Modeling Aspects of the Dynasty 3-D Game

Frederick C. Harris, Jr., Brent Devaney,
John Kenyon, Charles Robertson, Tchad Rogers

Department of Computer Science and Engineering
University of Nevada, Reno, NV 89557
{fredh, devaney, kenyon, robertso, trogers}@cs.unr.edu

Abstract

This paper presents the motivation as well as the development for Dynasty, a 3-D game. Dynasty is a multiplayer 3D first person shooter game that is based in Ancient China. Players compete against each other to collect the majority of chi (energy) in the world and restore it to order. This document also discusses the Software Requirements Specification and Design documents used to develop Dynasty. The specifications were created using the Unified Modeling Language (UML). The implementation of Dynasty followed a variation of the Unified Process (UP).

Keywords: UML, UP, 3-D game, requirements specification, use cases, class diagram.

1 Introduction

Dynasty is a multiplayer, 3-D, first person shooter game. Dynasty was loosely inspired by Bullfrog Production's 1996 game Magic Carpet. Dynasty is not just a clone of Magic Carpet but is an evolutionary step from its predecessor. Dynasty will be emphasized as a multiplayer game; the interactions between players and the world is the most important part of our system. Emphasis will be placed upon the communication between a server and multiple clients. Due to the limited time given to implement Dynasty, no single player option will be given at this time to help us concentrate on the game aspect of our program.

The Application Programming Interfaces (API) we will be using include: WIN32, OpenGL, DirectInput, fmod, WinSock 2.1, and the STL. The Win32 API will be used to create the main window for the game to run in. Initially it will also be used to create our menu system as well, but we hope to create a unique menu system using OpenGL. Details of the OpenGL system can be found in [3] and [6]. OpenGL will be used to display our world for the client gamers; objects in the world will either be billboarded or modeled using MD2 models. DirectInput will be used to handle client commands for the control of the character. Fmod will be used to play music and sound effects. WinSock 2.1 will be used for the networking side of the game. We will be using the Unreliable Datagram Protocol for our data transportation due to its speed. STL will be used as an easy way to handle data, especially the vector and string classes in the STL specifications.

The novel parts of Dynasty include: random terrain generation, a reliable client-server architecture, a multi-layer artificial intelligence (AI), a predictive physics engine, and all original models and sounds. The terrain will be generated based upon a determined seed that will be sent to all clients. This is to provide each player with a

unique experience every time they play the game. The emphasis placed on the architecture will allow the users to play without noticing lag (breaks in operations) in the game. The multi-layer AI is important to provide users with a reasonable challenge during the game. A predictive physics engine will be used on each of the clients to predict what the server is doing. At set intervals, the server will make sure each client is working properly and correcting any anomalies that appear. We will develop all original models and sound to further test our abilities and to avoid any royalty issues that may come up upon any distribution we may partake in.

This paper presents an overview of Dynasty and the design of Dynasty using the *Unified Modeling Language* (UML) [5]. UML is used to specify the use cases, scenarios, and class diagram. It was helpful in letting us visualize the design of Dynasty before implementation.

We have organized the remainder of the paper according to the following sections: In Section 2 we provide a brief overview of Dynasty. In Section 3 we present the Requirements Specification. In Section 4 we have provided the UML specification: use case diagrams, scenarios, and the class diagram. In Section 5 we look to the future and provide some future work for Dynasty. Finally, in Section 6, we conclude the paper.

2 A brief overview of Dynasty

Dynasty is a modern 3-D multiplayer game that is designed to be both simple to learn and use while simultaneously being deep and intricate in content. Dynasty is based on a client-server model, where the user has the option to easily create their own Dynasty server, or join a server that is already up and running. These options, as well as other settings, are available from the main-menu, which due to its industry-standard layout, is very user-friendly and simply to navigate. This main menu contains options for Join Game, Create Game, Options, and Exit. When a server creates a game, all computers on the Local Area Network (LAN) will be able to join the game and begin playing. Play will then continue until all the monsters in a level are destroyed or the players decide to quit the game.

The theme of Dynasty is going to be Ancient China. We have used [2] for a visual guide to the models we will be creating for our game. We will also use [2] for further ideas for integrating our theme into the game.

3 Requirements

We have determined the functionality that we wish to achieve for Dynasty. Our requirements are separated into three groups: functional requirements, game requirements, and non-functional requirements. We have included the most important requirements here. The *Functional* (and *Game*) *Requirements* list the behavior of the system, while the *Non-functional Requirements* list a number of specific properties the software should have as discussed in [1].

3.1 Functional Requirements

The functional requirements will provide Dynasty with a base functionality for the system. These requirements pertain to user specific interactions and behaviors. These are all denoted using the format <R#>.

R01 The system shall render the terrain with optimizing techniques such as quadtrees.
R02 The system shall generate the terrain using a random terrain generator.
R03 The system shall have an object dedicated to the physics of the game.
R04 The system shall allow a user to exit a game at any time without disturbing the

game by substituting a bot in their place.

R05 The system shall provide the user with a HUD (Heads Up Display) with information regarding health, chi, radar, and current spells.

R06 The system shall provide AI for the monsters and bots.

3.2 Game Requirements

The game requirements pertain to the functionality of Dynasty as a game. They provide constraints on how the system will interact with the user through the game world. These are all listed using the format <G#>.

G01 The radar shall provide the user with information regarding monster position, opponent position, and pagoda position.

G02 The player shall choose an element to follow through the game: earth, water, fire, metal, or wood.

G03 The player shall use chi to upgrade their levels.

G04 The player shall respawn 15 seconds after they are eliminated. They shall respawn at their pagoda.

3.3 Non-Functional Requirements

The non-functional requirements place constraints on how the system will be built. These are all denoted using the format <N#>.

N01 The system shall be written in C++.

N02 The system shall use a client-server based communication protocol.

N03 The system shall be created using pure WIN32 API functions to provide an optimized interface.

N04 The system shall use OpenGL for the graphics engine.

N05 The system shall use fmod for the sound system.

N06 The system shall use Direct Input (DX8) to manage input from the user.

N07 The system shall use the MD2 format for the models.

N08 The system shall use the context of ancient China for the themes associated with the game.

4 UML Specification

After laying down the requirements and specifications for the game, we were able to complete the UML design. Included are examples of the use cases and scenarios as well as an excerpt from our class diagram.

4.1 Use Cases and Scenarios

Using the UP approach and the UML notation, we created the use case diagram for Dynasty. Figure 1 shows the use case diagram. Figures 2 and 3 are example use cases. Figures 4 and 5 are two sample scenarios.

4.2 Class Diagram

Our class diagram is show in Figure 6. Due to the extensive span of our class diagram, it contains only those classes pertaining directly to the Entity Manager. It is complete but it gives a good idea about the complexity of the project.

5 Future Work

Given the limited amount of time to complete this project, there are many things that should be expanded upon before a final version of this product is made available to the general public this list includes:

- Greater optimization of data transfer between clients and server. Support for multiple screen resolutions.
- Single player campaign

Figure 1: Dynasty: Use Case Diagram

- Changing level of detail support for in game models and terrains.
- Greater technology tree for leveling and different character types.
- Varying Artificial Intelligence levels for computer controlled players.
- Enhanced intelligent behavior for creatures in the world.
- Additional player and enemy models, animations, and skins.
- Model and world deformations
- Dynamic and random ground texturing.
- A particle engine for spell and environmental effects.
- In-game text-messaging.
- An introduction animation.

Use Case: Change Spell
ID: UC13
Actors: • player
Preconditions: • The player is logged in to a server • The player is alive

Flow of Events:
1. The player presses the key bound to 'change spell'
2. The spells in that category appear above the current spell
3. If the player presses the 'next spell' button
  1. The next spell in the list cycles into the current spell slot
4. If the player presses the 'previous spell' button
  1. The previous spell in the list cycles back into the current spell slot
5. If the player presses the 'change spell' button again
  1. The selection system disappears, and the new spell is selected

Postconditions: • A new spell has been selected

Figure 2: Dynasty: The Change Spell Use Case

6 Conclusions

We have presented in this paper a computer game, named Dynasty, intended to be a fun and fully featured form of computer entertainment. What distinguishes Dynasty from other available computer games is its unique game play style and setting; and its open ended game play due to randomly generated worlds, AI, and multiplayer mode.

Use Case: Join Game
ID: UC01
Actors: 1. Player 2. Server
Preconditions: 1. The player has launched the game 2. A server on the same network is running
Flow of events: 1. The client moves to the 'join game' menu 2. The client broadcasts a request for all servers 3. All local servers respond, providing IP addresses and game information 4. The client lists the active servers for the player to select from 5. The player selects a server from the list 6. The client connects to the specified server 7. The server adds the player to the game
Postconditions: 1. The player is in the active game

Figure 3: Dynasty: The Join Game Use Case

Dynasty was created according to a very structured and planned manner, leading to a very structured and organized software package, which could be easily extended in the future. The main components of Dynasty at the specification level have been presented in this document. Possible extensions to Dynasty as a game and as a software program have also been outlined in this document. It is our belief that we have created a fun and unique game, with fundamental design concepts and infrastructure that could lead to a marketable title with some additional extension to all areas of the game.

Scenario for Use Case: Entity is killed
ID: UC12
Actors: 1. Server
Primary Scenario: 1. The server game logic detects that a entity has been destroyed 2. The server awards points to the player who destroyed the entity 3. The server sends all clients a message to remove that entity from the world
Secondary Scenario: 1. The server game logic detects that an entity has been destroyed 2. The entity happens to be a player 3. The server awards points for the kill to the appropriate player 4. The server tells all client to re-spawn the player at their pagoda

Figure 4: Dynasty: Entity is Killed Scenario

Scenario for Use Case: Promotion
ID: UC14
Actors: 1. Server 2. Client
Primary Scenario: 1. A player gains enough energy to be given a promotion 2. The server sends to the player's client the update 3. The server awards applicable spoils to the player

Figure 5: Dynasty: Promotion Scenario

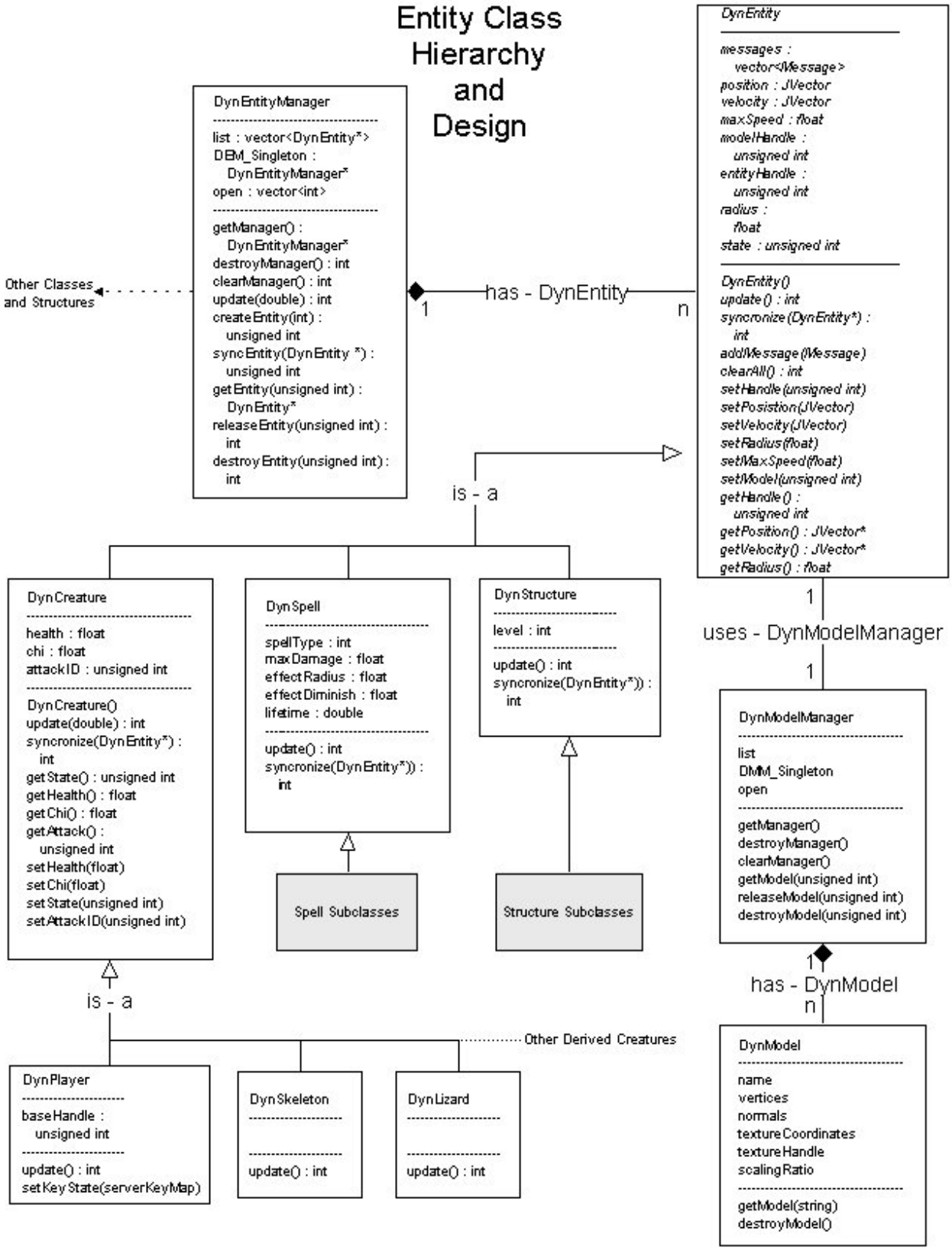

Figure 6: Dynasty's Class Diagram

References

[1] J. Arlow & I. Neustadt. *UML and the Unified Process: Practical Object-Oriented Analysis & Design.* Addison-Wesley: Great Britain. 2002.

[2] K. Buchanan, C. Fitzgerald, & C. Ronan. *China*. Crown Publishers Inc.: New York. 1980.

[3] J. Foley, A. van Dam, S. Feiner, J. Hughes, & R. Phillips. *Introduction to Computer Graphics*. Addison-Wesley: Reading, MA. 1997

[4] J. Rumbaugh, I. Jacobson, & G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley. 1998.

[5] OMG's UML Resource Page, available at: http://www.omg.org/uml/, accessed April 15, 2004.

[6] OpenGL – High Performance, available at: http://www.opengl.org/, Accessed April 15, 2004.