CHAPTER 3.

TECHNICAL GAME DEVELOPMENT I (IMGD 3000)

MARK CLAYPOOL[1]
WORCESTER POLYTECHNIC INSTITUTE (WPI)

Course Title: IMGD 3000: Technical Game Development I
Course College/School: School of Arts and Sciences
Course Department/Program: Interactive Media and Game Development
Course Level: Undergraduate
Course Credits: 3
Course Length: 7 weeks
Course Medium: Face-to-face
Course Keywords: game engine, programming, C , object-oriented, design patterns

CATALOG DESCRIPTION

This course teaches technical Computer Science aspects of game development, with the focus of the course on low-level programming of computer games. Topics include game engines, resource management, graphics and rendering, player input, collision detection, debugging, performance tuning and AI. Students will implement a game engine from scratch using C and make a complete game using their own engine.

COURSE PURPOSE AND OBJECTIVES

The purpose of the course is to provide a hands-on, in-depth exposure to technical (programming and software design) concepts related to game development, which a specific focus on the game engine. The course is intended to combine concepts learned in previous classes – such as data structures, algorithms, software design patterns, and game design – by applying them to a game and game engine. This combination and application of previous concepts provides both re-enforcement of topics learned earlier and also illustrates a holistic view of technical game development. In doing so, the course is to provide experience creating a large code base from scratch, as and experience working on teams on a game development project. Lastly, the course also intends to provide a bridge to

1. Mark Claypool (claypool@cs.wpi.edu) is a Professor of Computer Science and Interactive Media & Game Development at Worcester Polytechnic Institute. He teaches courses on technical game development, and does research in network game systems, with an emphasis on latency and games.

additional technical content on game development (i.e., senior-level game development and computer science classes).

The objectives of the course are for students to:

- Understand the structure and design of a game engine.
- Understand the trade-offs between complexity, fidelity, and interactivity in game engines.
- Demonstrate understanding of a game engine from a game programmer's perspective by extending a simple game.
- Use a game engine to create a complete, original game from scratch.
- Use iterative design and development practices to create a playable game.
- Understand how software engineering techniques can be applied to creating the parts of a game engine.
- Gain experience and develop skills in working in a team on a software project of significant size, with a short deadline.

COURSE CONTEXT

At WPI, this class is intended primarily for juniors seeking a B.S. degree in Interactive Media and Game Development. Such students are required to take at least 11 Computer Science (CS) classes en-route to the degree, and most have taken at least half of their required CS courses before IMGD 3000. While the only pre-requisite is a systems programming course, most students will have also taken at least introductory programming, objected oriented design, and an algorithms course. About half the students will also have taken computer architecture and software engineering, and somewhat fewer students will have taken operating systems, database systems and human-computer interaction. Nearly all students will have done one or more introductory courses in the game development process, game design, critical studies of interactive media and games, and storytelling and games.

COURSE PEDAGOGY

The goal of this course is to have students learn the fundamentals of a game engine and core game technology by building and engine (and a couple of games using it) themselves.

In doing so, the intent is for students to gain an in-depth understanding of a game engine – not only know how a game engine is implemented, but also *why* it is implemented the way it is, understanding choices required to achieve general purpose functionality to support a variety of games. Students should also gain an understanding of programming from the game programmer's point of view, being able to differentiate functionality in game code versus functionality provided by the game engine. This understanding is reinforced by making a game using full-featured, fully functional game engine.

The course arose from the desire to convey details on how a game engine works, both to provide the technical acumen to create and understand existing engines and to train better game programmers. The idea of building a game engine from scratch was inspired by MINIX, a Unix-like operating system (and precursor to Linux). With MINIX, a student could study an entire operating system, even building it if desired, since the whole system was about 12,000 lines of C-code. From this inspiration

arose the idea to create a MINIX-like experience for operating systems, but in this case for game engines.

Throughout the course, the requirements for a game and game code are discussed, design rationales are explored to figure out how and why a game engine might support the requirements, and students implement their designs, step by step, for a fully working engine. The use and understanding of the engine (as well as some additional game development topics) are book-ended by two games students make using the from-scratch engine, thus solidifying their use and understanding of a game engine.

As a bonus, students have the opportunity to use materials produced in the class (e.g., the engine and a game) as part of a portfolio, a showable record of what they can do. The built-from-scratch engine itself can be shown with one or more games demonstrating its functionality, and with the potential for an in-depth conversation (say, with a potential employer) about how, exactly, the engine and game(s) are implemented.

COURSE TEXTS, GAMES, SOFTWARE, AND HARDWARE

The only required book is:

- Mark Claypool. "Dragonfly – Program a Game Engine from Scratch", *Interactive Media and Game Development*, Worcester Polytechnic Institute, 2014. Online: https://dragonfly.wpi.edu/book/index.html

The book is freely available online in separate chapters. It can also be purchased (for a modest price) as a complete ebook, black and white print or color print.

Students need access to a computer (Microsoft Windows, Apple MacOS, or Linux are all supported) with a suitable programming environment. Supported environments include: 1) Windows with Visual Studio, 2) MacOS with XCode or Homebrew and the GNU compiler, and 3) Linux with the GNU compiler and a text editor. All the development software mentioned is freely available. Students are free to choose whichever platform they prefer and they only need to develop with one target platform.

Setup and configuration guides for each operating system platform are provided online: https://dragonfly.wpi.edu/engine/index.html#setup

COURSE ASSIGNMENTS

The individual projects can be found online at: http://www.cs.wpi.edu/~claypool/papers/dragonfly-projects. Point breakdowns and grading rubrics are also provided for each project.

Project 1 – Catch a Dragonfly

The first project is for students to get used to Dragonfly, typically their first exposure to a text-based, 2d game engine. Students work through a tutorial that has them make a simple, "stock" game using Dragonfly. This helps students better understand a game engine by developing a game from a game programmer's perspective, providing the foundational knowledge needed for building their own Dragonfly game engine in project 2, and designing and developing their own game from scratch with it in project 3.

In project 1, students:

- Visit the Web page (https://dragonfly.wpi.edu/) and briefly familiarize themselves with the contents. The Web page includes a download of the Dragonfly game engine (compiled – no source code), documentation with details on the classes and methods for the game programmer, and links to games and utilities that may be helpful for subsequent projects.
- Download the Dragonfly game engine for the environment of choice (Windows, MacOS, or Linux) and setup their development environment. This means ensuring all the needed external libraries are in place (e.g., SFML: https://www.sfml-dev.org/), installing the Dragonfly libraries and header files in an appropriate place (root/administrator privileges are *not* required), and testing a simple program to be sure development can proceed. The same basic setup is used for the students' own game engine development in project 2.
- Complete the tutorial, available from the Dragonfly Web page: http://dragonfly.wpi.edu/tutorial/. The tutorial has students build an arcade-style shooting game, called *Saucer Shoot*, where the player flies a space ship into combat against an ever-increasing number of enemy saucers. The tutorial has all the sprite files and sound files needed for development as well as working sample code for students to reference.
- Extend the Saucer Shoot game in a meaningful way by adding 10% or more functionality. For example, student's may add additional weapon types or enemies, health and/or multiple lives, a high score table, or something entirely of their own creation. The actual 10% extension done is up to each student, but s/he indicates what is done with brief documentation when submitting the assignment.

Students work alone for project 1. When done, students turn in a source code package with all code necessary to build their games, including header files and any needed additional sprites (depending upon their extension). In addition, each student includes a Makefile for compiling their game, a README file explaining their platform, files, code structure, and anything else needed to understand (and grade) their game, and a GAME file providing a short description of the additional 10% functionality extension to the Saucer Shoot tutorial game, including indicating the code written.

Grading

Breakdown

Tutorial – 40% : Doing the tutorial without any additional customization is worth just under 1/2 the grade. While doing the tutorial will provide a substantial amount of knowledge about the Dragonfly game engine, it will by itself demonstrate an in-depth understanding of the engine principles.

Customization – 50% : Extending or modifying the tutorial game with custom work is worth 1/2 the grade. Doing so will begin to flex technical muscles and show mastery of the basic concepts of the Dragonfly game engine. This is essential in moving forward.

Documentation – 10% : Not to be overlooked is including the documentation provided, as well as having that documentation be clear, readable and pertinent to the assignment. This includes the README described above as well as the GAME document. Having well-structured and commented

code is part of Documentation, too. Getting in the habit of good documentation is important for large software projects, especially when done in teams (e.g., project 4).

Rubric

100-90. The submission clearly exceeds requirements. The tutorial game works without problems. The custom extensions exhibit an unusually high degree of effort, thoughtfulness, technical ability and insight. Documentation is thorough and clear.

89-80. The submission meets requirements. The tutorial game works without problems. The custom extensions exhibit substantial effort, thoughtfulness, technical ability and/or insight. Documentation is adequate.

79-70. The submission barely meets requirements. The tutorial game may operate erratically. The custom extensions exhibit marginal effort, thoughtfulness, creativity and/or insight. Documentation is missing details needed to understand the contributions and/or to build the programs.

69-60. The project fails to meet requirements in some places. The tutorial game may crash occasionally. The custom extensions are of minor scope, or exhibit perfunctory effort, thoughtfulness, technical ability and/or insight. Documentation is inadequate, missing key details needed to understand the contributions and/or to build the programs.

59-0. The project does not meet many of the requirements. The tutorial game crashes consistently. The custom extensions exhibit little or no evidence of effort, thoughtfulness, technical ability and/or insight. Documentation is woefully inadequate or missing.

Project 2 – Dragonfly

In the second project, students build their own version of the Dragonfly game engine. Project 2 is broken into three parts: A) *Dragonfly Egg*, B) *Dragonfly Naiad*, and C) *Dragonfly*, that build upon each other to end with a fully functional, full-featured game engine. Since it is critical that game engine code be easily understood (from the game programmer's perspective) and, equally importantly, robust, the three projects are structured such that completing parts A and B provides for a fully functional, if somewhat limited, game engine. This level of proficiency enables students to proceed to project 3, where they make a game using their engines. Completing part 2C provides for a full featured game engine, with functionality that makes it easier to create a broader range of games.

For timing and grading, the due dates are staggered so that most of the time is allocated for part A and part B, but there is still time for completing part C for the top students in the class. In addition, points are allocated such that completing part B is sufficient for earning a "B" grade for project 2, while fully completing part C provides an opportunity for earning an "A" grade for project 2.

Students are informed that it is much better to have tested, trusted robust code that only implements part A and part B than it is to have buggy, partially working code that attempts to get into part C. Since students make use of their own engine for project 3, most tend to heed this advice.

Students work alone for all of project 2. While group work is important for many aspects of software engineering, including game development, developing the engine solo ensures students have complete

and deep understanding of both the game engine and the programming skills needed to develop it — there is no way to "hide" behind a more experienced teammate. That is not to say students are alone, however — discussing the project with other students is encouraged (e.g., via Slack or Discord), even for help in debugging each other's code. The line is drawn at not allowing sharing of code in that each student must write all the engine code him/herself.

All development is done in C . Students are expected to be familiar with C from earlier computer science classes, but are not expected to be experts in the language. While development is done as "homework" outside of class, the requirements and design of Dragonfly are presented in class, with discussions of design rationale, implementation choices and alternatives, and more advanced features.

Individual classes, with high-level descriptions of attributes and methods, are provided in the project writeup at: http://www.cs.wpi.edu/~claypool/papers/dragonfly-projects/proj2/.

Project 2a – Dragonfly Egg

Part A of the project is to construct the foundations of a game engine that provides the following capabilities:

1. Game initialization: Start and stop gracefully.
2. Logging: Write time-stamped messages to a file.
3. Object support: Add and remove game objects. Objects support 2d game world positions for objects.
4. Game loop: Run a game loop with: a) A fixed update rate (e.g., 30 Hz), and b) updates sent to all objects each loop

To implement this functionality, students develop, code and test about a dozen base classes.

No visual depiction of the game is required for part A. Instead, all output is done via printing to the screen or to a log file via the logfile manager functionality built into the game engine. As suggested above, at the successful completion of part A, students do *not* have a game engine. Instead, they have a robust, foundational code base they can build upon to get a functional game engine in part B.

Project 2b – Dragonfly Naiad

Part B is to continue construction of the game engine, each student using their own code base from part A, adding the following additional capabilities:

1. Output: Support 2d, text characters with color. Provide a clean refresh each game loop.
2. Input: Accept non-blocking keyboard and mouse input. Send input to interested game objects.
3. Object Control: Support "velocity" for game objects with automatic updates.
4. Collisions: Provide a "solid" attribute for game objects. Detect collisions between solid objects. Send an event to both objects involved in a collision.
5. Misc: Provide deferred, batch removal of game objects. Provide support for an "altitude" attribute for game objects to support layered drawing. Signal game objects that travel out of

bounds with a custom event.

All of the above capabilities must be thoroughly tested, bug-free and ready for a game programmer to make a game (the students themselves, in project 3).

Project 2c – Dragonfly

Part C is to continue construction of the game engine, each student using their own code base from part A and part B, adding the following additional capabilities:

1. Sprites: Provide multi-character frames. Associate one or more frames with a sprite. Play sprite frames in sequence to achieve animation. Support "slowdown" of animation to less than one frame per game loop.
2. Resource Management: Read sprite data from files. Provide bounding boxes for game objects. Allow game objects to be larger than a single character (for movement and collisions). Associate bounding boxes with sprites.
3. Camera Control: Allow the game world to be larger than the screen, providing a "viewport". Enable free viewport movement around the game world, including the ability to follow one object (e.g., the player's avatar).
4. View Objects: provide an alternative (to game objects) object that supports "heads-up display" functionality for UI elements.

As for project 2B, all of the above capabilities must be thoroughly tested, bug-free and ready for a game programmer to make a game (the students themselves, in project 3).

For each part, students turn in a package with all code necessary to build their game engine, including header files and a Makefile for building their engine. Game programmer code (i.e., code someone would write using their engine) is required to demonstrate the full functionality of what they have built (so far). This can be more than one program, if needed. Documentation is required to explain the platform, files, code structure, how to compile their engine and game code, and anything else needed to understand (and grade) their game engine.

Grading Rubric

Below is a general grading rubric:

100-90. The submission clearly exceeds requirements. The functionality is fully implemented and is provided in a robust, bug-free fashion. Full functionality is clearly depicted in one more more samples of game/test code and through clearly provided logfile messages. Documentation is thorough and clear.

89-80. The submission meets requirements. The basic functionality is implemented and runs as expected without any critical bugs. Functionality is depicted in one more more samples of game/test code and through logfile messages. Documentation is adequate.

79-70. The submission barely meets requirements. Functionality is mostly implemented, but may not be fully implemented and/or may not run as expected. There may be a few bugs, none critical.

The functionality is depicted in sample game/test code, but full representation is not shown. Documentation is inadequate, missing key details needed to understand the engine and/or to build the programs.

69-60. The project fails to meet requirements in some places. The game engine is missing critical functionality and/or what is there has bugs. The engine may crash occasionally. Game/test code demonstrating the engine is missing or incomplete. Documentation is clearly inadequate, missing key details needed to understand the engine and/or to build the programs.

59-0. The project does not meet core requirements. The engine cannot compile, crashes consistently, or is lacking many functional features. The sample game/test code is missing or incomplete. Documentation is woefully inadequate or missing.

Project 3 – Dragonfly Spawn

In project 3, students use the Dragonfly game engine they built in project 2 to make their own, original game from scratch. The end result is expected to be a robust (bug-free), playable, and balanced game (it may even be fun).

Like a typical large game development effort, the project is broken into several milestones: plan, alpha and final. Each milestone is submitted and graded separately, while all apply towards the total project 3 grade. The intent of the milestones is to provide production guidance to yield a fully-functional, complete, playable game built with their own game engine.

Students work in teams of two for project 3. Students are free to partition the work among the team as they see fit, but all team members are encouraged to help (say, with design and debugging) and be knowledgeable (in terms of how the game code executes) for all parts of the game.

Development must be in C using their game engine from project 2. Under exceptional circumstances (e.g., both partners not completing project 2b), students are allowed to use the pre-made Dragonfly engine from project 1. No engine source code is provided, however, only the pre-compiled engine.

Plan

Student teams provide a game plan within the first 1/4 of the project. The plan document provides a detailed description of the game they plan to build, including the technical challenges it entails, a bit about any significant artistic aspects of the game, and the timeline to successfully complete development in the time provided. In planning, students are asked to draw upon experiences from other classes (e.g., other game development courses), to inform the creation of the plan document. While the actual length of the plan is not a requirement, as a guideline the plan is expected to be approximately 2-3 pages – much less and students have probably have not supplied enough details.

For the plan submission, students turn in a written document.

Alpha

At alpha stage, the student games have all of the required features implemented, but not necessarily working completely correctly. Game code must be tested thoroughly enough to eliminate any critical gameplay flaws, but minor bugs or glitches may be present.

Games must compile cleanly and be runnable, even if all aspects of gameplay are not available from one program. Separate features of the game may be demonstrable from separate game code programs (e.g., separate game programs illustrating a kind of weapon or a specific opponent).

Games are likely not yet fully balanced nor the levels designed for all experiences (beginning to advanced) of the game player.

Games may contain some placeholder art assets. For example, in the alpha release, a simple, non-animated square may be used for an opponent with the intent of creating a figure and frames of animation for the final version.

For the alpha submission, students hand in a package with all the source code necessary to build their game engines and their games. All header files must be included, as well as Makefiles for building the games.

Final

The final version of all games has all game content complete – design, code and art. Games must be tested thoroughly for bugs, both major and minor, removing all visual and gameplay glitches. Game code must compile cleanly and be easily runnable. Upon startup, instructions for the player on how to play must be readily available, and with clear indications on how to begin play. Gameplay must be balanced, providing appropriate difficulty for beginners and/or early gameplay, with increased difficulty as the game progresses. Games must have a clear ending condition (i.e., winning or losing) and the player must be able to exit the game easily and cleanly.

For the final submission, students submit their engine and game, with necessary support files and Makefiles. The typical READMEs are required, as well as DESIGN documents providing all the details in the plan, but updated to reflect the games as actually built. For example, the functionality, milestones and work responsibilities need to be updated from the plan to reflect the development. Major deviations from the original plan must be be noted.

Grading

Under most circumstances, both team members receive the same grade. Students will, however, be given the chance to provide your own feedback (e.g., a grade) on their project and on their partner privately to the professor when the project is complete.

Note, for the final release, the grade will be based on the version of the project submitted online by the due date, not on the version presented in class.

Breakdown

- *Plan* – 10%
- *Alpha* – 25%
- *Final* – 40%
- *Design* – 10%
- *Presentation* – 10%

- *Promotional materials* – 5%

Rubric

100-90. The submission clearly exceeds requirements. The game is fully implemented, playable from start to finish in a robust, bug-free fashion. Gameplay is balanced throughout, providing appropriate difficulty for beginners while getting more challenging as the game progresses and/or the player obtains skills. Instructions are provided in-game for how to play. The required documentation (plan and design) is thorough and clear. The group presentation is well-organized, well-rehearsed and introduces the team and game in a fun, yet professional manner. The promotional material is clear, complete, and very presentable.

89-80. The submission meets requirements. The game is implemented and playable from start to finish, in a mostly bug-free fashion. Gameplay is mostly balanced, providing adjusted difficulty for beginners and more advanced players Instructions are provided in-game for how to play. The required documentation (plan and design) is complete. The group presentation is organized, rehearsed and effectively introduces the team and game. The promotional material is clear, complete, and presentable.

79-70. The submission barely meets requirements. The game is implemented and playable but may have some minor bugs or glitches. Gameplay is balanced, but may have some aspects that are too easy or too hard for either beginners or advanced players. The required documentation (plan and design) is intact, but may be unclear and/or missing some sections. The group presentation introduces the team and game, but may suffer from lack of preparation or organization. The promotional material is presentable, but may have shortcomings in appearance or substance.

69-60. The project fails to meet requirements in some places. The game is playable, but has minor to moderate bugs or glitches. Levels are incomplete or gameplay is unbalanced, and there are aspects that are too easy or too hard for either beginners or advanced players. The required documentation (plan and design) is unclear and incomplete or missing sections. The group presentation is not well-organized and suffers from lack of preparation. The promotional material is incomplete or not very presentable.

59-0. The project does not meet core requirements. The game may not compile cleanly or has major bugs. Levels are incomplete or not even playable. The required documentation (plan and design) is incomplete or missing. The group presentation is poorly organized and suffers greatly from lack of preparation. The promotional material is missing, or incomplete and of low quality.

COURSE ASSESSMENT

Projects (*80%*) – The bulk of the course grade involves programming projects. The grading policy for each project is provided at the time the project is assigned. In general, for each project there is a basic objective for the majority of the assignment points. There may be an extended objective for demonstrating additional work and understanding. Projects, including all data and source code, as appropriate, are to be turned in online as specified in the writeups.

Quizzes (*15%*) – There is a quiz at the start of almost every class. These are designed to test important

class concepts from the previous class(es), especially concepts that may not have been adequately demonstrated in the programming projects. Quizzes are closed book and closed notes, unless otherwise indicated. All quizzes have an equal weight, except for the two lowest scores which are dropped.

Participation (*5%*) – Showing up to class is worth much of a class participation grade, but so is being engaged in the class material through asking and answering questions and participating in group exercises.

EXPANDED COURSE OUTLINE

The below outline assumes a 2 hour class taught twice per week.

In general, each class is about 1/2 lecture and 1/2 group work plus discussion. Lectures are short to medium – 5 to 30 minutes long – and are interspersed with active learning exercises. These active learning sessions have the students work in small groups to engage with game engine design and similar concepts talked about in the lecture. Students start by working solo, then in pairs or slightly larger groups, finally bringing material together with class discussions, facilitated by the course instructor and teaching staff.

Module 1 – Introduction

Objectives

1. Provide an introduction to the course as a whole.
2. Give a definition of a game engine, including examples.
3. Provide reference to foundations for building a game engine.

Materials

Chapter 1

- Book: https://dragonfly.wpi.edu/book/pdfs/1-introduction.pdf
- Slides: https://dragonfly.wpi.edu/book/slides/ch1-introduction.pptx

Length

2 classes

Module 2 – Setup

Objectives

1. Introduce software and libraries used for development in the course.
2. Guide student in setup of the development environment they will use for all projects.

Materials

Chapter 2

- Book: https://dragonfly.wpi.edu/book/pdfs/2-setup.pdf
- Slides: https://dragonfly.wpi.edu/book/slides/ch2-setup.pptx

Length

1/2 class

Module 3 – Tutorial

Objectives

1. Provide an overview of Project 1 (Catch a Dragonfly).
2. Give some hints for setup and planning for Project 1.
3. Motivate text-based (ASCII) graphics for use by the engine and games.

Materials

Chapter 3

- Book: https://dragonfly.wpi.edu/book/pdfs/3-tutorial.pdf
- Slides: https://dragonfly.wpi.edu/book/slides/ch2-setup.pptx

Length

1/2 class

Module 4 – Engine

Objectives

1. Provide the design and design rationale for the Dragonfly implementation.
2. Give "tips" on C coding, game engine implementations and large-scale software development.
3. Provide an overview of Project 2 specifics (Dragonfly Egg, Dragonfly Naiad and Dragonfly) at appropriate times.

Materials

Chapter 4

- Book: https://dragonfly.wpi.edu/book/pdfs/4-engine.pdf
- Slides: https://dragonfly.wpi.edu/book/slides/ch4-engine.pptx

Length

8 classes

Module 5 – Misc

Objectives

1. Provide an overview (and in some cases details) on technical game development topics that are not covered in the Dragonfly implementation.
2. Where appropriate, provide specific details on how the topics would be implemented in Dragonfly.

Topics

- AI for games
- Finite State Machines
- Pathfinding
- Scene graphs
- Game engine performance
- Testing

Length

3 classes

Module 6 – Closing

Objectives

1. Have students give a formal presentation of their final games (Project 3).
2. Use the remainder of the class as a "game fest" where students mingle, playing each others' games.

Length

1 class

Module – Groupwork

Groupwork sessions are 5-20 minutes in length, total. Generally, students do: 1) introductions and icebreakers, 2) think of the answers themselves, 3) share their answers with each other, and 4) report back to the class with their groups answers during discussion.

Objectives

The goal of the groupwork is to:

1. Help students meet and work with other students in the class and IMGD program.
2. Promote active learning.
3. Break-up class lectures.

Materials

- Online: https://web.cs.wpi.edu/~imgd3000/b20/groupwork/
- Note, answer keys for the instructor can be obtained by replacing the handout.html string in the URL with key.html. e.g., https://web.cs.wpi.edu/~claypool/papers/imgd3000-chapter-21/groupwork/2-log-manager/key.html

COURSE BEST PRACTICES

Students tend to come into the course with a variety of programming backgrounds. Most have been exposed to C but are not necessarily proficient. Most are well versed with Java but have not necessarily implemented a medium or large sized project in it. In order to help boost students to the same level, the student teaching assistants (TAs) offer recitation-type sessions (once per week) during the first three weeks of the course on topics such as: differences in C versus Java, Object-oriented Design, Development Environment Setup, and Testing and Using a Debugger. This can be a good professional development experience for the graduate student TAs, too, and help get students used to interacting with the TAs.

Each class usually starts out with a daily quiz (about 10 minutes) which is gone over immediately after. This serves as both an assessment and a quick review of material covered in the previous class.

Students have created a variety of innovative games using the Dragonfly engine, from RTS and platformers, to puzzle and horror. A sample of their game trailers can be found online: https://dragonfly.wpi.edu/games/index.html#trailers. These same trailers are shown to students before they make their own games (Project 3) in order to provide inspiration for the breadth and depth possible for their game implementations.

Extending Dragonfly has proved useful for seniors doing their capstone projects and graduate students in their research. Given that it is a small code base but still provides a full-functional engine, it can be fairly easily extended to study networking aspects (e.g., latency compensation) and performance aspects (e.g., scaling with numbers of objects).

An online version of this course has been taught successfully (as measured by comparing objective course outcomes for the online versus inclass versions). In general, the online version involved synchronous components for the lectures taught over Zoom, with groupwork done in breakout rooms. All class Zoom sessions were recorded and made available online for students that missed the class. Supplement help sessions were provided online via synchronous Zoom and screen sharing. It seems likely that the recorded lectures could be provided in a "flipped" manner, leaving synchronous class time for project questions and the groupwork.

FUTURE COURSE PLANS

Rather than implement the entire Dragonfly engine from scratch, engine components can be provided (e.g., the LogManager) in compiled form, saving the students implementation time. This time savings can be used by students to implement other technical game aspects, such as pathing or enhanced collision detection.

Pathfinding is currently taught as a concept only. Future plans may include incorporating pathfinding into the engine. This would require students to program an A* algorithm – an industry standard.

Scene graphs are currently taught as a concept only. Future plans are to have students implement scene graph elements (e.g., a quadtree) and evaluate the performance.

The two dominant forms of collision detection in game engines are *overlap testing* and *intersection testing*. The course currently describes overlap testing, which the students implement in their engines. Future plans are to incorporate intersection testing and have students evaluate the tradeoffs versus overlap testing.

The Dragonfly engine can be extended to 3d by extending the 2d Vector class that is part of the engine. Such an extension would stretch students to enhance the graphics in the display manager, useful for including advanced graphics topics.

Networking and general online connectivity is increasingly important for games. Future plans are to incorporate networking and networking materials into the second course in this sequence, *IMGD 4000: Technical Game Development II*.

REFERENCES

Mark Claypool. "Dragonfly – Program a Game Engine from Scratch", Interactive Media and Game Development, Worcester Polytechnic Institute, 2014. Online: https://dragonfly.wpi.edu/book/index.html

Andrew Tanenbaum and Albert Woodhull. "Operating Systems, Design and Implementation – The MINIX Book", 3rd edition, Pearson, 2007. ISBN 0131429388. Online: https://tinyurl.com/1izggtsf