A Design Goal and Design Pattern Based Approach for Development of Game Engines for Mobile Platforms

Ali Gökalp Peker
Department of Computer Engineering
Middle East Technical University
Ankara, Turkey
e1595503@ceng.metu.edu.tr

Tolga Can
Department of Computer Engineering
Middle East Technical University
Ankara, Turkey
tcan@ceng.metu.edu.tr

Abstract—**Nowadays, increasing number of game engines are being developed for mobile platforms. As game engines developed for mobile platforms are small scale compared to console and desktop alternatives, game engine development in mobile platforms uses basic software development methods. However, as mobile platforms enhance and development of more complex games become possible, basic development methods will not scale well for advanced game development requirements. This paper examines design of a mobile game engine that provides software maturity by using design goals and design patterns. Our results show that developing a game engine by using a design goal and a design pattern based approach provides a more regular and a clear path for building each aspect of the engine with a high-level perspective of design.**

Keywords-Mobile Application Development, Game Engine, Game Development, Design Patterns

I. Introduction

Beside its end-user features, the software design and architecture of a game is highly affected by the game engine used. Design and architecture include key elements for satisfying non-functional requirements such as performance, power efficiency and portability. As these requirements are very critical for desktop and console games, they also gain more importance in mobile games due to limited software and hardware capabilities of mobile platforms.

Supporting non-functional requirements in every level of game development is an important task of a game engine. A game engine also provides facilities for functional requirements. All these requirements suggest a mobile game engine should be designed to address each requirement with a formal, tested and proven solution.

In this paper, designing a game engine that satisfies functional and non-functional requirements by using design patterns is discussed. Also benefits and pitfalls of the proposed approach are demonstrated with a case study.

A. Mobile Game Engines

In this section, alternative open source and commercial game engines are discussed with respect to their software designs. How game engine features are designed, integrated and used is analyzed and discussed in the scope of the paper. Many mobile game engines are available in the industry for evaluation and it is not possible to discuss all these engines. So evaluated game engines are selected by considering their conceptual design.

Unreal engine and Unity 3D are the strongest commercial engines in the market and these engines provide applications of best practices. Both engines have one thing in common, their mobile versions are derived and scaled from console and desktop versions. They are evaluated in the context of mobile game engine approaches and platform specific features.

SIO2 is another alternative commercial engine that is targeted only for mobile platforms. This feature allows us to evaluate an engine that is purely developed for mobile platforms.

As an open-source candidate, cocos2D game engine is selected to evaluate a non-commercial, mature and platform specific game engine.

1) Unreal Engine and Development Kit For Mobile Platforms

In the mobile version of Unreal engine, developer interface is tried to be kept as much as the same as the console version of the engine. This is a practical example of a usability goal and achievement of this goal provides opportunities for unified development environment. Unreal engine limits some features in the mobile version due to the hardware limitations. Some of these limitations are adjusted automatically, making the game engine more adaptable. However, some adjustments had to be done by the developer, a step that leaves the adaptability attribute incomplete.

Developers use the scripting language provided by Unreal engine to build game modules. This scripting language abstracts developer from inner engine mechanics, but also decreases efficiency. Also the game engine provides frame rate limiter as a trade off between efficiency and game play smoothness.

2) Unity 3D

Unity3D is an exemplary engine that can be adapted easily for many platforms. Also the targeted platform can be changed on the fly during development.

Like Unreal engine, Unity3D abstracts underlying game engine architecture by providing a scripting language for the developer to manipulate game objects and binding game components. But unlike Unreal Engine, Unity compiles these scripts to native platform code to avoid performance inefficiency caused by using a scripting language.

3) SIO2

SIO2 game engine design includes cohesive, loosely coupled components. These components are distributed across the engine as models and methods that operate on these models. Thus SIO2 provides very well defined data structures and functions for building a game. This structural design approach provides advantages like lightweight execution and easy adaptation to other platforms. But it also limits dynamic extensibility features. This limitation is handled by providing flexibility via scripting support.

4) cocos2D

cocos2D has an inheritance based design. This design is flawless when developing a game using the engine. But inheritance based design does not allow base functionality to be extended or changed without modification of high-level classes.

cocos2D makes a trade-off between efficiency and extensibility. It highly depends on specified platform and there is no abstraction for platform specific modules. So the engine can use the whole potential of the underlying mobile platform. But design of the engine fails in portability and requires to develop the engine from scratch when porting it to another platform.

B. Evaluation of Game Engines

Discussed game engines have several aspects which make an engine more advanced in terms of efficiency, usability, portability and extensibility. But also they have weak aspects.

One of these weak aspects is that using inheritance instead of composition for deriving new components. This results in highly coupled systems. Inheritance have these disadvantages :

- Changing a parent class may affect all derived classes
- Customization gets difficult if the engine expands
- Strong dependency between classes makes them harder to reuse.

Lack of abstraction is another weak point for engines. This makes high-level implementation closely coupled with lower level implementations and interfaces. This coupling may be problematic in case of portability.

Trade-off between efficiency and other goals is very important in mobile platforms. Limited hardware capabilities should steer engine developers to choose more efficient solution that fails to accomplish other goals. SIO2 game engine gives up from extensibility for better adaptability and cocos2D gives up from portability for better efficiency.

So, visioning the entire picture and designing the engine systemically by keeping all design goals in mind, plays an important role in building an adaptable, usable and efficient game engine.

II. Mobile Game Engine Design Methodology

A. High-level Design of the Game Engine

A game engine requires a high-level design that reflects requirements of included features. High-level design affects how a game engine is formed. By describing the high-level design of a game engine,

- major extension points of the engine
- functional components(like graphics, sounds, and controls)
- integration of all the components

are defined. Defining the high-level design is done in following phases:

1) Defining features of the engine
2) Defining a set of design goals and design strategies
3) Extracting design goals from features
4) Defining design patterns for accomplishing design goals
5) Integrating all defined patterns and forming the complete design of game engine

1) Defining Features of the Game Engine

In the first phase, the tasks of the engine in the defined domain should be determined. A hierarchical feature set should be formed for functional and non-functional requirements.

Features of a game engine can be platform specific and these features should be defined by platform standards. But on the other hand a game engine that supports multi-platform is desirable. Features of a game engine are grouped as controllers, graphics, sound and physics support. With multi-platform support, included feature set of these groups changes. So separation of platform-independent parts of these features, allows us to design common features with no dependency to a specific platform.

Hierarchical design of features can be modeled with a feature tree. A typical engine feature set can be seen in Figure 1.

2) Defining a Set of Design Goals and Strategies

A design goal is a set of rules and principles that corresponds to a particular implementation goal. Design goals are major decisions about high level design characteristics and each design goal includes design strategies. Design goals change according to architecture and features.

For a mobile game, design goals are determined mostly by hardware and software specifications of mobile platforms. We can describe usability, efficiency(in the manner of power, memory, speed and size efficiency), portability and adaptability as major design goals for a mobile game engine.

a) Usability

Usability influences game engine in the way that how a game developer uses it as an infrastructure. Usability can be defined as readability and intuitiveness of the game engine and the game implemented on it. Usability design goal may be achieved by accomplishing other goals. A well designed package hierarchy or a clear design helps to achieve the usability goal. Achieving usability goals reduce the learning curve of the engine and increases productivity.

b) Efficiency

A game engine should use the underlying mobile platform efficient, in terms of power, memory and performance. Trade-off between efficiency and other design goals is a compelling part of the game engine design. Efficiency strategies can make the design more complex. This affects usability and adaptability goal. Also using underlying platform in an efficient way may create portability issues for the game engine.

c) Portability

Portability is an important design goal when multi-platform support becomes one of the goals of the game engine. Portability requires analysis of targeted and candidate platforms. Portability may bring structural and behavioral changes to software, making it mostly inefficient. However, portability provides productivity advantages for the game engine that is one of the most desired factors in the industry. Portability strategies highly depend on architectures of targeted platforms. As indicated in [5], strategies for supporting portability in software can be grouped in three categories:

- strategies that maintain identical execution-time interfaces by porting system components that form the interface
- strategies that maintain identical or nearly identical interfaces for different system components by adhering to appropriate standards
- strategies that assist in the adaptation of programs to a target environment

These strategies correspond to design patterns those are integrated to the design of the game engine.

d) Adaptability

Adaptability goal ensures game engine to run on different configurations. In mobile platforms, each device may have different screen resolutions, different input capabilities, and also different computing capabilities. Adaptability can be ensured in different levels of game engine design. It can be ensured in design level by using abstractions or in run-time level by using a scripting language.

3) Extracting Design Goals From Features

Design goals from this feature set can be defined by interpreting features in each leaf and node of the tree. Some goals are defined directly from the relationships of the features and some goals are defined by indirect interpretation of features. In this sample feature set described in Figure 1.

Common mobile platform capabilities and characteristics require efficiency goals, such as power, memory and performance efficiency due to platform characteristics.

“Platform support” node includes more than one leaf for platform. This incorporates portability design goals to engine. Input controller requires control of both touch screen and keyboard.

Also graphical support requires tile engine and layered viewing support. These factors make adaptability and extensibility design goal “a-must”.

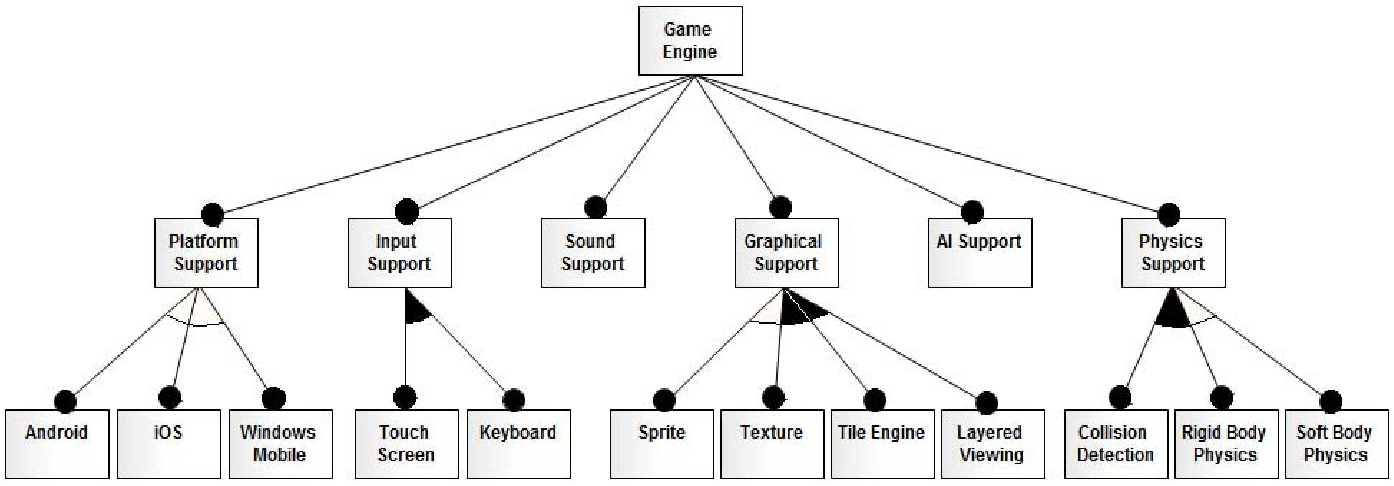

Figure 1.Feature Diagram For A Typical Game Engine

4) Defining Design Patterns

In this phase of the game engine design, design patterns are used to achieve design goals or to help achieving them. It is a known limitation that using design patterns reduces efficiency in terms of computational processing. Computational processing, as one of the most power consuming operation in mobile platforms, affects power efficiency directly. But as indicated in [1], most of the design patterns used in software development do not impose a significant penalty in energy consumption. Also as indicated in [2] and [3], it is possible to implement power conscious versions of design patterns to make them power efficient. This researches make it possible to use design patterns on mobile platforms with no sacrifice in power efficiency.

Used design patterns are based on industry standard resources in GoF book [5] and other tested and proven game development patterns [6]. Selected design patterns for achieving design goals defined in previous phase, are given as an example :

- Efficiency
  - Power Efficiency, Energy conscious versions of patterns, Frame Limiting
  - Memory Efficiency: Flyweight, Abstract Factory
  - Speed Efficiency: Proxy, Prototype
- Portability: Bridge, Adapter, Model-View-Presenter
- Adaptability: Decorator, Strategy

5) Integrating all defined patterns and forming the complete design of game engine

In this phase, all defined patterns are integrated together with the engine to form the main design of the engine. Integration of the game engine is done in a top-down manner. First, all design patterns are incorporated as the high level design of the engine. Implementations of functional requirements are relatively low-level parts of the game engine and these parts are placed beneath the high level design of the engine.

B. A Reference Design and Implementation for Proposed Methodology

A reference game engine that includes all the features defined in Section II.A.I and all the design goals stated in Section II.A.II is designed by using proposed methodology.

Additional features, design goals and related design patterns other than defined in previous sections are used in reference design. Used features, design goals, design patterns and mapping between them are given below.

- Game State Management Feature → Extensibility Goal → State Pattern
- Game Loop Feature → Extensibility Goal → Mini-kernel Pattern
- Multiple Physic Modeling Support Feature → Extensibility Goal → Strategy Pattern
- Level Manager Feature → Extensibility Goal → Abstract Factory Pattern
- Game Unit Re-usability Feature → Extensibility Goal → Model-View-Presenter Pattern
- Customized Game Units Feature → Adaptability Goal → Model-View-Presenter Pattern
- Game Element Creation Feature → Efficiency(Performance) Goal → Prototype Design Pattern, Proxy Pattern
- Game Element Creation Feature → Efficiency(Memory) Goal → Lightweight Pattern
- Extended Game Play Time Feature → Efficiency(Power) Goal → Frame limiter Pattern

Engine design is formed with all patterns are placed and integrated. The overview of the designed game engine is described in Figure 2.

A game is developed for applying and evaluating designed game engine. Game and game engine is developed on 2D for simplifying development process and focusing more on evaluation of accomplishment and evaluation of design goals. Also in this context, game features are kept as minimum as possible.

Developed game's genre is defense genre that one player tries to defeat another player by attacking and destroying game elements that other player tries to defend. This game genre allowed us to use modules of game engine easily and effortlessly.

III. Evaluation of the Design and Methodology

The design of the game engine is validated by testing achievement of each design goal. Each achievement is validated with different methods due to different characteristics of the each design goal. Some achievements are validated theoretically and some are validated experimentally. During these validations, engine functionality is abstracted from game engine design for focusing on validation of only design goals accomplishment.

A. Usability

There is no definite way of proving that an engine is usable due to the fact that it is a relative concept. Usability of an engine simply depends on intuitiveness, simplicity, and readability in addition to the provided feature set. Overall design in Figure 2 shows that cohesion and loosely coupled design provided understandable components.

These characteristics of design can provide ability to add new features even with visual design tools. Implemented prototype has proved that with a few classes (model, view and presenter), a new unit can be added to the game. However, usage of design patterns brings disadvantages in terms of productivity. For a new requirement, a design pattern based systems require much more classes to add. This is a sacrifice done for cohesion and extensibility.

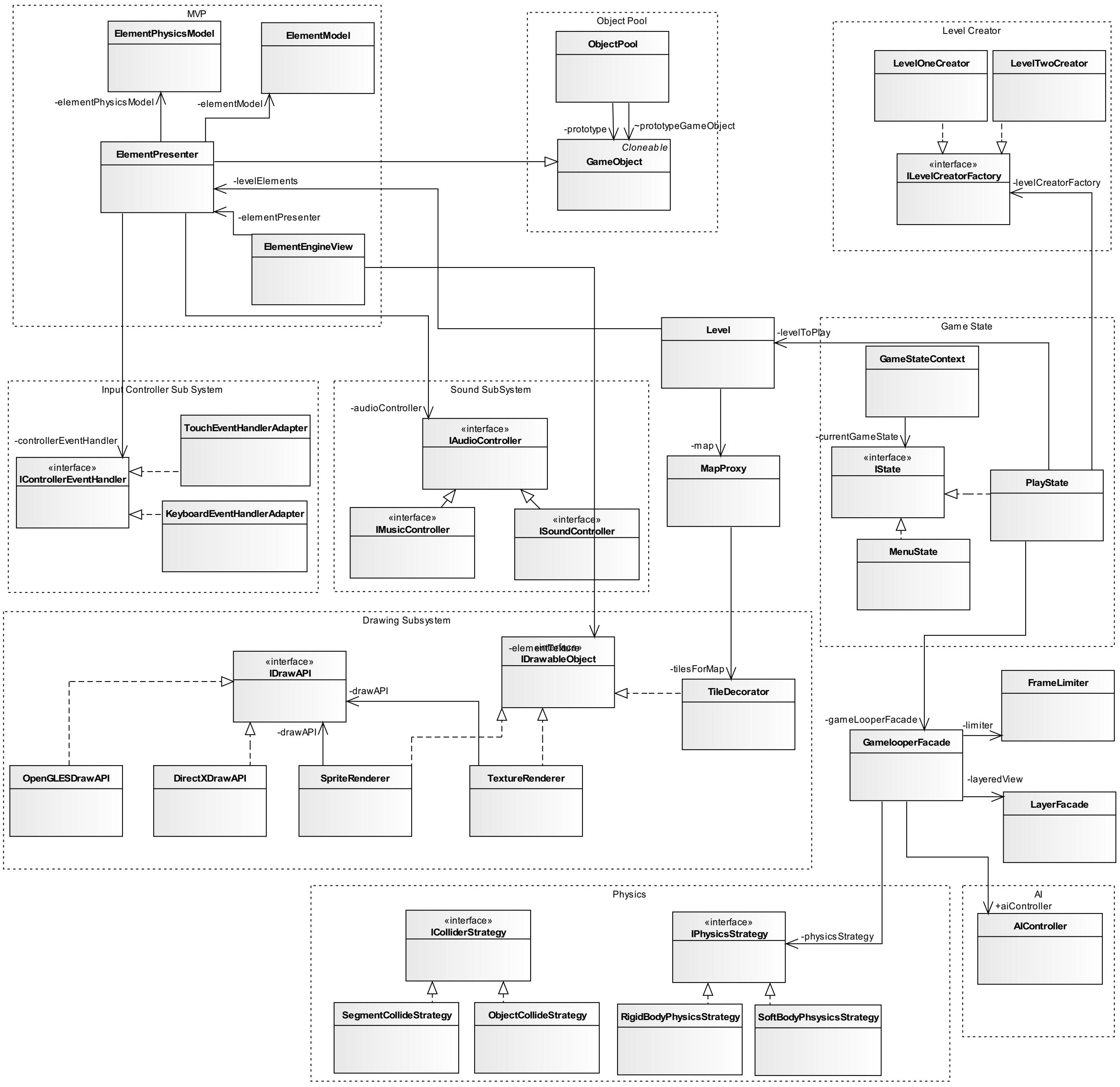

Figure 2.Overview of Game Engine designed by using proposed methodology

B. *Efficiency*

For evaluation of power efficiency, power consumption of engine is measured by enabling and disabling frame limiter. PowerTutor [8] is used for measuring power consumption of reference implementation. PowerTutor is an application for Android platform that displays the power consumption of CPU, display, GPS, and applications. Although, it is described that using PowerTutor in mobile devices other than specified hardware models may result in rough estimates, tour goal in this paper is to measure the difference between consumption with and without frame limiter. Power conscious versions of design patterns are not included in test due to it is partly proved in related papers [2][3].

For measurement of memory efficiency, memory consumptions are calculated for game elements created by using flyweight design pattern and raw allocation. Attacker element model from reference implementation is chosen as base model for measurement of the memory efficiency. This model includes :

- Maximum and current speed
- Width, height and position of element in two dimension
- Maximum and current health
- Enemy detection radius
- Gun damage amount, magazine size, current ammo count

In this element model, there are twelve primitives. Data related with all these properties are kept as integer variables in the memory. If it is assumed that an integer occupies 4 bytes in the memory, then the total size of an element is approximately 48 bytes. In traditional design, this data structure is created every time a new element creation is required. So this data type will consume 48 byte during object's lifetime.

These properties can be classified according to their variability in element's lifetime. A property can be a dynamic type of property (current speed, position of element in two dimension, current health, magazine size, current ammo count) that changes during the lifetime of element or it can be a static type of property(maximum speed, width and height, maximum health, detection radius, gun damage amount) that is constant during the lifetime of element. In Flyweight pattern, the static properties are held on a separate registry associated with each type of element. So when a new instance is requested, an element with newly allocated dynamic properties and mapped static properties, is created. And such a schema reduces memory consumption. In this example, size of an element in memory reduces to 24 bytes.

Performance efficiency benchmark is done by executing reference implementation with and without performance centric design patterns. Measuring performance efficiency is a difficult task to accomplish, due to the fact that processes are affected from other tasks and processes. Impacts of external factors are aimed to be minimized by isolating process from platform and calculating resulting values by taking average of multiple benchmarks. Isolation is realized by using simulators that emulate hardware logic. Efficiency results are reported on Table I.

C. *Portability*

For evaluation of portability, reference implementation has been compared to an existing game engine in terms of portability. cocos2D game engine is a good candidate for comparison due to its lack of abstraction and multi-platform support. In validation context, a new imaginary platform support is requested from each game engine.

In this comparison, some distinguishing features for a typical platform are defined and these features are implemented on both of the engines. Required changes are discussed for measuring portability level of each engine. New imaginary platform is assumed to have a new type of controller and a different graphical sub system. This new platform supports a new type of view, a dual screen that includes a main screen and another helper screen.

cocos2D game engine supports touch inputs by delegates. A delegate has to be implemented for controlling touch input. This means all classes that include controller operation may have direct association with the underlying controller system. So adding a new controller requires change of the delegate interface and all implemented delegates. Implementing an abstraction that encapsulates the system beyond the delegate may avoid spread of direct usage of controller classes. For the designed game engine, implementing a new event handler adapter for the new controller and binding it with controller's callbacks will handle the new controller support. This operation has no side effects beyond input controller's boundary, which means that the rest of the game engine and the game remains unchanged.

For the cocos2D game engine, changing the graphical subsystem requires replacing classes that use OpenGL ES bindings. These bindings are all spread across the engine. A new graphical sub system support requires changes to entire engine and game. For the designed game engine, new graphical sub system is added to the system by adding a new drawing API class. The rest of the system, even texture interface remains unaffected due to the usage of the bridge design pattern.

For the cocos2D game engine there is no presenter-view separation; so, support of dual screen is in the responsibility of game developer. If game developer can not foresee such a platform feature, porting all the game to this new platform may require change of classes that are spread all over the code. For the designed game engine, changing implementation of the view class is enough for supporting dual screen.

TABLE I. BENCHMARK RESULTS FOR EFFICIENCY DESIGN GOAL

Element Count	**Avg. power consumption with frame limiter(MW)**	**Avg. power consumption without frame limiter(MW)**	**Memory consumption with flyweight(bytes)**	**Memory consumption with raw object creation(bytes)**	**Execution-time with design patterns included(ms)**	**Execution-time with design patterns excluded(ms)**
2	744	794	72	96	32	467
4	780	840	120	192	56	824
6	789	837	168	384	82	92

D. Adaptability

Evaluation of adaptability goal of the reference engine is done by discussing adaptability points of the proposed game engine. Adaptability is provided in a variety of points in the game engine.

In tile manager, adaptability is provided by the decorator pattern. With the decorator pattern, tile manager is able to use any drawable object like texture or sprite as a tile and also the tile manager itself acts like a drawable object. And also tile manager helps developed game to adapt to new screen resolutions. In input controller, by using adapter design pattern , adaptability to different type of controls is ensured.

IV. Results

Game engine designed by proposed methodology accomplishes its design goals by using design patterns.

Tests shows that performance increases drastically when performance centric design patterns are included. This increase strictly depends on benefits of patterns like prototype and pooling. Context of the evaluation aims to test effects of design patterns locally, that makes these performance improvements specific to the creational and structural domain. Incorporating behavioral parts of the game gives different results depending on the behavioral domain implementation and it heavily depends on the target game features and technologies. Technologies like culling and spatial indexing can be analyzed separately for integrating them into a mobile game engine [9] [10].

Memory consumption is reduced up to %56 of total consumption when memory centric patterns are used . And this ratio increases when created element count increases. But game memory consumption includes many other types of allocations like textures and physics models, and it should be also evaluated by addressing all these allocations.

With the activation of the frame limiter, power consumption decreases up to %6 of total power consumption compared to the case where the limiter is disabled. It can be interpreted from frame limiting concept and result that power consumption gain increases with increasing amount of game elements up to a certain amount.

Other design goals , usability, adaptability and portability, have been satisfied by designed game engine.

V. Conclusions

Using design goals and design patterns, provided a clear and regular way for building each concept of the engine. Using design patterns allowed us to use industry strength, tested and proven methods in the mobile development context. As a higher level approach, design pattern based development has injected design goals in the high level design of the engine.

Mobile platforms are growing stronger and more powerful. This advancement seems similar to the advancement of personal computers. Like desktop platforms, mobile platforms may require higher level development methods to build large scale products. Design goal and design pattern based approach is one way to do this, especially on games and game engines.

References

[1] K. Chantarasathaporn, C. Srisa-an, “Energy Consumption Analysis of Design Patterns” In Proceedings of World Academy of Science, Engineering and Technology 2005, pp.86-90.

[2] K. Chantarasathaporn, C. Srisa-an, “Energy Conscious Builder Design Pattern with C# and Intermediate Language” In Proceedings of World Academy of Science, Engineering and Technology 2006, pp.134-142.

[3] K. Chantarasathaporn, C. Srisa-an, “Energy conscious factory method design pattern for mobile devices with C# and intermediate language” In proceedings of Mobility '06 Proceedings of the 3rd international conference on Mobile technology, applications & systems 2006(29)

[4] E.Gamma, R. Helm, R. Johnson and J. Vlissides ,Design patterns:elements of reusable object-oriented software., Massachusetts:Addison-Wesley 1995

[5] J.D. Mooney, "Strategies for supporting application portability", in IEEE Computer Volume: 23 Issue:11 pp.59-70, Nov.1990

[6] E. Welch, "Developing iPhone Games for Longer Battery Life", Technical Article. 2010

[7] R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, Pattern-Oriented Software Architecture Volume 1: A System of Patterns. Wiley 1995

[8] M. Gordon, L. Zhang and B. Tiwan, PowerTutor

[9] I. Antochi, B. Juurlink, S. Vassiliadis, P. Liuha, "3D Graphics Benchmarks for Low-Power Architectures". 2002

[10] H. Kim, "Frameworks of Indexing Mechanism for Mobile Devices" in BLISS '07 Proceedings of the 2007 ECSIS Symposium on Bio-inspired, Learning, and Intelligent Systems for Security pp.47-50.

Ali Gökalp Peker received his B.Sc. degree in Computer Engineering from Hacettepe University, Ankara, Turkey in 2006. He received his M.Sc. degree in Software Engineering from the Middle East Technical University, Ankara, Turkey in 2010. He is currently a Senior Software Engineer at MilSOFT Software Technologies.

His technical interests are in mobile application development, game engines, and software architecture.

Tolga Can received his B.Sc. degree in Computer Engineering from Middle East Technical University, Ankara, Turkey in 1998. He received his M.Sc. and Ph.D. degrees in Computer Science from the University of California at Santa Barbara in 2003 and 2004, respectively. He is currently an associate professor of Computer Engineering at the Middle East Technical University.

His main research interests are in scientific visualization, computational systems biology, and bio informatics.