Software Architecture

TDT4240

Architectural Description Document

Group 17 : "XNA videogame focus on modifiability"

Marius Greve Hagen

David Rozas Domingo

Maria Fernandez-Rodriguez

Jarle Lindseth

1 Index

2 Introduction

This document deals with the Architectural Description of the project “SuperPang videogame”. It starts mentioning the architectural drivers of the project; secondly it describes the different stakeholders and concerns followed. Next, each view is related with stakeholders interests.

The quality tactics we have chosen to achieve the previously mentioned quality goals is expressed and also an explanation of the impact of the choice to use XNA in our core game development. A brief description of the logical view is provided, and finally a rationale for our architectural decisions and a change log of how our architectural description has changed during the period of this development.

3 Architectural Drivers

In order to identify our architectural drivers, first of all, we should state our main business goals:

Our main goal is to have a modifiable game in which we are able to add more features as time permits us to extend the game. The possibilities of features are endless, but include two-player-games, potential online gaming, extra power-ups, more types of enemies, more gun-types and possibility to change the user preferences and control choices.

What is paramount here is that these changes can be added over time without changing the nucleus of the main logic.

A most important architectural driver here is the time constraint put on us in this project. We have a very short time span to complete the product and several major architectural decisions will be assumed automatically by the choices to use the XNA framework and XQuest.

As you can see from Requirements Documentation, relations between scenarios and quality attributes are as follows:

- Modifiability (M1,M2,M3,M4,M5,M6,M7).
- Testability (T1,T2).
- Usability (U1, M1,M2,M3,M5,M6,M7).

4 Stakeholders and Concerns

The stakeholders that are taking part in the system are the following ones:

Stakeholders	**Concerns**
End User (U)	Wants to be entertained. Needs a challenge and interesting experience. Pacing of game overall playability.
Architects and designers	Being able to produce the best possible product while learning the limitations and possibilities of the XNA framework dealing with its performance management and implied architecture. Dividing the functionality into different modules and retaining modifiability while extending our system.
Implementers	That the architecture and design is clear and precise enough that implementing it in the code is efficient and sustainable.
Testers	That the product works according to specification and tests can be carried out while checking for well-defined results on individual modules and the end product.
ATAM partners	That the specifications are well defined and easy to understand so the can provide a way for them to analyze and give constructive feedback while also learning and being introduced to possible choices for their own project.
Course staff	That the product is interesting and fun to use while easily being analyzed and having a transparent implementation based on the project specification. That the product incorporates applicable theories and patterns and that the developers learn and improve as designers and implementers.

5 Selection of Architectural Viewpoint

The following table describes if the view is useful for the stakeholder. A detailed description of the views is shown in the next sections:

Stakeholders/Views	Logical	Process	Development
End User (U)	Yes	No	No
Architects and designers	Yes	Yes	Yes
Implementers	Yes	Yes	Yes
Testers	Yes	Yes	Yes
ATAM partners	Yes	Yes	Yes
Course staff	Yes	Yes	Yes

Physical view is not reflected in this document because of the nature of our application which is running in a personal computer, where physical resources are bounded and, although they can vary, difference in performance can be negligible.

6 Quality Tactics

6.1 Performance:

For the game to perform at a steady frame-per-second rate or have a predictable performance it’s important to manage the events and methods in an efficient manner.

The most important task is to use algorithms that compute as little data as possible to maintain this overall efficiency. We might also introduce concurrency to compute data simultaneously, but it’s important to keep the threads to a minimum so they don’t take up to much of the resources (given that the concurrency is thread-based).

6.2 Testability:

In order to make it easy to test the game architecture for faults and failure it’s important to generate a lot of output to control the estimated computation. This could be done by implementing thorough logging and well-defined interfaces. The interfaces should be separated from the implementation.

It should also be possible to test each component separately to easily determine where problems originate from and then make it easier to deal with.

6.3 Modifiability:

In order to make the controller architecture and implementation easy to change for adding or modifying functionality we should first and foremost anticipate expected changes in the architecture. For it to be easier when the changes arrive the system should be module-based and the module should be generalized and given a semantic coherence. We should also hide some information in means of private fields in the code.We could of course limit the number of features, but since we want our game to be scalable we should use interfaces and inherit from them.

7 Architectural Patterns

The architectural patterns are strongly influenced by the use of XNA since it implements the game loop and filter. As our game includes basically no AI our architectural pattern is limited to determining how we structure the flow of the game; the input from the user and updating the game elements and deciding interactions between objects. For this the loop and filter pattern of XNA works very well as it provides the critical foundation on which to structure our periodic object updates.

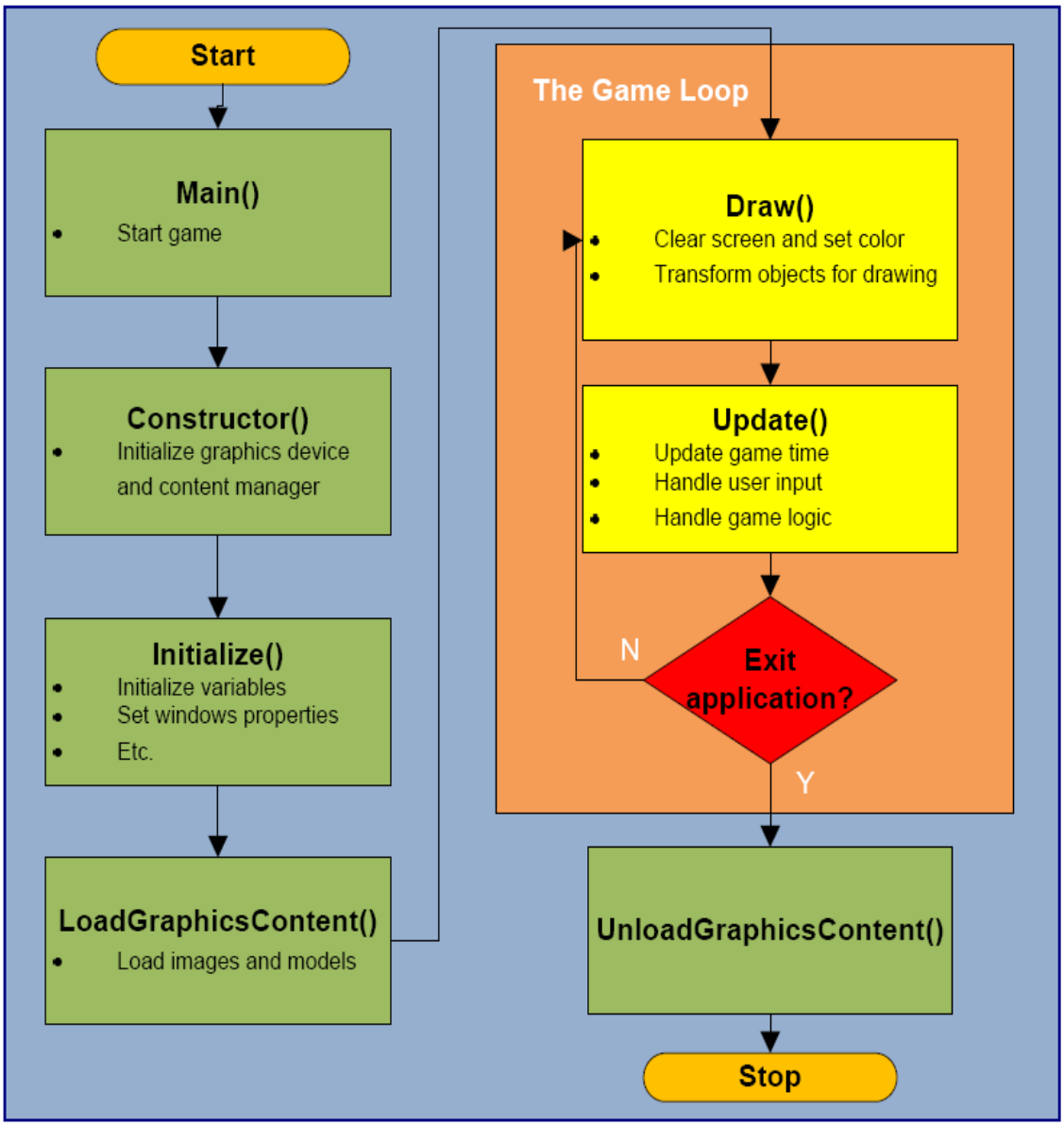


Figure 1: XNA application Model

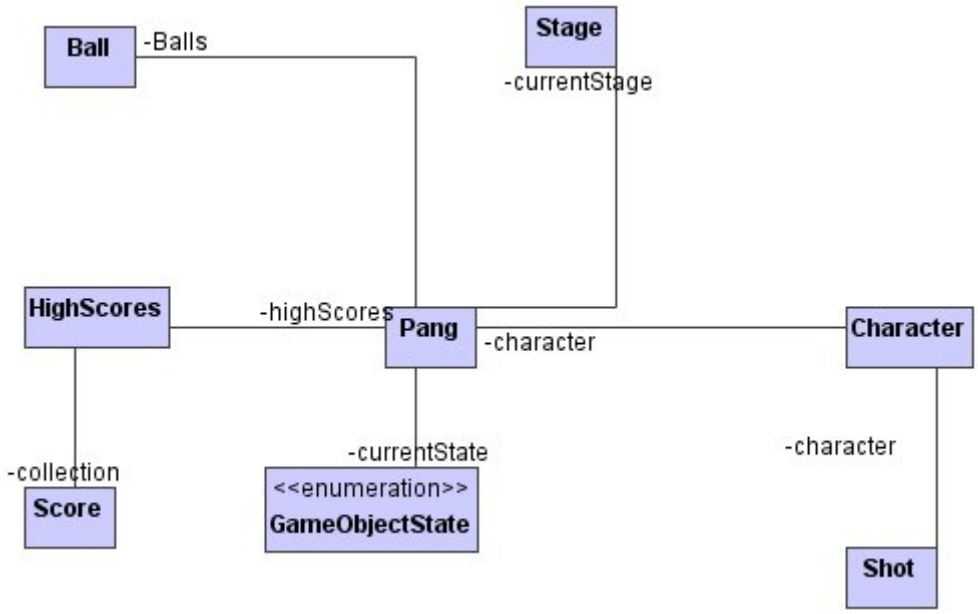
We decided to make use of XQuest and this helped us with controlling graphic elements and user input, but the main architectural pattern was still strongly set by XNA.

Layers, an architectural design pattern that structures applications so they can be decomposed into groups of subtasks, such that each group of subtasks is at a particular level of abstraction, can be useful to divide and dispose responsibilities in the development team. The way layers were organised is reflected in the development view.

8 Views

8.1 Logical view.

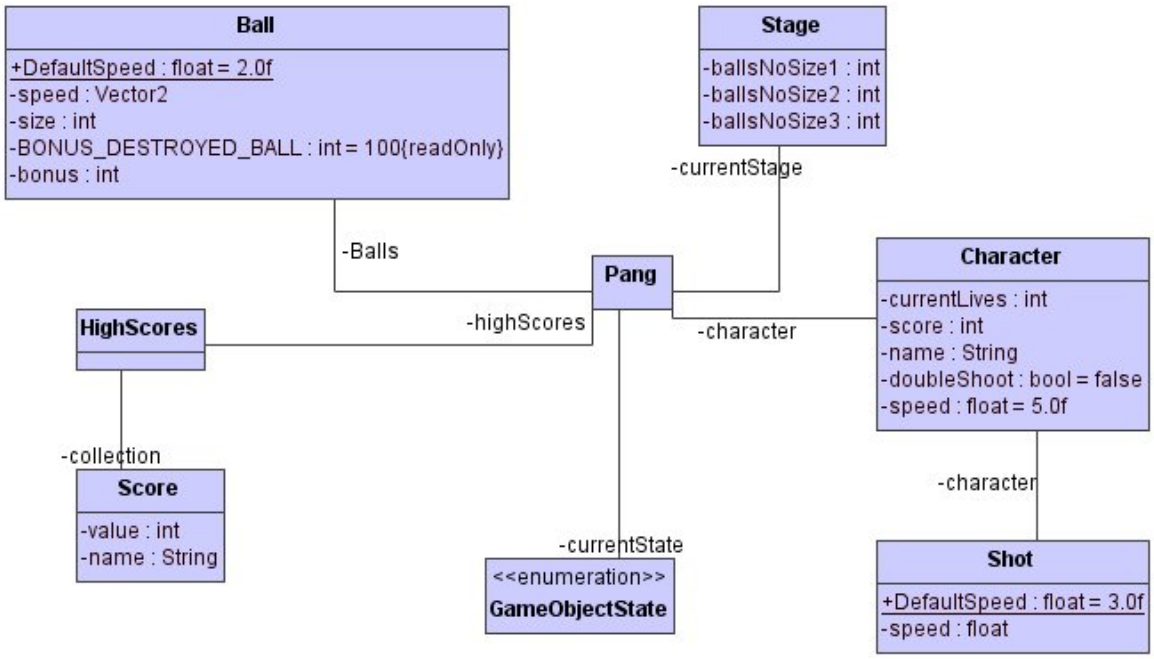
The logical view primarily supports the functional requirements - the services the system should provide to its end users-. If we discompose our domain model, that is to say, the logic of Pang game , we get a set of classes associated as follows:



Pang class represents the Game class. Pang contains crucial information to the game such as which character is playing, in which stage and the game state.

Stage should be responsible to establish the level of difficulty in terms of number of balls and size of balls. A character should be able to shoot balls thanks to a Shot that can be single or double.

To keep a record of higher Scores reached by users, a Score class and its composition HighScores were also added.



8.2 The Process view.

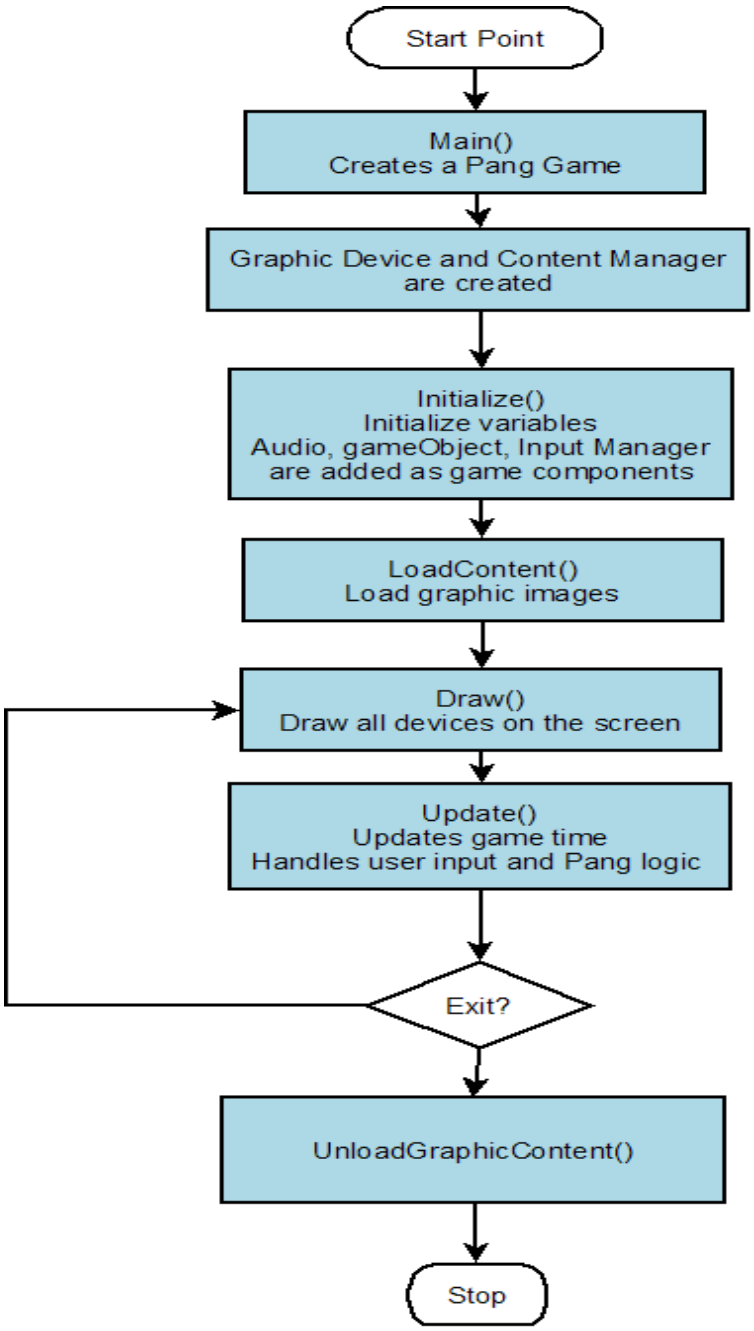
The process view describes mainly the design's concurrency and synchronization issues. Our videogame is not really concerned about these aspects for the moment. There is no point to distribute processing load or improve system availability, since our videogame is running in a local system.

Anyway, if we consider a process as a group of tasks that form an executable unit and that can be grouped into two groups:

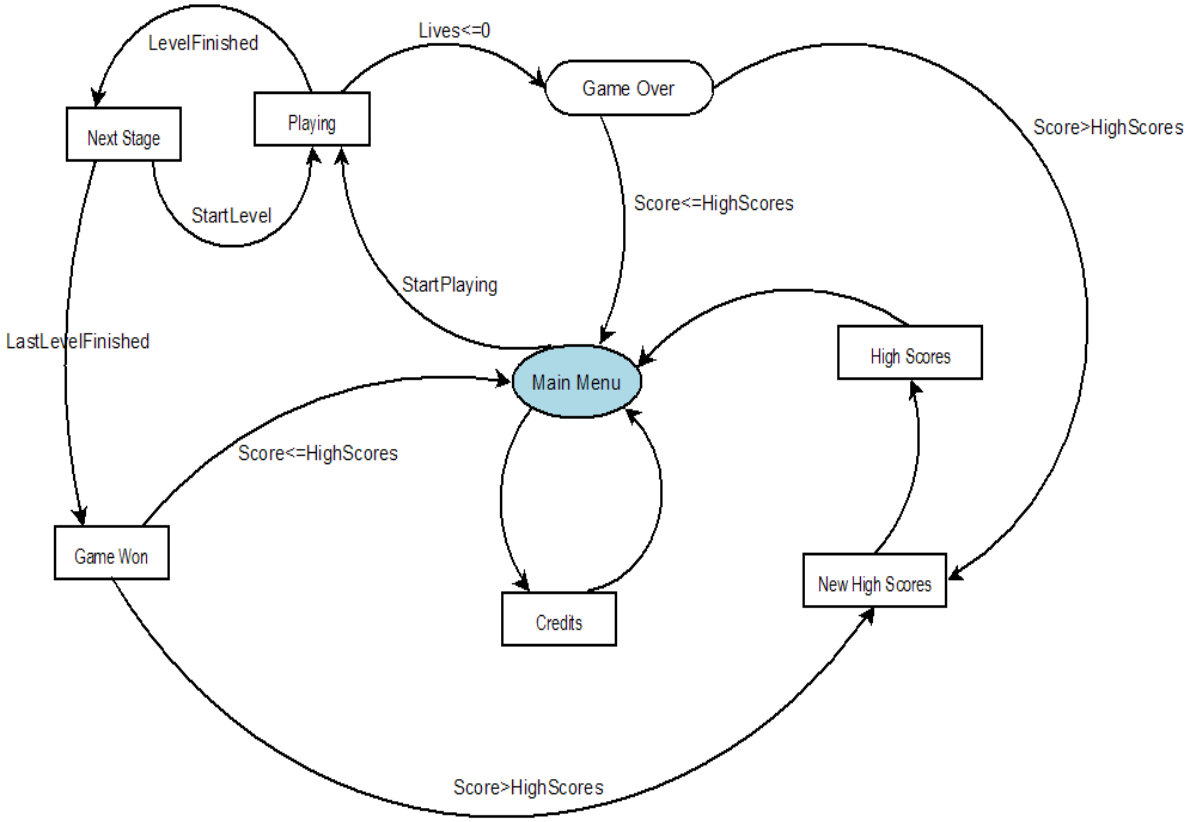
- Major tasks – those which are the architectural elements that can be uniquely addressed (designated from another task). They communicate through a set of well-defined intertask-communication mechanisms- .

- Minor tasks – those which are additional tasks introduced locally for implementation reasons such as cyclical activities, buffering, and time-outs.

Given this classification, we can consider XNA as a big process divided in major tasks as it is shown in the next figure.



As XNA is a constraint in our project, these processes and their interaction are also a constraint to develop the game logic. According to the previous classification, Pang logic resides in our minor tasks, where the logic process was divided in different states. Transitions between states are depending on conditions fired by the user and continuously checked in the game loop in Update method. States and transitions between them are shown in the next diagram.



According to the logic of a videogame, a user can play OR interact with Menu options, but not both at the same time. Thus, we also have two different processes we have to take care of: On one hand, the “Menu mode” and, on the other hand, the “Game mode”.

Transitions between both modes is dispatched by two kind of events: User active events (keyboard input) or game events (final stage was accomplished). In both cases, a game state fires the transition, which is handled by the Pang class.

8.3 The Development View

The development view focuses on the organization of the actual software modules in the software-development environment. This is supposed to make easier to divide software in *chunks* or subsystems that can be developed by one or more different developers.

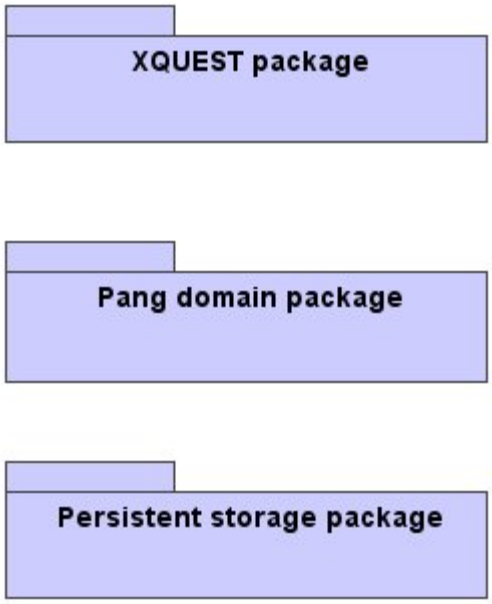
In our concrete case, three main layers were established:

1. Technology dependent layer: XQUEST framework which encapsulates main functionalities from XNA in order to create new 2D Videogames.
2. Domain layer, containing all classes which take care of the main logic of the application. In this layer, game states transitions are followed. This layer is directly

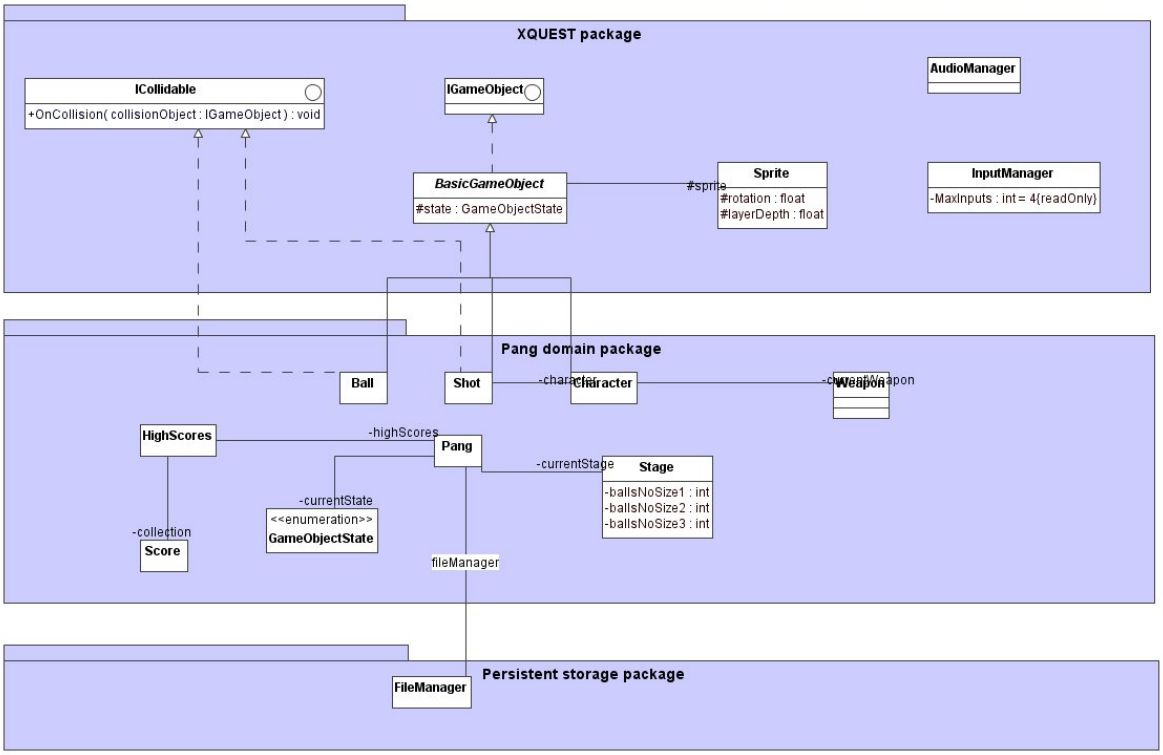
related with the upper layer since most of the classes are inheriting from classes in XQUEST, so division is not sharp.

3. Persistent storage layer to provide store settings and scores. This layer contains the logic in charge of provide data to the upper layer. Settings and score are stored in XML documents.

This division is illustrated in next figure.



Main classes taking part in this division and its relationships are shown in the next figure:



In Pang domain package, the central and main class is *Pang*. This class is the nucleus of the logic since it communicates with *FileManager* class to get and save data from persistent storage, set correct parameters to the *currentStage*, it is in charge of updating properly game states, since when a collision takes place, elements involved in a collision notify *Pang* class.

8.4 The Physical View

It is not relevant for this application as we mentioned in previous section.

9 Architectural Rationale

For our project our choice of architectural design has worked quite well. Our general lack of AI ensured that each objects complexity of update would be fairly low and the main challenge of managing the flow of the game would be in making sure all the objects properly interacted with each other. With XNA's game loop and the implementation of the interface ICollidable XNA made sure that every item properly ran its update function on every cycle and that all objects were checked for collisions with each other and the corresponding behaviours were triggered.

Outside the main game loop we had some choices as well; we needed to read a few configuration files at game start up and we originally wanted to use a Singleton for the FileManager. In the end we realized that framework we used to implement the reading and writing of xml-files together with the fact that we ever only read or wrote files from our main class inside our main game object at startup or at game restart we found out we really did not need a Singleton after all. The design goal of using a Singleton was already fulfilled by us maintaining the one file reader object created at startup and used for all read/write operations.

10 Changelog

15.04 – We eventually started to realize how the XNA model actually worked. Allthough we didn't exactly change anything in our specification, it was not until now we really understood what the architectural design that we implicitly had chosen actually meant.

17.04 - We realized that we would not be able to implement all the features into the project that we had originally wanted. A lot of our envisioned features would simply take too long to implement. In a way our goal shifted from wanting to show how we could easily add all sorts of features to wanting to show that our project had a design that would easily allow those features to be added had we had the time.

18.04 – Our idea of wanting the file reader and any similar classes to be implemented was scrapped because in our limited scope there was no need for having access to it from any other class than the game-class that would originally create it at load time.

18.04 – We refined the concerns of the stake holders and also added our ATAM partners and the class teachers as stake holdes.

18.04 – We added as a architectural driver our limited time scope. This had proven to be a most severe driver as it defined most of our choices in the game development. The focus on the game being easily modifiable for future installments was also made more important as an architectural driver rather than all the extra features we wanted to add as we started out this project

18.04 – Our logical view was changed substantially to illustrate the new and more limited class and logical structure of our game.

11 References

"Software Architecture in Practice, Second Edition", Len Bass, Paul Clements, Rick Kazman (Tapir).

XNA Intro Slides from TDT4240 course.

A 4+1 Views overview from http://www.ics.uci.edu/~andre/ics223w2006/kruchten3.pdf