The Making of an Interdisciplinary Games Project

Rafael Bidarra
Delft University of Technology, Mekelweg 4 , NL-2628 CD Delft , The Netherlands
r.bidarra@ewi.tudelft.nl

Jerke Boers, Jeroen Dobbe, Remco Huijser
Cannibal Game Studios, Rotterdamseweg 145 , NL-2628 AL Delft, The Netherlands
{j.boers, j.dobbe, r.huijser}@cannibalgamestudios.com

Abstract

Many universities with a Computer Science (CS) curriculum now offer a game development course in a variety of flavors. However, the fundamental standpoint that leads their particular course design is not always clear. Delft University of Technology introduced project-based education in its CS curriculum five years ago, including a second-year games project. Initially designed as little more than a companion to the computer graphics course, the games project matured into a large project integrating a broad range of computer science topics. More importantly, though, the current games project brings CS students together for the first time to work in a realistic and interdisciplinary game development team, involving students pursuing a Game Design and Development degree at the Utrecht School of the Arts. We believe that the key to the huge success of our games project lies in the consistent combination of this careful interdisciplinary organization with the deployment of professional technology and working environment specifically crafted for an educational environment. We also conclude that a streamlined collaboration among students of related disciplines works as a very powerful catalyst in their personal and academic development.

KeyCategories and Subject Descriptors
K.3.1 [**Computers and Education**]: Computer Uses in Education—Collaborative learning
K.3.2 [**Computers and Education**]: Computer and Information Science Education—Computer science education, Curriculum
K.8.0 [**Personal Computing**]: General—Games

General Terms: Design, Experimentation, Human Factors

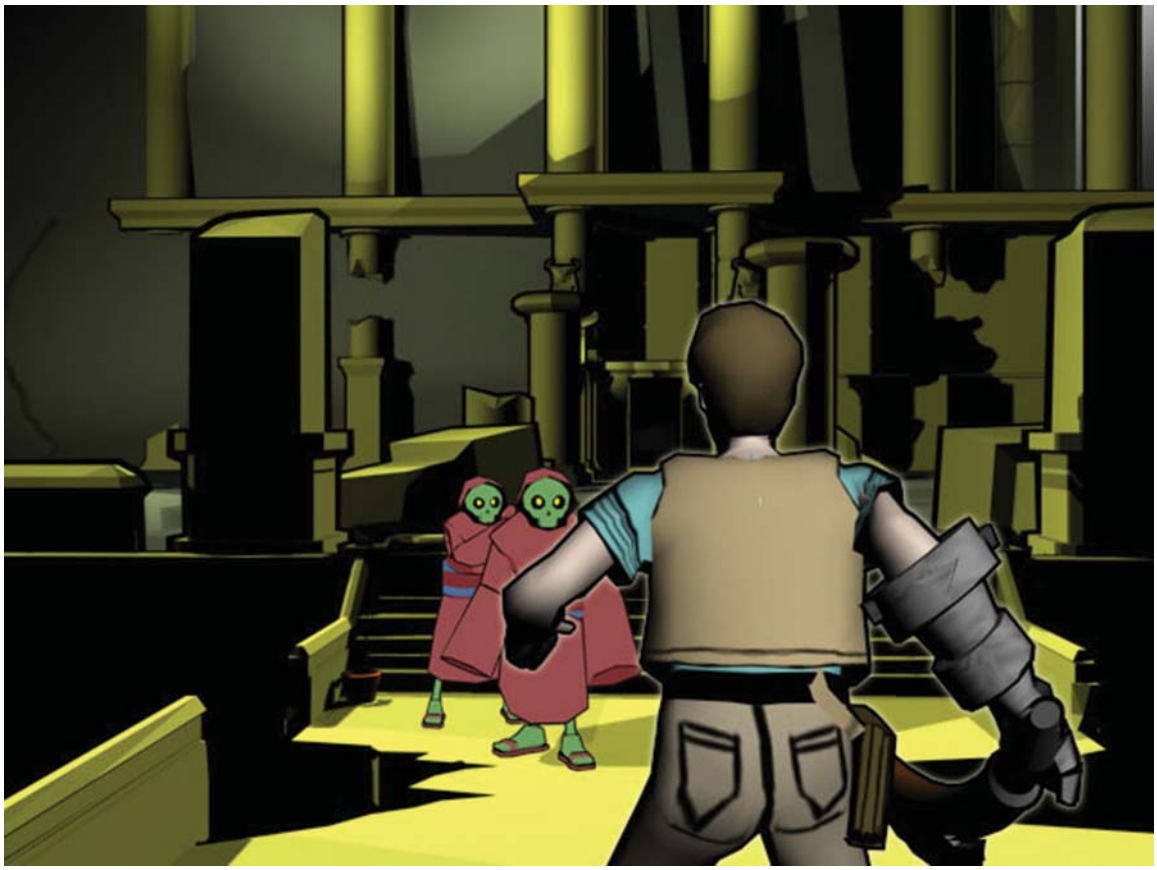
Keywords: Games education, project-based education, interdisciplinary education, game design and development, industry involvement

Introduction

Five years ago, Delft University of Technology introduced project-based education in the Computer Science (CS) curriculum. One of the new project-based courses was the second-year *games project* [Bidarra et al. 2003]. This games project was initially associated with an introductory course on Computer Graphics (CG), and as such, the primary goal was to have students apply computer graphics techniques in a practical setting.

While first running the games project, it soon became apparent that it had more potential and a bigger scope than was initially envisioned. CG was covered, and it naturally required students to learn more about other game-related aspects.

Because of its potential and the enthusiastic reaction from the students, the project has been actively improved over the years. Combined with valuable industry input, the games project has now matured into a multidisciplinary course covering all aspects of game development, and therefore better reflecting real-life game development environments. See Figure 1 for some sample impressions of last year's games.




8:32
SUPER-COMBO + +
34.354
N


the god of weather
was going for a little stroll..

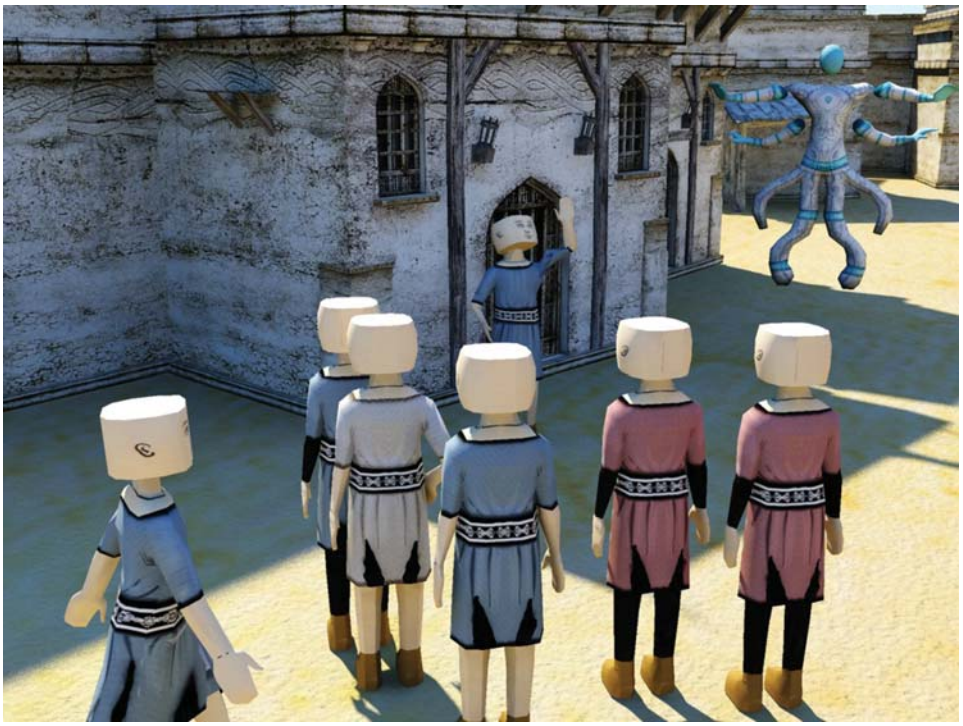


FIGURE 1 Some representative scenes of the games developed in 2007.

In this article, we describe and motivate the evolution over the past five years, from a pioneer to a professional games project, from both an academic and industry perspective. We start by summarizing the project organization, followed by a discussion about the technology deployed and the working environment provided. During the course of the project, several external parties were involved in the project. We conclude with an evaluation of the project run in 2007, and some general conclusions about running such a games project.

Project Organization

Project-based education very much responds to the basic concepts behind *constructive alignment* [Biggs 1999], a rather influential stream, in particular in higher education, which advocates among other things that "students construct meaning from what they do to learn." In line with this, an advantage of including such projects in a curriculum is that the acquisition of knowledge is strongly motivated by its immediate application in a practical environment. In addition, it encourages students to actively learn to value and promote the teamwork process, instead of focusing exclusively on the final product.

Characteristic for CS project courses is that students have to work in groups on a more or less open assignment [Schaefer 2004]. In our case, they design and implement a computer game from scratch, using the technology provided, while working in a team. This section describes how we organized the project as a whole.

Course Goals

At first, the games project focused on teaching students to apply computer graphics techniques in practice, which has been the main motivation of similar projects (e.g., [CS248 2001]). However, as game development involves more than just CG, we also wanted the students to be able to focus on software engineering, artificial intelligence, user interaction, and other techniques involved in game development.

As we strived to continuously improve the project and better prepare students for work after their study, we also wanted to make sure students learned how to work within the context of a realistic software project, while learning how to cope with the challenges of interdisciplinary collaboration.

Considering this over the past few years, we gradually expanded the course goals to comprise a wider range of games-related issues, eventually leading to the current set of learning objectives. We say these goals have been achieved when the student has demonstrated proficiency in:

- Applying media and programming techniques within the context of computer games, and in relating them to particular game effects
- Striving for the balance between the effectiveness of a programming technique and the desired quality of a game effect
- Describing the main modules of a game engine and purposefully use their functionality

- Deepening object-oriented programming skills while building a complex and large software system in an agile context
- Developing and contrasting teamwork skills within the context of a realistic interdisciplinary team

Teams

For several years, the games project had been run in groups of about five to seven CS sophomores, who had to handle alone both game design and implementation. The former not being part of their educational curriculum or goals distracted from their work as programmers, and did not allow them to focus their efforts on the course goals stated previously.

Therefore, in Spring 2007, the project entered a new phase: we started a pilot collaboration with the Utrecht School of the Arts (HKU), which offers a bachelor's degree on Game Design and Development. Their second-year students also have a one-semester project, focusing exactly on the game design process as a whole. Integrating their game design project with our game development project led to one large multidisciplinary project. In this integrated project, groups consisted of four CS students and five Game Design (GD) students. The CS students were mainly responsible for the implementation of the game, while the GD students were in charge of game design and artwork/content creation. In doing this, they worked as two departments of "one single company," with lead programmer and lead designer roles, respectively, assigned among them.

Integrating these two projects brought much more realism and power to the project: realism, because it more closely matches the actual team composition in a real-world game development company; power, because this interdisciplinary collaboration promotes that each team member contributes with his or her best skills. In other words, we fully confirmed the value of the splendid advice recently given by Randy Pausch: "(…) not to turn artists into engineers or vice versa, but to teach students how to work in teams that utilize the disparate talents of their members" [Pausch 2007].

These mixed groups, though having clear advantages over traditional uniform groups, also had some disadvantages; for example, more time was spent on communication, traveling, and appointments. In particular, everyone in these groups vividly experienced the additional challenges brought about by communicating with people from outside their own discipline, which requires a rather different way of thinking and explaining.

Significantly, after this interdisciplinary pilot experiment—which although facultative, was chosen by the vast majority of the students—all students unanimously recommended that next year we make it obligatory to work in such mixed teams.

Project Planning and Development Process

In line with other project courses in the CS curriculum, the games project at first consisted of three main phases: *analysis*, *design*, and *implementation*, where the implementation phase was by far the largest and most complicated. As this project is aimed at CS students, the focus is not

on the game design aspects of developing a game, but rather on the implementation of games. The students did not perceive the analysis and design phase as very useful. The analysis phase, in which the initial game design is created, was very short and did not provide much educational value for the students. The design phase was mostly used to create a technical design for the game; however, since the students did not have any experience with developing games, the designs created upfront were proven rather useless early in the implementation phase.

Key to designing and implementing a successful game in practice is having an approach in which you strive to have a playable and working version of the game as soon as possible: the so-called *first playable* [Laramée 2005]. After this version is established, different gameplay elements can be tried and changes can be made to the initial version. This usually occurs in multiple iterations leading to a more agile development process [Martin 2003]. To better accommodate this process and to make the project more interesting for students we decided to drop the classic waterfall style of development, incorporating several elements from the Agile Manifesto [Beck et al. 2001] and *eXtreme Programming* [Beck 2006] adapted specifically to what works well for the games project. Prototyping and iterative development are given a central role to help students cope with much uncertainty when it comes to potential technical solutions and the exact requirements. To give this shape we introduced new phases of a more iterative nature: *spikes*, *first playable*, *beta*, and *release*.

At the beginning of the project, students have no experience with the technology and, as they have no prior experience developing games, no knowledge of what developing a game entails. To smoothly introduce students to actual game development and the technology involved, the spikes phase offers room to try different concepts and technical solutions using small pieces of code (spikes), gaining more insight into important aspects of their game. The first playable phase is aimed at gaining the first playable by integrating all relevant spikes into one product. Both the beta and release phases are aimed at refining the previous versions and completing the game.

For the interdisciplinary groups this turned out to be a very important methodology that enabled both parties to work together on their game. During the first phase, GD students were able to concentrate more on setting up their initial game design, pitching their game concepts to the teachers and the CS students in their team. CS students were able to get a feel for the technology, the requirements, and what it takes to implement the concepts of the GD students. Once the game concept and initial design were finished, and the different spikes created, groups could start concentrating on creating their first playable. After the first playable was assembled, GD students could continuously evaluate their game (by play-testing) and come up with new requirements and changes. CS students could then evaluate the technical feasibility, prototype (spike), and incorporate these changes when desirable for the game design. At the start of each phase, CS students were encouraged to evaluate their current design, improving it where necessary to incorporate desired changes, also called refactoring [Fowler 1999], using the information gained from building their spikes.

Deliverables

To monitor the progress of the teams and to steer them toward an effective development process, students had to hand in three distinct deliverables at the end of each phase:

- The implementation of the game (working source code)
- A simple game design document (containing an explanation of the game and its key features)
- A technical document (linking the explanation of the game to the implementation)

As is to be expected from an iterative approach, each deliverable started from a basic version and evolved into the final product. With each phase, we could monitor how far the students had progressed (compared to the previous phase). These deliverables and the team progress also served as a valuable basis for the assessment.

Focus on Requirements

To provide students with a clear direction and a tangible approach to fulfill the course, a list of requirements was set up. While this list initially only contained some general requirements and computer graphics techniques, it has been expanded to include AI techniques and a number of other game-related elements/techniques. Students had to select from these different lists of requirements, as long as they incorporated all of the general requirements, two graphics and two AI techniques, and implemented one other technique/requirement of their choice. These requirements ensure that students build a 3D game involving interesting technical challenges. By offering a wide choice among many game-related techniques we guarantee that there are always challenging aspects for every student to explore. This, in turn, encourages students to remain motivated, to delve deeper into whatever study subjects required, and to exceed themselves in the implementation of the techniques of their choice.

Assessment

Several aspects are important when it comes to determining how to assess the students' work. From the course goals, it is apparent that we have to assess the final product and the process. This led to both a product mark and a process mark:

$$\textbf{final grade} = \frac{6 * \text{product mark} + 4 * \text{process mark}}{10}$$

The product mark takes into account, among other things, the quality of the game (various aspects of gameplay), the quality of the software (e.g., architecture, modularity, clarity, choice of technical solutions), technical realization of the different requirements, and the quality of the project documentation. Placing great emphasis on the technical realization supports the focus on the requirements.

The process mark takes into account the collaboration between team members (e.g., use of working environment, task planning and assignment, communication with GD colleagues) and the individual contribution of each group member in the entire development process (e.g., attitude, dedication, initiative, leadership, performance).

To assist the tutors in performing the assessment of individual contribution and collaboration, the students perform several peer-evaluations throughout the semester, in which they anonymously assess each of their group members. Our experience has steadily confirmed that this peer assessment provides very valuable, reliable, and effective learning elements to each student [Liu 2006], in addition to assisting the tutors in their coaching and assessment responsibilities.

Adding the collaboration component and the quality of the game into the equation stimulated students to also focus on collaboration and get as much out of the group as they could. Including peer assessment assured that students would be motivated to cooperate with this collaborative process, thus avoiding negative peer-reviews.

Working Environment

An important part of any games project is the choice of the supporting technology to work with. In this section, we motivate the evolution of a successful framework applied in our games project throughout the years. We discuss the most important challenges faced while continuously using, configuring, and improving the framework, and describe the solutions we came up with to resolve those issues. At the same time, some conclusions are drawn that should be useful to anyone running game design and development projects or courses.

First Steps, First Lessons

After the first year run of our project, in Spring 2003, it soon became apparent that several students were somehow frustrated and disappointed with the results achieved. In fact, their huge initial enthusiasm was taken down quite a bit because they had not been able to concentrate on making their game, as they had to overcome many difficulties related to the programming language and support technology. Once students had finally learned how to work with the framework provided, they had very little time left to spend on creating their game, which should have been the focus of the course.

In this phase, the project was supported by the open source rendering engine OGRE [OGRE 2007]. This engine is written in C++, which was considered the industry standard. However, in the context of "inexperienced" students who have to develop a large and complex software product, C++ becomes a problem. Students were more focused on mastering and controlling the programming language than on the development of their game. A "higher-level" language like Java, which is taught throughout the CS curriculum, might help students to focus more on implementing concepts, rather than on "advanced" technical features (such as memory management and pointers).

Furthermore, OGRE, an open source rendering engine, was not easy to install and lacked quality support and documentation. Moreover, we needed the functionality of a complete game engine, which includes much more than just graphics. Therefore, students had to first set up OGRE, then choose a sound library, set that up to work, and share the right details with OGRE. In many cases, even a physics and collision engine had to be integrated, and all that in a language students felt uncomfortable with, before they could even begin working on their actual assignment: the game.

All in all, the first and most important lesson learned was that the choice of supporting technology for your course or project should be carefully made and aligned in straight relation to the learning objectives. To get students to focus on the actual course goals (e.g., applying computer graphics techniques and concepts in practice), you will have to carefully tune the choice of tools, language, and libraries in line with those goals [Biggs 1999].

Cannibal Birth: a Promising Alternative

During the second year this project was run (Spring 2004), we started to realize how we could improve the aforementioned situation. A couple of students, who were particularly keen with games and their technology, formed a development group called *Cannibal*, and started to work on a new game engine. To adopt this new technology in our games project, we had set as requirements that (i) it was *intuitive* to use, (ii) it *enabled* students to realize their dreams, and (iii) it was *fun* to work with.

A key aspect of this new engine was that it should favor usability over raw performance. This should make it easier to use and more manageable, leading to increased productivity and a better focus on developing the game. This motivation also led to the choice of C# as the main programming language. Although it is not an industry standard in game development, C# offers good performance, is of a higher level than C++, and has proven to be very easy to learn and work with [Bates 2004]. Complementary to the choice of C#, Managed DirectX [Miller 2003] was also chosen as a managed graphics framework. Together, the Cannibal Engine and a managed framework based on C# and Managed DirectX should encapsulate many of the technical details, so that students could focus more on the project requirements and the design of the system. A better focus on design is especially convenient for game development projects because they tend to grow exponentially in complexity as they grow in size.

In Spring 2005, we deployed the Cannibal Engine for the first time in the games project. Students' enthusiasm rapidly grew as they saw their programming speed increase and realized they were able to accomplish much more in a shorter time. This led them to request many exciting new features to accommodate their newfound wishes. To that, the Cannibal team corresponded with proportional enthusiasm, satisfying many of those requests, which in turn left hardly any time for proper testing, with the proverbial consequence of allowing a number of issues to creep in with every new feature released. The lesson for the future was clear: stick to one stable version of whatever software is going to be used in the project—game engine, libraries, modeling tools, etc. New intermediate releases or upgrades during the semester are a source of entropy that seldom pays.

At the end of the term, a survey was conducted among students to verify our observations. The most important results of this survey were that over 90% of all students found it very easy to install and start using the Cannibal Engine; not a single student faced major problems with the programming language C#; and over 80% of all students found the engine very intuitive and fun to work with. Therefore, we confirmed that the focus on usability and support, and not so much on features and performance, was indeed important. Students could get right into creating their game, instead of having to struggle over and over again with many "complicated" technology and language issues.

Cannibal Growth: Supporting Team Collaboration

After two years using and improving the Cannibal Engine, in Spring 2007 we had the opportunity to upscale the games project, giving the interdisciplinary step described in *Teams*. This larger and more complex course demanded a number of improvements, including a more complete game engine and supporting teamwork facilities to manage the larger teams.

Cannibal Game Engine

The evolution of the Cannibal Engine has been a continuous process driven by the eagerness of the Cannibal team to improve their technology. In this process, improving usability has always remained the focus point of the development, considering user feedback. In addition, the Cannibal team has worked on new techniques like collision detection methods and alternative dynamic lighting models. The latter resulted in an experimental version of the engine that used deferred lighting, theoretically allowing an unlimited number of dynamic light sources.

In this phase, one of the most important developments of the Cannibal framework has been the adoption of Microsoft's XNA [XNA 2007]. XNA enables Cannibal to provide cross-platform development using C# on both Windows and Xbox 360, and comes with a convenient development environment called Game Studio Express, further increasing the usability of the engine and the focus on students.

Online Collaboration Platform

With the introduction of multidisciplinary teamwork in a larger, rather diverse group of students, collaboration within each team became even more important. In addition, more and more game development aspects were included in the process, so planning became more important for the geographically dispersed teams.

To enable students to effectively learn from this collaboration process, an integrated working environment was provided in which they had access to a number of different collaboration tools. The most important tools provided to students were a Wiki [Leuf 2001], a Subversion repository [Pilato 2004], and a custom bug/task tracking system, all integrated into one. The Wiki system allowed for easy, fast, and collaborative editing of documentation for the game and communication. Students were motivated to keep their Wiki up to date throughout the project. This improved the collaboration between team members, and made it very easy to timely produce their deliverables, by simply extracting document data from the Wiki. The Subversion system was used to share code and assets among the group and to record the changes and different versions of the game. The bug/task tracking system also supported planning and was used to keep track of the project's progress. The team was encouraged to create milestones for each phase of the project, and to fill them with tasks assigned to each team member.

The tools presented here provided added value to the student teams, and were very useful to the tutors. By supplying these tools to students, tutors can meticulously follow the development process of the teams. This provides them with valuable insight and overview of the course, and helps them decide when and where to focus their guidance.

As it turned out, collaboration between different disciplines is much more difficult than between like-minded people, even with the right tools available. Suddenly, artists had to understand technical people and vice versa; content created by GD students now had to be adapted to suit the needs of CS students and the capabilities of the development platform. Since these different disciplines cause problems faced by many established game development studios in the past [Roathe 1998], some clashes were to be expected. In exceptional cases, mostly due to inexperience, this would lead to an "over the fence" culture, where GD students would create content, throw it over the fence, and blame the CS colleagues for it not working. These, in turn, would point the finger back at the GD students, leading to undesirable and nonproductive situations. However, learning through experience, all teams were eventually able to work out their differences and become rather productive.

Cannibal Experience: Learning by Doing

After the success of 2007, the Cannibal staff realized the potential of the "learning by doing" solution for educational settings, and started to work on their first professional product using the developed technology. This product, called *Cannibal Experience* [Cannibal Experience 2007], is directly aimed at higher-education institutions, supporting them to use game development as a means to teach their curriculum. Cannibal Experience mainly consists of two components: *Game Development Platform* and *Online Collaboration Platform* (see Figure 2). Both components are considered critical for the success of any game-related course, with an emphasis on project-based education. Of course, a support layer, including technical assistance, underlies both platforms, allowing students (and teachers) to ask questions directly to the Cannibal staff.

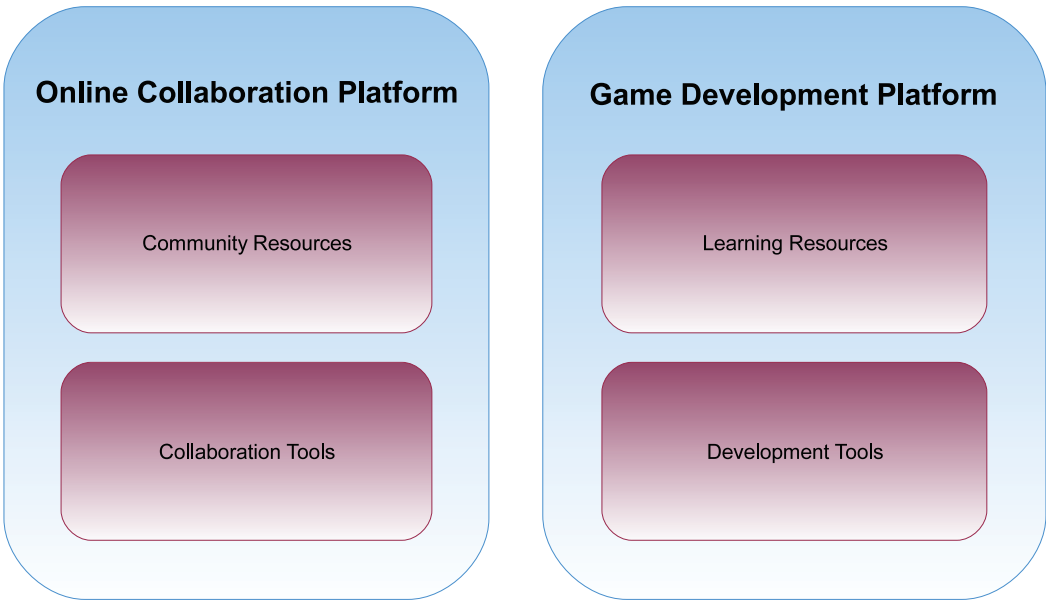


FIGURE 2 Overview of Cannibal Experience.

Game Development Platform

Regarding *Development Tools* (see Figure 3), the game engine plays a central role. Therefore, it has been carefully revised and more rigorously tested. More features and much flexibility have cautiously been added, without compromising the usability and cleanliness of the API. During development of the new version of the engine, special attention has been paid to allow students to work with the engine at their knowledge level. Students applying game technology for the first time can use the engine at a very high level, using only top-level features. When students want to delve deeper, they can start extending and modifying the behavior of the engine at any level they feel fit; they can, for instance, add new event triggers, add new sources of textures (e.g., a webcam), or implement new input devices (e.g., the Wii controller) [Thibault 2007]. The engine's main features include strongly object-oriented scene-graph management; asset management; different graphics elements (e.g., shaders, particles, lighting, animation); audio; intersection checking; a flexible and extendable event-driven programming system handling, among other things, user input and game logic; a user interface system based on common controls (e.g., button, slider bar).

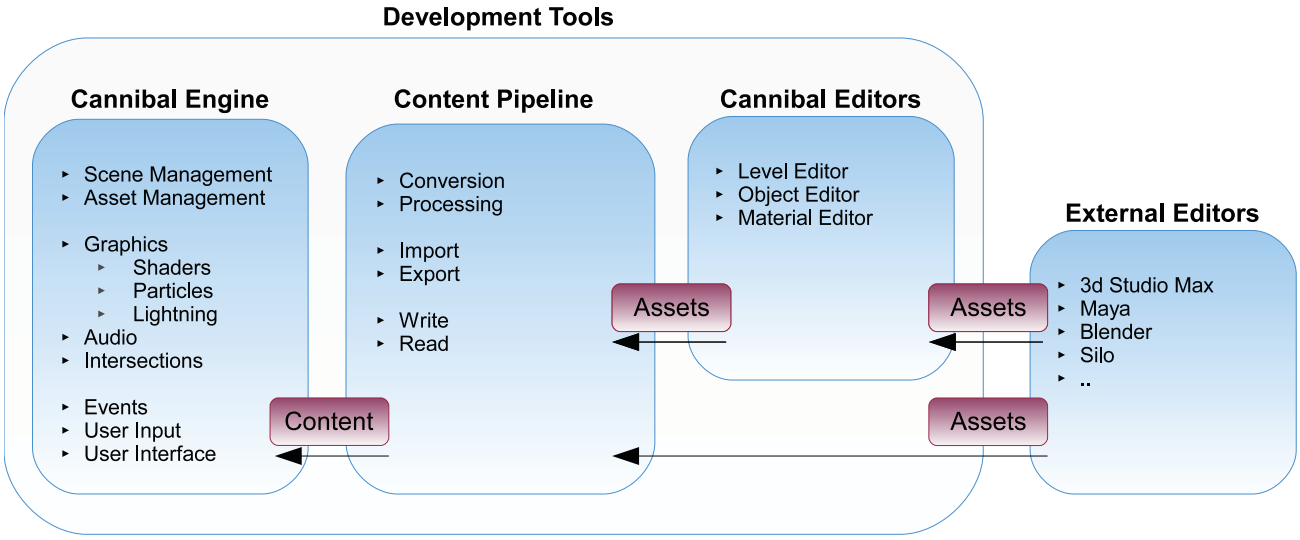


FIGURE 3 Overview of Development Tools.

Furthermore, Development Tools includes several editors; for example, a world editor and a game object editor. These tools can be used by nontechnical team members to verify and configure their content in the game environment. With the challenges introduced by interdisciplinary collaboration, the tool chain is an indispensable component, as it allows artists and game design team members to work independently on their content items and know that these will not cause complications when they hand their content to their technical team members. The content pipeline is used to import, process, convert, and serialize the assets so their contents can be used at runtime in the game. Assets created externally can also be imported into the Cannibal editors and be further configured for use with the engine.

From our experience of recent years, as students were moving faster and faster through course material and their requirements, their need for information on more (elaborate) game development techniques became increasingly apparent. Instead of having to explain techniques and concepts every once in a while, teaching staff became overloaded with questions. To resolve this issue, *Learning Resources* have been included in the Game Development Platform. These resources consist of a collection of tutorials and a starter kit designed specifically to help students to get started with concepts and techniques right away. These tutorials range from getting to start up the game to collision detection algorithms, character AI, and working with content like textures, sounds, models, and animations. The starter kit is a fully prepared virtual world where all these elements are covered. It comes with a full set of game content items like textures, sounds, various shaders, and different static and animated models. Having at your disposal all information on developing a game is of significant and valuable assistance in the whole process of learning by doing.

Online Collaboration Platform

As discussed in the section *Cannibal Growth: Supporting Team Collaboration*, the Online Collaboration Platform already contained several integrated *Collaboration Tools* for the students to share work and collaborate. A discussion forum has been integrated as well, as we realized that such functionality was particularly appreciated by GD students. A forum also helps to better facilitate online discussion between students at different geographical locations, while allowing students to work at different times, since a forum is by definition an asynchronous communication tool. Having gone professional, the Cannibal staff will usually be less involved in tutoring future educational projects than has been the case with the games project described so far. Therefore, different tools were provided for teaching staff to manage their course, enroll students, set up teams, and monitor progress. These new functionalities provide them with valuable insight, and help them decide when and where to focus their guidance.

Besides supporting project planning and team collaboration, *Community Resources* have also been integrated, where students and teaching staff alike can come together online and hold their discussions using Cannibal Experience. In addition, teaching staff have private forums where they can discuss course setup and other educational or tutoring aspects.

Community Resources also provide a way for all members to communicate directly with the Cannibal staff, by means of a forum, for feature requests, bug reporting, submitting suggestions, etc. The Cannibal staff, in turn, continuously provides the Cannibal Experience community with knowledge about the environment and its components. Creating a community and actively participating in it allows Cannibal and teaching staff to cope with the continuous requests for information and feedback from students.

Industry Involvement

From the very beginning of the games project, we have actively tried to involve a variety of partners with a relation to game development and game technology. In particular, involvement of real

stakeholders from the games industry has been an important success factor for the project because it strongly stimulates and motivates students. Furthermore, these parties enrich the project with significant game development experience and technical expertise. For example, throughout the semester, we schedule a number of guest lectures in which experts from renowned Dutch game developers (e.g., Streamline Studios, Triumph Studios, W!Games, Coded Illusions, Playlogic) tell about their experiences in games development, from a wide variety of viewpoints, ranging from design methodologies, through current CG or AI challenges, to commercial video game production and market aspects.

Another way to get the industry involved has been to invite companies to sponsor the *Game of the Year* competition, an exciting contest "unofficially organized" every year by our faculty among the participating teams. The basic idea is that the sponsoring company provides both a jury member and a prize for the winning team. This scheme gets the companies to promote their games, and helps them become acquainted with the best skills of our best students. No wonder that as a result of this close collaboration with the games industry, over the years many former students of the games project have found BSc internships, or even their final MSc research project, at selected Dutch game development companies. Eventually, some of these students found career opportunities, while several others formed their own game-related startups, as is the case with most of the authors of this article.

Finally, in 2007 the project was sponsored by Microsoft Netherlands. Because the Cannibal Engine is based on XNA, Microsoft Netherlands donated a number of Xbox 360 consoles to the faculty for use in this project, giving a significant boost to the students' enthusiasm. This year was also unique as both CS and GD students were given the opportunity to present their games at the Microsoft DevDays event in Amsterdam, at the end of which Microsoft handed a grand prize for the *Game of the Year*.

Project Evaluation

In 2007, for the first time, *all* CS students (approximately 30) of *all* participating groups successfully finished the project, whereas in previous years a couple of students might fail, typically due to lack of motivation or dedication on their part. As might be expected, the games developed by the six interdisciplinary groups were significantly more creative, consistent, and appealing than the game of the single group working alone. However, all games, although considerably simple, were recognized as a remarkable result for a one-semester design and development cycle (see Figure 1 for some representative screenshots). Please refer to the course Web site [MKT4 2007] for the description and sources of each of the games produced.

From the organization point of view, we very much profited from the accumulated experience, the biggest challenges having to do with the novel cooperation with the HKU colleagues (e.g., appointments, traveling time, language and culture clashes, etc.). However, learning to cope with this diversity was precisely one of the main reasons for the initial choice, and the consensus was that it had been very effectively achieved.

The working environment (see the section *Working Environment*) was generally acclaimed as rather helpful and pleasant to most tasks. The Cannibal engine was, this year, considered especially accessible, easy to use, and attractive, among other things, due to the Xbox 360 compatibility. The assistance and supervision tasks were now more directed toward architecture and conceptual issues, rather than having to concentrate on technical programming problems. In addition, extra time had to be dedicated to the coordination of the interdisciplinary groups to avoid or overcome conflicts at hand.

As usual, the games project was evaluated at the end of the spring term. For this, all students were surveyed on the most relevant aspects of its organization. The results of this evaluation (100% replies) were by and large rather positive.

First, and most importantly, the project goals mentioned in the section *Course Goals* were largely achieved. Indeed, most students acknowledged having attained a much deeper insight on many subjects. When asked to indicate the three areas most improved upon, students mostly indicated media and programming techniques, ranging from mathematical foundations (55%) to computer graphics (64%) and AI (45%). Programming and software design proficiency were mentioned most (90%).

Although apparently most creative work had been left to the GD students, CS students quickly realized that they had plenty of room left to exercise their own creativity, getting the most out of the engine (e.g., programming many gameplay, physics, and control elements of the game). The limited experience of OO programming at project startup was quickly overcome, which profited from the use of C#. The "fun factor" should not be underestimated, as 45% of all students attributed their learning experience was stimulated the most by their own enthusiasm and motivation for the project. Another group of 36% students indicated the room for choice in requirements led them to go that much farther for the requirements that interested them most. Finally, all groups recognized that carefully watching over their teamwork process made it possible to achieve their successful results.

In Table 1, we summarize several other results of the survey, underlining some more concrete, interesting aspects of the project realization. The table indicates, for each statement, the percentage of students who subscribed to it. Not surprisingly, every year many students point out that they would have liked to spend even more time to "get their product really satisfying," an interesting fact that, besides matching the reality of many game developers, leads some students to form development groups to create a new (and better) game, further increasing their computer science proficiency outside of the curriculum.

TABLE 1 Summary of Survey Results

The time I dedicated to this project was (much) more that the nominal (of its study credits)	59%
My dedication was (very) great	71%
We were given an interesting assignment	93%
I experienced the powerful capabilities of teamwork	92%
I am satisfied with the product delivered	63%
I needed more structure or help planning	63%
I learned more from this project than from any other in the curriculum	75%
This project was more fun than other projects in the curriculum	100%

Finally, we also gathered some lessons learned and recommendations aimed at further improving this project in its future editions. The most important are:

All communication facilities and, in general, methods for streamlining teamwork (online collaboration platform and tool chain) can better be introduced in practice and extensively explained right from the beginning of the semester (e.g., by means of a guest lecture).

Besides introducing the available tools, students miss the experience to successfully plan such a large and complex software project on their own. Most students indicate they would like more structure, but do feel they learn a lot from planning themselves. As hinted by several students, it would be wise to sit down and advise each group while they plan on their own.

Much attention has to be paid to the preparation and planning of all joint meetings, assignments and deadlines between the CS and GD (sub)groups of students, to ensure a successful and productive synergy.

Conclusions

Five years after the introduction of project-based Computer Science education at Delft University of Technology, we can safely conclude that its highly instructive and motivating potential has been more than confirmed, so much so that various departments and faculties started following the same approach. Initiated as a pioneer project on computer graphics [Bidarra et al. 2003], the *games project*, as it is known on campus, has now gained a prominent role as the integrator course *par excellence* of the Computer Science BSc curriculum.

In its current form and organization, including the input from the game development industry as described in this article, the project has achieved a substantial maturity, deploying a professional game engine, a fine-tuned working environment, and very experienced tutoring assistance. It is understood that by now, numerous former CS students of this project have graduated from Delft and have found their career in one of the various Dutch game developer companies, or established their own startup companies in the field, as is the case of most authors of this article. Furthermore, the increasing reputation and popularity of the games project is being very effectively exploited by the faculty for the urgent purposes of recruiting new CS students.

We believe that deploying adequate game technology, professionally crafted for this purpose, within a carefully setup working environment is crucial for the academic success of any integrated games project as the one described here. We also conclude that a streamlined collaboration among students of related disciplines is a powerful catalyst that can significantly raise the levels of knowledge, experience, and teamwork skills achieved by the students.

Games are, and have always been, all about fun. In pretty much the same way, our experience is that getting students in the position of making games can be even more fun. However, the most fortunate of them are those who realize how much they have learned in that process.

Acknowledgments

The authors are very grateful to all their (former) students for their patient and invaluable feedback throughout the years, and to all colleagues who contributed to the success of this project with their constructive ideas and criticism. Special thanks go to Natasha Tatarchuk and Alpana Kaulgud, from ATI Inc., for generously equipping our CG Lab, and to Maarten-Jan Vermeulen, from Microsoft Netherlands, for his enthusiastic and supportive involvement in our work.

References

[Bates 2004] Bates, B. "C# as a First Language: a Comparison with C++." *Journal of Computing Sciences in Colleges*, 19 (3): 89–95.

[Beck 2006] Beck, K. *Extreme Programming Explained: Embrace Change*, Addison-Wesley, Boston, MA.

[Beck 2001] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. The Agile Manifesto. Available at http://agilemanifesto.org.

[Bidarra 2003] Bidarra, R., van Dalen, R., van Zwieten, J. "A Computer Graphics Pioneer Project on Computer Games." Proceedings of CGME 2003—Workshop on Computer Graphics, Multimedia and Education, 8 October, Porto, Portugal, pp. 61–65.

[Biggs 1999] Biggs, J. *Teaching for Quality Learning at Un*iversity, SRHE and Open University Press, Buckingham, UK.

[Cannibal Experience 2007] Cannibal Game Studios Web site www.cannibalgamestudios.com/.

[CS248 2001] CS 248—Introduction to Computer Graphics Course. http://graphics.stanford.edu/courses/cs248-videogame-competition/ Stanford Computer Graphics Laboratory, Stanford University, CA.

[Fowler 1999] Fowler M., *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, MA.

[Laramée 2005] Laramée F. D. *Secrets of the Game Business*. Charles River Media, Boston, MA.

[Leuf 2001] Leuf, B. and Cunningham, W. *The Wiki Way. Quick Collaboration on the Web*. Addison-Wesley, Boston, MA.

[Liu 2006] Liu, N-F. "Carless D Peer Feedback: the Learning Element of Peer Assessment." Teaching in Higher Education, 11(3): 279–290.

[Martin 2003] Martin, R. C., *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall, Upper Saddle River, NJ.

[Miller 2003] Miller, T. Managed DirectX 9: *Graphics and Game Programming*, Sams. Indianapolis, IN.

[MKT4 2007] MKT4 Project Web site. Delft University of Technology. http://graphics.tudelft.nl/~mkt4/.

[OGRE 2007] www.ogre3d.org/.
[Pausch 2007] Pausch, R. and Marinelli, D. "Carnegie Mellon's Entertainment Technology Center: Combining the Left and Right Brain." Communications of the ACM, 50 (7): 50–57.
[Pilato 2004] Pilato, M. *Version Control with Subversion*. O'Reilly & Associates, Inc., Sebastopol, CA.
[Roathe 1998] Roathe L. and Fregien, C. CGDC '98 Roundtable Report www.gamasutra.com/features/gdc_reports/cgdc_98/roathe_fregien.htm.
[Schaefer 2004] Schaefer S. and Warren J. "Teaching Computer Game Design and Construction." *Computer-Aided Design* 36 (2004): 1501–1510.
[Thibault 2007] Thibault R. W. Wii Console. www.uweb.ucsb.edu/~rwthibault/Tech_Report.pdf.
[XNA 2007] http://msdn.microsoft.com/xna/.