2010 IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning

Extending Google Android's Application as an Educational Tool

[1]Bian Wu;[1]Alf Inge Wang;[1]Anders Hartvoll Ruud
[1]Norwegian University of Science and Technology,
[1]Norway
[1]bian,alfw,anderru@idi.ntnu.no

[2]Wan Zhen Zhang
[2]Guilin University of Electronic Technology,
[2]China
[2]zwan_zer@163.com

Abstract—**This paper introduces how to extend Google android platform as a game development tool to learn software architecture based on the double stimulation method. It starts with the motivation to choose the android platform since most of students in software architecture course from NTNU (Norwegian University of Science and Technology) have experiences of using java and eclipse platform before they starts this course. And then it describes the design and construct of extended android platform, called "Sheep" framework. Further, it presents the application of the Sheep framework as second stimulus means integrated in the game exercises in the software architecture course. Finally, the paper discusses the contribution from the aspects of technology, game ideas and pedagogy.**

Keywords- Google Android; Higher Education; Double Stimulation; Software Architecture; XNA; iphone SDK

I. INTRODUCTION

The rapid development of electronic devices and network communication provides a foundation for improving the learning and teaching environments through technology. A common phenomenon is that new game ideas grow up with distinctive technology or novel equipments used in learning, and it also brings a challenge to educational games, how we could integrate games in lectures, exercises, day life with recent technologies, such as 3G[11], PSP [10], iphone [3] or youtube, etc, to enrich the teaching or training environment and achieve better learning life.

However, when we live in and start to deliberate our learning and teaching in technology rich learning environments, we are facing some challenges and opportunities that arise from introducing technology into learning and teaching. Most of the theoretical literature on learning and teaching has not yet incorporated a perspective on technology as to how perceive learning and teaching, especially on game-based learning. As such, we would discuss it by present cases of how game technology to perceive the learning in this paper.

This paper's idea was inspired by the work on using XNA [13] successfully in teaching software architecture through students' teamwork on game projects [1]. Similarly we want to see if we could use other development frameworks than XNA for teaching software engineering and computer science. Currently, most attractive choices are the Google android [2] and iphone SDK [3], both are issued in 2007 and free to download from their official websites. After two years, these SDKs become more matured, the newest version of iphone SDK is 3.1, and android is 1.6. Both of them have potential power to enrich the learning life through diverse ways based on the various educational purposes.

This paper is organized as follows: Section 2 describes the theoretical context, previous works and investigates the features of Google android and iphone SDK. Section 3 introduces why and how to extend android platform as a game development tool for teaching purpose. Section 4 explains design issues and results. Section 5 presents how to integrate game development into teaching context based on our extended android platform for software architecture course. Section 6 presents a discussion of the teaching method from different aspects, and Section 7 concludes the paper.

II. RELATED WORKS

A. Theoretical Context

In schools, learners face a challenge, a problem, or a task that has been designed for a particular pedagogical purpose or they face situations that are likely to appear in work and public life. In both cases the purpose of exploiting tools is for learners to respond to such diverse challenges. Our focus is on the construct of the relationship between the educational tasks and the material artefact. This relationship is at the heart of Vygotsky's notion of double stimulation [14], a method for studying cognitive processes and not just results. In a school setting, typically the first stimulus would be the problem, challenge, task, or assignment to which learners are expected to respond. The second stimulus would be the available mediating tools. However, it is important to note that Vygotsky described this relationship in dynamic terms and where the second stimulus is not a discrete end point for this process but, "Rather, we simultaneously offer a second series of stimuli that have a special function. In this way we are able to study the process of accomplishing a task by the aid of specific auxiliary means" (p. 74, emphasis in the original). Note that Vygotsky identifies the second stimulus in the plural—a series. We take this to be most important when approaching the second stimulus in the form of digital tools [15].

Based on this point, we have a case design of learning environment present as below to describe how to construct the double stimulation in software architecture course. Also the design of the first stimulation (tasks) and criteria to

978-0-7695-3993-5/10 $26.00 © 2010 IEEE
DOI 10.1109/DIGITEL.2010.38

choose second stimulation (game development tools) are also given.

B. Previous works – Student projects based on XNA in software architecture course

In NTNU (Norwegian University of Science and Technology), the software architecture course is a post-graduate course offered to computer science students for one semester. Students were grouped in 3-4 persons and spent most time on the implementation phase (6 weeks) to finish game projects. The goal of the project is for the students to apply the methods and theory from the course to design software architecture and to implement a system (game) based on Microsoft XNA framework [4, 8]. The project consists of the following tasks:

1) COTS (Commercial Off-The-Shelf) exercise: Learn the technology to be used through developing a simple game.

2) Design pattern: Learn how to use and apply design pattern by making changes in an existing system.

3) Requirements and architecture: List functional and quality requirements and design the software architecture for a game.

4) Architecture evaluation: Use the ATAM (Architecture Trade off Analysis Method) evaluation method to evaluate the software architecture of project in regards to the quality requirements.

5) Implementation: Do a detailed design and implement the game based on the created architecture and on the changes from the evaluation.

6) Project evaluation: Evaluate the project as a whole using a PMA (Post-Mortem Analysis) method [12].

The second stimulus is chosen based on Malone's "What makes things fun to learn?" [16] and our own teaching experiences. The following are the criteria:

- Easy to learn and allow rapid development;
- Providing an open development environment to attract students' curiosity;
- Supporting programming languages familiar to the students;
- Not in conflict with the educational goals of the course;
- A stable implementation;
- Have sufficient documentation;
- Low costs to use and acquire. From our previous experiences, we found XNA to be a suitable tool in the software architecture course according to overall positive feedback from the students [1].

C. Features of the Google android and iphone SDK

This section compares the differences between android and iphone SDK. Table 1 is the summary features of the Google android and iphone SDK. Also, in order to get the overall understanding of game development platform, we also list the XNA's features for the comparison in the Table.

Both android and iphone SDK have strong support and market share. From technical perspective, they all have the potential value that could be extended as game development tools since most of their applications in the market are about games.

From the evaluation of the two SDKs [17], we decided to go for the android SDK to be used in the software architecture course to learn the syllabus through a project. However, we found that before android could be used, we had to tailor it for our educational purpose.

III. Extension of android for an Educational Purpose

This section gives motivation of improving the android for teaching purpose and direction of how to extend it.

Due to the different educational environments and teaching aims, there could be various extending methods and directions. Under this situation, we will give a case study on how to improve Google android platform under the direction of learning software architecture through game projects.

A. Motivation

From our experiences of using XNA in software architecture course, the survey of students' context in NTNU are nearly 90% have background of java programming [4], and less than 20% have the background of C# or even more less have the Objective C experiences. Most of students face the time consuming of learning new programming languages if they choose the game project in software architecture course. This point is very important, since they have only 6 weeks for the implementation, and they also involve in other courses. As such, Google android could give one more choice for students with java background and enrich the resources for the second stimulus (game development tools) during teaching process. Moreover, using android to develop mobile games also could attract students' attention. So, our goal is as follows:

- Extend the android platform as a game development framework to match the first stimulus (tasks) based on the double stimulation;
- Save students game programming time, let them have more time focus on the course theory.

B. Direction of improving android

From our knowledge, there is no paper that describes extending android or iphone SDK's application as a game development tool for an educational purpose, so not much previous experiences are available. We must start from validating which of the desired characteristics are present in the extended android platform--we called it "Sheep" framework. According to our goal, Sheep both enhances the students' learning experience and helps them achieving their goal faster by saving game development time.

While the android development kit provides a huge programming interface for general application development, the Sheep framework should not only focus on game

TABLE I. FEATURES OF GOOGLE ANDROID, IPHONE SDK AND XNA

Criteria		Google Android	iPhone SDK	XNA
Development Environment		Eclipse recommended by Google	Xcode provided by Apple	Visual Studio and XNA Game Studio provided by Microsoft
Operating Systems for Development		Windows, Mac OS X, Linux	Mac OS X	Windows
Documentation		Official developer website provided	Official developer website provided	Official developer website provided
Emulator		Provided	Provided	Provided
Programming Language		Java	Objective-C	C#
Mobile Devices	**Phone**	Google phone is available in most countries.	iPhone is available in most countries.	No mobile phones type.
	Digital player	No digital player type.	iPod touch is a great developer device, no SIM card required.	ZunePlayer accepts partly XNA games, no SIM card required.
Programming Interface		API contains key high-level abstractions which short development time.	Mainly rely on the low-level standards, like OpenGL ES and OpenAL.	Contains high level abstractions to ease the game programming.
Share of Applications		Publish/sell the applications on Google android Market.	Publish/sell the applications on iTunes apple store.	Publish on the XNA creator club websites.

development, but also on game development for the purpose of learning software architecture to meet the tasks design in the first stimulus in further.

C. Method

The method we used to get inputs for the required features of the Sheep framework was a survey of previous students' exercises in the software architecture course.

This section presents a survey of student projects based on XNA submitted in the software architecture course in spring 2008, and investigates games types the students made, game components they used in their games projects, and architectural patterns and design pattern they used the most. In total, 15 projects are analyzed. The use frequency of game components are list as follows:

- 100% of the groups chose to make a 2D game. The complexity of a game can be significantly reduced by developing a 2D game rather than a 3D game. Many of the architectural challenges which are present in 3D game creation still apply in 2D games.
- 100% of the groups used fonts and text to some extent in their game.
- 93% of the projects utilize collision detection. Many groups use simple rectangle or circle collision detection, some use per-pixel collision maps, and a few use collision detection with advanced geometry.
- 93% of the games contained graphics in multiple layers.
- 87% used graphical user interfaces to some extent.
- 87% also used game state logic in their games. Most games had at least a initial state with a menu, and a running state.
- 87% used sound and two variants are relevant: background music, which enhances the atmosphere, and sound effects, which are triggered when some events occurs in the game.
- 40% of the projects used tiled graphics. Tiled graphics makes sense in many contexts, especially role-playing games, strategy games and platformers.
- 27% of the projects used frame-by-frame animations.
- 20% used persistent data storage in their game, such as saving/loading of progress, or a simple high-score.
- Only 7% used particle effects, which are used to achieve certain visual effects like fire, smoke, snow, and so forth.

Certain elements, which we take for granted in any game have been omitted from above lists, such as input. This is simply to avoid inflating the list with entries of 100% frequency. These omitted parts will not be neglected in the design of the framework.

Also, the patterns that students used in game projects are also useful references for the requirements and design for the Sheep framework. The Model-View-Controller (MVC) [5, 6] is by far the most popular architectural pattern, with 46% of the groups use it. Other favourites include Pipes and Filters (23%), Layered (11%), Strategy (8%) and Client-Server (8%), showing in Figure 1. And the Figure 2 shows that Observer, Abstract factory, State and Singleton pattern are the most popular design pattern. All these patterns are the key concept in practice of software architecture.

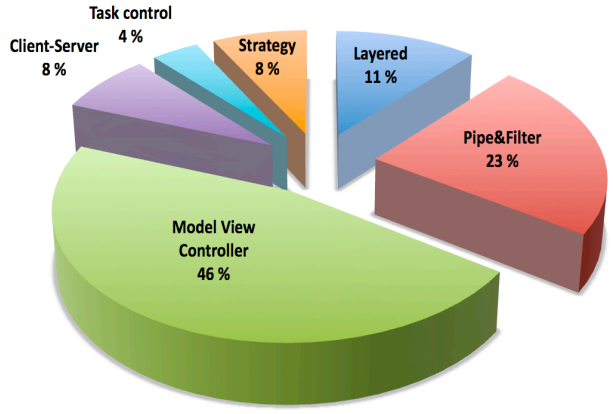

Figure 1. The distribution of chosen architectural patterns for game projects

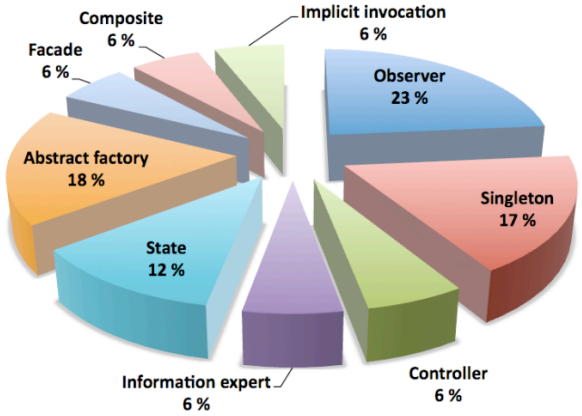

Figure 2. Distribution of usage of design patterns for game groups

D. *Student expectations — requirements for sheep framework*

As main criteria, the components with higher frequency will take a higher priority than the ones with lower frequency, but we must also take the usage frequency of patterns into account.

Under this point, we formed requirements as following for what the students expected to be able to use with the Sheep framework:

1) Graphics. The framework should be able to draw images, primitives and text on screen.

2) Math. The framework should be able to perform collision detection, and transformations on the graphics.

3) Audio. The framework should be able to play sound effects (non-streaming) and music (streaming).

4) Timing. The framework should be able to determine the timing of frames.

5) Storage. The framework should provide means to store persistent data.

6) Networking. It should be possible to transfer data over the network.

7) Resource Management. The framework should provide classes, which makes resource management as easy as possible.

8) Input. The framework should provide means of accessing input information.

Under these requirements, we also investigate what is available in the bare Android API. It is expected that some of the students' requirements will be satisfied fully by the bare Android API, such as networking.

IV. THE "SHEEP" FRAMEWORK

This section introduces the structure of the Sheep framework and the key components' value.

A. *Design goals*

From our previous experiences on XNA, students should not be involved in the programming too much time and cause less time on software architecture study, so the main goal of the Sheep framework is to allow the students to save time in game programming. In a nutshell, the two overall goals for all major components in the Sheep framework are:

- Simplify a common task in game development, so the students can spend more time on structure or course theory and less time on technical issues.
- Use known patterns to interact with client code, as to teach students these patterns, let them to perceive the course theory through using this framework.

According to these goals, we classify the components values in the Sheep framework as:

*1) **Practical value** means that components which simplify common tasks without requiring the use of any particular patterns. The primary goal of these components is to allow faster development, and save time for student to focus on the course content.*

*2) **Academic value** means that components which require the use of certain patterns. The primary goal of these components is to illustrate the usefulness of a certain technique, let students could handle or use this pattern.*

Not all components achieve both goals. Some may be of no direct academic value to the students, and may simply exist as a convenience, some components may be of great academic value, but may not be practical in a certain game genre or specific game design.

B. *Structure of Sheep*

According to our design goals, we could describe Sheep structure in two ways in which the Sheep framework makes the android platform more feasible for game development to learn the software architecture.

From aspect of time saving goal for game programming, the Sheep structure is organized as packages as follows:

- Sheep.audio provides components for loading and playback of sound.
- Sheep.collision contains collision detection and spatial partitioning components.

- Sheep.game assists in structuring the game logic (the model) of the game.
- Sheep.graphics contains components for loading images and fonts.
- Sheep.gui holds the graphical user interface system.
- Sheep.input contains the input devices, and the interfaces needed to subscribe to events.
- Sheep.math contains some math classes which aren't directly related to collision detection.
- Sheep.util is meant to contain miscellaneous components, but for now it only contains a singleton which keeps track of time between frames.

From aspect of encouraging or requiring the use of patterns, the components in the framework are:

- Sprites, which uses the Model-View-Controller.
- Game states, which uses the State pattern.
- Collision detection, which uses the Observer pattern and the Template pattern.
- Spatial partitioning, which uses the Visitor pattern.
- Graphical user interface system, which uses the Observer pattern and Chain of command.
- Other components without expected patterns.

All above pattern concepts are from the software architecture course, and students should master them during the process of using this framework.

C. Packages analysis

Three packages will be examples to explain the components values.

1) Sheep.game package. It provides components, which help organize the game model. Game State pattern is one of the main design concepts in this package. It keeps track of the high-level states of the game. Its main controller object contains methods for loading content, updating its internal state, drawing itself, and responding to input events. The practical value is that having a complete state system in place is beneficial because it allows relevant input events to be presented more clearly and quickly to the students. And its academic value is that State pattern is a well-known pattern, which allows an object partially changes its class at run-time. Specific game behaviour should be implemented via subclasses of the State class. Each state represents a different view of the game, when students use it in programming, they probably would understand it clearly.

2) Sheep.collision package. It provides functionality for detecting interactions between objects in the game world, and generates collision events, which may be subscribed by observers.

The practical value is that collision detection was used in most of the student projects, and getting the details of such collision systems to work right can be incredibly time consuming. When providing the students with a full collision detection system, they could use it directly to work efficiently.

The academic value is that two patterns will be visible from the perspective of the student, the Template pattern and Observer pattern. The Template pattern is in the Shape class, where the overall algorithm is fixed, but some sub-parts are modifiable by derived classes. The Observer pattern will be used for custom collision responses. As an example, perhaps a player should lose health when it is hit by another object, A "Lose Health Listener" could then be attached to listen for collision events occurring to the player object.

3) Sheep.gui package. It provides a graphic user interface system and can be used to create complex windowed menus or simple buttons. The practical value is that a few buttons were present to allow the user to start and quit the game, and also provide functional kit for the extensibility. The academic value is that Observer pattern is used to listen for events. The Chain of command pattern is used to control how input events are passed through the widget class hierarchy.

V. Integrate Sheep in the Software Architecture Course

This section presents how patterns work in Sheep framework, and how students interact with the framework based on the double stimulation. We choose two design cases to explain it.

A. First Task: Sheep and patterns

When the students start to use Sheep, they are inevitable to get into the code of Sheep. So the first task is to let the students become familiar with Sheep by list some patterns that they could find (or construct) in Sheep framework.

Here we give three exercise examples to explain the patterns that the Sheep framework used. Also, other patterns, such as *Template, Visitor, Singleton,* etc also can be found in Sheep, but not list here.

1) Exercise 1: Model-View-Controller. Student should find a Model View Controller design in the Sheep.

In the Sheep framework, the Sprite class acts as the superclass for all models. When a method on the Sprite itself is called to draw a sprite, this call is either redirected to the associated SpriteView, or ignored in case a SpriteView is not set. If the client code wishes to change the way that a Sprite is represented, for instance an animation instead of a static image, the client can simply create a new SpriteView subclass. The logic in the Sprite remains the same.

Figure 3 is one example to the exercise: a PlayerController listen for events on the keyboard. This controller can for example cause the Player to shoot bullets when a certain key is pressed.

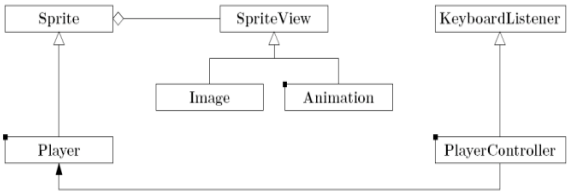

Figure 3. MVC pattern in Sheep framework

2) Exercise 2: State. Students should find an example of State pattern used in Sheep.

State pattern causes an object to appear as if it has changed its class. It can be used in the Sheep framework by adding subclasses of State to the instance of the Game class. Figure 4 is an example of State pattern used in Sheep.

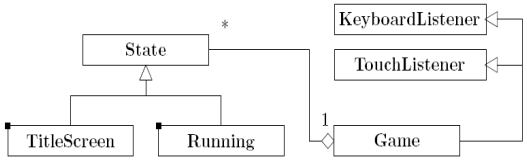

Figure 4. State pattern in Sheep framework

3) Exercise 3: Observer. Students should find an example of Observer pattern used in Sheep.

The observer pattern could be found under some conditions, such as, a) When listening to input devices, either via the keyboard or touch singletons or via a State; b) When listening to events from the collision detection system. Events are issued when Sprite objects collide; c) When listening to events from the graphical user interface system. Events are issued for various reasons, for instance when a button is pressed.

Figure 5 is an example in Sheep that PlayerController responds to events from the keyboard, touch or collision detection system.

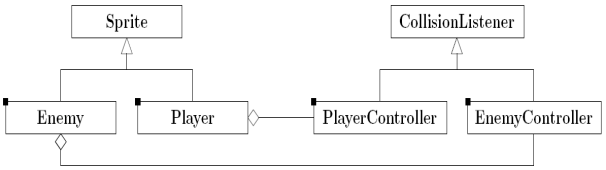

Figure 5. Observer pattern example in Sheep framework

B. Second Task: Patterns design and game implementation

We propose three exercises to implement small games by apply the patterns design through Sheep.

1) Exercise 1: Moving Sprite. The requirement is to make a simple game where a Sprite is controlled by the user. This could be done by subscribing to events issued by the Touch singleton.

The purpose of this task is to show how the observer pattern can be used to respond to input events in a way which is familiar to gamers.

In a solution, the students could create a subclass of Sprite, which listens to events directly; or simply instantiate Sprite and use the main state class as the controller; or they could create a separate controller class (Figure 6). There are also other possibilities of the solutions.

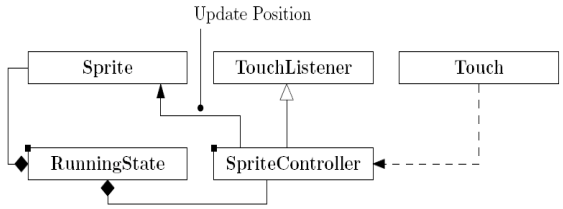

Figure 6. Possible solution to the exercise 1

2) Exercise 2: Game States. This task is to make a game with at least three States: a title screen, a main running state, and an in-game menu. The game can be as simple as Pong or Tic-Tac-Toe, as long as these states are present.

This exercise shows how an object may change its perceived class using the state pattern. A solution would consist of three (or more) subclasses of State, and some mechanism for transition between the states.

3) Exercise 3: Racing Game. This task is to make a racing game architecture with following characteristics:

a) It should be possible to change between two sets of graphics in the middle of the game; one with graphical sprites, and the other using primitive shapes only, for instance rectangles for cars and lines for the racetrack.

b) There should be more than one car; all cars except the player's car are controlled by the computer.

c) It should be possible to click on other cars and take control of them. In so doing, the computer should take control of your old car.

The racing game does not need good artificial intelligence or realistic car simulation, but these characteristics should be evident in the game.

This exercise shows how to decouple the visual representation, input handling, from the Model of Racecar. The solution here is to use the Model-View-Controller (Figure 7).

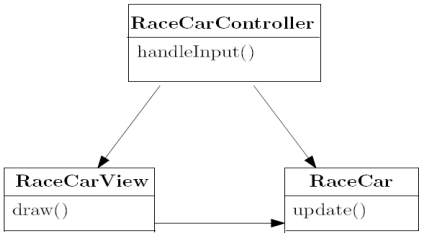

Figure 7. Solution to the exercise 3

VI. DISCUSSION

The software architecture course at NTNU is taught in an untraditional way, in that the students in addition to designing and evaluating their software architecture have to implement the architecture in a game project as well. The

main advantage with this approach is to let the students feel the "pain" of making their design decisions, as complicated and look-nice-on-paper architectures can be very difficult and time-consuming to implement.

From Sheep's application view, the Sheep framework is a very useful tool to help the students with the transmission between the design and implementation by offering high-level components based on architecture and design patterns. The most difficult task for the students when implementing a software architecture is to decompose a high-level architecture into classes and design the interaction between these classes. The Sheep framework will make this transition easier, as the built-in architecture and design patterns is the first step in decomposing a high-level architecture. Due to this type of design in Sheep, the students could find appropriated available components to start with. The Sheep framework also enables the students to focus more on the architecture and less on issues related to the programming.

From a view of edutainment, the android platform was chosen and extended based on the Malone's "What makes things fun to learn?" [16] and our own experiences [1]. We believe that it is useful to teach the students about design and architecture patterns in a practical way through the suitable game exercises proposed. This game domain is likely to motivate the students to put an extra effort when learning the patterns through various exercises. The students are motivated by learning how to program on the android platform as well as programming interesting games.

Game development for devices like android phones and iPhone/iPod Touch can also be motivating for the students from a business point of view, as development of games on these platforms can be low-cost and low risk. The result of a game project in a software architecture course might end up as a continuing hobby game project uploaded to android Market or AppStore to sell or as a student start-up game company.

From the pedagogical point of view, the design and application of the Sheep framework is also an example how to bridge pedagogy, technology and game ideas to enhance teaching in a reasonable way. During this double stimulation process, students seek to align their continuous interpretation of a task and tools. Also, the second stimulus is provided with series of tools that can be classified in horizontal-vertical orientations. From horizontal aspect, XNA and android are provided in parallel for the same task; from vertical aspect, XNA is used with other related tools, such as XNA club website for the help and sharing games, Zuneplayer for testing game demos, and PowerPoint for the final presentation. Corresponding, android is used with android Market, android phone and PowerPoint too.

We believe our analysis points to the necessity for further pedagogical and technological co-design to better facilitate awareness of game-based learning, better conduct the direction of how to design the knowledge construction process of involving individuals and small groups to stimulate their initiative and creativity in game related activities. This indicates that future evaluation of using the Sheep framework for teaching the course is also beneficial, it reveals not only the efficiency of using the framework along with how much the students actually learn from game projects, but also the social relationships of learner-learner and learner-teacher. We also need to further investigate the relationship between games, tasks, and tools in technology-rich and collectively oriented knowledge construction in order to better understand and support the game-based learning.

VII. CONCLUSIONS

From our previous experiences in the software architecture course, we would like to offer a new choice for the double stimulation in this course. And we found that Google android is a suitable tool for the educational use. In this way, we extended the android platform mainly based on the requirements from previous students' projects. Further, we have developed a game development platform called Sheep based on android. We also described how to integrate the game technology and software architecture learning in Sheep framework to explain one perspective of how technology perceives learning.

From the discussion, we found that there are various orientations to apply or extend a tool according to the previous experiences, context of students, local environment and technology and teaching aims. Based on these conditions, the game ideas, technology and learning should be integrated in a reasonable way to let the second stimulus match the first stimulus. This paper is an example from this idea that applied theoretical and empirical context to support the design process of game-based learning.

ACKNOWLEDGMENT

We would like to thank Anders Hartvoll Ruud for implementing Sheep framework and for his inputs to this paper.

REFERENCES

[1] Wang, Alf Inge; Wu, Bian. "An Application of a Game Development Framework in Higher Education." International Journal of Computer Games Technology 2009 ;Volume 2009.(2)

[2] Google. "Android developers" http://developer.android.com/index.html, Retrieved September 22, 2009.

[3] Apple. "iPhone Dev Center", http://developer.apple.com/iphone/, Retrieved September 22, 2009.

[4] Bian Wu, Alf Inge Wang, Jan Erik Strøm and Trond Blomholm Kvamme: "XQUEST used in software architecture Education", IEEE Consumer Electronics Society's Games Innovation Conference , August 25-28, 2009, London, UK.

[5] Trygve M. H. Reenskaug, MVC, XEROX PARC, 1978-79. Accessed March 16th, 2009

[6] Koen Witters, Game Architecture: Model-View-Controller, 2008. http://dewitters.koonsolo. com/gamemvc.html, Accessed March 16th, 2009.

[7] Gamma et.al, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Publishing Co, 1994.

[8] Bian Wu, Alf Inge Wang, Jan-Erik Strøm, Trond Blomholm Kvamme,"An Evaluation of Using a Game Development Framework in Higher Education," 22nd Conference on Software Engineering Education and Training, pp.41-44, 2009.

[9] T. Blomholm Kvamme and J.-E. Strøm, "Evaluation and Extension of an XNA Game Library used in Software Architecture Projects", Master thesis at NTNU, June 2008.

[10] PSP, "Sony PlayStation Portable" http://www.us.playstation.com/psp, Retrieved September 24, 2009

[11] 3G "Definition" http://en.wikipedia.org/wiki/3G, Retrieved September 24, 2009

[12] A. I. Wang, T. Stålhane. Using Post Mortem Analysis to Evaluate Software Architecture Student Projects, Conference on Software Engineering and Training 2005, 8 p.

[13] XNA, "Microsoft XNA" http://www.xna.com, Retrieved October 5, 2009

[14] Vygotsky, L. S.. Mind in society: The development of higher psychological processes. Cambridge, MA: Harvard University Press, 1978

[15] Lund, A., & Rasmussen, I. (2008). The right tool for the wrong task? Match and mismatch between first and second stimulus in double stimulation. International Journal of Computer-Supported Collaborative Learning, 3(4), 387–412.

[16] T. W. Malone, "What makes things fun to learn? Heuristics for designing instructional computer games", In SIGSMALL '80: Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems, pages 162–169, New York, NY, USA, 1980. ACM Press.

[17] Anders Hartvoll Ruud, "Designing a Game Development Framework for Teaching Software Architecture on the Android Platform", Master thesis at NTNU, June 2009.