An Interaction Design Project, spring 2009
Chalmers University of Technology

Richard Fredriksson
Robert Krantz
Olof Millberg
Mikael Nilsson
Fredrik Wendt

Abstract

This report present the design and development process of Iridium: Twenty Million Leagues Above the Sea. Iridium is a single-player ship simulation game set in space in an alternate universe where steam power is the primary source of energy. The main features include a self-maintained AI, multiple missions and combat. The user controls only the captain and has no direct control over the ship; instead he must rely on the crew and issue the correct orders to survive. Iridium was developed with Microsoft's XNA framework as part of an Interaction Design project at Chalmers University of Technology.

Den här rapporten presenterar design- och utvecklingsprocessen av Iridium: Twenty Million Leagues Above the Sea. Iridium är ett single-player spel som simulerar livet ombord ett rymdskepp. Spelet utspelar sig i en alternativ verklighet där ångmotorer är den primära energikällan. Spelet innehåller en avancerad Artificiell Intelligens, flera uppdrag och strider. Spelaren styr endast kaptenen och har ingen direkt kontroll över skeppet; istället måste han förlita sig på besättningen och ge rätt order för att överleva. Iridium utvecklades med Microsofts XNA-platform som en del av ett projekt i Interaktionsdesign på Chalmers Tekniska Högskola.

Contents

Introduction

The aim of this project was to, after a predetermined theme, plan, create and implement a game in a steampunk inspired visual style. This report outlines the goals, techniques and methods used during the development process of the game Iridium.

The game Iridium is a single-player ship simulation set in space in a alternate universe where steam power is the primary source of energy. The main features include a self-maintained AI, multiple missions and combat. The user controls only the captain and has no direct control over the ship, instead he must rely on his fellow shipmates in order to control the ship, the captain can besides controlling himself, only give orders to the crew.

The project was also a source of experience for everyone involved and gave insight into how a larger game project can be created and what it takes for a game to be realized. At the end of the project the game was showcased at an exhibition at which it received great response, both good and bad.

Background

Using personalities in games is an interesting subject. In books and movies the characters and their relationships and interactions are probably the most important part and in order to be convincing they all need their own fleshed out personalities. Games open up new possibilities with regards to interaction between the player and the non-player characters. In most games though, any characters other than the player only serve to drive the story forward and any interaction tend to be scripted.

Non-player characters are a big part of role playing games and are becoming more important in first person shooters thanks to games like Half-Life. The dialogues and interactions are often very intricate and interesting but they are all scripted and most of the time the characters are controlled by the same artificial intelligence so it is difficult for the player to get any feelings for them outside of the dialogues and cut-scenes. Implementing dynamic real-time dialogues is an interesting research area that deserves to be explored in more depth but it will not be covered by this report.

A few game developers have taken the opposite approach. In Dungeon Keeper for example each type of minion has their own personality and some types get along better with others while some cannot stand each other. Another example is “The Sims” where each character has its own personality traits and behaves accordingly. In both of these games however, dialogues are non-existent. What keeps the players interested is managing the relationships of the characters and making sure they get along as well as possible.

Concept

The project was started by forming a concept on which to base the game. This concept was then formalized by creating a set of goals for the project.

Project concept

The idea of the project was to make the original constraint "Interactive Interacting Personalities" presented at the start of the course into a form of gameplay. The group discussed and talked much about games and titles that we liked and that might fit the description of "Interactive-Interacting Personalities". The group quickly decided that the game should be in situated in space, mainly due to the fact that there are less content in space and therefore appropriate considering the time aspects of the project.

With a setting for the game, three ideas were presented by the members of the group. The three ideas were: a space flying shooter, a “The Sims”-like game situated on a space ship and a real-time strategy game with focus in interplanetary relations. All these options had different game styles and gameplay opportunities as well as drawbacks. The description of each concept can be found in Appendix B. Through voting the group selected the second idea, to make a “The Sims”-like game situated in space. The decision fell to this option mainly since the concept felt original. It also suits the project theme as it contains characters, each with a unique personality, interacting with each other. The concept’s one-sentence-pitch is:

> *- As Captain of a starship you seek your destiny in the galaxy while commanding and managing your crew.*

For a more detailed concept description, see the design document.

Main goals of the project

With less than six months of development time, there was a limited amount of time to fully realize the game concept. All the group members had personal goals with the project; however the main goals were the same for all participants.

One of the motivations for creating the game was for the group members to be able to use it as a reference in any future job applications. To further motivate the development effort, the group decided to send the game to *Swedish Game Awards* (SGA). The benefit of this was possibly two-fold; exposure for the game and group as well as professional feedback from the SGA jury.

Due to the nature of the game an important aspect of the gameplay resides in the fact that there is an interesting AI for the crew. The player should not feel that the crew is ridiculously stupid and do not get anything right. If that is the case then the player will most definitely stop playing the game. The group therefore concluded that the AI should be good enough to support the following features:

- Unique personality for each crew member
- Need-driven behavior
- Interaction between the player and the crew
- Interaction between the different crew members
- Interaction between the crew and their surroundings

Influences

To come up with ideas for the game a number of movies and games were discussed. It was early decided that the game should take place in space and this led the discussion to well known series such as *Star Trek*, *Battlestar Galactica*, *Firefly* and *Red Dwarf* as well as horror movies like the *Alien* series and *Event Horizon*. The concept of being a captain giving orders to his crew as well as ideas for all the various situations that can occur came from these discussions.

The gameplay takes influences from various games. The crew management is similar to games like Maxis' *The Sims* and Bullfrog's *Dungeon Keeper*. In *The Sims*, the goal is to build a house and manage a family, choosing career paths and generally keeping the characters happy. What keeps the game interesting is watching the interactions between the characters. (1)

In *Dungeon Keeper* the player manages a dungeon, orders his minions to gather gold and attack the enemy. By researching and building rooms in the dungeon the player attracts various demons. These need to be kept happy or they will start rampaging and kill each other. The demons have different personalities and some work well together while others try to eat each other. This makes the game different from other strategy games since the minions may refuse to follow the player's orders. (2)

The missions in Iridium work similar to other space trading games such as David Braben's *Elite*, Digital Anvil's *Freelancer* and *Eve Online* by CCP Games. In general they revolve around picking up something on one station and taking it to another. There can also be escort missions or bounty hunting missions. As the player earns money he can upgrade his ship and accept more challenging missions.

To make Iridium stand out from all existing space games the art direction was discussed extensively. The decision fell on steampunk and in order to get a good picture of this style existing steampunk games and movies were examined. The heaviest influences probably come from Troika Games' *Arcanum*. The game takes place in a fantasy setting where the industrial revolution has just begun. Mixed with elves, dwarves, gnomes and magic are steam engines, guns and electricity. The visual style resembles Victorian England in clothing and furniture which is the direction that was chosen for Iridium. Other games with steampunk influences, although less "on the spot", include *Chrono Trigger* and *Final Fantasy VI* by Square (now Square Enix) and *Syberia* by Microïds.

A few movies that were looked at with regards to the visual style are *The League of Extraordinary Gentlemen* directed by Stephen Norrington and *Skycaptain and the World of Tomorrow* by Kerry Conran. In litterature the biggest influence comes from the books by Jules Verne. This is reflected in the subtitle of *Iridium: Twenty Million Leagues Above the Sea.*

Realization

In order to realize Iridium, a large amount of content and programming was done. The entire game including all assets, with the one exception of the sound, was created within the group. The group had to create the visible world by modeling the rooms of the spaceship, contents of each room and characters living in it. Each room, furniture and character also needed a texture and some of them needed animations. The most elaborate animation work was for the character model which required an advanced rig – a set of bones that controls and restricts the movement of the model – as well as a large amount of different animations.

On the programming side several systems needed to be created and adjusted to work well together. Among these were a scene graph for rendering assets, hit detection for selecting objects and crew-members, AI to control the crew, a sound manager, a game-state manager and many other modules. A big part of the programming side was writing code in such a way that all the different parts worked well together.

Critical design decisions

With a relativity short amount of time to create a game of this magnitude, a lot of hard choices have to be made quickly without much time for review. Some of these decisions are more important than others, typically due to the amount of code they affect.

XNA

XNA is a game development platform provided by Microsoft. It provides a simplified interface to DirectX, accessible through C#. The framework provides a content pipeline as well as rendering, input and sound functionality. However, choosing the XNA framework meant imposing a number of constraints on the project, such as the lack of buffered input. The decision to use this framework was made mainly based on three points:

- A large community
- Portability to Xbox 360
- The chance to win the XNA category of the SGA

3D World

The choice of setting the game in a fully realized 3D world had a direct impact on both the workload and what kind of work that had to be done. Models, animations and textures have been one of the largest workloads for the group and have also required a certain degree of added competence with programs and technique.

In-House Developed Content

Having the group create every model, texture and animation themselves, meant a lot of work being taken away from gameplay design and programming. However, having full control over every asset is a large plus.

Lua

To be able to quickly and easily write and edit gameplay code, the programming language Lua was chosen. It is designed as a scripting language with extensible semantics. The main advantages with scripting languages are that the code does not need to be compiled and that it can be altered during runtime. Lua was used because it is currently the most popular scripting language and is used extensively in the industry. The idea was that the game engine would be coded in C# using XNA while game specific code like the interface and game mechanics would be written in Lua.

Threaded agents

In order to run the logic of the agents in a more direct way, a decision was made to thread the agents. Instead of running their logic per frame, doing many function calls, the agents now instead live inside a continuously running while-loop. Furthermore, using semaphores helped a lot when syncing two agents occupied in a joint action.

Organization

Most weeks the group had one large meeting each Monday, and if deemed necessary another one on the Friday. The Monday meetings were for discussing the work that had been done during the previous week, and to decide what should be done the following week. The Friday meetings were used as backup in case extra time was needed.

Everyone in the group had different areas of responsibility, including:

- Project management
- Software architecture
- Graphical User Interface
- Artificial Intelligence
- Rendering
- Sound
- Content
    - Models
    - Animations
    - Textures

Each area of responsibility came with a certain control of what and how things would be created. To keep people on the same track it made sense to have one person in charge of the people in the group and another person in charge of the software. All the big decisions were taken democratically, even across different areas of responsibility making it possible to criticize and comment on the work done by other group members.

Software framework

Once the goals were more or less clear, work began on the software framework. XNA already contains functions for loading content and rendering it, and there is also an update-function and a draw-function that are called each frame. Using XNA as a base the first thing that was written was a scene graph. It was set up as a simple tree where each node has a position, rotation and scale as well as a pointer to the entity attached to it. Simple view frustum culling was implemented by calculating bounding spheres for the entities.

At this point work was divided into Lua, graphics and artificial intelligence. The idea was that all gameplay code would be written in Lua while the graphics would be handled in C# using XNA. Since AI was such an important part, its development was started as early as possible. Lua support was implemented using the LuaInterface library (3). To simplify debugging a console was implemented. Since XNA does not support buffered input the XNA rendering window had to be hooked to capture window messages (4). Once Lua was in place a resource manager was written and load functions were exposed to Lua so that content loading could be handled dynamically. Functions for adding nodes and entities to the scene graph were also written and made available to Lua so that the scene could be set up and changed without recompiling.

To handle input, gameplay mechanics and the graphical user interface, a game module was written in Lua. Its update and draw functions are called from their respective functions in C#. The update function handles input and the game state while the draw function is responsible for drawing the graphical user interface. The mouse state is passed from C# into a handle input function in an input module. This function also updates a set of Boolean variables for each action, such as move left, move right and select. These are then read in the update function which performs the appropriate action.

When an entity is created it is placed both in the scene graph in C# and in an entity list in Lua. A visible entity can be a room, an item or a character. There are also non-graphical entities such as triggers and navigation points used by the AI. C# entities listen to an update event and Character entities are additionally accessed from the AI threads. In Lua, all entities are iterated through in the update function in the game module.

Gameplay Mechanics

The game state handler is a module that keeps track of where in the game the player is, what he's doing right now and what is going on around him. It is completely done in Lua in order to separate the game logic from the graphics, which are done in C#.

The game state handler is more than anything a state machine that keeps track of the ships current state and changes it depending on what input it receives. The different states with descriptions are listed below.

- **Docked:** The ship is in a station or docked to one. This gives the player access to missions posted on the current station.
- **Safe:** The ship is in transit between two stellar bodies and no one is pursuing or engaging it. The player is unable to start new missions at this point since the ship is not connected to a station computer.
- **Pursued:** The ship is being pursued by hostile ships.
- **Engaged:** The ship is under attack by hostile ships.
- **Fleeing:** The ship is escaping from pursuing or attacking enemies.

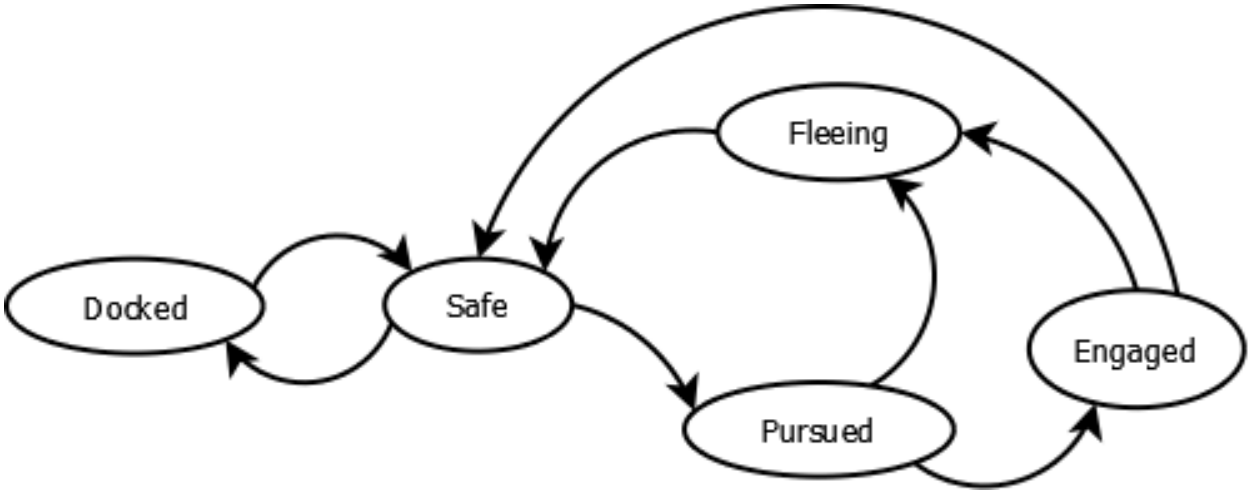

Figure 1: Diagram of the states of the ship

Combat is handled during the Engaged state and is mainly a matter of keeping track of turret firing timers and shots fired. Each hostile ship gets a firing rate based on the type of turret mounted on the ship, which specifies the time left until the next shot for the corresponding ship. The state handler decreases this time each frame and fires a shot when it reaches zero. Whether or not the shot hits is based partly on chance and partly on the precision rating of the turret. The same system applies to the player's ship but while the hostiles are simplified with only one turret per ship, the player's ship is equipped with *n* turrets and therefore it has to keep track of *n* turret timers and *n* shots.

If the player ship is hit, the impact will deal damage to both armor, which is basically the health of the ship, and the systems onboard the ship such as turrets and engine parts. The number of parts damaged is completely based on chance but the damage dealt to them and to the armor is based on how much armor the ship has and how many parts were damaged.

In addition to updating the ship state it keeps track of the status of the ship, such as armor and condition of the turrets. It also keeps track of the in-game time, i.e. how much time has passed in the game world. This is used to switch the work shifts during the day so that the AI can control what the crew is doing during different times of the day.

The game state handler also has a few connections to C# in order to set certain states and to call functions specific to a certain action. Since the combat system is tied to visual and audio feedback it has to send combat state changes to C#. When it changes shifts the AI needs to know which characters are supposed to do what.

In order to keep the game state handler legible it has been divided into sub-modules that encapsulates a certain logical area of control. These modules are described in detail below and also shown in the structure diagram below.

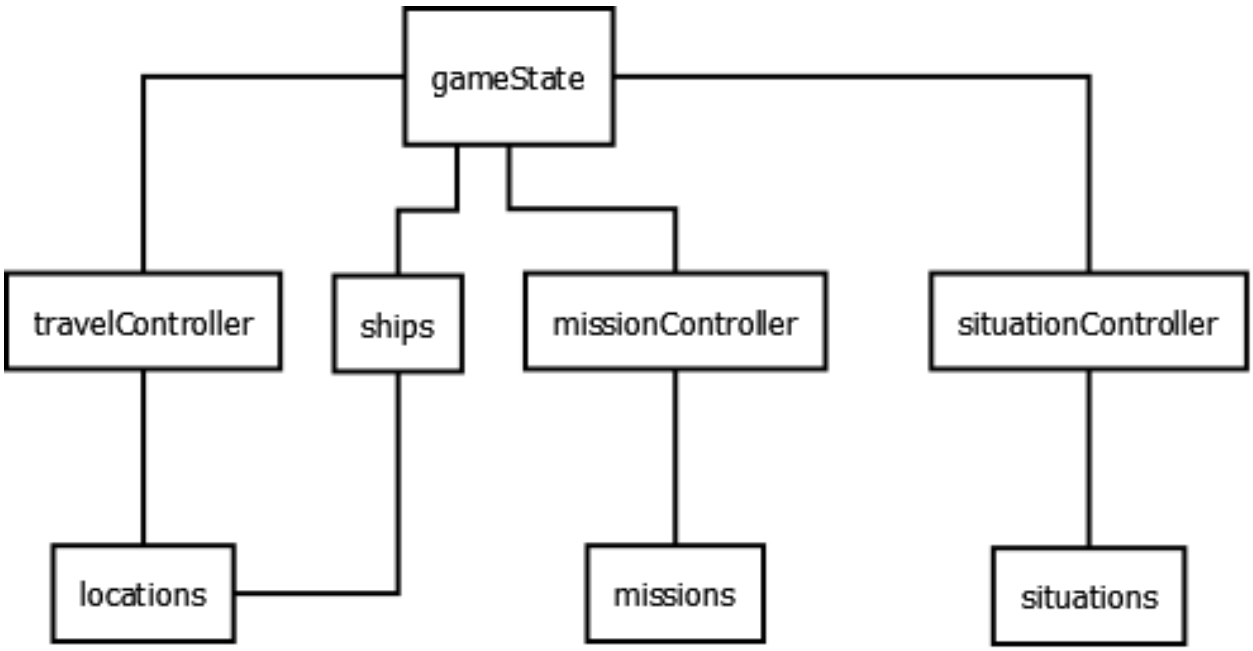

Figure 2: Overview of the game state controller

The travel controller module is responsible for keeping track of the ships movement in the game world. When the player starts travelling from point *A* to point *B* the travel controller will retrieve the distance between these two points and for each frame it will subtract the distance travelled since last frame, based on the speed of the ship. The ship has reached its destination when the remaining distance is zero.

The mission controller keeps track of what missions the player has active at any given moment and will continuously check if one or several goals have been completed. Currently there are only trading missions implemented in the game. This means that the only check made is whether the ship has reached the destination of one of the goals of the mission. The mission controller also adds and removes cargo from the ship's cargo bay when the player accepts, completes or cancels missions.

Since a goal can have several sub-goals it is necessary for the controller to recursively check the goals in order to make sure all sub-goals are fulfilled before the main goal can be completed. Upon completing a

mission's main goal the reward attached to the mission will be added to the player's inventory. The mission will also be made unavailable to the player.

The situation controller works much like the travel controller except that the player has no control of what situation is started and when. When a ship starts to travel between two points in space a probability of enemies occurring will be calculated based on the distance between the points. The reason for this is that a fixed probability would result in a large amount of situations for larger distances. As with missions there is only one type of situation implemented and that is the pirate attack. To successfully complete such a situation the player either has to destroy all the enemy ships or flee the scene.

An addition that was made to the situation controller was that during the first twenty and last ten percent of a journey an attack cannot occur. The reason for this is that these areas are more likely being patrolled by naval forces and pirates will therefore not venture that close to a planet.

Graphical user interface

The user interface is implemented as a sort of hybrid between retained mode and immediate mode user interfaces. It is handled completely in Lua and consists of a set of tables, one for each element. The tables hold information such as type, position, size and texture. Buttons also contain a function which is called when the user clicks on the element. The items are put in a parent-child hierarchy similar to a retained mode user interface and the entire interface is first updated and then drawn. However, all of this is done every frame as in immediate mode user interfaces, and all code needed for an element is in the same place, meaning there are no callbacks or unnecessary data.

The updating and drawing of the interface is handled recursively by a UI manager. The update function traverses all visible items and checks whether the mouse is hovering over any of them. It also updates any animations. The hot[1] item is stored so the appropriate on click function can be called when the user presses the mouse button. The draw function simply renders the visible items by calling a draw sprite function in C#.

Context Menu

To give orders to the crew a context sensitive menu is used. It appears when the player clicks the right mouse button. Depending on which character is selected and what the player right clicked on various options appear on a four-button dial. If the player clicked on the currently selected character the four buttons represent the work priorities of that character. The player can for example choose whether an engineer should focus on the engines, armor or turrets. If the player clicked on a character other than the selected one the buttons represent what the selected character can do to the targeted character. A medic can for example hand out various drugs to improve the performance of the crew. Lastly, if the player right clicks

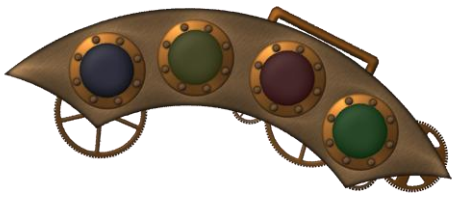

Figure 3: Context menu.

[1] An item hovered by the cursor

on an item the system looks at a table to see what the currently selected character can do to it. This is determined by what job the character has. An engineer can for example repair a turret but not man it and vice versa for a gunner.

In-game menu

The in-game menu consists of controls for a large number of gameplay elements. It is divided into a number of different screens, each containing the controls available to the player for controlling a specific element.

The menu system is designed with a number of different patterns in mind. The definitions of these patterns vary, but in this report the definitions used by is used. The description of how the patterns have been used is divided into the different interface screens, starting with patterns used in the interface in general.

General

Almost all elements in the in-game menu utilize shading in the textures which makes them stand out from the background, effectively giving the background a depth relative the elements. This is referred to as "deep backgrounds" in (5). All elements are beveled to give them a physical volume and to reinforce this they also have rounded or otherwise stylized corners.

The appearance of a button changes depending on the state of the control. When a button is hot, it will be highlighted if it is clickable at that moment. Buttons can be disabled when they are unavailable for use, e.g. scroll buttons are grey and inactive when scrolling is unavailable. When a button is disabled it will not be highlighted if the player should scroll over it.

Figure 4: Button appearances

Tooltips are used on most clickable objects in the menu. The exceptions are buttons with obvious functions. Many buttons in the menu system are grouped into sections or button groups (5), indicating that they have common functionality.

Heads-Up Display

Only a few interface elements are visible at all times. These are the clock and the in-game menu selection buttons. The clock is a simple time indicator with only one pointer. Each lap the pointer makes represents 12 hours. The current shift is highlighted in green to make the player aware of which characters should be currently active.

Figure 5: Heads-Up Display

The in-game menu selection buttons group is, by default, displayed on the left side of the screen and hovering over a button will highlight that button. When the player clicks one of the buttons the corresponding menu screen is opened. When the menu is opened the button group is moved to the right side of the menu frame, at the right side of the screen. At any time the menu screen can be changed to one of the other menus by clicking one of the buttons on the side of the menu. The menu can be closed by clicking the "confirm" button on the lower right side of the menu.

Captain Menu

The captain menu contains information about the captain character and controls for changing his attributes and uniform. There are five fields in this screen:

- The basic information field contains the captain’s name, rank and experience.
- The rank field contains a visual representation of the captain's current rank, in the form of a symbol comprised of one or several nautical stars.
- The uniform field specifies what uniform the captain is currently wearing and also has a button for changing his uniform.
- The skills field specifies the number of points the captain has in the different skills, as well as how many points he has in his point pool. Each skill has its own field with a number panel and buttons for changing the skills, if there are skill points available.
- The portrait field contains a portrait of the captain and a plaque with his name.

Figure 6: Captain Interface Screen

The purpose of the captain interface is mainly to be informative and give the player a special connection to the captain compared to the rest of the crew, which is why it houses a very limited amount of controls.

Crew

The crew menu screen has two looks, depending on which state the ship is in. If the ship is docked in a station or on a planet the player is presented with controls for hiring new crew and firing current. In this mode the crew menu system is almost identical to the mission menu system described in *Mission Selection*.

When the ship is not docked, the player is presented with controls for arranging the crew in different shifts and with different tasks. A day-night cycle in the game is divided into three shifts, each being eight hours. In the crew menu system the player can decide which crew member should work which shift or shifts as well as what they should do during their shifts. The menu takes the form of a list builder (5) with four different lists.

Figure 7: Crew Interface Screen

On the left side of the screen there is a list of all the available crew. Each crew member has a plaque with a portrait, utilizing illustrated choices (5), and two descriptive fields. The first field has three icons, the top one a little bit bigger and more highlighted. These three icons represent the characters three best skills, the top one being the best one. The second field has the characters name as well as more detailed scores for the characters three top skills.

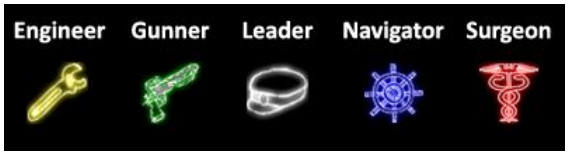

Figure 8: Skill and Job Icons

On the left side of the crew list is a button group housing four buttons. The top three buttons will either add or remove a character to the three different shifts. The fourth button will clear all the shifts.

When the player clicks a character plaque the three buttons in the middle button group are highlighted and their icon and function change according to what functions can be performed on the selected character. If a character is in a shift the corresponding shift button will change to a remove-from-shift button. Each character can work, at maximum, two shifts each 24-hour cycle. If a character is already in two shifts the third button will be grayed out and unavailable.

To the right of the button group are the three shift lists, one for each shift. Adding a crew member to a shift adds a smaller character plaque to the corresponding shift list. The small plaque consists of the portrait and a big icon representing the crew member's job during that shift. When adding a crew member to a shift he will be assigned the task he is best suited for, i.e. the task utilizing his top skill. This

is in adherence to the pattern “Good defaults” (5). The player can change the characters job during a shift by clicking the toggle buttons under the icon field on the plaque.

Mission Selection

As with the crew menu, the mission selection menu can take two forms, depending on whether the ship is docked or not. In both cases the menu consists of a list on the left side of the screen as well as two descriptive fields in the middle.

Figure 9: Mission Selection Screen

In the list all currently accepted and unfinished missions are listed. On the left side of this list there is a button group housing two buttons: the set-active button and the cancel button. The set-active button can be used to set a currently selected mission as active. This will display the waypoints connected to that mission on the map described in *Map*. Only one mission can be active at a time. The cancel button removes the selected mission from the mission list and any progress made in that mission is forfeited.

The middle section of the screen has a monitor at the top and a text field at the bottom. The monitor prints information about any currently selected list item, such as current progress. The text field gives a more detailed description of any selected mission.

When the ship is docked the right side of the menu screen contains a list of all the missions available to the player at that time (Figure 9: Mission Selection Screen). Clicking an item in this list will give a more detailed description in the text field as well as print some related information in the monitor. When the ship is not docked the right side of the screen contains a picture frame, which is empty if no mission is selected (Figure 10: Mission View Screen). By clicking a mission a picture representing that mission's type, e.g. trading or combat, is displayed in the frame.

Figure 10: Mission View Screen

Map

The map screen contains one major element, the map frame, which in turn has two elements in it: the map inside the frame and a button group on the lower part of the frame.

The map has an icon for each location the player can travel to, e.g. the different planets in the game. When the player hovers over a planet, it is slightly enlarged and highlighted. The map also displays the possible routes between the different planets as gray or white lines, depending on whether a planet is accessible from the player's current location.

Figure 11: Map Interface Screen

Apart from locations and routes the map can also display waypoints if there is an active mission (see *Mission Selection*). These appear as crosshairs in different colors depending on what sort of waypoints they are and whether or not they are currently available. There are two kinds of waypoints; Sub-goal waypoints or main goal waypoints. The former is gray while the latter is green and a little bit larger. If the player needs to complete another goal before the goal the waypoint points to can be completed, the waypoint will be unavailable, which will make it transparent. When hovering over a waypoint the player will get information about what task that waypoint represents, as in the pattern "Datatips" (5).

Figure 12: Waypoints

The button group on the lower part of the frame has two controls: The depart button and the waypoints toggle. The depart button will be available if the selected location is reachable from the players current location and clicking it will set the ships course towards that location. The waypoints toggle toggles whether to show waypoints on the map or not.

Visual Style

The visual style of the interface has been designed to follow the game's steampunk theme as much as possible. Surfaces have metallic textures and colors to give the felling of old machinery, and cogs and wires have been used excessively. Many of the textures used were found at CGTextures (5). To further strengthen the feeling of industrial machinery, overlays of dirt, wear and tear have been used, e.g. cracks in glass panels, rust spots and scratch marks on surfaces.

To make the interface look like a steampunk contraption rather than just an old, worn piece of machinery, colorful lights and luminescent borders and highlights have been added on top of the mechanical metal parts. In the context menu the colors are very divergent to help the user quickly distinguish between different options. In the in-game menu however, the light color used is primarily a shade of light blue to give an almost mystical glow to the interface elements. In the case of skill icons, different colors have been used, again to help the player quickly distinguish between them.

Artificial intelligence

One of the most demanding aspects of the concept for Iridium was to create an AI that would be independent, take rational decisions and interact with the rest of the ship crew. With the theme "Interactive Interacting Personalities" to live up to, two main objectives where created for the AI. The first was to have an academic basis for the underlying structures and fundamentals for the AI. The second, to create a flexible foundation that can be adopted to both work in a restricted manner where the AI has little "free will", or to be completely set free to take any desirable action.

Research

The main source for researching the AI is a web-portal for AI in games (6). At this page there is a blog, a forum, articles and other useful resources for anyone interested in AI. Research for this game started with reading an article about the AI in "The Sims" (7). Some of the most important insights from here is the object driven AI they use, and the *Goal Oriented Behavior* (GOB) used. In order to find more details about a suitable GOB the book "Artificial Intelligence for Games" by Ian Millington (8) was consulted. The main ideas had formed and at this point the concept needed some feedback. The AI still lacked the academic foundation that was mentioned earlier, but after consulting the forums at (6) the resulting feedback pointed towards some research that would fit very well with how the AI was to be realized.

Implementation

The bulk of the AI is written in C#, but in order to more efficiently extend the amount of actions that agents can do, the GOB and complete listing of actions where written into Lua. This helped as adding new actions and tweak existing ones could be done without recompiling any code.

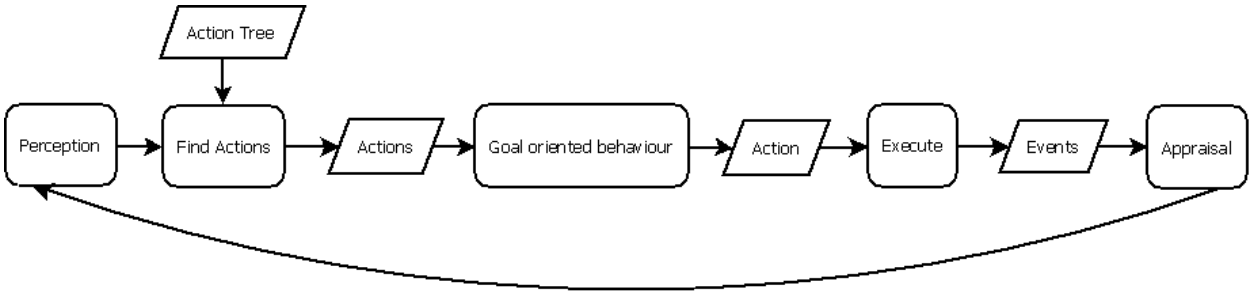

Figure 13: Sequence diagram for the agent thread

The main objective for the AI is to perceive its environment, use this information to select an action depending on its current state, and finally execute this action. As code this results in a large part consisting of the perception, one for the GOB (finding the right action) as previously mentioned done in LUA, and finally one for executing actions.

Agents, events, actions and items

One of the critical decisions regarding agents as mentioned was to have them separately threaded. This means that they are cut off from the main thread of the process and therefore no longer have any access to certain resources that this provides. It also meant that handling actions and especially actions between two agents was greatly simplified because of the use of semaphores and the ability to synchronize threads. In reality an agent consists of a large set of variables and lists. It contains everything from its mood to the current need, target and action it is involved with.

The internal events sent between agents were created in order to simulate reactions. It is a pre-set reaction to a certain thing an agent could do. The resulting influence it had on the agent is much more personal and takes many runtime variables into account.

Actions are stored in Lua and contain a certain number of variables that are everything from the required emotions to execute the action to what animation should be cued for it.

Items are also stored in Lua but have a C# copy. This is in order to let agents walk to and find the items in the world. They are in some cases categorized depending on their use or characteristics. To further make the interaction between agents and items interesting an item has a number of properties which determine what an agent thinks of it. An important addition to some of the more interact able items is adding certain points of reference to them. This makes it possible to walk up to items and cue animations in a controlled manner regarding position and direction of the agent.

Perception and available actions

In order to let agents understand where things are occurring and where other agents are, they are given a field of vision and hearing. At its core the perception consists of information about objects, agents and their whereabouts.

The GOB needs a set of actions and it made sense to prune this selection as early as possible down to only the actions that an agent can do at that point in time. The main conditionals for the pruning are the current need of the agent and its emotions.

GOB (Selecting an action)

With a set number of actions the goal oriented behavior will for each action calculate a score that represents how good this action is for the agent. This part of the code was written in Lua because it deals directly with actions that are also present there. One important aspect when choosing the best action with the GOB is that the score can be based on information that does not necessarily have anything to do with the actual effect on the agent. The score would be something fabricated that represents something much more prejudice, i.e. how the agent “thinks” the action would affect him. This is a powerful tool when designing actions and more interesting scenarios.

Execute

Executing an action was done as general as possible. The algorithm starts by “Resolving the target”. This means parsing certain keywords and could involve finding a suitable target that would not always be the same. The action might ask for a bed, and it would be up to this part to find a not already occupied bed.

The second step of the action moves the agent from its current position to the target position using a special lookup table containing agents, rooms and items and their location.

At this point animations, sounds, icons and any other player feedback that is part of the action would be triggered. A lot of what an agent does is represented with a simple visual cue combined with having the agent thread sleep for a while. Apart from animations and other visual cues it will also send out a special event for the other agents to receive, this is done in order for the other agents to react to what it is doing.

The third step branches into two possible directions. One is dependent on if the target is another agent; it has logic to take care of things like syncing two agents in order to have them chat to each other. Any action that needs two agents to meet, and work through a number of things in a certain order require certain care and has a lot of use of the threaded nature of the agents.

The second branch is for target items, as mentioned these use a special list of points of interest, and in some special cases more care is needed to handle these right. One example is using these points as some finite resource that can be replenished with a special action. At the end of executing an action the agent falls back into an idle state until a new action is chosen.

Appraisal

A very important part of the game is for agents to react to what they themselves do and other agent’s actions. Events contain information about how an agent should react, which is processed by the appraisal. A large amount of tweaking is involved with the appraisal as it is a matter of designing what and to what extent an effect should affect each emotion and memory of other agents. The appraisal starts by for each event update emotions and memories. It then applies a small decay to each memory and emotion and updates the agents needs based on its current mood and emotions.

Tweaking

In order to find the right correlations between different aspects of items, agent personalities and emotions and everything else, Excel is very useful. It was first determined what item properties should affect each class of agents. Secondly each item property is mapped onto each personality trait (9). This results in finding a base set of values that represents each class’ personality, which is in turn checked against each characteristic item that each class would come in contact with. This helps in finding a good balance for needs and how emotions are affected by events and everything that happens in the game.

	Neuroticism	Extraversion	Openness	Agreeableness	Conscientiousness	
Quality	1	1	4	1	3	10
Weapon	4	-1	-1	-2	10	10
Gadget	-1	-1	8	3	1	10
Aesthetic	3	6	-2	2	1	10
Grand	8	-1	-3	8	-2	10

	Gunner	Surgeon	Engineer	Navigator	
Quality	3	1	5	1	10
Weapon	12	-4	-1	3	10
Gadget	-12	12	13	-3	10
Aesthetic	5	-4	-2	11	10
Grand	-4	9	2	3	10

	Gunner	Surgeon	Engineer	Navigator
Neuroticism	46	33	-2	73
Extraversion	37	-40	-21	64
Openness	-94	85	123	-54
Agreeableness	-79	109	58	32
Conscientiousness	130	-47	12	35

Neuroticism - Adjustment vs emotional instability // insecure - secure
Extraversion - Amount and intensity of social interaction; activity level; need for stimulation; capacity for joy // Sociable - reserved
Openness - Active seeking of experience for its own sake; tolerating and exploring the unfamiliar // Curious - conventional
Agreeableness - Quality of interpersonoal orientation // Soft-hearted - cynical
Conscientiousness - Degree of organization, persistence, and motivation in goal-directed behaviour // Hardworking - lazy

Figure 14: Excel sheet for finding correlations between items and personality.

The first cell grid is the relation between item properties and character personality. It creates a connection that simply determines how important a particular item property based on each personality trait.

The second cell grid is the relation between item properties and different character classes. This connection is important as it determines how well each class will do when working with a particular item in the game world. E.g. an engineer should like gadgets (like the engine) and a gunner should like weapons (like a turret).

The third cell grid is the result of these two relations. This gives a base set of values that is suitable for a certain type of character. Each character will have a variation of these, but to find the right job for a character this is very useful.

Another tool for tweaking and debugging was an in-game output of each agent's state. It shows the logical position, current actions and any other variable that is useful to track in real-time. In the picture this is the red text, the three colored spheres are navigation points that are used for the AI to navigate the ship, and the yellow blocks are triggers used for moving agents between rooms in the scene graph.

Figure 15: AI debugging

On top of the in-game output there is also a debugging system that streams some of the information into a file that can be reviewed to search for what is causing a bug.

Rendering

This section will describe some of the fundamentals of 3D-drawing in Microsoft XNA and how it is done in this project.

Drawing in XNA is done by retrieving or creating a set of vertices and a list of indices which indicates in what order to draw the vertices. To make XNA more user-friendly this has been wrapped inside a draw function that draws an entire model mesh at once. If more control over the drawing process is needed the more manual approach is still available. Each time a model is drawn it has to have an effect attached to it, which in most cases is a pair of shaders, one vertex and one fragment shader. In this case, “attached" means that each mesh of a model holds a reference to one or several effect files. An effect, a shader pair, is needed to actually be able to draw something to the screen.

When a model is loaded through the XNA content pipeline it will get a basic effect attached to it. This is the default effect used by XNA and has a number of options and parameters that can be modified by the user to achieve a satisfactory result. However, this is usually not enough which means a custom effect has to be created and then, as mentioned previously, attached to the model by replacing the existing base effect.

When drawing to the screen in-game, care has to be taken with what is rendered and when it is rendered. If this is not done the game would have to render every object currently in the scene, which quickly becomes unfeasible. This is solved by putting all objects in the scene in a logical and spatial

hierarchy, a scene graph, which holds their position and the volume or area they occupy. This structure takes on the form of a tree and is traversed recursively to identify which nodes are visible or not. This is done by checking if an object intersects the view frustum of the camera. If it does the object is drawn, otherwise it is culled away, which reduces the load since the game only have to draw what is actually seen by the player.

For this project all models are drawn directly by using the models vertices and indices. However, since the game utilizes the XNA Animation Component Library (4) this is wrapped inside a model class. The library wraps the model, animation and other information in a custom model class that has its own draw function.

In order to be able to put more specialized effects in the scene, the scene is not rendered to the screen but to a texture. This is then modified to achieve certain effects, such as blurring or camera shaking. This technique is called post processing. Once all post effects are done the final image is rendered to the screen.

Effects

In order to achieve a more convincing and visually pleasing environment in the game a few effects has been applied to the models and to the scene.

The first of these is lighting. As the game takes place onboard a space ship, which is divided into rooms and corridors, it's assumed that there is some form of lighting. So for each room a number of light sources has been placed to light up the environment but also to darken areas that are further away from the lights. The light sources are modeled after a form of incandescent light source, i.e. light bulb, and therefore take on a yellowish tint instead of clear white. These light sources can change color based on the alert level of the ship and currently switch to a darker pulsating red when in red alert. The pictures below show the difference between using only the texture color and adding lighting.

Figure 16: A scene without lighting

Figure 17: A scene with lighting

The second effect is a post process and goes hand in hand with the red light effect. When the ship comes under attack it will be shot at by hostile ships and when a shell hits the hull the screen will shake

for a short period of time, simulating a shockwave. This is done by rendering the scene to a texture and feeding it to a shader that moves the screen around based on sine and cosine functions.

Animations

A system or library is needed to show and update animations. Early on it was decided to use an already existing third party library and the choice eventually fell on the XACL.

This library services as both an importer and a processor meaning that it will parse the model file to retrieve both the model information, as the standard XNA importer does, and the animation information. It also handles all information about the model, i.e. it's a wrapper for the XNA model class and it has its own draw and update function in order to correctly update the animation and draw the model according to the current state of the animation.

With the framework in place each model can have a custom xml file to load animations. This did not contain the exact details needed, and instead a custom xml file parser was written. This made it possible to set names for the animation controller independently of what animation is called. Which is useful when using the same animation, or parts of the same animation for things like a "fast walk" and a "slow walk", both based on the same walk using different time factors when running the animation.

Bone groups were also created in order to divide animations across the body. In some cases it was of interest to have different animations for different body parts at the same time. In theory it also means that animations can be done with this in regard, creating upper and lower body animations that are not directly dependent and can be switched around to create more variations.

Thanks to the animation library it is possible to blend two animations by a factor. One of the interesting areas for this is when an agent starts walking. This works by slowly blending from the idle animation into a walk, depending on the agents speed, and as the speed continues to increase the walk starts blending into a run animation.

The last interesting aspect of the animation library is the possibility to attach items to a certain bone, and receiving a calculated transform for the item that follows the bones position. This makes it possible to attach things like a hat to the agents head or a tool to its hand.

Sound - effects and music

Sound and music was for a long time put aside in the project due to lack of time and the notion that it was less important than other areas of development. However, when the project moved into a phase where gameplay was more important it turned out that audio feedback and music was one of the things that was missing. Due to the late stage of the project it was decided that only simple audio support should be implemented.

To play sounds in XNA there are two approaches that are part of the framework itself and a multitude of third-party libraries. It was decided that for this project an approach that was native to the XNA framework should be used since it gives the most flexibility. The first of these is a sound effect class which is the simpler of the two and allows for playback of sounds and music.

The second approach, and also the one used in this project, is an audio engine class together with sound banks and wave banks. This approach requires that all sound files are compiled into a XNA specific format by XACT, a tool provided by the DirectX SDK. XACT provides a wide range of functions that sound artists can use to add effects to their sounds and music. It also provides support for 3D and directional sound. In XACT the sound artist organizes the audio into different cues that represents playlists. These are then compiled by XACT into an audio engine file and a wave and sound bank file which can be loaded into the XNA project.

The music in the game was created by Kevin MacLeod at (10) and is registered under the *Creative Commons* license. Sound effects are from (11) and then modified in XACT to achieve a close enough approximation to what a firing turret and a hull being hit would sound like.

Content

Throughout the development of the game much content had to be created. One of the largest workloads was the visual content. To ensure a certain art direction of the content, all of it was made in-house.

3D Models

When doing the models much thought was spent on planning how the models were to be seen in the game, the idea was that every model could be optimized in order to make it as efficient as possible. Every model was optimized so that all the faces that were not to be seen were removed, making the object as lightweight as possible. All objects were also simplified so that each model's detail level reflected the detail needed to display the object properly inside the engine from the camera distance intended. All objects were also exported with normals so that the shaders running in the program could utilize them in order to light the objects properly.

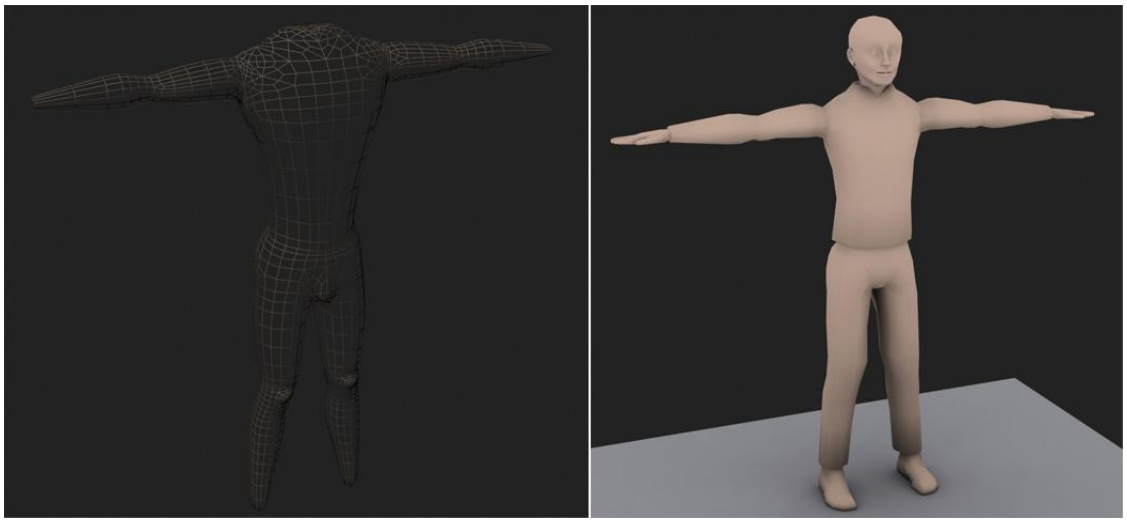

Figure 18: Character model during development

The visual style in the game is very much focused on making the game feel old and used. The chosen style is steampunk which is a mix between the Victorian era, industrial period and science fiction. This is

a rather challenging style to follow and in order to make the style clear in the game some decisions concerning content were made; the ship's engine should be a steam-powered, big piece of industrial machinery; all the furniture in the ship should be of the Victorian era style with, e.g. lion-feet and dark heavy wooden materials mixed with stone and textile.

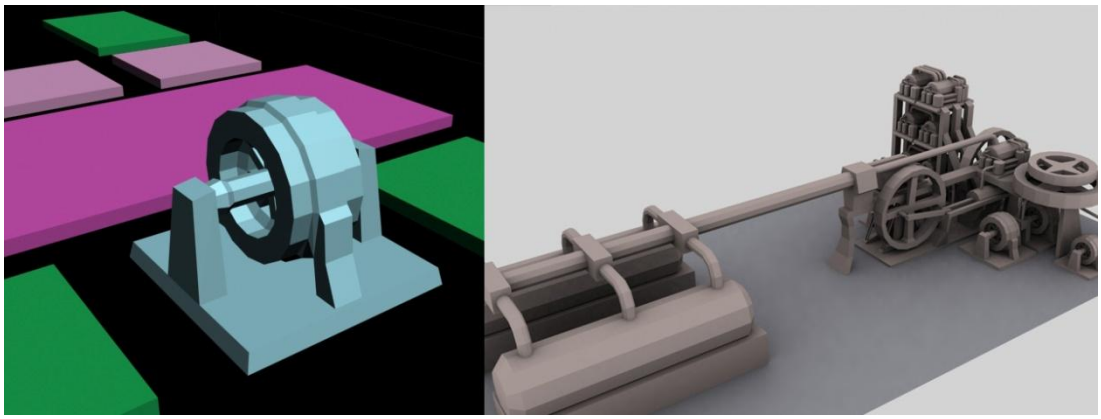

Figure 19: Steam engine during development

Textures

With each model a UV-map was exported as a picture. In turn this was used as the base for drawing the material on top of. Working with the rooms in the ship meant mixing a lot of metal, wood and a few textiles. A lot of work is also done with using smaller material samples and resizing them by fading out the seams. With the materials in place the last step is working with the saturation and hue.

Figure 20: Texture and UV-Map

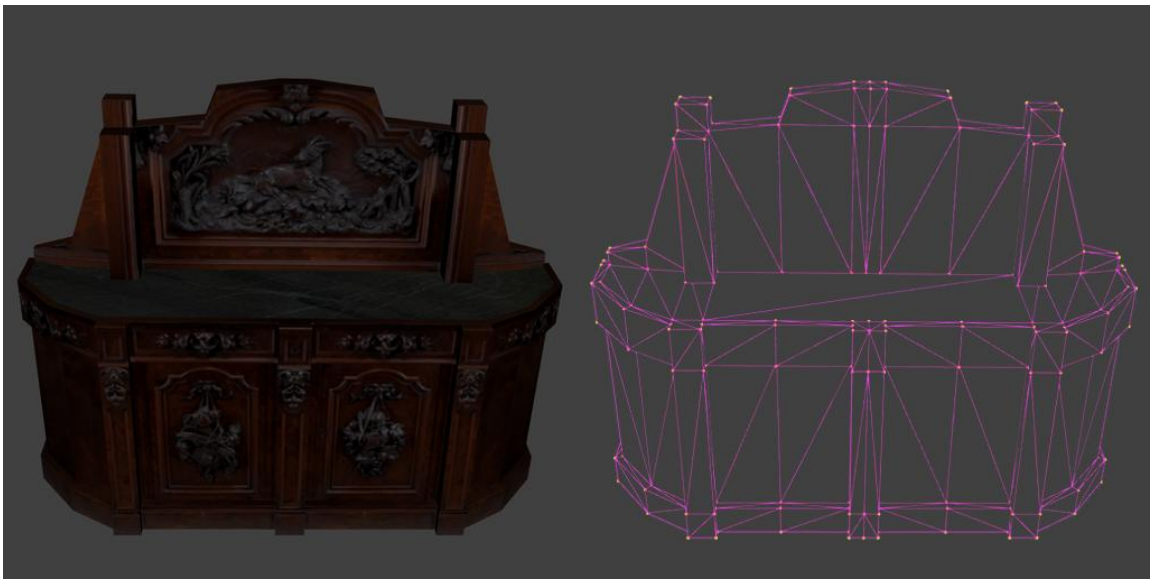

Figure 21: Model and wire frame

To match the visual style as mentioned a lot of different metals were used, the grittier and torn the better. Mixed in with the metal some rooms would use wood for the walls or the floor, and also at some places heavy Persian mats. To create a more gritty and torn look the materials were augmented with a lowered brightness and depending on the case, the saturation can be lowered to get a dirty look, and raised to get an enchanted look. The hue can also be used to better stick to a certain palette of colors. Many of the textures used in the game were found at CGTextures (5).

Animations

Some animations like a running or walking cycle are done by starting with a start pose and then inserting the middle pose. Imagine the motion spanning across 30 frames. At the first and last frame the character has the same pose (only 1 through 29 is later used), in the middle frame, 15, the middle pose is inserted. The animation tool will then according to your wishes interpolate between the two states which creates a very basic animation. To fine tune this, more middle frames are added, the middle of each new span, 7 and 23 are created, and the animation slowly grows into something decent looking.

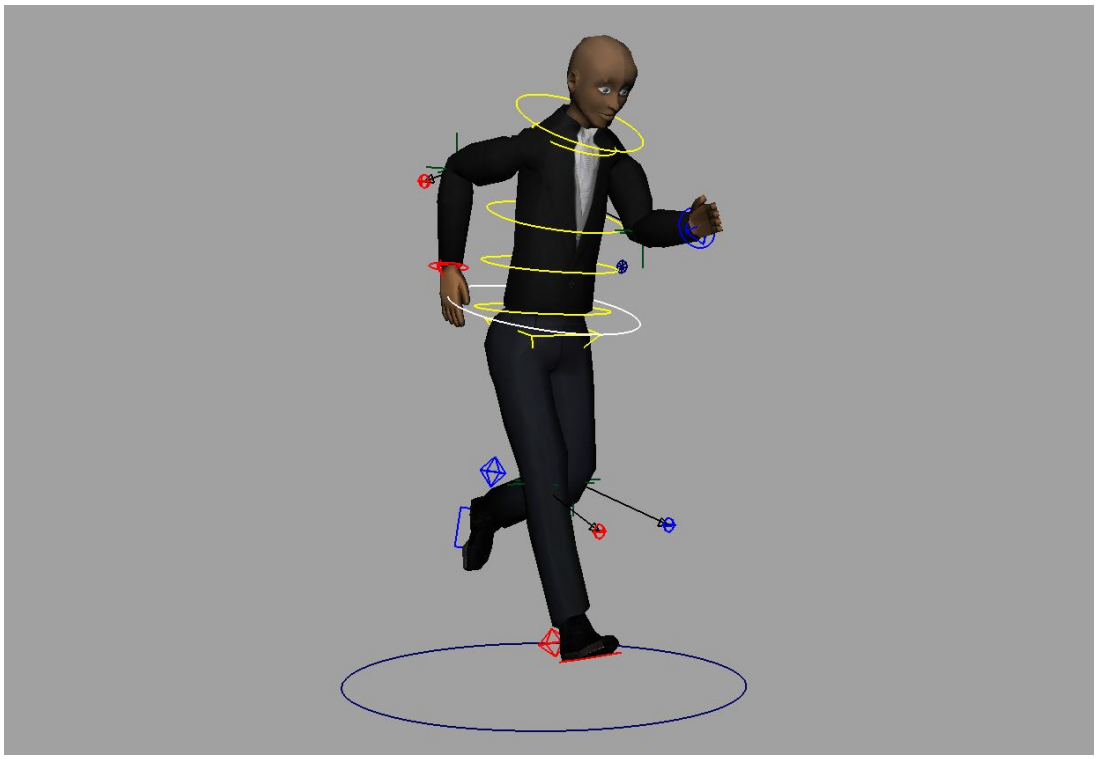

Figure 22: Character model being animated

Animations that are not looped are done by moving from the start of the animation onward, and depending on the total length, a different step size is used. For a longer animation it can make sense to start by moving the model each 5th frame, but this is really a much more free process where length between frames is used for timing and speed of the animation.

Assets pipeline

The content in the game was created through a content pipeline; the first step in the pipeline was the concept stage. In the concept stage we made a rough sketch of the object on a piece of paper and then used that as a reference when doing the actual model. Next step is the modeling stage, which is where the model is converted into digital format and created in a 3D program. After this is done the UV-mapping begins, which is where the model gets a basic texture template which can then be used to create a real texture for the object. After the UV-mapping is done the pipeline splits, one of the paths handles the texture and its creation and finalization, and the other path covers the finalization of the model. During the last stage the model is just exported to the format needed and then put in the right directory in the game-engine. As for the texture, it is created in an image program using the basic texture template. Often several iterations are required before the texture is finished; the artist then uses the game engine to verify that the result is as intended.

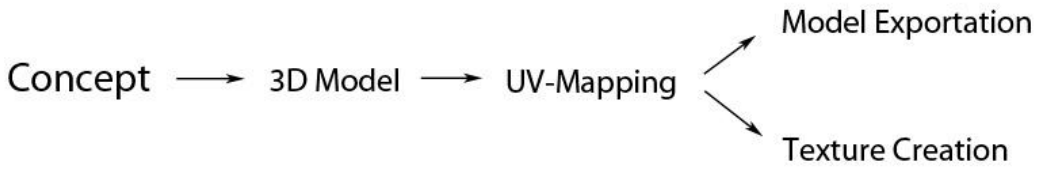

Figure 23: Asset pipeline

From Asset to Content

When the 3D model and its texture have been created they are considered to be assets of the game. Then the model has to be imported into the engine, and this is made via the XNA content pipeline. First the model and its texture have to be copied into the engine. This is done by only copying the files to a specific folder inside the engine. The next step is to add the file and selects its content processors inside the development environment. After this is done, the model has to be loaded into the project using Lua commands as well as added to the game. The last step is to place the model where it is supposed to be in the 3D environment.

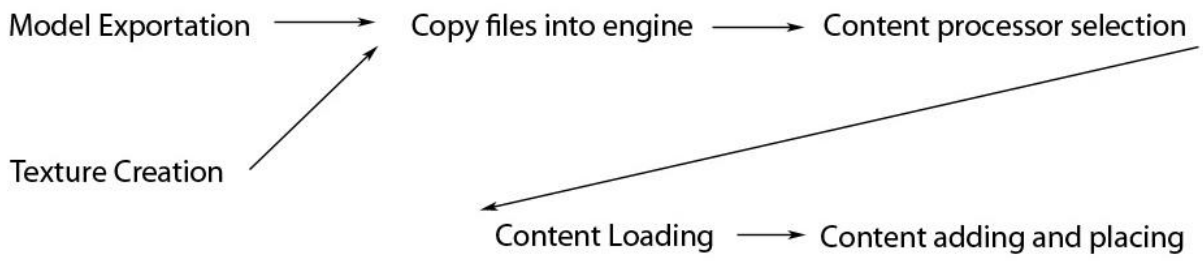

Figure 24: Content pipeline

Evaluation

With this being the first real game project for the group setting the right level for the goals was a real challenge. Though the ambition never was to create a complete game, but rather a small piece of what could be, even this turned out to be right outside of our reach. The final exhibition prototype could, at best, be classified as an advanced tech demo. It consists of a massive base of functionality, which in its current state is not used for as much as was intended for the exhibition release.

The main thing that got in the way of the final goals for the exhibition was numerous systems that needed to be in place in order to facilitate with the underlying mechanics that where needed.

Organization

The group started out well with regular meetings and though there where less meetings in the end of the project this was more due to the workload rather then not being able to. Meetings were used to make sure everyone was working and get everyone on the right track for the next week. But towards the latter part of the project meetings became obsolete as the group would meet more or less everyday to work together for several hours.

The group worked very well and people got along. It is important with projects like this to divide work and friendship, and leave room to be able to criticize and comment each other's work, without letting this get in the way of working together.

A thing that worked less well at times was decision making. Smaller decisions would at times take time as it felt unclear whether or not it was the right decision, and the group was too busy to sit down and all decide this for each time. The problem was more a communication issue where it was unclear what kind of decisions each one could take on their own. From the start the idea was to more often take decisions in the group, but as time went this was too slow, and instead decisions were encouraged to be taken more freely.

The other problem relating to the first one was that of not having a complete design for everything being done. When the need arose to give orders to agents with the controls, there was not a design decision taken on what orders should be available to give, which greatly slowed down the process. These problems are, in part, a direct fault of the way the course is laid out, with a bad structure between working with the design and working with the implementation. It is also a fault of choosing a concept that the group in no way could undertake and completely design in the time period, which led to a lot of "holes" in the concept. In term this leads to taking a lot of decisions with very little review.

Software framework

Using XNA made it possible to focus more directly on the game without having to worry too much about the engine. The simple scene graph that was implemented worked well. Working with a hierarchy and relative positions makes things easier, although moving characters between rooms was a bit tricky. It also makes the picking algorithm more efficient with a O(log *n*) complexity instead of O(*n*). There were a few problems with the view frustum culling since bounding spheres were used and the algorithm which calculated them was not accurate enough. It would probably have been a better idea to use bounding boxes, something that would not have been very complicated since they were already being used for collision detection.

One bad thing with XNA is that, for some reason, it does not support buffered input. This means that the keyboard and mouse states need to be polled every frame and if the user presses a button quickly in between two frames the input is missed. It also makes text input very complicated. For this reason a method of hooking the window and catching window messages was used for input to the console. For some reason this prevented the mouse wheel state to be read from XNA's input handling. In retrospect it would probably have been a good idea to use this method for all input, but at the time the decision to drop XBox 360 support had not been made and hooking window messages obviously only works in the PC.

When the decision was made to use Lua the idea was that C# and XNA would be used to write the game engine and all game specific code would be written in Lua. Partially due to inadequate planning and lack of time the separation did not become that clear. The different entity types needed to be rendered differently and therefore rooms, characters, items etc. are separated both in C# and Lua. Ideally there would only be a general entity class in C# or even no entity at all and only models. Another problem is that the update functions of the entities in Lua are not used at all at the moment since everything is handled in C#.

Since the coding of the AI began before there was a clear software framework, much of the code is written in C#. This leads to some jumping back and forth in the code which might give some overhead, and it's one of the reasons why the entities could not be unified in the C# code. On the other hand, having the main AI code in C# allowed for threading which probably increases performance.

Graphical user interface

The graphical user interface system, while simple, worked very well. It lacks a few nice features such as scroll bars and combo boxes, but all these are possible to construct from simple buttons. In the case of scroll bars a functionality that is not currently supported is dragging the bar with the mouse. With a little work it would probably be doable but the lack of buffered input makes implementing it a bit tricky.

Context Menu

The context menu needs a lot of more work in order to be practical. It is not clear to a first time user what the buttons do and why they differ depending on what he clicks on. This could be remedied by having the mouse pointer change appearance when the cursor hovers over different items. The interface also needs to show what character is currently selected. Something like a portrait and status of

the character in the corner of the screen would probably be enough. A lot of work needs to be put into the appearance of the buttons. Right now the buttons are only a single color with a tool tip saying what it does, but obviously there needs to be icons representing what the button does.

In-game Menu

The in-game menu is still a work in progress, but what has been implemented works quite well. Highlighting buttons and fairly descriptive icons will make it possible for a first-time user to orient himself reasonably quickly. There are a few exceptions to this, such as the shift arranging interface where the add-to-shift buttons are not as intuitive as, e.g., a drag-and-drop solution. The game is not meant to be playable after having only tried it for a few minutes, however, so this is not considered to be a problem.

The main problem still existing in the interface is the lack of scalability. From the beginning of the development process the intention has been to allow for a crew of something like 30 men. With a crew of 30, the crew list system would quickly get annoying, allowing no easy and quick way to scroll the shift lists or, more importantly, the crew list. One possible solution to the problem would be to make the crew plaques smaller and therefore allow for more of them to be displayed without the need of scrolling. Another solution would be to implement a scroll bar, allowing the user to drag the scroll up and down and thereby quickly traverse the list.

Even with the problems mentioned above, however, the interface gives the functionality needed for the game and stays true to the artistic direction of the game.

Feedback

During the occasions where we have showed the game the general feedback on the interface has been that it looks professional and fits the theme very well. The people who have tested have done so for a very limited amount of time and have therefore not had the time to fully understand all the functions in the system. When the game is finished there will be a tutorial going through all the gameplay elements, such as the controls, thereby remedying this problem.

Artificial intelligence

When looking at a project like this it could be easy to imagine that the AI would be the bottleneck, that it would be to slow, or simply too “stupid” to live up to the expectations of what the player requires of it. In reality the AI for games is very often simplified in order to better work with the player. The biggest problem is creating an AI that fits with the animations, sound cues and models created for the game, and making sure that the player can see what it is doing, and most of all, understands what it is doing.

The AI was written completely in C# from the start, and later parts of it where moved into Lua. Moving a small part (actions and the GOB algorithm) into Lua proved useful, but a more radical decision would have been to move the complete AI code into Lua. Doing this would for the sake of the framework and extensibility be sound, but at the time only moving a small part proved very time consuming, and it made more sense to keep working on what we had.

The impact as previously discussed for having the AI split between Lua and C# creates some overhead when sending information between them. This is fortunately not to bad as C# only will access the functions in Lua once for each time it needs to find an action (between 10 and 30 seconds interval), and Lua will at the end of its execution send information back to C#. Additionally C# will call two Lua functions for each action; one dealing with the action starting and one with the action stopping. But as it is, the number of calls is very limited and the overhead from this is a very small problem.

Requirements for the AI

- The game must contain AI for in-game characters
- The game AI must be able to simulate a subset of human emotion and interaction
- The game AI must feature path finding functionality for the crew members

A version of the AI was quickly up and running and it turned out that the AI would not really be a bottle neck. The hard part with a game and AI is creating enough assets (models, animations and sounds) to create that feedback the player needs to understand the AI and what it is doing. This requirement is specified separately but the project fell short of a lot of intended content, a direct consequence of building a game of this scale and ambition. Though it is hard to estimate, probably around 80% of what the AI is doing is not sufficiently showed to the player and therefore, in reality, void. The content side of the game is however intended to be expanded over the coming year and has a lot of base to feed of.

Currently, the AI has a lot of room for new animations and sounds to be utilized in order to better express what it is doing, which is what was required of it.

Feedback

Talking about the AI with people made it sound very promising. As mentioned earlier the game has a lot more AI then there is evidence of it. But as the exhibition had a lot of room for talking to the people watching, AI was a topic that would come up with some people. The overall impression of the AI and that it indeed was based on some real academic basis, seemed positive.

Rendering

At the start of the project, the general idea was that the game would some visual effects to help set the atmosphere and make the game more visually pleasing. The hope was to also get some other visual feedback, such as smoke, flashing lights and sparks. In retrospect, that was probably a quite reasonable goal but there was a lot of trouble just getting it to draw in a desirable way and getting the underlying logic to work. There simply was not enough time to do all those things.

The main problem was to get the various models to display correctly in the game. For a reason still unknown to us, the models came out of the modeling program with normals pointing inwards, meaning the model looked like it had been turned inside out. That meant that most of the time set aside for graphics was focused on getting it to work instead of implementing the things we wanted. Towards the end of the project we had to create a small application that would take a model file as input and invert all the normals so that the model would draw correctly.

Doing post process rendering was, technically, never an issue because XNA has a great system of render targets and texture rendering through sprites. The problem was to create or find a shader that did what we wanted.

As mentioned in the realization chapter, when loading a model an effect file is attached to each mesh of the model and that is then used when rendering it to the screen. This works well as long as all the techniques you need is in that one effect file. It definitely becomes trickier when you want to logically divide your effects into different files because for each time you draw a model with a different effect you have to switch file for all meshes in the model.

One of the things that were both good and bad when it comes to the actual rendering was the use of XACL (XNA Animation Component Library). By using it we also had to use the library's own draw function which meant we had no say in how the rendering was actually done and you more or less assume that it does it in a correct and efficient manner. But it also meant that if something went wrong it was a bit trickier to figure out what the problem was. Most of the time the library would say what was wrong but sometimes it could throw an error saying what was wrong and how to fix but then the correction would not have any effect. But in general, the library worked well for us when it came to rendering.

Animations

When we started working with XNA we were aware that it had no native support for animations and that we would have to make use of a third-party library.

While our choice eventually fell on the XNA Animation Component Library we had to evaluate a number of libraries before we could decide. It turned out that there are three prominent libraries, XACL (XNA Animation Component Library), XNAnimation and KiloWatt Animation. Unsurprisingly, all of these had their own problems and in the end we had to make a compromise between usability and functionality.

The first library we looked at was XNAnimation (12) which was developed by Bruno Evangelista. The reason why we chose this first is because one in the group had worked with it before and because it's a relatively easy library to use. And this was for a time the library we were going to use but it turned out it that animation blending, i.e. the process of playing two animations at once for one object, had been removed in the latest build. We reasoned that we needed blending to reflect the mood of a character, e.g. by blending a happy animation with a sad animation.

At this point we had found out about both the KiloWatt (13) and XACL and did a parallel evaluation of them. The KiloWatt library seemed to be the more comprehensive of the two and the developer of it had created it because he needed a library for his animations in XNA. However, for that very reason it seemed that he viewed documentation as unnecessary and any information about the actual functionality and usage of the library should be retrieved from the code itself. That did not really work for us so we quickly abandoned the idea of using it.

The last library we looked at was the XACL, which apparently stopped being developed in 2007. That was, however, not a problem and the library appeared to contain all the functionality we needed. Most of all, it had a whole tutorial section dedicated to explaining how it worked.

While we could, with more work, most likely have gotten the KiloWatt library to work we felt we did not want to linger too much on trying to get a third-party library to work as intended. And since the one we eventually chose had a comprehensive description of how to use it we could easily get it to work in an hour or two. But that is not to say it was without its own problems. Once we strayed from the path of the tutorial and tried to get our own models to work as they should we started having a lot of problems just getting the models to be drawn at all. Some problems also arose when we tried using custom effect files for drawing the models since the library required a skinning matrix and a special function for calculating the positions of the vertices as the model is being animated. This eventually resulted in us using the shader that came with the library but modifying it to a point where it matched our needs.

The result was not what we wanted but better than we expected. When in the planning stage it is easy to say that we should have animations for a number of things, but they quickly pile up and considering none of us had any real animation experience it became quite the challenge. In view of this lack of experience the resulting animations came out better than anyone had expected.

Working with third party code is always a hassle, but this has the benefit of being open source. The large benefit of a third party framework for the animations is that writing the code can be pretty complex. Handling skinned models and animations with blending can be a lot to learn. As a complete framework for this was used this was not a problem. Things that are a problem however is that everything does not always work as intended.

One thing that never did work very well was attaching objects to characters. Each character has an absolute transform in the world, so attaching a hat to this character first means moving it along with this position. The hard part is when the character also is animated, because this movement is not considered in the absolute transform. The animation framework has an interface for handling this which essentially is attaching something to one of the bones in the character; this calculates a position relative to the character that moves with the bone. All this sounds great but in practice it proved to be pretty complicated as attached items never did turn up where intended without being moved, further more with two attached items none of them worked at all.

Sound - effects and music

Unlike animations XNA has built in support for sound effects and music which is actually quite good. Granted there is manager for sound but that does not matter since there is no manager that fits all projects. Beyond the built in support there are at least one third-party library and the DirectX SDK provides the tool XACT for sound editing. In fact, that is the tool to use if you want to go beyond the simple sound effect class in XNA. In order to use the more powerful audio engine class, XACT is required since audio engine only uses precompiled sound- and wave bank files.

As discussed in the realization section, sound support was a very late feature of the game so we had no time to do a comprehensive sound manager. However, XNA made the process of actually adding rudimentary sound support extremely easy.

What was not so easy was working with XACT. Admittedly, it is a powerful tool that offers a lot of functionality but it is not a beginner's application. The biggest problem we had was that it was

impossible to play the sounds from XACT itself, instead requiring us to compile it and run the game to check if the sound worked. It's not that the program did not offer that functionality, the problem was that it could not connect to the audio server that was needed to play sound, and if it did not connect it crashed.

Sound and music never was a big issue during this project and considering the time put into selecting appropriate sound effects and music the result was fairly good. We tried to get music that was not typical Victorian music but more general classical sounding music, a lot of strings, piano etc. There is not that much to say about the sound effects since the only ones used are the turret sounds, hit sounds and simple explosion sounds modified to fit the source they originated from and to give some audio feedback.

Content

Working with content was not without problems to say the least. A lot of the problems can be blamed on inexperience but also different tools simply not cooperating as one would expect. One of the ongoing biggest problems was that the 3D model exporter insisted on not working correctly without the normals of each model being flipped, this led to the engine instead having to compensate for this, causing further problems with models without the flipped normals. In the end we managed to solve some of the problems by converting the models through a normal-inverter which we wrote in Java. This was certainly not a perfect solution but it got the job done.

In large the content managed to give the game the visual appearance intended for it, a steampunk aesthetic with a lot of wear and tear. A lot of time was put into getting to this point and one could argue that a lower focus on getting the visuals right could have freed up time to work with gameplay.

The content and visuals took a lot of time because of the goal of having a game in full 3D. There is no getting around the fact that working with full 3D as a goal is going to put a lot of burden on the content creators. Considering this the situation could have been worse as the top-down view puts less pressure on the level of detail required compared to a first person view.

Discussion

Some decisions can be hard to make, and there is always room for discussion afterwards on the matter of how each choice affected the rest of the project and whether or not it was the correct one. It is not about the blame game, but about learning by the mistakes and refining the good decisions. Also discussed in this section are improvements that can be made to the current code, without looking into future work.

Lua

Object oriented programming is well suited for big projects and help keep the code readable and easy to extend. Sometimes though, it can be overly complex with code spread out over many classes, often with many layers of inheritance. This is where scripting languages step in. They don't need to be compiled and the code is often very straight forward. For games this is very useful since the gameplay code needs to be extended and tweaked often. Using a scripting language also keeps the engine code separated from other code. Most scripting languages are easy to learn so the artists can in a sense become programmers themselves and code interfaces and maps without having access to engine code.

Lua is currently the fastest and most popular scripting language and it is used in a lot of games. When deciding what language to use the choice wasn't difficult. In order to get Lua to work with C# a library called LuaInterface (3) was used. This made things very easy since it has straight forward functions for executing strings and files as well as a simple way of registering functions for access in Lua by using C#'s reflection functionality. This removes the need to fiddle around with the Lua stack like you would have to in C and C++. It also makes exposing class member functions to Lua very easy. In fact, it is possible to expose entire classes, but that functionality was not used in this project since most of the C# classes were used as singletons and there was no need to create more instances from Lua.

One problem with scripting languages though is debugging. Lua can throw error messages if there are any syntactical errors but it's not possible to step through the execution of the code, and when calls are being made back and forth between Lua and C# things can get messy. Also, when Lua runs into an exception it stops executing and returns to the function that called it which can lead to mysterious bugs. There are a few debuggers such as Decoda (14) and RemDebug (15) but there was trouble with getting them to work with C#.

Other scripting languages include Python and JavaScript. Python was considered shortly but the reasoning behind choosing Lua was similar to the reasoning behind choosing XNA. It is fairly new and seems to be popular in the industry so learning it should prove useful in the future. In a later project Python may be chosen in order to explore the differences.

Graphics

In the beginning of the project we discussed the way the game was supposed to look like, since we had chosen the visual style of steampunk then we needed to show this style to the user as clearly as possible. The choice fell on to 3D with a few exceptions. The game was never intended to be visually stunning on par with professionally developed games today. Instead the game should be fairly good looking and work on old computers as well as new ones. The moment the group selected 3D as the way

to go, we realized that this would take a lot of work to get going compared to 2D which is pretty straight forward. In 2D it is easier to make a game without having to put down heavy work on graphics. However, now in the latter part of the project we really can see the benefits of our choice, many of the assets are now built and we have gotten much positive response from those who have seen the game. But the most important part of all is that by putting a lot of effort in creating a solid base, we can now continue to make the game into what we want. And this will definitely be something to show on future job interviews.

Besides the fact that 3D looks visually more intriguing than 2D we also must consider the knowledge we gained from doing the game in 3D. Many of us had not made a 3D game previously and most of us never had any real experience with 3D graphics whatsoever. This made this project a perfect opportunity to learn of how 3D suits and 3D game engines interact and how the content pipeline of a modern game actually might look like. Since most of today's titles are made in 3D this will naturally affect our ability to understand how different parts of a game creation work together into making the final product.

In 2D games, effects like shadows and lighting are almost always on a simple level. In a 3D game however, these can be made as complex as the creator wants. We saw our game as the perfect testing ground for implementing shaders, lighting techniques as well as advanced shadows. Even though many of these effects never made it the final version showcased in this course, it was always interesting to see how much performance was affected when an effect or shader was added. This was mainly due to the total control which we had over the content of the game. And performance is as known, always an issue of games.

Artificial Intelligence

With the AI as an integral part of the game, much time was spent on research and making important decisions for it. Many of these choices, as well as future work, are discussed below.

Threaded agents

At the point of threading agents they had already been working linearly for a while as a part of the main update. When running the agents in a linear fashion every part of the loop must be of a consistent length, as there are very clear time constraints on the main loop, the main one being that it must be quick enough to keep the frame rate up. This for example means that anything an agent does over time needs to be cut into slices that are run per frame.

When moving the execution of agents over to a thread everything they do is done completely in one run without looping, until they start a new action. Things that should run across several frames do this by looping inside of a while loop and simply sleeping each turn of the loop for some set amount of time.

The main advantage of the threading is disconnecting the agents from the main update loop. This means not having to do 60 function calls for the agents per second. Instead the threads are only ever started once, and then run inside of a while loop. Anything that should take a certain amount of time is then simply run consequently until done. Threading also means accessing a large amount of synchronizing tools like semaphores, which is the most frequently used for this game.

One of the clear downside is a certain loss of control over the execution. It can be hard to ensure that each agent is receiving the same amount of processor time which could affect the gameplay. To avoid this problem the appraisal of the agents was moved to the fixed update loop outside of the thread.

Planner: Look-Ahead VS Reaction Based

This is a hard comparison to make without having tried the alternative. From what has been read on forums, such as those over at (6), the difference is hard to perceive but points for both can still be made.

The real difference between them is that the look-ahead planner, much as the name suggests, looks at each possible action, evaluating these for the agent, and then picks the one perceived to be the best choice. The reactive planner instead looks “back” at the agent’s current state and picks an action only based on the current state taking into account what the agent needs. The latter approach is also often done with a reactive plan as the terminal for each reaction, which in itself will find an appropriate action based on its context.

Rational Thinking VS Control

When working with research and AI, emphasis is often put on creating something new and advanced, compared to with a game, where the most important aspect is the game being fun. This often gets in the way of the more ambitious designs and concepts, and results in more scripted AI that can be controlled to ensure a degree of quality. It is a risk/reward situation where executives often opt for the less risky version, and sadly the less complex.

In our case the situation was not as risky and creating a more complex AI was feasible as we did not risk commercial failure. The reward for creating a more complex AI is learning about it and finding what works well in the result. The hope is to find more interesting behaviors that seem to be more rational and intelligent. The downside, even in our case, is that the AI over time can be hard to predict and balance; things like every agent always ending up very sad or very happy because of the AI, instead of having a varied behavior that might be sought after.

Something that should not be underestimated is the value of being able to control the AI to a high degree when needed, while maintaining the "free will" for when it is useful. One such situation is the critical situations used in the game where, if the crew does not react fast and do what they need to, the ship may go under.

Future Work

The main thing is making the code path for an agent more general and less cluttered. The former wish mostly means working on how the same result can be achieved but with code that branches less. Branches are not a problem on today's CPUs, but start to become a problem when code is moved over onto today's consoles, which use strict in order instruction processing. More improvements along those lines are also simply optimizing the existing code.

Graphical User Interface

The use of Immediate Mode Graphical User Interface made it easy to quickly create an interface with the functionality we required. With the time constraints imposed on us in this project, this was very helpful. If, however, there had been more time we would probably have included more room for different functionality in the interface elements, and perhaps created an editor program for creating the interface screens with instead of specifying everything in text. That being said, there is no doubt in my mind that we would not have achieved an equal or better interface using a more traditional model.

The next step in the development is to implement the still unfinished menu screens. These include the ship menu, the tactics menu, which consists of two different screens, and a number of merchant screens using the same layout as the mission selection screen.

Conclusion

Almost half a year of work, over four hundred revisions and many hours spent with full focus on creating a game comes to an end.

The one thing nobody could say at the end of this project is that the group did not spend a lot of time working towards their goal. Not reaching it is a consequence of inexperience and unrealistic goals. As a conclusion the game created, and the work done is by the group viewed to be a success. The game that is running is at a point where content can be added with ease, and where different parts of the engine are running very well, instead of being poorly, but more quickly, implemented.

Part of the feedback from the exhibition was that despite the lack of gameplay, people understood the large amount of work undertaken to get it to where it was. The visuals of the game were very appreciated and people understood the theme the group had wanted to envision.

The gameplay is where the game was lacking but talking to people, they could still, with what was there, understand and see how the game would work down the road. But this is still being very tough on us, the game is running, decisions are being made by the crew, the entire bottom ship is modeled and a large amount of the rooms are decorated with furniture that the characters can interact with. There are critical situations that have the agents run to their battle stations and that have complete gameplay code running.

One of the bigger downsides, apart from the lack of content, is the lack of help for the player. It was never a very easy concept and game to show at an exhibition as the game requires a proper time investment to familiarize with and learn. With this in mind however there needs to be a tutorial, and more onscreen cues to help the player understand what is going on, and what the controls do.

Future work

Some work needs to be done before this game can be playable, and the group will continue to develop it. For the next milestone, focus is put on making the game more of an actual game, needed is a tutorial level where the user gets introduced to the game in a slow and systematic way. Also we want to further add levels and missions that users can choose from as well as more variety in mission style. The group also wants to enhance the visual style of the game by adding more shaders and more detailed graphics. Sound and music is to be enhanced in order to support 3d sounds as well as more variety of music. Furthermore, the interface needs to be finished and enhanced to allow for new gameplay elements.

The group has also discussed the matter of a homepage, and by creating a good looking homepage with ties to the visual style in the game we hope to interest users into try and playing the game. Through and through there are several different aspects that might be improved and further development is surely needed in order to make the game into the creation which we had in mind when design started.

Bibliography

1. **Park, Andrew Seyoon.** The Sims Review for PC. *GameSpot.* [Online] GameSpot, February 11, 2000. [Cited: May 26, 2009.] http://www.gamespot.com/pc/strategy/sims/review.html.

2. **Ward, Trent C.** Dungeon Keeper Review for PC. *GameSpot.* [Online] GameSpot, July 9, 1997. [Cited: May 26, 2009.] http://www.gamespot.com/pc/strategy/dungeonkeeper/review.html.

3. **Presti, Craig, et al.** Lua Interface. *LuaForge.* [Online] March 8, 2009. [Cited: May 26, 2009.] http://luaforge.net/projects/luainterface/.

4. **dastle, mnikonov.** XNA Animation Component Library. *CodePlex.* [Online] April 2, 2007. [Cited: May 16, 2009.] http://animationcomponents.codeplex.com.

5. CG Textures. *CG Textures.* [Online] May 26, 2009. [Cited: May 26, 2009.] http://cgtextures.com.

6. Game AI for Developers. *Game AI for Developers.* [Online] May 23, 2009. [Cited: May 26, 2009.] http://aigamedev.com/.

7. **Champandard, Alex J.** Living with The Sims' AI: 21 Tricks to Adopt for Your Game. *Game AI for Developers.* [Online] [Cited: May 26, 2009.] http://aigamedev.com/open/highlights/the-sims-ai/.

8. **Millington, Ian.** *Artificial Intelligence for Games.* s.l. : Morgan Kaufmann, 2006.

9. **Adesso, Vincet J.** Costa & McRae's Five-Factor Theory. *UW-Milwaukee: Something Great in Mind.* [Online] [Cited: May 26, 2009.] http://www.uwm.edu/~vince/psy205/wwwcourse.205.lec12.fivefactormodel.handout.htm.

10. **MacLeod, Kevin.** Incompetech. *Incompetech.* [Online] May 24, 2009. [Cited: May 26, 2009.] http://incompetech.com/.

11. SoundSnap. *SoundSnap.* [Online] [Cited: May 26, 2009.] http://soundsnap.com/.

12. **Evangelista, Bruno.** XNAnimation Library. *CodePlex.* [Online] July 17, 2008. [Cited: February 25, 2009.] http://xnanimation.codeplex.com/.

13. **jwatte.** KiloWatt Animation. *Enchantedage.* [Online] November 28, 2008. [Cited: March 5, 2009.] http://www.enchantedage.com/xna-animation.

14. Decoda. *UnknownWorlds.* [Online] Unknown Worlds Entertainment, April 22, 2009. [Cited: May 26, 2009.] http://www.unknownworlds.com/decoda.

15. **Mascarenhas, Fabio and Carregal, André.** RemDebug - Remote Debugger for the Lua programming language. *Kepler Project.* [Online] July 27, 2006. [Cited: May 26, 2009.] http://www.keplerproject.org/remdebug/.

16. **J, Tidwell.** *Designing Interfaces.* 2006.

17. **Cygon.** Capturing Keybard Input in XNA. *Nuclex.* [Online] January 10, 2009. [Cited: May 26, 2009.] http://www.nuclex.org/articles/capturing-keyboard-input-in-xna.

Appendix A – Project Plan

Project Members

- Richard Fredriksson - Project Leader and responsible for AI and animation
- Robert Krantz - Responsible for graphics, sound and tools
- Olof Millberg - Responsible for framework and Lua
- Fredrik Wendt - Responsible for 3D assets and animation
- Mikael Nilsson - Responsible for Interface and AI

Design document version

- Version one - 12/2
- Revision one - 5/3
- Revision two - 2/4

Project Report

- Project homepage - 30/4
- Final project report - 23/5

Swedish Game Awards

- Concept documents - Period one, week 6, 25/2
- Game - 25/5

System versions

- Low-fi demo - 25/2
    - Content (In order of priority)
        - One-Room ship
        - One model (with 2 different textures)
        - Interface + basic controls
        - Small behaviour AI
        - Walk and sit animations
        - Graphics, shading
        - Other animations
    - Focuses of the demo (In order of priority)
        - AI powered interaction
        - Animations and visual style of the game
- System version two - 27/3
- System version three - 24/4
- System pre-exhibition version - 1/5
- Final version (SGA) - 25/5

Weekly project plan

Week 6, 2-6/2

- Decide art direction as well as making sure that all group members are aware of the visual style of the game.
- Concept art and samples
- Work with the Framework, Sound, Graphics and AI to be started.
- Present final project idea, 5/2

Week 7, 9/2-13/2

- Charm presentation, focus on design and art, and a few gameplay examples so that we can describe our game in a fun way, 11/2
- Design document: Version one - 12/2

Week 8, 16/2-20/2

- Development work in all areas
    - AI: Create the foundation and create a basis for a sandbox to test the AI in
    - Framework: User-interface, visible and partly functional
    - 3D assets: A low-fi character with a few animations
    - 3D engine: Able to load models and animations, basic effects, camera and lightning

Week 9, 23/2-27/2

- Low-fi demo presentation, 25/2
- SGA: Concept document for Swedish game awards, 25/2
- Low-fi prototype feedback, 26/2

Week 10, 2/3-6/3

- Design document: Revision one - 5/3

Week 11, 9-13/3

- Exam week

Week 12, 16/3-20/3

- AI: Integrate AI into the main project. Create effects of actions and events for the characters
- Software framework: Look into different event passing systems
- Content: Finish the first floor
- Graphics: Look into different animation libraries
- Peer review of revised design document

Week 13, 23/3-27/3

- System version two - 27/3
    - Graphics: Billboards / thoughts / reactions / feedback
    - AI: Orders and actions
    - Interface + controls
    - Content: One character model with two looks
    - Content: The entire ground level textured + shading
    - Content: Furniture

Week 14, 30/3-3/4

- Design document: Revision two - 2/4
- Controls:
    - Graphics: list
    - Hit detection
- AI:
    - Triggers
    - Orders
    - Actions
    - Path finding
- Graphics:
    - Shading
    - Post processes
- Models:
    - Engine
    - Furniture
    - Beds

Week 15, 6/4-10/4

- Easter holidays, work work

Week 16, 13/4-17/4

- Re-exam week, work work

Week 17, 20/4-24/4

- System version three - 24/4
    - Graphics: Lightning working with correct normals
    - Graphics: Animations running with blending
    - AI: Work actions running
    - Interface: Captain and crew ready
    - Content: More more more

Week 18, 27/4-1/5

- Project homepage, 30/4
- System pre-exhibition version, 1/5
    - Graphics: Critical situations graphics
    - AI: Critical situations running
    - Interface: Mission and starmap
    - Controls: Selecting and giving orders
    - Content: More more more

Week 19, 4/5-8/5

- Exhibition

Week 20, 11/5-15/5

- Present and receive feedback on exhibition, 13/5

Week 21, 18/5-22/5

- Catch up with other courses
- Work with report, report, report

Week 22, 25/5-30/5

- SGA: Game, 25/5
- Finalize report and submit on the 27/5

Appendix B – Concept Ideas

Concept one

The first concept is basically a regular space combat game seen from a first person perspective. Unlike previous games in this genre, where the NPCs are basically just tools, the focus would be on the other characters in your squad and the interaction between them. In the game you take the role of a pirate or bounty hunter and fly on missions with the squad. To maximize the chance of success you would have to take command and give tactical orders. This could backfire though, because if the other characters in your squad don't trust and respect you they will refuse to follow your command. There wouldn't be any strict hierarchy in the squad but the characters would have different personalities. Some would be the leader type, some would be good team players and some would be lone wolves. In the beginning there would already be a leader type character in your group and you will have to increase your reputation before you can start giving orders yourself.

Concept two

Concept two is a trade and diplomacy game set in space. The player establishes trade stations on different planets and space stations. He can then establish trade routes between his different trade stations, as well as trade with other characters, all in real time. The player needs to develop his relationship with the other characters to be able to trade with them or establish trade stations, factories or similar on their planets. This is done through dialogue, gifts and other similar interactions. The non player characters will attempt the same thing the player is, namely to expand their trade empires by developing relationships with other races and factions. In other words, the game world will change if left alone. The game uses a tactical overview with iconic ships and similar, all done with symbolic distances.

Concept three

For the third concept our idea is a kind of the Sims in space, but without the furniture, no making babies, and no dressing up. The aim is to show the inside of a space ship from a top down 3D view. The game would involve two phases; it would start with a phase where the player would, with a given budget, recruit a crew. This would be done with a list of proposed people looking for work, and showing the player a file on each. The files would contain all the information about the person, his experience, what ticks him of and what makes him happy. So a rookie would have little experience, leaving his resume thin and not very informative, which would be reflected in the price to recruit him. That in contrast to the older experienced person, whom would cost a lot more, and a lot more would be known about him.

When a crew is recruited the player enters the second phase by going on a mission. The mission could be a mining run or trading goods in various parts of space. The control over the crew and ship is only through the captain, and done by giving orders to the crew. There is a big social aspect to this part of the game which is the crew members interacting with each other. One whistling might cause someone else to go mental, or one being happy might lift the mood of everyone around him. The trick will be for the player to think a lot when recruiting people, finding ones that fit together and cooperate well. If not, the player will quickly get problems with crew members going insane. When the mission is done, and the player is rewarded, the player can again recruit more people for the crew and upgrade the ship.

Appendix C – Swedish Game Awards Concept Document

Game name: Iridium - 20 million leagues above the sea

Team name: Dao

Game genre: Simulation

Multiplayer support: No

Platform: PC Windows

Participates in Mobile Games with Netbeans 6.0: No

Game play

The core game play is played from a top-down 3D view with pan, zoom and rotate capabilities. The player controls a spaceship by giving orders to crew members, assigning tasks and dealing with situations that may arise.

To efficiently control the crew the player must monitor how the crew members perform their duties and interact with each other. Scheduling the work shifts, making the right people work with each other, will be crucial.

The player will take his ship and crew on missions, such as trading, rescue and elimination missions. To make the missions more interesting situations can arise which requires the players immediate attention. These can be minor situations – e.g. the ship getting attacked or a flue spreading amongst the crew – or critical situations – such as an alien incursion on the ship.

Goals

There is no clearly defined main goal in the game. Instead the game is played as a sandbox type game where the player sets up his own long or short term goals. Short term goals could be to improve the ship and increase crew size while a long term goal could be to buy a new ship.

Visuals

The main visual theme of the game is *steampunk*, which can be described as a mix of Victorian era technology and fantasy/sci-fi. This is to give the ship and equipment a more mechanical feel, as opposed to the environments used in e.g. Star Trek or Star Wars.