

Bodaborg som spel

Programmering

Nicklas Larsson
Anders Pousette

Luleå tekniska universitet
Högskoleingenjörsprogrammet
Datorspelutveckling
LTU Skellefteå

THE BORG



Nicklas Larsson & Anders Pousette

Förord

Syftet med denna rapport är att klargöra för utomstående vad vi har gjort under vårt examensarbete. Vi har fokuserat denna slutrapport på de problemområden som funnits under arbetets gång. Vill du ha en bättre helhetsbild av hela examensarbetet, eller mer detaljerad information, titta då i vår planeringsrapport som finns bifogad.

Vår uppgift under examensperioden har varit att utvecklat ett mindre dataspel åt en av Aptera Reklambyråns kunder, Äventyrshuset Bodaborg i Skellefteå.

För ett lyckat genomförande vill vi tacka följande personer/företag som gjort detta möjligt:

Patrik Holmlund, examinator & LTU i Skellefteå, för stöd genom examensarbetet, lån av hårdvara. Per-Olov "Prolle" Wiklund, LTU Adm. för hjälp av frakt. Ett stort tack till Aptera Reklambyrå AB, Skellefteå, Stefan Burman, grundaren till examensuppdraget, support i stort sett allt kring om! Expolaris i Skellefteå, för lån av de fina lokalerna. Äventyrshuset Bodaborg, Skellefteå förstås! Slutligen tack till alla andra som har hjälpt till!

Abstract

The purpose with this project was to see if there was a market for commercial computer games, games that directly promotes the company in question. The work was carried out in Expolaris Center, through an advertising company called Aptera. We were four people working on the crew, two CG-artists and two game programmers. The programming aspect was mostly a matter of putting a lot of parts together and make it work, such as audio, graphics and physics and then program the rooms so they will be entertaining and challenging. We worked with Visual C++ and the Ogre source code, and interfaced it with Newton Game Dynamics, OpenAL and CEGUI.

Sammanfattning

Vi har gjort vårt examensjobb på reklambyrån Aptera. På efterfrågan av Äventyrshuset Bodaborg, en av Apteras kunder, fick vi uppdraget att göra ett litet spel för reklamsyften, som ett sätt att nå ut till konsumenterna på ett underhållande och lite nytänkande sätt. I reklambranschen är det viktigt att man har bra och nyskapande koncept för att folk ska lägga märke till, och ännu viktigare, gilla reklamen. Och reklam finns ju i alla möjliga former, så det kändes som ett naturligt steg att även använda dataspel som ett reklammedium.

Vi var ett team på fyra personer som skulle jobba med projektet, två grafiker och två datorspelstekniker och det skulle pågå under tio veckor. Tanken var att vi skulle få färdigt en spelbar demoversion och eventuellt att man skulle kunna distribuera den som reklam till företag etc. Spelet skulle innehålla en bana, vilken består av tre rum med en knepig uppgift i varje rum, precis som på Bodaborg.

Från början var den stora frågan vilken spelmotor vi skulle använda, men efter många överläggningar bestämde vi oss för att använda grafikmotorn Ogre, vilket fungerade överraskande bra. Tack vare det har hela projektet gått relativt smärtfritt.

Innehållsförteckning

THE BORG	1
1. Introduktion	1
1.1 Problemställning	1
1.2 Bakgrund	1
1.3 Syfte med arbetet.....	1
1.4 Förkortningar.....	1
2. Genomförande/Material och Metoder	2
2.1 Beskrivning av arbetet.....	2
2.2 Arbetsprocess	2
2.3 Ogre	2
2.4 Newton	2
2.5 OpenAL.....	3
2.6 CEGUI.....	3
2.7 Spelmotor	3
2.7.1 Design	3
2.7.2 Beskrivning av states	3
2.7.2.1 GameState	4
2.7.2.2 Loading/Intro	4
2.7.2.3 Huvudmeny	5
2.7.2.4 PlayState.....	5
2.7.2.5 Rum 1 (Öppna luckor)	6
2.7.2.6 Rum 2 (Laser)	6
2.7.2.5 Rum 3 (Kasta boll).....	7
2.8 Objekt.....	8
2.8.1 NABaseObject	8
2.8.2 NAObject	8
2.8.3 NASoundObject	8
2.8.4 NAAnimationObject	9
2.8.5 NATriggerObject	10
2.9 Managers	11
2.9.1 NAObjectManager	11
2.9.2 NAEEventManager	11
2.9.3 NAAnimationManager.....	11
2.9.4 NATriggerManager.....	12
2.9.5 NAAudioManager.....	13
3. Resultat.....	14
4. Diskussion	15
4.1 Slutsatser	15
4.2 Förändringar/Förbättringar.....	15
4.3 Vidareutveckling	16
5. Referenser.....	17
Appendix A – Objektrelationer	18
Appendix B – Översikt över klasser och relationer i spelmotorn	19

1. Introduktion

1.1 Problemställning

Eftersom vi bara har 10 veckor på oss att genomföra examensarbetet så var vi i stort sett tvungna att välja en färdig spelmotor/renderingsmotor. Detta innebär att vi måste spendera en hel del tid bara för att få saker att fungera.

1.2 Bakgrund

Examensarbetet bestod av att tillverka ett dataspel till Apteras kund Äventyrshuset BODABORG, Skellefteå. Spelet är ett mindre spel/demonstrationsspel som visar vad och hur det kan gå till när man besöker Bodaborg på riktigt. Eftersom varken reklamföretaget Aptera eller Bodaborg har haft något tidigare samröre med tekniken och spelbranschen, kommer examensarbetet vara något av ett experiment. Aptera ser detta som ett tillfälle att prova på ett nytt sätt att marknadsföra en kund. Spelet ska fungera som en reklamprodukt.

Framtida distribution av spelet kan vara:

- Köp av CD-skiva på Äventyrshuset Bodaborg
- Reklam CD för utskick till företag

Spelet kommer att innehålla en temabana vilket inkluderar tre olika rum. Dessa tre rum existerar inte i någon äventyrslokal. Rummen finns planerade på riktigt men går ej att lösa tekniskt, därför passar dessa bra att använda i ett spel.

1.3 Syfte med arbetet

Syftet med projektet är dels att locka och uppmuntra folk att besöka Bodaborg, men också för vår egen del att utvecklas och lära oss mer om dataspelsutveckling samt projektarbete. För Apteras del var det också en fråga om att undersöka hur spel kan användas i reklamsyfte.

1.4 Förkortningar

OGRE	Object-oriented Graphics Rendering Engine
OpenAL	Open Audio Library
GUI	Graphical User Interface (Användargränssnitt)
CEGUI	Crazy Eddies GUI (GUI – se ovan)
ODE	Open Dynamics Engine

2. Genomförande/Material och Metoder

2.1 Beskrivning av arbetet

Det första vi gjorde var att besöka Bodaborg tillsammans med Stefan på Aptera, för att diskutera med personalen. Där fick den information och materialet vi behövde för att komma igång med arbetet. Vi fick också tillfälle att prova på olika rum, detta för att få en bättre inblick i hur det fungerar. Under besöket noterade vi även hur lokalerna var utformade i färg, form och funktion.

Examensgruppen samlades tiden därefter till möte för diskussion och val av lämpliga rum till spelet. Valet av spelmotorn Ogre [1] togs efter den övergripande bild vi fick av det kommande arbetet, utifrån detta skapades även en planeringsrapport. Gruppen installerade sig i Apteras lokaler på Expolaris strax före examensperiodens början.

2.2 Arbetsprocess

Vi började med att få Ogre att kompilera, och att sedan skaffa fram de nödvändiga komponenterna vi behövde för att kunna bygga vår spelmotor utifrån Ogre. Eftersom Ogre endast är en renderingsmotor så behöver vi separata delar för att hantera ljud och fysik. Vi valde att använda OpenAL [2] för att hantera ljudet och till fysiken använder vi oss av Newton Game Dynamics [3] samt OgreNewt [4] som är ett skal för Newton till Ogre.

Första veckorna spenderade vi mest tid med att testa de olika sakerna som vi hade tänkt använda samt gå igenom diverse guider för att lära oss mera. Efter det började vi bygga upp vår motor.

När vi fått motorn att fungera på tillfredställande sätt så spenderade vi den sista tiden på att färdigställa de tre rummen vi skulle ha med i vårt speldemo.

2.3 Ogre

Som tidigare nämnts använder vi oss av Ogre för att visa grafik. Ogre är en objektorienterad grafikmotor. Eftersom Ogre är open-source så finns det många eldsjälar som jobbar hårt för att utveckla den. Detta gör att det finns en stor community runt Ogre och det kommer ständigt nya uppdateringar med nya funktioner och finesser.

2.4 Newton

Newton Game Dynamics är ett fysikbibliotek som har det mesta inom fysiksimulering. Det enda vi ville ha från Newton var dock kollisionsdetektion (så att man kan gå omkring på våra banor utan att falla igenom), kastsimulering (så att vi kan kasta bollar) samt kollisionsrespons (så att de olika fysikobjekten påverkar varandra på riktigt sätt).

2.5 OpenAL

För ljud använder vi openAL. Det har mycket bra stöd för det mesta och har 3D-ljud.

2.6 CEGUI

För att sköta interface och pekare är det standard i Ogre att använda sig av CEGUI [5]. CEGUI står för "Crazy Eddies Graphical User Interface" och är ett bibliotek för att bygga upp menyer.

2.7 Spelmotor

2.7.1 Design

Tidigt under projektet så hittade vi en guide för att göra en state-baserad spelmotor med hjälp av Ogre på Ogres wikipedia. Den använde vi som bas för vår motor och har jobbat utifrån den. Den bygger på att man har ett antal singleton [6] states och en game manager som håller reda på vilka som körs med hjälp av en stack [7].

Game managern tar han dom all indata till programmet från mus/tangentbord och skickar det sedan vidare till det state som ligger överst på stacken (d.v.s. det state som körs just nu). Detta ger möjligheten att köra states ovanpå varandra (t.ex. pause ovanpå ett rum utan att avbryta rummet) men det är något vi inte använder oss av i nuvarande implementation.

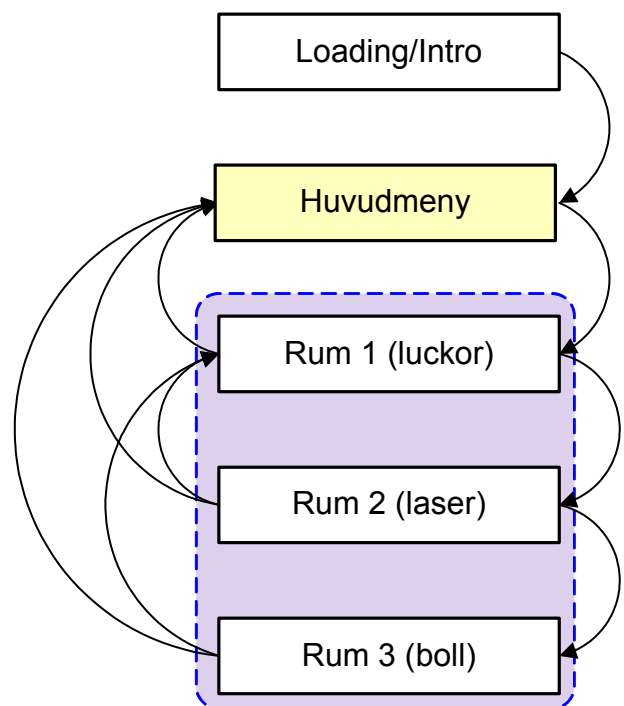


Fig. 2.1 – States och övergångar mellan dessa

2.7.2 Beskrivning av states

Som synes i figur 2.1 så består vårt program av 5 states, Loading/Intro, Huvudmeny, Rum 1, Rum 2 och Rum 3. Pilarna visar hur övergången mellan de olika statsen sker. Applikationen startar i Loading/Intro-state och när det statet har gjort sitt så hoppar programmet automatiskt vidare till huvudmenyn. I huvudmenyn så styrs allt av användaren och inget händer utan att användaren gör något.

De tre rum-staten i den blå fyrkanten motsvarar en bana. När man väljer att starta ett nytt spel så byts menystaten ut mot Rum 1. Alla rum har gemensamt att om man misslyckas med uppgiften så flyttas man tillbaka till rum ett och ett nytt spel

på börjas, samt att man när som helst kan avbryta spelet och gå ut i huvudmenyn. Dessutom så om användaren skulle klara av uppgiften i rummet så flyttas användaren vidare till nästa rum (rum 1 → rum 2 → rum 3) och klaras rum 3 av så hamnar användaren återigen i huvudmenyn (mer om det senare).

Detta är dock inte alla states, vi har även två stycken abstrakta states, GameState och PlayState. GameState är mallen för alla states så att dom ska fungera tillsammans med GameManagern. PlayState ärver från GameState och alla rum måste ärva från PlayState. PlayState innehåller alla funktioner som är gemensamma för alla rum.

2.7.2.1 GameState

GameState
<pre> +enter() +exit() +pause() +resume() +keyClicked(in game : GameManager*, in e : KeyEvent*) +keyPressed(in game : GameManager*, in e : KeyEvent*) +keyReleased(in game : GameManager*, in e : KeyEvent*) +frameStarted(in game : GameManager*, in/ut evt : const FrameEvent) : bool +frameEnded(in game : GameManager*, in/ut evt : const FrameEvent) : bool +mouseMoved(in game : GameManager*, in e : MouseEvent*) +mouseReleased(in game : GameManager*, in e : MouseEvent*) +mousePressed(in game : GameManager*, in e : MouseEvent*) +changeState(in game : GameManager*, in state : GameState*) #GameState() </pre>

GameState är en abstrakt klass som finns där för att GameManager ska kunna hantera alla olika sorters states som ärvs från GameState. Enter och Exit är funktioner som körs när statet läggs till respektive tas bort ur listan på aktiva states i GameManager. Eftersom varje state är en singleton så tas de inte bort när staten byts och detta leder till att konstruktor/dekonstruktor inte direkt används utan man gör motsvarande saker i Enter/Exit i stället. Pause körs när state inte längre är överst på stacken av states och resume körs då state återigen ligger överst.

Alla funktioner med key, frame och mouse i början är funktioner som anropas av gameManagern när det händer någonting så att det state som körs får nya data. Key-funktionerna tar emot knapptryckningar från GameManagern, som i sin tur fått dessa av användaren. Mouse-funktionerna fungerar på samma sätt som key-funktionerna men för musen istället. FrameStarted och frameEnded är events som genereras av Ogre i GameManagern skickar vidare till aktivt state.

2.7.2.2 Loading/Intro

Från början skötte detta rum inläsningen av nödvändig data samt visningen av diverse ”splash screens” med information om spelet. I slutversionen är dock det enda som fortfarande används i Loading/Intro just laddningsbiten, då introduktionsdelen flyttades ut till en för-renderad film istället.

2.7.2.3 Huvudmeny

I huvudmenyn har användaren fem stycken valmöjligheter, Introduktion, Starta Spel, Highscore, Alternativ, Avsluta Spel. Om man klickar på introduktion visas lite information om hur man spelar spelet. Starta spel gör precis som den låter, startar ett nytt spel. Highscore visar highscore-listan, Alternativ visar olika inställningar och avsluta avslutar spelet.



Det enda speciella här är highscore-listan. När man klarat det sista rummet i banan (Rum 3) så kommer man som tidigare nämnts tillbaka till menyn. Om man hade en tid som platsar på highscore-listan så får man få en möjlighet att skriva in sitt namn och listan visas. Om man inte fick en tid som platsar får man ändå se listan så man vet vad man måste slå.

Allt grafik i huvudmenyn styrs av CEGUI som sedan manipulerar Ogre.

2.7.2.4 PlayState

PlayState
+setupPlayer(in startPos : Vector3, in direction : float) +processMovement(in evt : FrameEvent) +processLook(in evt : FrameEvent, in game : GameManager) +processEvents(in game : GameManager) +processEvent() +updateCamera(in evt : FrameEvent) +updateGui(in evt : FrameEvent, in game : GameManager) +switchCursorOnObject() +customGravityForceCallback(in me : Body) +customBallGravityForceCallback(in me : Body) +cleanupGui()

Playstate har alla vanliga funktioner för att spelaren ska kunna interagera med rummen. SetupPlayer måste läggas till i enter-funktionen för varje state som ärver från PlayState. Vad funktionen gör är att den skapar alla nödvändiga variabler och objekt för att kunna spela rummet, så som kollision på spelaren, gui, och ljud.

Alla funktioner med ”process” i början (utom processEvent som är ”pure virtual”, d.v.s. det finns ingen kod i PlayState för den utan det måste läggas till själv i varje state.) måste läggas till i frameStarted för varje playState. ProcessMovement hanterar förflyttning av spelaren i x/z-led och processLook tar hand om vilken riktning spelaren tittar. ProcessEvents är en loop som går igenom event-listan i EventManagern och skickar vidare events dem till processEvent (mer om den senare).

De två update-funktionerna uppdaterar bilden med det nya datat från process, updateCamera flyttar kameran och vänder den åt rätt håll, och updateGui ritar om interfacet med senaste datat.

ProcessEvent är en specialfunktion som skrivs speciellt för varje state som ärver av playstate. Den får som tidigare sagts events från processEvents-loopen, och det den gör sedan är att kolla vad det är för event och vad som ska göras med den. På så sätt så håller man funktionerna i respektive state men kan ändå ha en eventhanterare som är gemensam för alla.

Sedan finns det ett par andra funktioner som har lite blandade uppgifter. De båda forceCallback-funktionerna styr hur mycket gravitation olika objekt har. CleanupGui döljer all Gui som updateGui visar på skärmen. SwitchCursorOnObject byter med hjälp av en RayTrace [8] muspekare/hårkors om spelaren pekar på ett objekt som kan användas.

2.7.2.5 Rum 1 (Öppna luckor)

Rum 1 går ut på att hitta fem rätta luckor av 20 st. Varje två sekunder lyser 5 stycken slumpmässigt valda luckor. Det är endast i luckor som lyser som man kan hitta rätta luckor i. Om man lyckas hitta rätt lucka så kommer spelas ett ljud upp och en lucka i gui visar på att detta har uppdagats. När alla luckor är klara så kommer en ljudsignal att ljuda samt pilen ovanför dörren bli grön. Då är det bara att gå vidare.



2.7.2.6 Rum 2 (Laser)

Rum 2 går ut på att man ska leda en laser till ett mål vid dörren, det finns även en ledtråd i rummet som man kan hitta. För att leda lasern genom rummet måste man använda sig av speglarna på väggen. Dessa går att markera och rotera på med hjälp av musen, när man har markerat en spegel visas två stycken pilar åt vilka håll man kan rotera (höger eller vänster). Om man klickar eller håller in musen ovan någon av dessa kommer spegeln att rotera åt detta håll. När man slutligen lyckats leda laser strålen till målet så kommer en ljudsignal samt att pilen ovanför dörren blir grön berättar att man har klarat rummet. Då kan man gå vidare genom dörren.



2.7.2.5 Rum 3 (Kasta boll)

Rum 3 går ut på att man med hjälp av ledtrådarna från föregående rum lyckas att lista ut vad det "hemliga" ordet är. Om man har tryckt på ledtrådarna, kommer dessa att synas på skärmen. När man väl har ett ord som man tror är rätt så ska man kasta bollarna på respektive bokstäver i ordning och på så sätt bilda ordet. Om man har stavat rätt ord kommer en ljud signal och den gröna pilen ovanför dörren att tala om att man har klarat rummet. Om man får slut bollar eller tiden går ut så misslyckas man med rummet.



2.8 Objekt

2.8.1 NABaseObject

NABaseObj
#m_Name : String #m_Entity : Entity * #m_Node : SceneNode * #m_SceneMgr : SceneManager * #m_Body : Body *
+NABaseObj() +~NABaseObj() +setPosition(in pos : Vector3) +getBody() : Body * +setBody(in world : World*) +setName(in name : const String) +getName() : const String +setEntity(in meshFileName : const String) +getEntity() : Entity * +getNode() : SceneNode * +setSceneManager(in type : SceneType) +createAndAttach() +buildTangentVectors()

En abstrakt klass som ligger till grund för NAOBJECT och NATRIGGER. In kapslar de viktiga variablerna Ogre::Entity [9] och Ogre::SceneNode [10]. Innehåller även ett antal viktiga hjälpfunktioner för att förenkla användandet av objektet. SetBody är en sådan funktion, genom att kalla denna och skicka in en ”newton värld” så skapas automatiskt kollisions mesh av objektet.

2.8.2 NAOBJECT

NAOBJECT
+NAOBJECT() +~NAOBJECT() <u>+create(in name : const String, in meshFileName : const String) : NAOBJECT *</u>

Ärver från NABaseObject, har en static create-funktion implementerad. En aktualiserad version av NABaseObject, används som ett grundobjekt i spelmotorn. De viktigaste funktionerna här är setBody som skapar en kollisions mesh av den inlästa meshen i objektet.

2.8.3 NASoundObject

NASoundObject
-m_Handle : int -m_Name : String -m_Node : SceneNode * -m_gotNode : bool -m_isPlaying : bool -m_isLooped : bool -m_isOgg : bool
+NASoundObject() +~NASoundObject() +create(in name : String, in soundfileName : String, in loop : bool) : NASoundObject * +attachToObject(in node : SceneNode*) +setSourceRelative() +getName() : String +update() +getHandle() : int +play() +stop() +rewind() +pause() +setPosition(in pos : Vector3) +setGain(in gain : float) +isPlaying() : bool +setIsPlaying(in value : bool)

NASoundObject är en klass som har alla nödvändiga funktioner för att hålla koll på ett specifikt ljud d.v.s. ett hjälpmedel för att lätt kunna komma åt och styra ljudbufferten där ljudet ligger. Den har endast ett ”handtag” till bufferten, alltså en variabel med positionen i bufferten. NASoundObject använder sig endast av NAAudioManagers funktioner för att göra det enklare att använda ljud. Alla buffrar och inställningsmöjligheter ligger i NAAudioManager. Med hjälp av NASoundObject kan man enkelt hänga fast ljud på en SceneNode och på så sätt inte behöva bekymra sig om att flytta på objektet själv detta görs med funktionen attachToObject. SetSourceRelative sätter ljudpositionen relative d.v.s. positionen har ingen betydelse, ljudet är alltid lika högt. Används bland annat till musik och gångljud.

2.8.4 NAAAnimationObject

NAAAnimationObject
#m_AnimState : AnimationState * #m_AnimationSpeed : float
+NAAAnimationObject() +~NAAAnimationObject() +create(in name : const String, in meshFileName : const String, in animStateName : const String) : NAAAnimationObject * +loadAnimationState(in animStateName : const String) +getAnimationState() : AnimationState * +setAnimationSpeed(in speed : const float) +getAnimationSpeed() : float

Ärver från NABaseObject, och har några skillnader i utformningen. Innehåller ett AnimationState som laddas vid skapandet av objektet. Detta används för att utföra en skelett-animation.

2.8.5 NATriggerObject

NATriggerObject
<pre>#m_Triggered: bool #m_Action : String #m_Delay : float #m_DefaultDelay : float #m_ActivatorID : MaterialID *</pre>
<pre>+NATriggerObject() +~NATriggerObject() +create(in triggerName : const String, in action : const String, in meshFileName : String, in size : Vector3, in world : World*) : NATriggerObject * +setBody(in world : World*, in size : Vector3) +setAction(in action : String) +getAction() : String +getActivator() : MaterialID * +setActivator(in id : MaterialID*) +setDefaultDelay(in def : float) +getDefaultDelay() : float +setDelay(in delay : float) +getDelay() : float +setTriggered(in triggered : bool) +isTriggered() : bool</pre>

Trigger objektet har en överlagrad setBody funktion, den skapar en ”box kollision” d.v.s. den använder en box som man kan ändra storlek på som kollisionsyta.

2.9 Managers

Alla NAManagers har det gemensamt att de är singleton klasser. Detta gör att det är lätt att få tag på dem genom två funktioner:

- `getSingletonPtr()`
- `getSingleton()`

För att kunna använda dem måste man dock inkludera rätt header-fil och använda scope-operatoren, t.ex.:

```
NAObjectManager::getSingletonPtr()
```

Alla NAManagers skapas vid initieringen av GameManager-klassen och tas bort vid borttagningen av GameManager klassen.

2.9.1 NAObjectManager

NAObjectManager
#m_ObjectList : NAObjectList
+NAObjectManager() +~NAObjectManager() +addObject(in obj : NAObject*) +removeObject(in name : const String) +removeAll() <u>+getSingletonPtr() : NAObjectManager *</u> <u>+getSingleton() : NAObjectManager &</u>

Innehåller en `std::list` av `NAObject`, med hjälp av denna så kan man enkelt hålla koll på att alla objekt tas bort på rätt sätt.

2.9.2 NAEventManager

NAEventManager
#m_EventList : NAEventList
+NAEventManager() +~NAEventManager() +update() +createEvent(in code : String) +addObject(in evt : NAEvent*) +removeAll() +getEvent() : NAEvent * <u>+getSingletonPtr() : NAEventManager *</u> <u>+getSingleton() : NAEventManager &</u>

Innehåller en `std::list` av `NAEvent`, med hjälp av denna så kan man enkelt hålla koll på att alla objekt tas bort på rätt sätt.

2.9.3 NAAnimationManager

NAAAnimationManager
#m_AnimObjects : AnimationObjectList
+NAAAnimationManager() +~NAAAnimationManager() +processAnimations(in/ut evt : const FrameEvent) +addObject(in obj : NAAAnimationObject*) +findObject(in name : String) : NAAAnimationObject * +removeObject(in name : const String) +removeAll() +getSingleton() : NAAAnimationManager & +getSingletonPtr() : NAAAnimationManager *

Innehåller en std::list av NAAAnimationObject, med hjälp av denna så kan man enkelt hålla koll på att alla objekt tas bort på rätt sätt. Man kan även söka rätt på objekt i listan. Har även en processAnimations funktion som varje ”frame” körs i PlayState::framestarted() denna funktion går igenom alla objekt i listan och ”lägger till tid” till animationerna. Detta leder till att animationerna animeras

2.9.4 NATriggerManager

NATriggerManager
#m_TriggerList : NATriggerList
+NATriggerManager() +~NATriggerManager() +addObject(in obj : NATriggerObject*) +removeObject(in name : const String) +findObject(in name : String) : NATriggerObject * +removeAll() +process(in/ut evt : const FrameEvent) +getSingletonPtr() : NATriggerManager* +getSingleton() : NATriggerManager &

Innehåller en std::list av NAAAnimationObject, med hjälp av denna så kan man enkelt hålla koll på att alla objekt tas bort på rätt sätt. Man kan även söka rätt på objekt i listan har även en process() funktion som körs varje ”frame” i PlatSate::framestarted(). Syftet med denna funktion är att gå igenom alla triggers och se om triggeren har blivit ”triggad”, om detta skett så skall en delay tid räknas ned innan triggeren kan återställas. Triggers är synliga i debug.

2.9.5 NAAudioManager

NAAudioManager
<pre>#buffer[256] : ALuint #source[256] : ALuint #nextBuffer : ALuint #nextSource : ALuint #m_Listener : Camera * #m_ListenerNode : SceneNode * #m_SoundObjects : SoundObjectList #relative_Path : String</pre>
<pre>+NAAudioManager() +~NAAudioManager() +init() +setCameraAsListener(in camera : Camera*) +setListenerPosition(in x : float, in y : float, in z : float) +setListenerPosition(in position : Vector3) +setListenerOrientation(in f : Vector3, in u : Vector3) +setListenerOrientation(in fx : float, in fy : float, in fz : float, in ux : float, in uy : float, in uz : float) +loadFile(in filename : String, in loop : bool) : int +play(in handle : int) +stop(in handle : int) +pause(in handle : int) +rewind(in handle : int) +setSourcePosition(in handle : int, in position : Vector3) +setSourcePosition(in handle : int, in x : float, in y : float, in z : float) +setSourceVelocity(in handle : int, in velocity : float*) +setSourceRelative(in handle : int) +setSourceGain(in handle : int, in gain : float) +getListenerGain() : float +setListenerGain(in gain : float) +update() +addObject(in obj : NASoundObject*) +findObject(in name : String) : NASoundObject * +removeObject(in name : const String) +removeAll() +getSingletonPtr() : NAAudioManager * +getSingleton() : NAAudioManager &</pre>

Skiljer sig från de övriga hanterarna.

Använder sig inte av en lista av objekt, utan har en buffert som sedan objekten sitt index till.

NAAudioManager använder sig i sin tur av OpenAL för att skapa och spela upp ljud.

Audio lyssnaren sätts i `PlayState::setupPlayer()` enligt koden nedan:

```
NAAudioManager::getSingleton().setCameraAsListener(mCamera);
```

Detta gör att man får 3d ljud till lyssnarens position.

3. Resultat

3.1 Spelmotorn

Slutresultatet blev en komplett spelmotor, som har det man förväntar sig av en spelmotor, så som kollision, triggers, events, ljud och meny. Eftersom grundjobbet vi gjort på motorn är fristående från själva rummen och ”bodaborg-grejjorna” så är det lätt att återanvända motorn i andra spel.

3.2 ”The Borg”

Speldemot av ”the borg” blev också det klart i tid och enligt specifikation. Vi fick med de tre rummen och en meny som fanns i planeringen och utöver detta också en introfilm och en modell av lobbyn i bakgrunden på menyn som man kunde titta runt i.

4. Diskussion

4.1 Slutsatser

Det är en stor nackdel med korta projekt inom spelprogrammering om projektet kräver en ny/egen-utvecklad spelmotor. Mycket tid måste läggas ner på något som måste vara bra, men det tar tid från utvecklingen av innehåll i spelet. För att undvika detta måste man antingen ha längre projekt eller börja arbetet från en plattform som är komplett från början. Det optimala skulle vara att ha en relativt universell spelmotor som programmerarna kan väldigt bra och sedan bara göra nya spel ovanpå den så man slipper utvecklandet av motorn. Problemet var att för detta projekt fanns ingen sådan.

Våra tidigare erfarenheter byggde på AgentFX [11] som, även om den är mycket bra, kändes lite otrygg på grund av bristen på bra guider samt Torque [12] som även den är en komplett motor. Nackdelen med Torque är att den är väldigt låst i "source"-delen och det mesta manipuleras med script. Ogre verkade vara ett bra val, dels för att det är ett gratisalternativ (open source, sprids under LGPL), dels för att den verkade väldigt omtyckt och slutligen för att man styr det mesta själv och eftersom det är open source så har man full inblick på vad som pågår.

Det är också spännande att jobba med open source, vilken dag som helst kan det komma nya uppdateringar som ger oss nya möjligheter. Till exempel kan nämnas maya-exportern, som hade några skavanker i början, men ungefär halvvägs genom projektet så släpptes en ny exporter som hjälpte både grafikerna och vi programmerare signifikant.

För att hantera fysiken i vårt program valde vi att använda det tidigare nämnda "Newton Game Dynamics"-biblioteket samt OgreNewt för att integrera det i Ogre. Även här fanns andra alternativ (ODE [13], novodex [14]) men Newtons lättanvändhet samt OgreNewt gjorde att det verkade vara det bästa alternativet, och det blev faktiskt mycket bra i spelet också.

4.2 Förändringar/Förbättringar

Tyvärr blev det lite sämre lösningar på vissa saker på grund av den korta tiden vi hade på oss. Antingen blev lösningarna väldigt problemspecifika (d.v.s. icke återanvändbara, så dom måste skrivas om helt om samma sak ska användas på något annat ställe) eller helt enkelt dåliga. Dock är funktionaliteten alltid okej, det är bara koden som är ineffektiv.

En stor förbättring för alla inblandade skulle vara en ny exporter/importer så att grafikerna får större möjligheter att påverka banorna genom att låta dom sätta ut triggers, lampor och objekt och sen i kod läsa in motsvarande så att programmerarna slipper göra det för hand. Under vår korta tidsram fanns det dock ingen möjlighet att utveckla en exporter och en importer.

Man skulle även kunna bygga ut motorn med support för något skriptspråk för att förenkla styrningen av rum och triggers och sånt.

4.3 Vidareutveckling

Eftersom grundstrukturen i vår motor är väl uppbyggd så är det relativt lätt att bygga vidare på programmet genom att lägga till nya rum och banor. Vår kunskap om Ogre har ökat avsevärt så att vi kan göra mer på kortare tid än i början då vi fortfarande höll på att lära oss Ogre-grunderna.

Vi har jobbat mycket med att kapsla in Ogre-bitarna i enklare "NAObject" och managers som sköter hantering av objekten. Detta gör att det går väldigt smidigt att bygga upp scener för oss.

Om man ska fortsätta utveckla rum i stor skala så måste man nog bygga ut exportern åt grafikerna som tidigare nämnts.

5. Referenser

- [1] Streeing, Steve. Object-oriented Graphics Rendering Engine (Ogre) 2001.
<http://www.ogre3d.org>
- [2] Newton Game Dynamics. 2004.
<http://www.newtondynamics.com>
- [3] OpenAL. 2004.
<http://www.openal.org>
- [4] Walaber. OgreNewt 5th release (May 9th). 2004.
<http://www.ogre3d.org/phpBB2/viewtopic.php?t=7857>
- [5] Turner, Paul. Crazy Eddies Graphical User Interface. 2005.
<http://www.cegui.org.uk>
- [6] Gamma, Helm, Johnson, and Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.
- [7] Lafore, Robert. Data Structures & Algorithms in Java (Second Edition). Sams Publishing, 2003.
- [8] Hill, F. S. jr. Computer Graphics Using Open GL (Second Edition). Prentice Hall, 2001.
- [9] Ogre API Reference. Ogre::Entity Class Reference. 2005
http://www.ogre3d.org/docs/api/html/classOgre_1_1Entity.html
- [10] Ogre API Reference. Ogre::SceneNode Class Reference. 2005
http://www.ogre3d.org/docs/api/html/classOgre_1_1SceneNode.html
- [11] Agency 9 AB. AgentFX 3D-engine. 2004
<http://www.agency9.com/products/agentfx/>
- [12] Garage Games. Torque Gaming Engine. 2003
<http://www.garagegames.com/mg/projects/tge/>
- [13] Smith, Russell. Open Dynamics Engine. 2002
<http://ode.org/>
- [14] AGEIA™ Technologies Inc. NovodeX. 2005
<http://www.ageia.com/novodex.html>

Appendix B – Översikt över klasser och relationer i spelmotorn

