Harri Hakonen | Tuomas Mäkilä | Jouni Smed | Andy Best

Learning to Make Computer Games: An Academic Approach

Turku Centre *for* Computer Science

TUCS Technical Report
No 899, June 2008

Learning to Make Computer Games: An Academic Approach

Harri Hakonen
Department of Information Technology, FI-20014 University of Turku, Finland.
Email: harri.hakonen@utu.fi

Tuomas Mäkilä
Department of Information Technology, FI-20014 University of Turku, Finland.
Email: tuomas.makila@utu.fi

Jouni Smed
Department of Information Technology, FI-20014 University of Turku, Finland.
Email: jouni.smed@utu.fi

Andy Best
Digital Arts, Arts Academy, Turku University of Applied Sciences,
Joukahaisenkatu 3–5, FI-20520 Turku, Finland.
Email: andy.best@turkuamk.fi

TUCS Technical Report

No 899, June 2008

Abstract

Anything that combines aspects of play, challenge and conflict can be seen as a game. Therefore, games provide a versatile but demarcated setting for education. From the students' perspective the concept of 'game' is familiar, approachable and diverse, which makes it motivating for them. From the teachers' point of view, a game can serve as a mixing pot for various study subjects. For example, making a game can concretise study topics as well as work practices and on-the-fly collaboration across many different disciplines. Therefore, games as an educational form seem to suit to situations where the study material cannot be partitioned into strict independent fragments. Because our areas of teaching are digital arts and software development, it is natural to instruct students using computer games projects.

However, the selection of a game as an educational form does not solve problems in learning such as misconceptions due to simplification. To deepen the students' understanding we have to cope with at least the following issues: how to verify and validate acquired knowledge, how to distinguish and intermix the essentials, and how conflicting interests in the domain of computer games are resolved. From our experience, in digital arts and software development a teacher cannot—and should not—expect pre-made plans to hold when teaching more advanced knowledge: To understand the 'whys' we have to consider holistically the contexts, contents, and organizations of the study topics. To meet these prerequisites with tested practices we (i.e., the teachers and the students) create computer games in actual software development projects.

There is a myriad of ways for arranging a software development project. We present three project configurations that depend on how closely the teachers participate to the actual project work. We describe and analyse our experiences from several course instances and student projects on computer game development, and discuss their implications for the teacher.

Keywords: Computer games, education, student project

TUCS Laboratory
Algorithmics

1 Introduction

The computer game industry has been historically founded on self-taught game programmers who mostly earned their credits from other fields. Even in the mid 1980s many commercial games were created by small groups who had an intriguing idea for a new kind of end-user experience and mutual ambition to put it into a publishable form. Since then, the industry has grown steadily more professional and large-scale. The increasing complexity of the computer game development demands advanced technologies and practices, and consequently the first academic programs for teaching game making started in the late 1990s.

International Game Developers Association (IGDA) laid down their principles for academic curriculum in 2003 [6]. It divides the area to different approaches (called "core topics") and assigns them as fields of study. The core topics are

- critical game studies,
- games and society,
- game design,
- game programming,
- visual design,
- audio design,
- interactive storytelling,
- game production, and
- business of gaming.

Each core topic has then subcategories that have a specific focus. For example, game programming is broken down to

- math and science techniques (e.g., Newtonian physics),
- style and design principles,
- information design (e.g., data structures),
- prototyping,
- testing,
- programming teams,

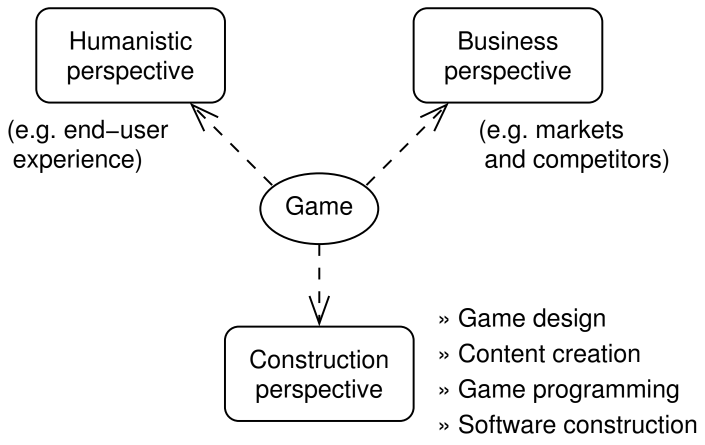

Figure 1: The three perspectives for making a computer game.

- design/technology synthesis,
- system architecture for real-time game environments and simulations,
- data-driven systems,
- game logic,
- multimedia programming,
- artificial intelligence,
- networks, and
- tools for designers and play analysis.

One can argue that the IGDA classification is just a list of techniques and approaches lacking broader perspective in the academic field. For this reason, we summarize three academic perspectives for computer game research and education (see Figure 1):

- Humanistic perspective is about how a game affects and changes the users, gaming communities, other social networks, and society at large.
- Business perspective concerns economics around computer games (e.g., competing on the markets, productisation, and investment strategies).
- Construction perspective deals with the actual making of an executable computer game.

Although each of the perspectives provides essential topics for study, in this paper we focus on the construction perspective. In other words, our viewpoint is the constructional aspects of software development within the domain of computer games. This has four subcategories:

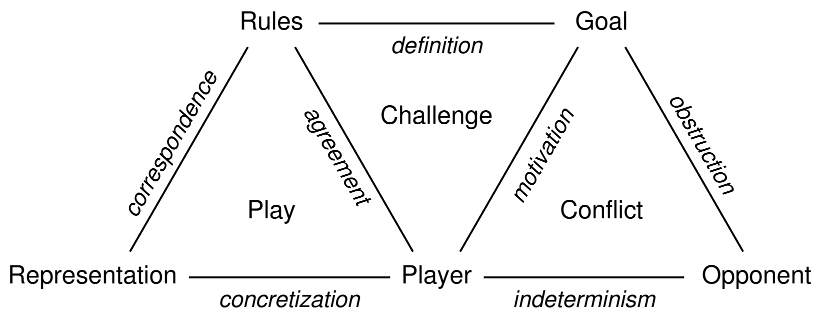

Figure 2: Components, relationships, and aspects of a game [9, p. 2].

1. Game design: What is the intended end-user experience and what game elements contribute to it?

2. Content creation: What are the entertainment artefacts and functionalities and how they are represented?

3. Game programming: How to implement the software mechanisms that serve the content creation so that the game design is achieved?

4. Software construction: How to weave the digital game components into an executable game?

These perspectives target product design and development and they categorize the main artistic and technical concerns in the making of computer games.

Let us consider how these perspectives relate to one another. The game design concretizes the vision, intention, concept and theme of a game. Among other things it includes activities for designing the back-stories, composing the opposing forces and the major roles, and developing the synthetic participants (e.g., in-game characters, sidekicks, and *deus ex machina*). Game design is a creative process but it is governed by the rules of play, since we need an outcome that fulfils the definition of game. Any system that consists of the following properties can be seen as a game: *players* who participate the game, *rules* that set the limits to the game, *goals* that motivate the players towards achievements, *opposing forces* that obstruct the players, and *representation* that concretizes the game. These elements form the defining aspects of challenge, conflict, and play (see Figure 2).

Nowadays, the demands on the aesthetics, end-user experience, and replayability of a game are high. To meet them cost-efficiently the game design is implemented in two fronts at the same time. Simply put, the intended game features can be divided into art and technology. 'Art' comprises the entertainment content, which is run on the technology platform. The platform requires specialized programming effort in some form (even if it is purchased from a third party). These two fronts require deep expertise, and thus, they form their own special areas. However, they are not dissociated in the game development, but one of the key challenges is bringing them together. In other words, when making a game we have to consider, for example, programming, 3D modelling, and animation together. The game industry has pioneered finding practices and technologies to have the interaction between many disciplines such as computer science, art and design, business, management, and productisation. One can argue that game development is an expertise area in itself by connecting these disciplines.

Because all these perspectives are present at the same time in a game, they are often developed in parallel: At first the emphasis is on the game design and gradually, as the game attributes and features become fixed, the focus moves towards content creation and software system issues. However, it is only seldom possible to stabilize and freeze the whole game design. Thus, there must be feedback activities that evaluate and control the possible changes to the game design. This uncertainty is intrinsic to software development in general and it arises from the digital nature of the computer programs. Especially due to different cost structures, software construction differs considerably from the manufacturing process for physical products.

Games are useful in education because they provide versatile but demarcate setting that can include almost any kind of a topic. Furthermore, games tend to intrigue the humans intrinsically, which can be used for motivation and encouragement. Games are interactive when playing and making them. This allows a teacher to blend in topics that develop social skills and group work practices.

Making computer games—and specialized education for it—teaches more than just creating the games. We claim that the domain of computer games is one of the most challenging software product domains, and thus, the students also learn beneficial practices used in the software industry [4] and in multimedia industry in general. To facilitate this extrapolation the teacher (or the group of teachers collectively) should have profound experience on the whole computer game domain, teaching methods, and project work.

In this paper, we discuss how to teach the making of computer games. To simplify the effort we place emphasis on the construction perspective. In Section 2 we introduce the main factors of game development that influence the education. In Section 3 we examine how to organize the study fragments into a curriculum. Section 4 consists of an introduction to project attributes, forces, and configuration. Sections 2–4 summarize the rationale of our educational decisions for a project where students construct a computer game. Section 5 presents models for game production, and their impact on teaching is discussed in Section 6. After that, we present three schemes of how to arrange a student project: In Section 7 the game development project is configured as a traditional assignment. In Section 8 the project is run as a research laboratory course where the teachers participate as on-site customers or consultants. In Section 9 we describe a project where the teachers work together with the art and technology students. Then, we discuss the excluded issues and sum up the lessons learnt in Section 10. Finally, the concluding remarks appear in Section 11.

2 Domain of computer games

Expertise on a complicated domain is not only about understanding advanced concepts and topics at hand. Deep know-how is also about knowledge with respect to the context in a given situation. For example, one must be proficient in applying various operation strategies and methods, evaluating relevancy, changing the context or the perspective, interconnecting the details with the abstractions, and recognizing failures. These skills cannot be categorized into isolated disciplines nor be learnt one by one. In our case, this observation applies to the digital arts as much as to the game software development, which together form the domain of computer games. However, the setting is not necessarily doubly complex, because in computer games the notion about 'context' can be approached from two aspects. These aspects are present at the same time but they can be considered sequentially. More specifically, in computer games it is possible to fractionate—with a certain cost and effort—the game content and the underlying technology. Computer game construction succeeds in joining two areas of expertise, the digital arts that creates the game content and the game software development that provides the game technology.

The bipolar characteristics of computer game development offers us two directions for scaling the game problems. On one hand, the amount, quality and finesse of the content concern mainly the effort put in by the artists; from the technology perspective these are managed via resource parameterization, not by changed functionality. On the other hand, the technology issues, such as state-of-the-art algorithms, networking, and compensation of resource limitations [9], matter mostly for the game software development; from the arts perspective they set the boundaries for the creative mind. In addition to these two polars, there must be a context that connects them. The game developers have many, often ingenious solutions for this purpose, but this topic is outside the scope of this paper.

From the education point of view, the bipolarity of the computer games has a significant consequence: There are many aspects that can be adjusted without losing the characteristic attributes of computer game construction. In other words, the central part of the work (in both the digital arts and game software development) made in large commercial games can also be demonstrated with small non-commercial games. This property of multiscalability makes the domain of computer games well-suited for teaching the actual profession of computer game making.

3 Organization of the study fragments

The selection of topics to be studied is not the only educational problem, because we have to decide how the content to be taught is organized so that new things do not pile upon the student all at once. Spiral curriculum [1, p. 13] is a common approach to achieve this at the introductory education level or with the topics that have a clear conceptual structure. For example, the fundamentals of software construction can typically be taught apart and without knowledge about application domains. In principle this is also the case with digital arts but the approach is often considered impractical. The reason for this difference is that the basic software concepts are mostly based on mathematical formalisms and detachable details, but in digital arts it is significant that the wholeness binds the minor features.

In the spiral approach, a topic is taught starting from a reduced and simple instance to more diverse and complex occurrence. This can be accomplished by organizing the study fragments of the topic by the following guideline:

1. Introduce the topic as simply as possible by a concrete example or instance. The students should be able to understand what is the topic compared to the other topics.

2. Demonstrate the most important variations of the topic. The students should be able to recognize the forms in which the topic appears: What are the common factors and how the changes relate to them.

3. Invite the students to create simple new variations from the topic. The students should be able to discern the topic and imitate the presented examples.

4. Present the topic as an isolated concept having a static aspect (e.g., structure) and a dynamic aspect (e.g., change) with respect to its context. The students should learn the schema of the topic and the fundamental rules associated with it.

5. Show the boundaries of the topic by exceptions, misuses, and conflicts. The students should grasp the applicability of the topic.

6. Invite the students to apply the topic in new non-trivial situations and to assess the outcome. The students should be ready for unaided study of the topic.

The phasing of the study fragments according to increasing complexity is the essence of the spiral approach. It underlines that recognition of a topic and understanding of it are different. For example, knowing a programming language does not make one a programmer or knowing how to use an image editor does not make one an artist. From the education perspective, the spiral curriculum specifies how the knowledge-base evolves, and thus, the selection of a study level becomes straightforward. For example, a teacher can adjust the starting level to match the students' skills or set prerequisites for the course. Furthermore, changing the pace of study, omitting some details, giving specialized teaching for the more advanced students, or jumping from one spiral topic to another do not mix up the curriculum. The advanced phases act often as a context for the simpler ones, which forms a continuum of abstractions. From the student's perspective this gives time for comprehension because falling behind does not blur the direction.

The downside of the spiral approach is that the student must be able to cope with uncertainty and to have confidence that the topic gets more understandable later. From the teachers' perspective the spiral approach requires deep understanding of the topics. Otherwise, the teacher is not able to prepare the spirals beforehand and the possible chains of dependencies between the spirals are lost.

Despite many pros and few cons, the spiral approach is too limited for advanced studies or with intertwined topics and we have to use more elaborate curriculum form, such as lattice [3]. As an example, we demonstrate the following higher-level issues to our students:

- what kind of contexts, aspects, viewpoints, and perspectives must be considered and how they are related,
- how the knowledge appears in various situations and how to interpret the knowledge,
- how the balance of conflicting interests reflects the intention behind a solution,
- what knowledge-seeking strategies suit for a certain kind of situations,
- how to apply the general lessons learnt in software development into the context at hand,
- what conceptual elements are relevant and how they combine,
- how the evolution of the solutions is managed,
- how to cope with inaccuracy, uncertainty, risks, and mistakes, and
- how to plan and what, and especially, how to change on-the-fly the approach or the way of working to resolve surprising problems.

We teach these issues inside the specific domain the students are working with. The idea is that the more advanced the topic is the more the students should form the idea of it by themselves; the teachers' role is just to be catalyst by giving concrete instances.

Because the making of computer games includes both well structural and non-structural concepts, we apply both spiral and lattice approaches when organizing the study fragments. The overall setting from the technical perspective is depicted in Figure 3. The computer game has some use features that can be observed and experienced by the player. For example, the player can share the game world with a remote player via networking or the player can record the game play for later viewing. These use cases of the game are implemented by some technical functionalities that are, due to a need for resource optimizations, only seldom in one-to-one relationship with the features. For example, 'start and stop' of the game is often considered in isolation from the actual game flow. Furthermore, 'start and stop' can be structured, and thus, it can be approached by the spiral curriculum. Because the features and functionalities have complex many-to-many relationships, the game is not made one spiral at a time but the spirals are advanced gradually over time in a suitable order. The making of the game involves also non-structuralities that require more sophisticated approach. For instance, let us consider decision-making at the strategic level of software development. We must take into account among other things conflicting interests, different perspectives, the current situation at each workflow discipline, the upcoming situations, the workers' capabilities, and risks involved. At this level, to learn methods for knowledge navigation and knowledge transferring one has to have experience from concrete project cases.

4 Project as an instruction form

The organization of the study fragments does not assert how the actual instruction is carried out. This leeway to choose how the teachers and the students co-operate narrows down when we take into account, for instance, the following issues:

- resource limitations, such as time schedule in the study period, available learning materials, and the number of students per teacher,
- risk management policy, for example concerning the absences and dropouts, and
- goals of the course, such as acquired proficiency levels in the study fragments.

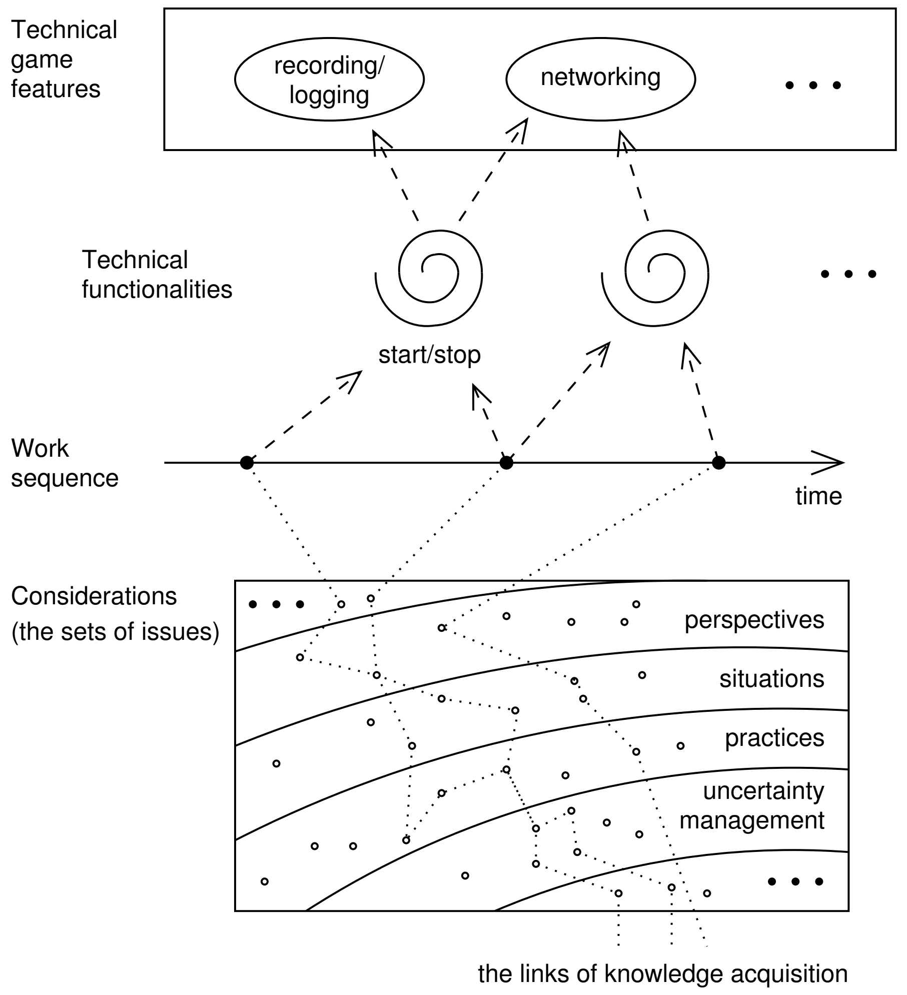

Figure 3: The organization of well structural and non-structural topics in a technical software development. The arrows indicate conceptual dependency from dependant towards dependee. Dotted line connects the considered issues in the 'landscape' of software development concepts.

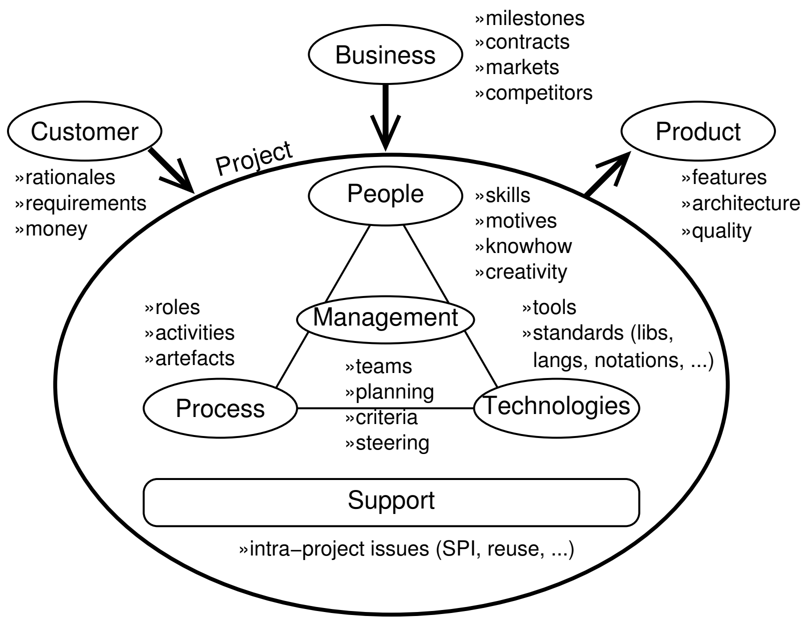

Figure 4: The typical attributes of a project that produces a product [4].

It is not uncommon that in the advanced education the resources are scarce, the working environment is volatile, and the planned results change over time. Although this is also the case in the digital arts and in the software industry, we are in a beneficial circumstance: We have opportunity to put the students in the situation that resembles the actual work in the companies. This gives the idea to arrange the instruction as a project.

A project is an undertaking that lasts for limited time period (i.e., has start and end) and aims to produce an unique result. In a company a project can be depicted as in Figure 4: The needs of a customer and the company's way to make business shape the project so that its outcome, in this case a product, is beneficial for all the participants. Typically, a project has five main attributes: The *people* operate together by following some *process* and utilizing development *technologies*. This effort is facilitated, co-ordinated, and supervised by *management* activities. If many accomplished projects are similar the collected expertise can be used to *support* the new resembling projects.

We configure the instruction of the selected study topics as a project on the computer game domain. To concretize the topics the end result of the course is a finished product (i.e., a computer game). This choice gives us methods to validate and verify the acquired knowledge (e.g., have the students made rational decisions and does the game fulfil the given requirements and specifications). Also, due to long history in the software development industry, the project-like form of instruction advises how to distinguish and intermix the essential activities and work products. For example, there are many proven practices for prioritization and organizing parallel work. The project practices are imperative also in resolving the conflicting interests and forces that are involved in the making of computer games.

A computer game development project is demanding educational method because it integrates the education aspects with many other areas of knowledge and with technical capabilities. For example, all the participants, while attending the instructions, must have adequate knowledge on teamwork, general software development, applicable technologies, tools, project management, and the specific problem domain. This results often in a conflict, especially when the educational goals are high and the time schedule becomes tight. To keep the project feasible and manageable we have to reduce the requirements for the teaching. Apart from getting rid of the irrelevant details, there are two dimensions for the simplification: We can leave out the project attributes, as summarized in Figure 4, or we can specialize more the project features.

We claim that ignoring the key project attributes eradicates the whole idea of real-world experience and without the attached interest groups the over-simplification leads apace to ordinary group work assignments. This increases the possibility for misconceptions and introduces conceptual distortions that can interfere with the more advanced learning given later. Thus, we are interested in preserving the inherent complexity of the domain knowledge; instead of mock-up projects we recommend true projects where the project, product, and process issues are present together. To make the project simpler we inject more assumptions into it by adding further non-volatile requirements and forces. These additions are educational in nature, and they can modify the project so that it can be carried out with the given resources (especially with respect to time, because it is a course with a set end date).

The problem with this idea of simplification by assumptions is how to actually configure the project. This depends on the case, but as an example, we can

- ascertain the students' starting level to match the pre-requirements so that a student does not exhaust and quit the course,
- run greenfield projects over clear application programming interfaces (APIs) rather than extend a legacy system so that the progress is not buried,
- make variable details static so that the structure of the topics emerges in such a way that serves the course,
- prioritize the usual 'happy' cases over the exceptions and error cases,
- detract the productisation issues by introducing intermediate objectives that are valid handover points of the project results, and

- assume short product life-cycle so that there is no need for maintenance and reuse issues.

In general, the setting of a project configuration goes as follows: Before starting the project the teacher must determine what are the educational goals to be demonstrated with respect to the actual real-world issues. Next, the teacher establishes the perspectives from where the selected study topics are to be approached. Then, by using hands-on experience and imagination, the teacher collects the simplifying assumptions so that the project becomes feasible without losing the intention and the central aspects of the real-world work. In our case, we select and abstract the topics in the computer game development. For example, the goal can be an introduction to multiplaying via a local area network (LAN) and the perspective can be a construction of communication technology. From this sub-domain we can select the topic, for example, the implementation of proper balance between consistency and responsiveness for a given game. To simplify the project we can stipulate that the LAN connections are slow and have narrow bandwidth, the self-steering software team is located into one room, and the customers want only to see the alpha release of the game with just the placeholder graphics in place.

It is clear that there is a correspondence between the problems in learning and the practices in a software development project: Obvious matters need not to be treated with a project, and the non-obviousness arises especially from the unknown. Let us cover the key issues of learning (italicised in the list below) and how they are matched up in a game development project. The prime learning aspects and issues are summarized, for example, in [3].

- *Sufficient base-knowledge gives the students readiness to learn advanced and abstract topics.* Before the project starts we can arrange selection interviews to assess the students' skills and knowledge on technologies, mechanisms, and application domains.

- *The novices differ from the experts in relation to amount of knowledge, way to acquire further knowledge, way to apply knowledge, and thinking strategies.* The diversity can be put to use because proper mixture of novices and experts in the project steps up the activities in peer-learning, mentoring and coaching.

- *Prior learning, including misconceptions, affects the later comprehension.* Because the misconceptions are inevitable, we should detect them as promptly as possible. Thus, the project must have many levels of feedback loops that makes it possible to monitor the ongoing work. For example, the feedback can go through the co-workers, the testing systems, the team leader, the teachers, and the customers.

- *A complex topic can be understood by going through many different learning episodes from it.* In the project this is captured by iterative-incremental development where, essentially, iterative development is about proceeding in small pieces and incremental development is about building on the previously acquired system.

- *The more advanced the knowledge to be learnt is the less there are right and wrong solutions; instead the solutions should become more credible.* The same applies in the software development: The system is not right or wrong, it is usable or less usable for a customer. Furthermore, the system testing does not prevent all faults but it increases the belief and plausibility.

- *The advanced knowledge cannot be taught in isolated categories if they overlap or have deep interconnections.* Analogously, in the making of a software system the most of the challenges are caused by such parts that comprise many, even conflicting, decisions. In the project a complicated situation can be simplified by stabilizing the non-structural issues in order to make them more compatible with the structural ones. For example, in the computer game development some flexible description interfaces can be utilized to separate the construction of the arts and the technologies.

- *The assessment of the knowledge acquisition must take into account the desired educational goals as well as the causes for the comprehension failures.* When the project is concluded it can be analysed in a feedback meeting. One practice is to write a post-mortem document from the project which lists a few most critical reasons for the successes and the failures.

5 Game production

The goal of game production is to produce fully-working game products. It builds on the tradition and the methodologies of disciplines such as project management, software engineering, and culture production. Game production life-cycle defines the basic activities which have to be accomplished before a game idea actualizes into a working game product. Although every game and every production team has its own ways of working, some generic models for game production life-cycle activities have been proposed (e.g., game production process and game production hand-book).

Figure 5 visualizes the main phases in two game production models [8, 2] and in one software product development model [5]. The essential activities in the production and development phases of each model are also listed in the figure.

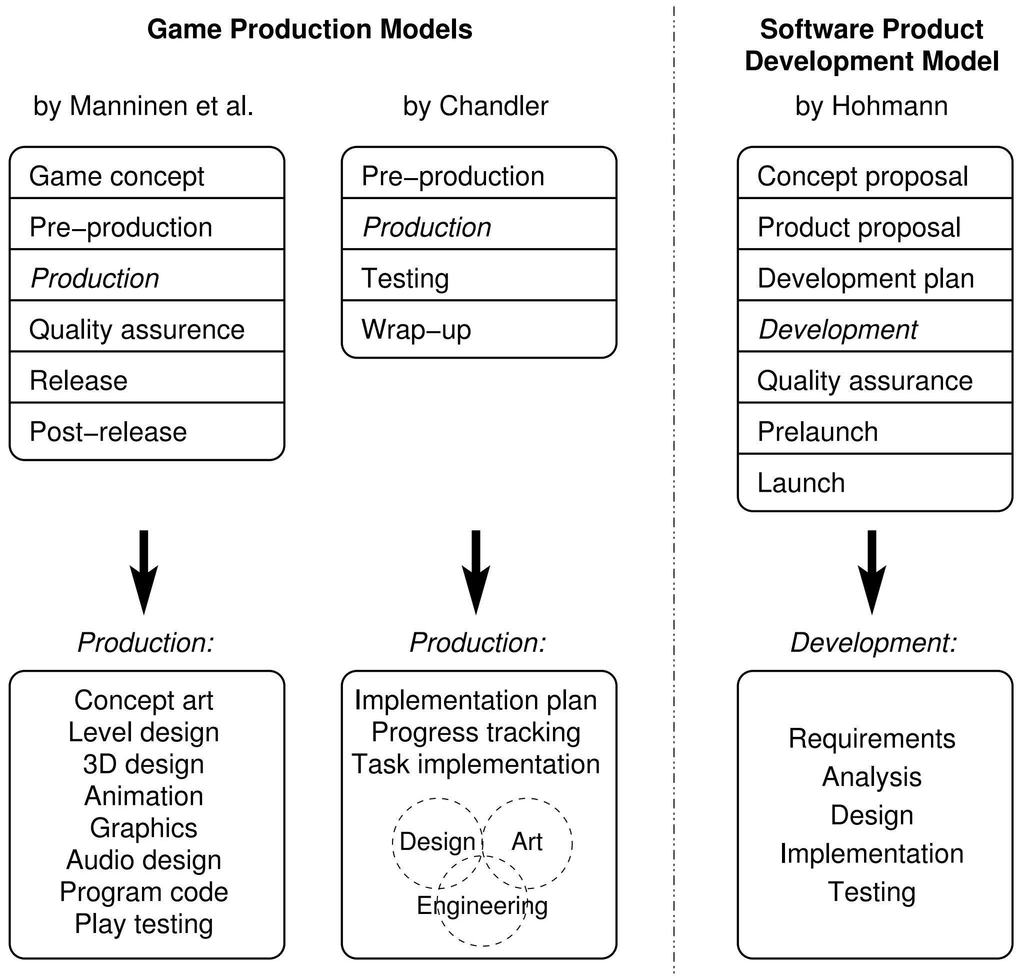

Figure 5: Comparison between game production models of Manninen et al. [8] and Chandler [2], and software product development model of Hohmann [5]. The development phases are described on the upper half and more detailed production phase steps on the lower half.

If we compare game production models to the generic software product development model we find notable similarities. All models have a starting or a pre-production phase when a (game) product idea is defined and its feasibility is evaluated from the production and business viewpoint, a production phase when the actual product is developed, a quality assurance phase when the final quality of the product is tested, and a release phase when the project is wrapped up and the product is released to the customer. Manninen et al. [8] address the post-release activities (e.g., bug-fixes and user support), which are a natural part of the product life-cycle.

Development and production activities are also quite similar between the game production models and the software product development model. The multidisciplinary nature of game production is the main difference. While the activities in the software product development model focus solely on the software development and testing, game production activities include co-operative work of artists, animators, game designers, level designers, musicians, and programmers.

Although production modes and best practices exist, surprisingly many game production projects do not utilize well-known project management practices. Some of these productions still manage to be highly successful [7] which implicates that there is not one right way to carry out (or teach) game production but also that the production training has its place.

6 Teaching game production

Our curriculum has included other game related issues, such as game algorithms, artificial intelligence, object-oriented programming patterns, interactive storytelling, game design, and co-operation between artists and programmers. Since our universities do not have a degree program in game development, our goal is to teach the students general skills on computer science, software engineering and digital arts. Although teaching game production process itself was not our main goal, we constantly ended up with course formats where the students go through the game production cycle. We observed that using a game production as educational format creates several opportunities:

- *Versatile but demarcated setting.* From the teachers' point of view, a game can serve as a mixing pot for various study subjects to be included into a curriculum. Also, the project-like form of instruction helps us to distinguish and intermix the essential activities and work products.

- *Pragmatic teaching approach.* It is easier for the students to understand advanced issues connected to the game content when they actually make games themselves.

- *Higher motivation.* The students are motivated by real game development projects (i.e., simple “tic-tac-toe assignments” do not hold students’ interest long enough).

- *Visible results.* A working game demonstrates the acquired knowledge. For example, the students can assess themselves whether they have made rational decisions or fulfilled the given requirements and specifications. Also, the user features are more visual in a game than in other kinds of software products.

- *General applicability.* We claim that the domain of computer games is one of the most challenging software product domains, and thus, the students learn also beneficial practices used in the software industry [4] and in multimedia industry in general.

There are also several challenges in using game production as a teaching method:

- *Sufficient multidisciplinary teaching skills.* To facilitate this extrapolation, the teacher (or the group of teachers collectively) should have profound experience on the whole computer game domain, teaching methods, and project work.

- *Readiness to learn.* Students participating game development courses must be proficient at their own study field. The digital design and art students should have traditional visual art skills as well experience on the software tools used for digital art creation. The software developers should master programming and information system design. However, they must also be able to apply the acquired knowledge in the game domain, be ready to work in teams and willing to quickly learn new work practices.

- *Uncertainty.* It is not uncommon that in advanced education the resources are scarce, the working environment is volatile, and the planned results change over time. For this reason, the students must be able to cope with uncertainty and to have confidence that the topics get more understandable later.

- *Controlled simplification.* Over-simplification of complicated and intertwined topics can lead to misconceptions and conceptual distortions that ruin the learning. However, to keep the project feasible and manageable we have to reduce the requirements for the teaching. Apart from getting rid of the irrelevant details, we recommend that the inherent complexity of the domain knowledge is preserved. We claim that injecting more assumptions (i.e., non-volatile requirements and forces) into real-world project carries out the actual simplification.

The next three sections present three different approaches—traditional assignment course, research laboratory course, and intensive course—which we have used in our curriculum.

7 Traditional assignment course

In traditional assignment course[1] students form a team and solve the given problem assignment as a homework exercise. In the software development industry this kind of arrangement is called outsourcing, and the goal of the assignment is to implement a complete computer system. To concentrate the students' effort on the relevant issues the teachers lecture the theory behind the selected study topics beforehand. Hence, the assignment should demonstrate the practical applications of those theories.

Although an assignment course concentrates on the production phase of a game project, the preceding phases must also be taken into account. To give the students a grounding for ideas each student analyses individually some existing game that fits in with the course topics. Then each student writes a game treatment that suggests the main ideas and features for a new game. These game treatment documents form the basis from where the actual game concept is developed at the beginning of the production phase. To keep project management issues simple one team consisting of three to five students runs the game production life-cycle. If more students enrols the course, each teams is given the same assignment.

The main activities of this course type are illustrated in Figure 6. The project work can be divided into three phases: inception, construction, and conclusion. At the inception phase, the teachers prepare the assignment and the theory lectures. This activity begins with selecting the study topics, outlining the game concept, and determining the results of the preproduction phase. These three work products determine the perspective and depth of the theory issues that are tackled in the project. After that, the project setting is introduced to the students in lectures where the theoretical background is taught. To support the game production aspect the lectures should include practical guidance for solving the most crucial problems. Whereas the lectures ascertain that the students' starting level matches the prerequisites, the presentation of the actual assignment acts as a decision point where the students either commit themselves to the course or drop out. The presentation of the assignment is one of the key activities for the successful execution of the assignment. The teacher should write down explicitly and

[1]Acknowledgement: The case is based on the on-line course Games and Virtual Environments ("Pelit ja virtuaaliympäristöt") lectured by Dr Tony Manninen from the University of Oulu. The case study presented here is based solely on the assignment work carried out by the teachers and students of the University of Turku.

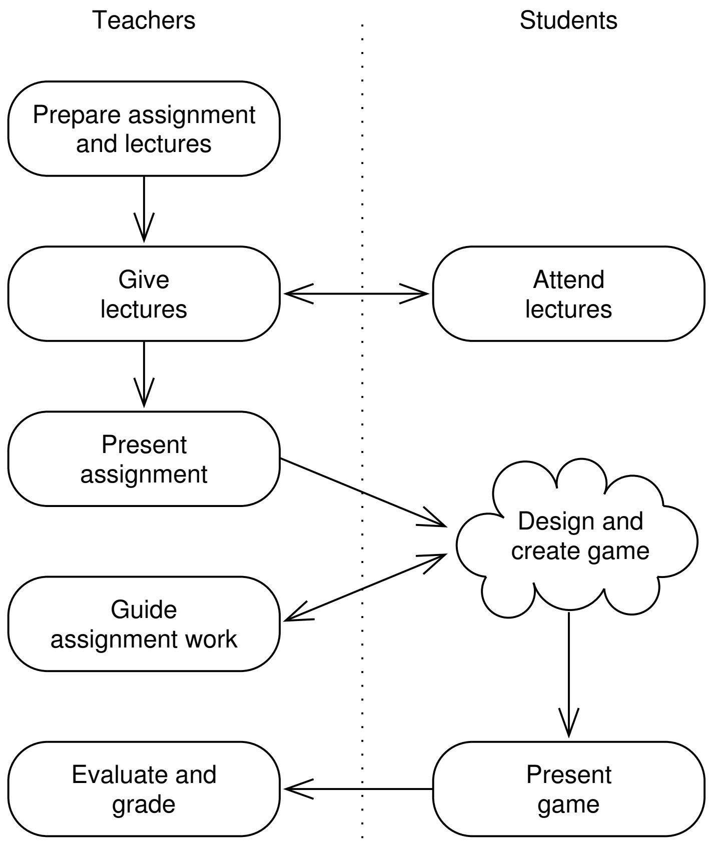

Figure 6: The main activities in a traditional assignment course.

explain the assignment milestones, communication mechanisms and practices between the teachers and the students, references to guidance and extra resources, and the grading principles. This focuses the students' work on the assignment and they do not have to consider the organizational details of the assignment.

In the beginning of the construction phase, the students are taught and prepared for the forthcoming teamwork. Especially if the students already have experience on teamwork and the assignment is extensive, the teachers can advise how to divide the work into sub-teams. However, the main idea of traditional assignment course is that the students organize their own work when making more elaborated design and creation of the game. The teachers' role is not to control how the team operates, how the development conflicts become resolved, or what tactical decisions are made. Thus, this is the most demanding project course format we give to our students. The teachers act as outside customers and they follow the progress through the project milestones set in the assignment. For example, the teacher assesses the feasibility of the work plans. The teachers also provide external support on request for the project, for example, by advising, guiding, and coaching in situations that halt the project. In an extreme case, the teachers can intervene and freeze the project until the problems are analysed and further actions are found.

The rationale behind this kind of autonomous assignment is to teach tacit knowledge about software development in teams. For example, the students should find out themselves how to take responsibility, communicate, and share know-how. From our experience, if the students have sufficient initial competence and the preceding activities are effective, they do not need extensive assistance to accomplish advanced results.

As in projects in general, the assignment has a strict deadline after which the students present the game, demonstrate its features and write a post-mortem document from the project. The writing of a post-mortem about the game project is a retrospective practice, where the participants list and analyse the most influential successes and failures. In addition to concluding discussion, the post-mortem document sums up the tactical and strategic lessons learnt and provides the teachers feedback from the inception and construction phases. Lastly, the teachers evaluate the results and grade the students on the team level. We prefer not to evaluate each student individually because a team should be seen as one development unit; the balancing and fairness issues must be considered during the project, not after it. In principle, a student can be expelled from the project but in our course instances this situation has never occurred but the students have kept their enthusiasm throughout the course duration.

8 Research laboratory course

In a research laboratory course, the students alongside the teachers examine a scientific problem, work to find a practical solution to it, and report the results to the whole class. Usually, the course is aimed at advanced level because it requires that the participants to have pragmatic skills of collaboration and knowledge acquiring methods. In the software development industry, similar kind of work is carried out in customer-on-site projects. This course format has two goals: Firstly, the students learn about the latest scientific topics and they have hands-on training into scientific work. Secondly, the teachers—who are also researchers—benefit from fresh ideas and possibly new solutions to the research topic. In our case, this kind of co-operation has often led to further studies and even to joint research papers.

Our latest instance of this course focused on software technologies, and especially how certain object-oriented design patterns affect the software architecture of a multiplayer game. When the research question is specific and mainly technical one, the influence of the preceding and succeeding phases of the game project can be reduced. This reduces the risk of failure, although the outcome of the project tends to stay unpredictable until the deadline.

Nevertheless, this poses also an educational problem because we are leaving out real-world problems concerning, for example, the launching of the project. Fortunately, there are adequate solutions to this such as organizing a separate project course without detailed construction phase. To keep the project management as simple as possible one team includes five to seven students. If more students have enrolled the course, each team is given the same problem and the teams are encouraged to find a balance between co-operation and friendly competition on technical and content assets. Also, if the problem turns out to have many differentiating aspects, the teams can decide to distribute them.

The main activities of this course type are depicted in Figure 7. The project work consists of three phases: inception, co-operative construction, and conclusion. At the inception phase, the teachers conduct also the whole preproduction phase. At first, the teachers prepare the assignment based on their research questions which determines the main concerns of the project. Then, to make the experiment repeatable and to have a case setting the teachers design the game concept and features. The game design is used to constrain the general research question; the idea is to have a non-trivial case example that is feasible to solve with the given resources. The assignment and the game design are presented to the students after which they commit themselves to the course. Finally, the teachers guide the students to organize the teamwork and to select practices, methods, and development tools. Unlike in the traditional assignment course, the inception phase can be rather informal if the research problem is well-defined

The construction phase of the pre-made game design is realized in several iterations where the teachers and the students work alongside. The students make decisions and construct the game mainly at the operational and tactical levels. They are responsible for software and game level design, iteration planning, system and content development, and unit testing. To facilitate the construction the teachers can operate also at the tactical level but they should avoid affecting the results. Instead of giving ready solutions, the teachers should reveal on need-to-know basis what aspects and alternatives the students should consider. In other words, the teachers act as on-site customers who are interested in the outcome and not the means. This puts the students into a situation where they have to think proper questions and possibly find out unforeseen answers. The teachers control the strategic work by organizing frequent follow-up meetings where they verify the progress, prioritize the game features, and guide the forthcoming tasks.

The construction phase has a strict deadline after which the project proceeds to the conclusion phase. At first, the students present the game demonstrating its features and discuss the project in retrospective. Unlike in the traditional assignment course, the teachers should write the post-mortem document because they are more experienced in reflecting scientific issues.

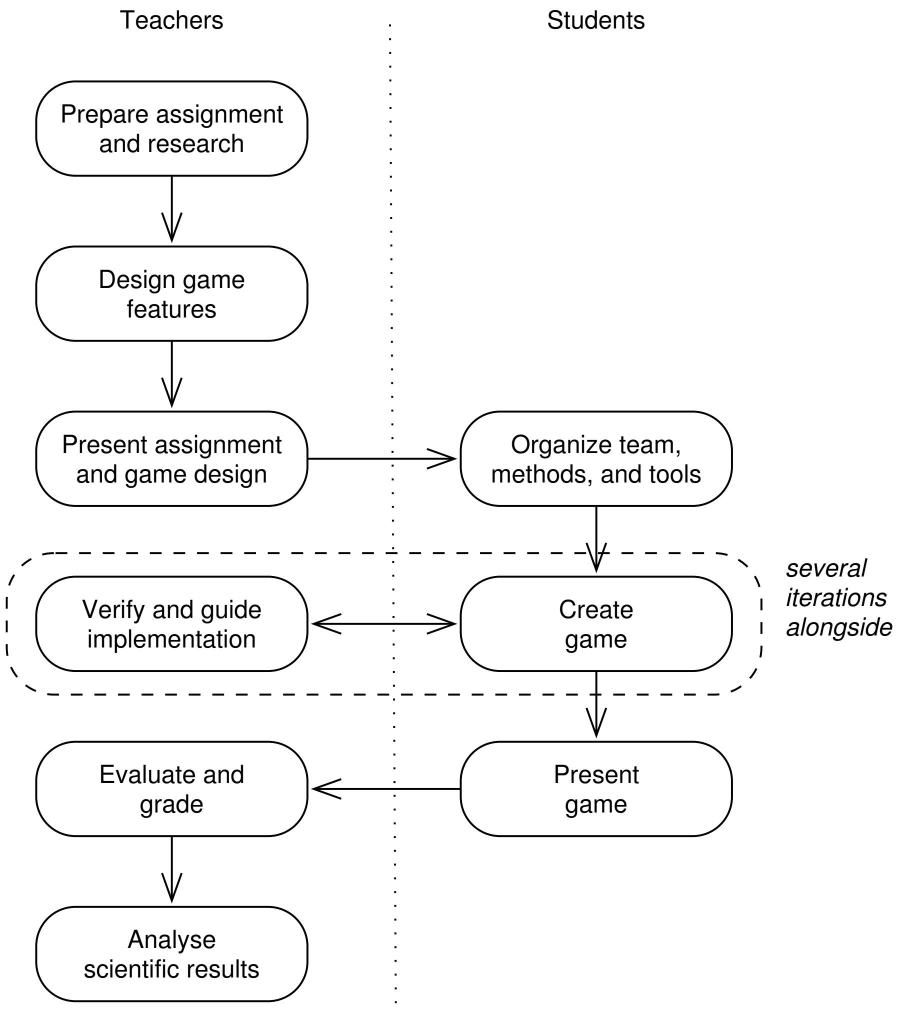
Teachers
Students
Prepare assignment and research
Design game features
Present assignment and game design
Organize team, methods, and tools
Verify and guide implementation
Create game
several iterations alongside
Evaluate and grade
Present game
Analyse scientific results

Figure 7: The main activities in a research laboratory course.

After that, the teachers evaluate the outcome and grade the students. Although we consider the team as one, individual grading is also possible because the iteration work has been transparent and the game design is complex enough so that it leads to clearly specialized and dedicated roles. Lastly, the teachers, preferably together with the students, analyse and report the scientific results. Often this can result in other research questions, study papers, or thesis work.

9 Intensive course

In an intensive production course, the students and teachers work together as one team to produce a computer game that demonstrates both artistic and technological skills. The production goal is to create a functional game prototype in a short period of time. The project proceeds in intensive workdays of six to eight hours at one site with interconnected computers. In the software development industry, this is called as customer-as-expert project. Due to the intensiveness, each participant must attend to all the workdays to be at the team-mates' disposal and to keep up with the project's pace. The educational goal of the course is to encourage the students with different backgrounds (e.g., digital arts and software technology) to work together. Also, we foster mutual learning and extend understanding of the multidisciplinary nature of game development.

The main focus of the course is collaboration between different disciplines so that it concretizes as a computer game. This approach gives us leeway to adjust our emphasis for the course without changing its format. For example, if all the participants are already familiar with the development tools in their own expertise area and skilled in handling versioning tools, the course can tackle more content creational and technological challenges. On the other hand, the course can emphasize the challenges in multimedia design and art by selecting ready-made technology platforms with uncomplicated scripting languages. Thus, the teachers can put the students in a situation where they are kept motivated: They have meaningful tasks from their own discipline but, at the same time, they have to understand their impact to the whole project. The latter is not possible without collaboration over the discipline boundaries.

As with the other course types, we prefer one-team co-located projects because we want simple communication practices. Because the project has many disciplines and the teachers also work as team members, the teams become relatively populated. To keep the number of participants between ten and twenty the students apply to the course with an informal application letter including a curriculum vitae. This allows the teachers also to plan the course adjustments, possible sub-teams, and role descriptions beforehand.

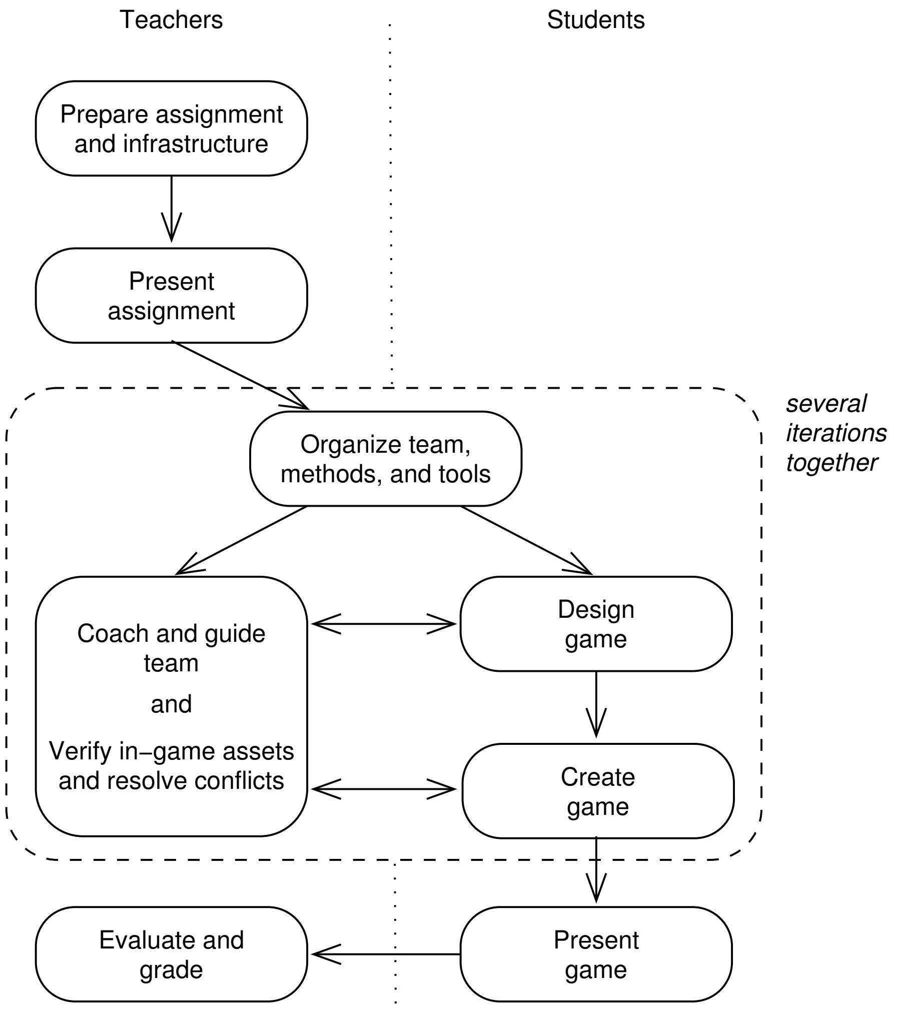
Teachers
Students
Prepare assignment and infrastructure
Present assignment
several iterations together
Organize team, methods, and tools
Coach and guide team and Verify in–game assets and resolve conflicts
Design game
Create game
Evaluate and grade
Present game

Figure 8: The main activities in an intensive production course.

From the three course types, intensive course is the most challenging for the teachers because the schedule is tight, the workdays turn sporadically into chaos that must be managed, and in-advance and on-the-fly planning have to be constantly balanced.

The main activities of this course type are presented in Figure 8. The project work has three phases: inception, continuous construction, and conclusion. In the beginning of the inception phase, the teachers prepare a preliminary schedule for the course assignment and set up the infrastructure for the project. The actual game idea and design are still left open. The inception ends with assignment presentation where the students are introduced to the project topics and facilities. Because the students have passed the application process they have already been committed to the course.

The continuous construction begins with an open debate and brainstorming on game ideas, their possibilities and the students' interests in them. It is important that in this stage the students form a mental bond to the game, and thus, the teachers should act only as moderators trying to shape up even the oddest ideas. In our courses, the proposed game idea have included, for example, ladies' tea-party poisoning contest, medieval time-travelling to fetch the last virgin, and a non-violent first-person shooter (of which the last one got selected). Then, the most promising ideas are refined and among them the core game idea is selected. This is where the teachers experience is needed to consider the amount of required work, required production tools, and overall technical feasibility.

The actual project work iterates the idea into a computer game. The teachers take the responsibility for project management and product level quality assurance. Their first activity is to organize teamwork, practices, methods, and development tools. Then, they coach and guide the students, verify art assets and software quality, resolve workflow conflicts, and plan and prioritize the creation process. In other words, the teachers act as on-site customers who are experts on the disciplines and actively participate in the development of the game. The students are responsible for the game design and its implementation. In other words, they have to make decisions on the game creation from the operational level up to the strategic level. For example, the students must assign a level designer who looks after the playability and immersion of the game. The most difficult problems arise from where the disciplines are bound tightly together into one.

As with the other course types, the construction phase has deadline when the conclusion phase begins. Despite the teachers participation to the project, the students present the game in the final project meeting. The relaxed atmosphere usually creates long retrospective discussions and the teachers can verify their understanding about the successes and failures before writing the post-mortem document. Finally, the teachers evaluate the results and grade the students. As in the traditional assignment course, we prefer to evaluate the team as one unit, because in this kind of intensive project individual work is subject to circumstances.

10 Discussion

The preceding sections described the main activities involve in three different teaching approaches and discribed the differences between them. Let us now list the common characteristics of the three approaches and summarize the key lessons we have learnt during the courses.

Lesson 1 *Game production provides a natural framework for project courses.* The game production provides characteristically suitable context for project-oriented courses, because the projects are not mock-up projects constructed to meet only pedagogic needs. Instead, the game production acts as a practical framework to which the pedagogical elements are injected. This partly explains why the students are motivated to pass these courses and feel that the course setting is quite realistic.

Lesson 2 *Larger team sizes has to be managed.* Other characteristic of the game production is that the productions are usually self-organizing one team projects. This can make course organization and evaluation harder when the course size increases. We have found two methods to handle the problem: (1) divide the students into smaller teams each with own responsibilities, or (2) divide the students into smaller teams and give the same production task to each. Surprisingly, using the latter method the teams still end up with completely different results.

Lesson 3 *Course formats do not restrict what kind of games can be developed.* In the courses presented here, students have produced a wide variety of different games from 2D real-time strategy games to 3D first-person shooters. Although none of the games meets the modern commercial standards, with a small further development effort many of them could reach the level of shareware or independent games. Since the majority of the students have been computer science and engineering students, the emphasis of the game development has been on the technical issues of the game production. Still, some of the games have also been visually appealing, partly because the student groups have been experimenting with unorthodox game concepts. The course formats presented here do not set restrictions themselves on what kind of games can be produced, but the length of an academic term sets the timeframe and limits what can be done and how polished the resulting games will be.

Lesson 4 *Assignment format may seem easy for the teachers but it has pitfalls.* For the teachers, project courses are basically easy because students do most of the work. Naturally, the preparations for the course and assignment have to be prepared well so that students can focus on the assignment work itself. In the research laboratory course and intensive course approaches, the preparation activities can be seen as a part of the pre-production of the game and, therefore, they affect directly the end result of the course. The teachers have to follow the students' work throughout the course and assist them as needed. The effort for this task depends on the activity and skills of the students.

Lesson 5 *Pre-production and production phases are emphasized during the project, but it still includes the whole production cycle.* Although main goal of our courses has been teaching game production related issues and not the game production itself, we have to know which parts of the game production used approach teaches. If we compare the workflow models of each teaching approach and the generic game production models, we can see that the activities related to the pre-production and production phases are emphasized. However, all characteristic activities of game production are present in the course structures in one way or another. The only significant omission is the business aspect of game production. Because the basic activities of a non-commercial small-scale game project are the same than the activities of a large-scale commercial game project, each teaching approach gives students a good insight into actual game production.

In addition to the production phase, we have to consider the production disciplines. All game production activities are done in small scale during the courses and, therefore, all game production disciplines are covered from digital arts to software technology. The emphasis between the disciplines can be adjusted by selecting the course objectives and needs. The teachers have to be set the main focus of the course and understand that there are students who do not have excellent skills in each necessary discipline.

Lesson 6 *Game production teaches skills that can be used outside the game production domain.* The teachers should pay attention also to how well the skills learnt can be generalized outside the domain of game production. During a game production course, software engineering students can deepen their knowledge of the software engineering issues, since the game production process resembles general software engineering processes. The skills obtained are also applicable to other kinds of software project outside game production. The multidisciplinary nature of game production teaches software engineering students that software projects are not only about programming but there are also other things to consider.

Although the multidisciplinary nature makes a game production project more challenging to manage, it makes concepts like project roles and release integration more concrete to the students. From this perspective, game production assignments are good all-around project management and team work practices.

The comparison done between the game production models and the software product development model in Section 5 reveals a rather interesting fact: Game production is actually a product development process rather than just a software engineering or art production process. This means that the end result of the process is a complete package, which can be evaluated more easily than an individual algorithm implementation or a prototype application. At the same time, the prerequisites have to be set more carefully and end-user needs have to be taken into consideration throughout the process.

11 Final Remarks

In computer game industry, a production project binds together many perspectives and disciplines by inventive balancing of conflicting interests. Because the balance of decisions culminates in the game production, game creation is a holistic process. In other words, we cannot isolate parts like digital arts and software technology assets, project activities (e.g., requirement management, risk management, budgeting, resource allocation, prioritization, quality assurance, and motivation), productisation, product line management (i.e., strategic business decisions on products), and marketing from each other without losing a realistic representation of the game development. However, due to time and budget limitations, the game creation education cannot take into account all of these perspectives at the same time.

Instead contenting with mock-up projects where some of the production perspectives are just ignored, we have identified three project schemes used in software development industry that also suit student projects. The traditional assignment course emphasizes game production work, the research laboratory course focuses on technological design and implementation of a game, and the intensive production course stresses collaboration among the selected game disciplines. These three case schemes demonstrate that it is possible to arrange game projects that have educational goals and reflect the projects in the software industry in general. There are game production models and general software product development models that support this observation. The successful injection of educational goals into the game projects is also well-founded.

The three game project schemes involve much software technology work, because computer games are executable programs, where the software platform runs the game content. We utilize this relationship to give the project adaptations concrete form and to determine the consequences. For example, there are tried practices for estimating how much time a given software implementation work takes. From study topic aspect the reason is that we have wide know-how on how to implement the game technicalities [9]. However, it is a misconception to believe that the essence of a computer game comes from technological solutions alone. The game must have intriguing end-user experience (entertainment, education, or other kind) that is only obtained by creative content and playability. For this reason the involvement of artistically oriented students is important.

To teach game production the teachers have to jointly understand the practicalities of computer game domain, education methods, software development projects in general, and ludic values of game playing. Only then they can educate the students not only about game production but also general software production and modern team practices.

References

[1] J. S. Bruner. *The Process of Education.* Harvard University Press, 1960.

[2] H. M. Chandler. *The Game Production Handbook.* Game Development Series. Charles River Media, 2006.

[3] A. D. Efland. The spiral and the lattice: Changes in cognitive learning theory with implications for art education. *Studies in Art Education*, 36(3):134–153, 1995.

[4] H. Hakonen. Game industry versus software industry: Similarities and differences. In A. Tuominen, editor, *New Exploratory Technologies 2006*, pages 124–132, Salo, Finland, Sept. 2006.

[5] L. Hohmann. *Beyond Software Architecture: Creating and Sustaining Winning Solutions.* Addison-Wesley Signature Series. Addison-Wesley Professional, 2003.

[6] International Game Developers Association. IGDA curriculum framework: The study of games and game development, 2003. Available at ⟨http://www.igda.org/academia/IGDA_Curriculum_Framework_Feb03.pdf⟩.

[7] S. Larsen. Playing the game: Managing computer game development. Technical Report International Edition, Version 1.1, Blackwood Interactive, 2002. Available at ⟨http://blackwood.dk/pdf/PlayingTheGame-IE.pdf⟩.

[8] T. Manninen, T. Kujanpää, L. Vallius, T. Korva, and P. Koskinen. Game production process—a preliminary study. Technical Report v1.0, LudoCraft/ELIAS-project, 2006. Available at ⟨http://ludocraft.oulu.fi/elias/dokumentit/game_production_process.pdf⟩.

[9] J. Smed and H. Hakonen. *Algorithms and Networking for Computer Games.* John Wiley & Sons, Chichester, UK, 2006.

ISBN 978-952-12-2105-7
ISSN 1239-1891