.Resize((bitCount + 7) >> 3, array);
}

inline int Get_BitCount() const
{
 return this->BitCount;
}

inline int Get_LastIndex() const
{
 return this->LastIndex;
}

inline bool Get_LastBitSet() const
{
 return this->LastBitSet;
}

inline void Set_LastBitSet(bool set)
{
 this->LastBitSet = set;
}
```

```
 inline const VectorClass<unsigned char> &Get_Vector() const
 {
  return this->Vector;
 }

private:
 static void Set_Bit(unsigned char *vector, int bitIndex, bool value)
 {
  int bitIndex2 = bitIndex;
  if (bitIndex < 0)
  {
   bitIndex2 = bitIndex + 7;
  }

  int bitIndexMask = 1 << (bitIndex - (bitIndex2 & 0xF8));
  if (value)
  {
   vector[bitIndex / 8] |= bitIndexMask;
  }
  else
  {
   vector[bitIndex / 8] &= ~bitIndexMask;
  }
 }

 static bool Get_Bit(const unsigned char *vector, int bitIndex)
 {
  int bitIndex2 = bitIndex;
  if (bitIndex < 0)
  {
   bitIndex2 = bitIndex + 7;
  }

  return ((vector[bitIndex / 8] & (1 << (bitIndex - (bitIndex2 & 0xF8)))) != 0);
 }

 static int First_True_Bit(const unsigned char *vector)
 {
  unsigned char *vectorPtr = const_cast<unsigned char *>(vector);

  int index;
  for (index = 0; !(*vectorPtr); vectorPtr++, index++);

  int bitIndex;
  for (bitIndex = 0; bitIndex <= 7; bitIndex++)
  {
   if (Get_Bit(vectorPtr, bitIndex))
```

```
   {
    break;
   }
  }

  return (bitIndex + sizeof(unsigned char) * index);
 }

 static int First_False_Bit(const unsigned char *vector)
 {
  unsigned char *vectorPtr = const_cast<unsigned char *>(vector);

  int index;
  for (index = 0; (*vectorPtr) == -1; vectorPtr++, index++);

  int bitIndex;
  for (bitIndex = 0; bitIndex <= 7; bitIndex++)
  {
   if (!Get_Bit(vectorPtr, bitIndex))
   {
    break;
   }
  }

  return (bitIndex + sizeof(unsigned char) * index);
 }

 int BitCount;
 bool LastBitSet;
 int LastIndex;
 VectorClass<unsigned char> Vector;
};
#pragma warning(default: 4521) // Multiple copy constructors

#endif // include guard
```

vector.cpp
Toggle Spoiler

```
#include "General.h"
#include "vector.h"
```

encyclopedia.h
Toggle Spoiler

```
#ifndef SCRIPTS_INCLUDE_ENCYCLOPEDIA_H
#define SCRIPTS_INCLUDE_ENCYCLOPEDIA_H
```

```
#include "SaveLoadSubSystemClass.h"
#include "vector.h"

class DamageableGameObj;

class EncyclopediaMgrClass : public SaveLoadSubSystemClass
{
 public:
  enum TYPE
  {
   TYPE_CHARACTER = 0,
   TYPE_WEAPON,
   TYPE_VEHICLE,
   TYPE_BUILDING,
   TYPE_MAX
  };

  static void Initialize();
  static void Shutdown();
  static void Build_Bit_Vector(TYPE type);
  static bool Reveal_Object(TYPE type, int classId);
  static bool Is_Object_Revealed(TYPE type, int classId);
  static void Reveal_Objects(TYPE type);
  static void Reveal_All_Objects();
  static void Hide_Objects(TYPE type);
  static void Hide_All_Objects();
  virtual bool Save(ChunkSaveClass &csave);
  virtual bool Load(ChunkLoadClass &cload);
  static void Load_Variables();
  static bool Reveal_Object(DamageableGameObj *obj);
  static void Display_Event_UI();
  static void Store_Data();
  static void Restore_Data();

  virtual uint32 Chunk_ID() const
  {
   return 0x40148;
  }

  virtual const char *Name() const
  {
   return "EncyclopediaMgrClass";
  }

 private:
  static BooleanVectorClass KnownObjectVector[TYPE_MAX];
  static BooleanVectorClass CopyOfKnownObjectVector[TYPE_MAX];
```

```
};

#endif // include guard
```

encyclopedia.cpp
Toggle Spoiler

```
#include "General.h"
#include "encyclopedia.h"

#include "cGameType.h"
#include "DamageableGameObj.h"
#include "SoldierGameObj.h"
//#include "GlobalSettingsDef.h"

BooleanVectorClass EncyclopediaMgrClass::KnownObjectVector[];
BooleanVectorClass EncyclopediaMgrClass::CopyOfKnownObjectVector[];

void EncyclopediaMgrClass::Initialize()
{
 for (int type = 0; type < TYPE_MAX; type++)
 {
  Build_Bit_Vector(static_cast<TYPE>(type));
 }

 Store_Data();
}

void EncyclopediaMgrClass::Shutdown()
{
 for (int type = 0; type < TYPE_MAX; type++)
 {
  KnownObjectVector[type].Clear();
 }
}

void EncyclopediaMgrClass::Build_Bit_Vector(TYPE type)
{
 static const char *ENCY_INI_FILENAMES[TYPE_MAX] =
 {
  "characters.ini",
  "weapons.ini",
  "vehicles.ini",
  "buildings.ini"
 };

 int highestId = 0;
```

```
 INIClass *ini = Get_INI(ENCY_INI_FILENAMES[type]);
 if (ini)
 {
  List<INISection *> &sectionList = ini->Get_Section_List();
  for (INISection *currentSection = sectionList.First(); currentSection; currentSection =
currentSection->Next())
  {
   int currentId = ini->Get_Int(currentSection->Section, "ID", 0);
   if (currentId >= highestId)
   {
    highestId = currentId;
   }
  }

  if (ini)
  {
   delete ini;
  }
 }

 if (KnownObjectVector[type].Get_BitCount() < highestId + 1)
 {
  KnownObjectVector[type].Resize(highestId + 1);
 }
}

bool EncyclopediaMgrClass::Reveal_Object(TYPE type, int classId)
{
 bool result = false;

 if (cGameType::GameType == 1 && classId < KnownObjectVector[type].Get_BitCount())
 {
  if (KnownObjectVector[type].Get_LastIndex() != classId)
  {
   KnownObjectVector[type].Fixup(classId);
  }

  result = (KnownObjectVector[type].Get_LastBitSet() != true);

  if (KnownObjectVector[type].Get_LastIndex() != classId)
  {
   KnownObjectVector[type].Fixup(classId);
  }

  KnownObjectVector[type].Set_LastBitSet(true);
 }

 return result;
```

```
}

bool EncyclopediaMgrClass::Is_Object_Revealed(TYPE type, int classId)
{
 bool result = false;

 if (classId < KnownObjectVector[type].Get_BitCount())
 {
  if (KnownObjectVector[type].Get_LastIndex() != classId)
  {
   KnownObjectVector[type].Fixup(classId);
  }

  result = KnownObjectVector[type].Get_LastBitSet();
 }

 return result;
}

void EncyclopediaMgrClass::Reveal_Objects(TYPE type)
{
 KnownObjectVector[type].Set();
}

void EncyclopediaMgrClass::Reveal_All_Objects()
{
 for (int type = 0; type < TYPE_MAX; type++)
 {
  KnownObjectVector[type].Set();
 }
}

void EncyclopediaMgrClass::Hide_Objects(TYPE type)
{
 KnownObjectVector[type].Reset();
}

void EncyclopediaMgrClass::Hide_All_Objects()
{
 for (int type = 0; type < TYPE_MAX; type++)
 {
  KnownObjectVector[type].Reset();
 }
}

bool EncyclopediaMgrClass::Save(ChunkSaveClass &csave)
{
 for (int type = 0; type < TYPE_MAX; type++)
```

```
 {
  csave.Begin_Chunk(0x9221215u);

  const VectorClass<unsigned char> &currentVectorClass =
KnownObjectVector[type].Get_Vector();
  if (currentVectorClass.Length() > 0)
  {
   TT_ASSERT(0 < currentVectorClass.Length());

   unsigned char &vector = const_cast<unsigned char &>(currentVectorClass[0]);
   csave.Write(&vector, currentVectorClass.Length());
  }

  csave.End_Chunk();
 }

 csave.Begin_Chunk(0x9221214u);
 csave.End_Chunk();

 return true;
}

bool EncyclopediaMgrClass::Load(ChunkLoadClass &cload)
{
 Hide_All_Objects();

 int type = 0;
 while (cload.Open_Chunk())
 {
  int curChunkId = cload.Cur_Chunk_ID();
  if (curChunkId == 0x9221214u)
  {
   Load_Variables();
  }
  else if (curChunkId == 0x9221215u)
  {
   unsigned long curChunkLength = cload.Cur_Chunk_Length();
   int bitCount = KnownObjectVector[type].Get_LastIndex(); // Yes, it gets LastIndex and not
BitCount

   if (static_cast<int>(curChunkLength) > bitCount)
   {
    bitCount = curChunkLength;
   }

   KnownObjectVector[type].Init(sizeof(unsigned char) * bitCount);

   TT_ASSERT(0 < KnownObjectVector[type].Get_Vector().Length());
```

```
   unsigned char &vector = const_cast<unsigned char
&>(KnownObjectVector[type].Get_Vector()[0]);
   cload.Read(&vector, curChunkLength);

   type++;
  }

  cload.Close_Chunk();
 }

 return true;
}

void EncyclopediaMgrClass::Load_Variables()
{
 // Empty
}

bool EncyclopediaMgrClass::Reveal_Object(DamageableGameObj *obj)
{
 bool result = false;

 if (obj)
 {
  if (cGameType::GameType == 1)
  {
   const DamageableGameObjDef &def = obj->Get_Definition();

   int encyclopediaType = def.Get_Encyclopedia_Type();
   int encyclopediaID = def.Get_Encyclopedia_ID();

   if (encyclopediaType != -1)
   {
    if (!Is_Object_Revealed(static_cast<TYPE>(encyclopediaType), encyclopediaID))
    {
     Display_Event_UI();
    }

    result = Reveal_Object(static_cast<TYPE>(encyclopediaType), encyclopediaID);
   }
  }
 }

 return result;
}

void EncyclopediaMgrClass::Display_Event_UI()
```

```
{
 /*
 Source not available of below code

 int encyclopediaEventStringId = GlobalSettingsDef::GlobalSettings->EncyclopediaEventStringID;
 if (encyclopediaEventStringId)
 {
  SoldierGameObj::Say_Dynamic_Dialogue(encyclopediaEventStringId);
 }
 */
}

void EncyclopediaMgrClass::Store_Data()
{
 for (int type = 0; type < TYPE_MAX; type++)
 {
  CopyOfKnownObjectVector[type] = KnownObjectVector[type];
 }
}

void EncyclopediaMgrClass::Restore_Data()
{
 for (int type = 0; type < TYPE_MAX; type++)
 {
  KnownObjectVector[type] = CopyOfKnownObjectVector[type];
 }
}
```