Game Programming and XNA in Software Engineering Education

Oliver Denninger
FZI Research Center for Information Technology,
Germany
denning@fzi.de

Jochen Schimmel
University of Karlsruhe,
Germany
schimmel@escde.net

Abstract

Game programming can help students gain practical experience with software engineering. Game programming covers a wide range of software engineering topics – including algorithms, data structures, team work, and software tools. Unfortunately, game programming usually involves many repetitive and time consuming tasks such as accessing hardware resources and managing game content. In this paper we present our experiences utilizing game programming for project courses. We discuss two conceptually different game project courses along with the results. The XNA framework relieves programmers from many of the tedious tasks mentioned, allowing them to develop a feature complete game and to gain experience with the process of software development. Students were so fascinated by the subject that they spent more time on the courses than required.

Keywords

Game Programming, XNA, Software Engineering Education

1. Introduction

Regardless of how and when programming is taught in academic curriculums, students do not gain real world programming experience in lectures. To deal with this problem many universities use project courses or capstone projects. Such projects are also recommended by the Curriculum Guidelines [1] published by ACM and IEEE. Unfortunately most real world problems are too complex to handle them in time and resource limited academic curriculums. Ghezzi and Mandrioli [2] discuss this problem and suggest choosing projects that can be reduced until they fit into the limits of academic courses. In our opinion game programming perfectly meets this requirement.

In the next two sections we describe our motivation for utilizing game programming in software engineering education along with the reasons for choosing XNA as programming technique. The main section describes the details of two project courses we held.

We use game programming only for project courses, teaching game programming skills is not our primary intent. In the classification of Sung, Shirley and Rosenberg [3] – describing different categories of courses with game programming as topic – our courses belong to category three, that is, courses with the primary intent of general computer science education.

2. Why Utilizing Game Programming for Education?

Exercises, as they are typically used in lectures, are not adequate to train skills for real world projects. Special project courses or capstone projects are necessary to give students the chance to work on large-scale projects over a period of several months. However, it is difficult to find interesting and suitable topics for projects. We think that game programming is such an interesting and suitable topic. The complexity of a game can easily be adapted to a project course.

Developing modern computer games covers a broad cross section of computer science, including mathematics, physics simulation, algorithms, data structures, concurrency, and graphics. Students may need to apply, for example, sorting algorithms or data structures they know from lectures. But game programming is not limited to algorithms; there is also user data – the game content – to be treated.

As most students are very enthusiastic about game programming, the motivation during the project course is quite high. According to our experience, students are eager to learn and work more than required.

Another advantage of game programming is that students have broad domain knowledge [4] – as most of them have several years of experiences in gaming. This background enables them to directly evaluate their results by comparisons with commercial games they know. Our courses showed that students are self-critical given their exposure to professional games. In courses with project definitions from industry partners only few students would have enough domain knowledge to evaluate their own work.

Game programming includes many algorithms and technologies which are not part of typical programming or algorithms courses. Therefore, students need to be willing to learn these technologies. Learning unsupervised is an important skill as the fast evolving computer technology will challenge them their whole life.

Furthermore, as performance is a core requirement of game programming, profound programming skills are necessary. Students in game programming courses not only have to be familiar with basic programming; they also need to estimate the cost of using them. In most applications, a delay of a few milliseconds won't be an issue. But in gaming milliseconds decide whether a game runs smoothly or not. Students get direct feedback for the quality about their code.

3. Why Using XNA?

In order to give students a „real-world experience“ in the project courses, our goal was to choose programming languages and APIs as close to professional game development projects as possible. Most current games are being developed for Windows using C++ and DirectX or OpenGL. The problem about C++ is that it requires a lot of experience with the language itself to write good code. But most students don't know C++ as Java is the most common language in software engineering education today. So we decided to use a more convenient language such as C# or Java. Besides the programming language, we had to choose a game and graphics API. Several Java wrappers like JOGL [6] provide access to OpenGL. But they cannot change the general characteristic of OpenGL being a pure graphics API. The benefits and drawbacks of Java / OpenGL are described in different books – for example [5].

We decided to use DirectX, as it is more powerful in terms of gaming tasks. Although DirectX has been designed for C++, it can be used with C# and other .NET languages. To further integrate DirectX into the .NET architecture, Microsoft published the “Managed DirectX” (MDX) API – a .NET layer built on top of DirectX. The “XNA Framework” [7] is the successor of MDX. It enables gaming and programming enthusiasts to easily write their own games for Windows. In contrast to MDX, XNA supports many standard tasks, leading to less code to be written by the developer for the same result. The so called content pipeline – responsible for uniformly importing any kind of game content (models, textures, sounds, shaders, etc.) – is an example of saving much work for developers. When starting a new game project the XNA Game Studio IDE – an extension to Visual Studio – automatically generates a basic game loop providing update and draws methods with proper timing. Adding a few lines of code to the project is sufficient to display a first figure on the screen without any thoughts about the graphic hardware or refresh rates. This is very helpful especially for beginners. A brief description of the differences between MDX and XNA for developers can be found in [8]. Generally speaking, using .NET libraries is easier than programming with the C++ Standard Template Library (STL).

At the moment we regard the C# / XNA combination as the most powerful game programming API for quickly achieving first results. The possibility to deploy games on Xbox 360 – requiring only a few code changes – is a special highlight of XNA.

4. Contents of the courses

Our two projects courses had completely different formats. While the first course took two weeks with a full day schedule, the second course took four months with a scheduled effort of four hours a week. Due to the short duration, the first course had to be well prepared and planned. We decided to implement the complete game before the actual course and to schedule the subject of each day. In contrast the long duration of the second course permitted a more flexible approach. The whole game was written during the course. The students planned the development and decided the game features, as part of the course.

The different formats also directly influenced the content of the courses: the first course focused on implementation of gaming algorithms and data structures, while the second course had a broader approach incorporating development process and creation of game content, such as 3D models.

4.1 First Project Course

The first project course itself was preceded by a short C# test, as more students applied than we could handle. Due to resource limitations, we accepted twelve students only. Preparation had a time limit of three months, including planning and implementation of the game as well as creation of presentations for the project course.

Preparation

To handle the considerable workload before the course we had a preparation team of six graduate students. As graduate students know best which topics are problematic and need to be stressed, it is a good idea to involve them in the creation of a project course of that kind. This team did not only prepare the project course, but also lectured and guided the participants throughout the course.

The first planning step was to decide the genre of the game. We chose to write a real-time strategy (RTS) game [9]. RTS games have the benefit of being fairly complex in terms of required data structures and algorithms. If we had chosen a first person shooter, the main focus would have been on computer graphics, including many non-programming tasks. Given course duration of two weeks, we could not handle such tasks.

The preparation team created a project plan and decided which features should be included in the game. Furthermore, the team collected the necessary literature and created an object model for the game. Examples of helpful books are [10], [11] and [12].

Each of the ten course days had a subject of its own, for example game mechanics, AI, or shader programming. In the morning we lectured to the students, explaining basic concepts and showing possible solutions. After that students started implementing what they had learned by reimplementing parts removed from the code. The completion of the game happened step by step. For instance, graphics was the first topic we covered, as this enabled us to visualize the later steps immediately.

Technical Details

The participating students worked using pair programming [13]. We chose pair programming to increase the quality of the code and decrease time spent on individual tasks [14]. As many game algorithms are rather complex, we wanted to enable the students to track problems down as soon as possible.

In order to visualize our game, we had to establish a graphics engine. Besides the fact that different noncommercial graphics engines are available for free, we decided to write our own. We did this for the following reasons: In the short time available for preparation, we were unsure if we could handle a huge and detailed graphics system available for multipurpose game programming. So we thought that we could proceed faster, by writing an engine designed exactly for our game. As we were able to keep our schedule, this decision was the right one. Another reason was that we wanted to explain how a graphics engine works. This can more easily be done using a small engine in which the reason for every single line of code is known.

The main focus on the project course was on game mechanics. These define the behavior of the game, the implementation, and the possibilities of a game designer, developing the “rules of the game”. Game mechanics includes tasks such as management of units and buildings (several hundred can occur in a game at a time), path finding, processing of unit logic, and collision detection. A problem students had to solve was, for instance, how to detect whether a unit can see another one or not. As these checks have to be done frequently, performance is an important issue. Another problem is how to keep track of finite state machines (FSM) for each unit. These FSMs are important, as they make up the basic behavior of a unit. Furthermore, the FSMs can store a sequence of behavior steps, enabling us to perform features such as units patrolling the map.

Sound is a feature every game must contain. We did not spend too much time on this task, as XNA takes control of all basic tasks and we did not have the sound resources to implement complex features. Nevertheless, problems such as how to decide which sound in the environment should be heard (in correspondence to the view) were discussed and solved.

For our user interface, we wanted to imitate the handling of popular RTS games and included a minimap. A unit can be selected by clicking on it and moved by clicking on the target. Groups can be selected by drawing a frame with the mouse around multiple units.
Artificial Intelligence was implemented in the terms of the game mechanics. The game mechanics decide whether a single unit automatically repeats fire when under attack or calls for reinforcements when opposed with a superior enemy. The intelligence can be seen as a virtual player, controlling the opponent of the human player. This virtual player places orders just like the human player in front of the monitor. When two contradicting orders are placed by the virtual player AI and the game mechanics (i.e. the mechanics wants to flee, but the AI wants to attack), the AI component has higher priority. Our virtual player AI collects resources, builds its own base, creates armies and tracks down the human player. It is also possible that several instances of the AI fight against each other.

Results

As expected, students' motivation was extremely high throughout the whole course. Often students stayed longer than required. At the end of the morning lectures, we had discussions about implementation alternatives and related problems. As already mentioned students are domain experts in gaming. Students compared their work with games recently released in terms of speed, polygon processing, and features, trying to compete with commercial games. Many students appreciated the experience of having to write fast code, as slow code gets punished with low video refresh rates. Another benefit of the course mentioned by the students was the possibility to write code for a GPU, which was new for nearly all of them, especially using the *NVidia FX Composer* [15]. For version control we used *Microsoft Visual SourceSafe* [16]. As a special appraisal for the work done, Microsoft asked us to provide our game as introduction material for game programming enthusiasts to the public [17]. Figure 1 shows a screenshot of the game. We will offer the course again.

4.2 Second Project Course

In the second course we accepted 16 students in the order of their application. The only requirements were that the students had already attended some basic software engineering lectures and had basic C# experience, but we did not check their knowledge.
The intention of the project course was that students could gain project experience. This includes brain-storming, transforming ideas into a common goal, defining tasks, estimating task complexity, building groups, and assigning tasks. To emphasize the project character of the course, all 16 students worked on a single game.

Preparation

To implement a nice looking and well playable game while keeping the effort controllable we decided to create a 3D Jump & Run game. Although game content creation is not part of software engineering education, we requested the students to create their own models and textures for the game as an example of handling non-programming tasks in a software project. To boost motivation for students we declared Xbox 360 support as a key requirement for the project course.

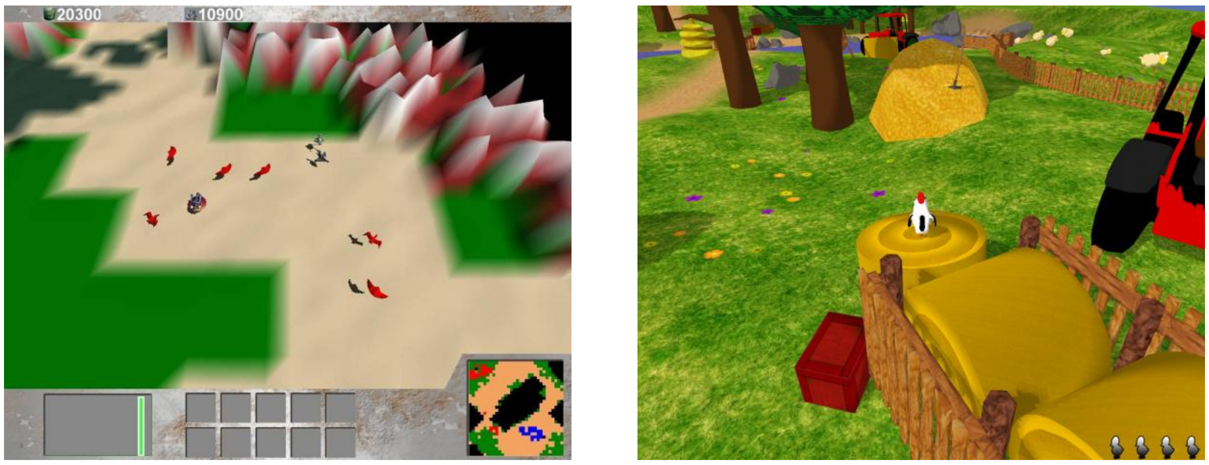

Figure 1: Resulting games: first project course (left) and second project course (right).

During this version of the course, we did not lecture regularly. Lectures were omitted due to time limitations and the fact that the students worked on different tasks, completely free in choosing an appropriate solution approach. We limited lectures to a single introductory session, in which the students received an overview of XNA as well as algorithms and data structures commonly used for game programming. We expected the students to work and acquire new knowledge on their own. They could ask for advice at any time of the project course.

At the beginning we had only little experience with game development and XNA. As the previous project course had a completely different focus – 3D modeling and Xbox 360 support had not been discussed – we could only slightly benefit from the experiences available. Therefore, all experiences we gathered during preparation of the course were recorded and provided to the students as support documents. During preparation we developed a basic game architecture and made it later available to the students to reduce time needed for project initialization. Furthermore, students would not have been able to design a suitable game architecture right at the beginning of the course. Beyond the architecture, we prepared a simple game story to provide a basis for discussions between the students. Without providing a common basis it is hard to achieve a consensus in a course of 16 students. The story expected that the gamer plays a hen which collects her lost chickens. The basic architecture consisted of a 3D landscape generated from a height map, a movable camera and some simple 3D objects. For demonstration purpose in the beginning, hen and chickens were represented by yellow spheres.

We organized the project course in a short introduction period and three development periods each lasting one month. Each development period started with a meeting of all students, in which students planed the goals for the current period, defined tasks and assigned them to groups, as in agile development teams. The meeting was moderated by a tutor to assure the progress of the project and to advice the students on defining feasible goals and tasks.

Technical Details

In the project course we used the following open source tools. We managed our code as well as game content with a *subversion* repository [18]. In addition, we used the project management tool *trac* [19] for bug-tracking and communication between the students. The 3D models and animations were created with *blender* [20].

The basic game architecture already featured a terrain generated from a height map. During the course the students enhanced it by a texture map. All game objects are organized in a quad tree. For the physics part, the students decided not to use the collision spheres provided by XNA. Instead they used an approach that supports several bounding boxes for a model. The bounding boxes were created during the modeling process and then imported into XNA. To resolve collisions and handle gravitation, a simple force system was implemented. As Jump & Run games normally focus on scripting to implement game stories, the students decided not to use any AI algorithms. In fact, all non player characters are controlled by triggers and scripting. Scripts are written in C# and compiled at runtime. They provide almost full access to all game objects and settings. A game level is defined by an XML file, including: game objects (initial transformation and references to model), triggers and the terrain settings for the level. To design new levels, the students developed an external editor for the basic layout of a level and an in-game editor for the exact positioning of game objects.

During the project course, the students had access to two Xbox 360 consoles, but used them rarely, as XNA development is taking place on windows computers. The same code can be used for windows and Xbox 360 games simultaneously; the few necessary platform dependent operations can be separated by preprocessor directives easily.

Problems we faced during the project course had basically two sources. Firstly, 3D models were often too detailed and textures too large. Secondly, we had a problem with small differences of XNA between Windows and Xbox 360. Although XNA makes it easy to develop for PC and a gaming console simultaneously, there are some minor differences. For example, the implementation of the garbage collection in the .NET framework is different on Windows and Xbox 360, see [21] for details. In some cases shaders also behave slightly differently on the two platforms.

Results

Figure 1 shows a screenshot of the game developed in the project course. The course concept worked quite well. Based on the basic game architecture the students quickly started creating own ideas. First they decided to enhance the Jump & Run game with some adventure-like quests. Furthermore, the students stayed with the story of playing a hen collecting her lost chickens and found it obvious to use a comic style for the game graphics.

Students' motivation was high during the whole project course. Most of the tasks defined by the students themselves could be finished within the periods planned for them. In the last period the students decided to realize animated models – they extended their models with bones, enhanced the XNA model importer to process bone data and wrote code for animation playback. During preparation we had discarded the idea of animation as we had expected it to be too complex. But students implemented it anyway. An interesting point is that the students primarily used the internet as source for game programming knowledge. Only a few used books and asked questions.

5. Conclusion

Our courses showed that game programming is suitable for software engineering education in diverse project formats. The first course had a fixed schedule accompanied by lectures. Thus, the content learned by all students was clearly defined. In the second course, students had to acquire gaming knowledge by themselves. As tasks were distributed among teams, students learned different topics. Although the students of both courses were graduates, we think that the stricter format of the first course is also suited for undergraduate students.

As we had expected, the students were highly motivated, enjoyed the course and willing to contribute to both courses more than required. In course evaluation all students named the game programming topic as primary reason for their motivation. Our decision to use XNA paid off, as we were able to focus on algorithms and project work during the courses. Almost as a side effect, students learned various software engineering skills, such as planning, coordination, use of configuration management and bug reporting tools, etc.

References

[1] Joint Task Force on Computing Curricula, *Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering,* tech. report, IEEE CS and ACM, 2004

[2] Carlo Ghezzi, Dino Mandrioli, *The Challenges of Software Engineering Education*, ICSE 2005 Education Track, LNCS 4309, pp. 115 – 127, Springer 2006.

[3] Kelvin Sung, Peter Shirley, Becky Reed Rosenberg, *Experiencing Aspects of Games Programming in an Introductory Computer Graphics Class*, SIGCSE'07, March 2007, Covington, Kentucky, USA.

[4] Marianne deLaet, Michael C. Slattery, James Kuffner and Elizabeth Sweedyk, *Computer Games and CS Education: Why and How*, SIGCSE'05

[5] Andrew Davison, *Killer Game Programming in Java*, O'Reilly Media 2005

[6] JOGL – Java Binding for OpenGL, https://jogl.dev.java.net/

[7] The XNA Framework, http://creators.xna.com/

[8] Microsoft Developer Network, *API Migration Guide: Managed DirectX 1.1 to XNA Framework*, White Paper, http://msdn2.microsoft.com/en-us/library/bb197956.aspx

[9] Mickey Kawick, *Real-Time Strategy Game Programming Using MS DirectX 6.0*, Wordware Publishing Inc., 1999

[10] Steve Rabin, *AI Game Programming Wisdom*, Thomson Learning, 2003

[11] Tom Miller, *Managed DirectX 9*, B&T/Sams, 2003

[12] David Scherfgen, *3D-Spieleprogrammierung. Modernes Game Design mit DirectX 9 und C++*, Hanser Fachbuch, 2003

[13] Kent Beck, Extreme Programming Explained: Embrace Change, Addison-Wesley Professional, 2000

[14] K. M. Lui and K. C. C. Chan, *When Does a Pair Outperform Two Individuals?*, XP2003, Italy, 2003.

[15] NVidia FX Composer 2, http://developer.nvidia.com/object/fx_composer_home.html

[16] Microsoft Visual SourceSafe, http://msdn2.microsoft.com/en-us/vstudio/aa718670.aspx

[17] Publication of the first project game, http://www.microsoft.com/germany/msdn/coding4fun/xna/seacraft/default.mspx

[18] Subversion – version control system, http://subversion.tigris.org/

[19] trac – Integrated SCM & Project Management, http://trac.edgewall.org/

[20] blender – 3D modeling tool, http://www.blender.org/

[21] .NET Compact Framework Team, *Managed Code Performance on Xbox 360 for XNA: Part 2 - GC and Tools*, Blog Entry, http://blogs.msdn.com/netcfteam/archive/2006/12/22/managed-code-performance-on-xbox-360-for-xna-part-2-gc-and-tools.aspx