Master thesis:
Game Mechanics Engine

Lars V. Magnusson
Østfold University College

January 3, 2011

Abstract

Game logic and game rules exists in all computer games, but they are created differently for all game engines. This game engine dependency exists because of how the internal object model is implemented in the engine, as a place where game logic data is intermingled with the data needed by the low-level subsystems. This thesis propose a game object model design, based on existing theory, that removes this dependency and establish a general game logic framework. The thesis then expands on this logic framework and existing engine design theory to create a concept of a genre-independent engine that can provide an alternative to the normal game engine. This new genre-independent alternative is referred to as a game mechanics engine.

Contents

List of Figures

Listings

Chapter 1

Introduction

Computer games have evolved greatly since their first appearances in the 1960s. The first computer games where simple software applications written by scientists on their spare time to run on hardware created for scientific purposes, while games today are complex pieces of software created by industry professionals to run on more or less specialized hardware. Despite this evolution in technology and complexity, all the computer games, created from the industry's humble beginning to the multi-billion industry it is today, have the common trait that they are all games.

1.1 Game logic and game engines

Games differ from other software applications in that they provide gameplay, a quality which has been described in number of ways, but has never been properly defined. Though, It is a common understanding that gameplay in some way or other are a property, at least in part, of a set of absolute game rules and logic. These game rules exist in all computer games, and they could as such serve as unifying element and provide a common ground for all computer games.

1.1.1 Game engine dependent game logic

In a modern computer game, logic are typically represented by a combination of game data stored in the internal game object model and scripted logic written in a language supported by the integrated scripting environment. The problem with this is that both the game object model and the scripting environment can be dependent on low-level subsystems in the game engine, which consequently ties the game logic to the game engine.

1.1.2 Inflexible game engines

Game engines are becoming more powerful and flexible with each new release, but the concept of game engines as a provider of a complete framework for game development has some issues.

Game engines are designed and created with a specific game or genre in mind, which have lead to genre-specific optimizations in the low-level components such as graphics. As a consequence there is no guarantee the provided solution is suitable for the next project. To make things worse game developers have been known to offer the majority of attention to low-level which such as graphics and physics, which in many cases has led to less than optimal design and flexibility in the overall engine architecture. This has the consequence that it can be difficult to replace one part of the engine with another.

This inflexibility has the consequence that in order to create a general game engine the integrated low-level components must be able fulfill all current and future needs of any imaginable computer game, making this approach seemingly insurmountable.

1.2 General game logic

In order to reach a situation where game logic can provide a common ground for all game, it seems like is required to come up with an alternative to the normal game engine.

1.2.1 Genre-independent game mechanics engine

It has been proposed that it could be possible to find a genre-independent game engine by identifying features and components in a game engine that can be used across the entire domain of computer games. It has also been suggested that a game engine taxonomy is needed in order to properly find these commonalities in game engines across different genres, but this has practical issues. This thesis will instead work under the assumption that established theory on game engine architecture and design will provide adequately detailed descriptions of the different components in a modern game engine to make the taxonomy unnecessary.

To ensure that the resulting engine technology still can be referred to as a *game* engine, it is essential to be able to include a foundation for general game logic, but this cannot be done without finding a solution to the already mentioned game engine dependency.

In order to separate the concept of a game engine made up of genre-independent engine features from the empirical understanding of normal a game engine, the genre-independent alternative will be referred to as a *game mechanics engine* in this thesis.

1.2.2 Game structure and patterns

With the introduction of our genre-independent game mechanics engine comes the opportunity to perform an analysis on the structures and patterns in the game object model with the goal of identifying reusable patterns. Any such patterns could be used in any number of games, since the game mechanics engine never changes. This could potentially simplify the job of the game designer tremendously, by allowing them reuse solutions from previously created games.

1.3 Research objectives

The main focus of this thesis can be summarized by the following three research objectives:

1. Separate the game logic completely from the low-level game components and their data.
2. Combine the established general game logic with other genre-independent game engine features.
3. See what game structures and patterns emerge, if any, when developing games with the stable game development environment.

1.3.1 Contribution

In this thesis we will attempt to create a genre-independent game mechanics engine with a general game logic foundation. This game mechanics engine could provide an alternative to the standard game engine approach, which although extremely powerful, has some issues in terms of flexibility, both in terms of the integrated low-level components and the inability to easily exchange one component for another.

We will also investigate the possibility of identifying emerging patterns in the game object model.

Chapter 2

Background

The research and literature that have provided the background to this thesis can be divided into the following three groups; *game studies*, *game engine design* and *other*. The chapter is concluded with a short summary of the theoretical foundation behind this thesis and presents the issues for which this thesis will attempt to provide solutions.

2.1 Game studies

Game studies, also know as *ludology*, is an area of research relating to the study of games, and as such it can provide some insight into what makes games games. There are several articles and books available that attempt to define what constitutes both games and gameplay, but not that many provide information applicable to the implementation of general game logic in our game mechanics engine. The few sources presented here, particularly the first one, have served as a theoretical and philosophical foundation for the approach taken in this thesis.

2.1.1 Half-real: Video games between real rules and fictional worlds

Jesper Juul has written a book [Juul, 2005] that points to the fact that computer games can be seen as two things at the same time; both *real* and *fictional*. In a game the player interacts with real rules to achieve a real outcome, but the game's events take place in a fictional world. This core philosophy of a duality in computer games have been important to the conception of this thesis, and that is why it is the first first source of background information presented.

Rules

Rules represent what the player can and cannot do in a game, which contribute to both the limitations and the possibilities of that particular game. The rules give meaning to the allowed actions by rewarding or penalizing the player, and this is what gives the player meaningful choices and provide structure to the game. The game rules are generally definite, unambiguous and easy to understand, but they still have to offer challenges that cannot be easily overcome, turning the recreational act of playing a computer game into a learning process.

Jesper Juul argues that the rules of the game can be considered as the state transition functions in a state-machine representing the current game state, but the game rules can also be thought of as algorithmic since they must be both *finite*, *definitive* and *effective*. Finite because it must be able to finish after a finite number of steps. Definite because the rules must be unambiguous and contain all contextual information needed to make all transitions. Effective because the amount of detail in the specification of the must provide sufficient information for a computer to determine the outcome.

He also argues that there are many different ways of structuring the rules in a game, but that the two most important ones are; *emergence* and *progression*. Games of progression will chain the challenges together to make a linear gameplay structure, while in games of emergence the challenges are set up indirectly when the rules interact, meaning that the game is not bound to *any* sequence, and the challenges stem from the combination of the rules in the game world. These are the two extremes, but most games end up somewhere in between.

The book also features a discussion on game theory, with focus on the differences between game rules and strategies. Jesper Juul states that while the rules are absolute, the strategy used by most players are typically *incomplete*, where *complete strategy* would be a strategy that specify what action the player should perform for every state in the game. A *dominant strategy* is a strategy that always better than all other possible strategies within the constraints of the game rules, and they are typically avoid to ensure both fair and interesting play.

Fiction

Most games have some form of fictional world that is projected using graphics, sound, text, advertising, the game manual and the game rules. He states that the game rules can operate without the fictional world, but the fictional world is dependent on the game rules. The fictional world is an attractive property of a game typically for the opposite reasons that game rules are: It is subjective, ambiguous and subject to discussion.

Jesper Juul says it is tempting to classify games as being either *abstract* or *representational*, but the implied separation between them does not exist. Instead he argues that this can better be viewed as a scale, where games can be either completely abstract or completely representational, but they can also be somewhere in between.

The concept of time in games discussed with the focus on the duality of time referring to both *real time* and *fictional time*. Real time is the actual real time spent playing, or the *play time*, while fictional time is the time elapsed in the fictional world.

2.1.2 Computer game criticism: A method for computer game analysis

Lars Konzack [Konzack, 2002] has written a paper on a framework for analysing games that breaks down games into seven different layers: *hardware*, *program code*, *functionality*, *gameplay*, *meaning*, *referentiality* and *socio-culture*.

The two layers *program code* and *functionality* together provide the technical foundation on top of which the games are built, and in that respect they roughly correspond to what this thesis refers to as the game engine, but the descriptions of these two layers are superficial and do not offer any details that could benefit this project. The meaning layer refers to the semantic meaning of a computer game, and he argues that this is best studied through the use of *semiotics* which is the study of signs and symbols. The referentiality layer relates to the game's virtual setting and genre. Lastly, the socio-culture layer contain the social and cultural aspects of a game, not just the relationship between game and player, but also between all the other players of the game.

Gameplay

The gameplay layer is described in the article as the first layer that provides the structure and mechanisms that make an application into a game, and he presents the following game factors as building blocks in gameplay: *positions*, *resources*, *space and time*, *goal* (sub- goals), *obstacles*, *knowledge*, *rewards* or *penalties*. The description of these game factors are not adequate to form a set of gameplay patterns for a game mechanics engine, but they can be used as a theoretical foundation for the different types of game objects needed to create games. This makes sense since the game factors presented in the article were not designed with game creation in mind, but game analysis.

2.2 Game engine design

While ludology may provide helpful information about the conceptual structure of games and gameplay, game engine design research and literature provide a foundation for how to actually build the technology to able to create a game. Although some good purely academic papers exist on this subject, the best and most popular resources are found in books written by experienced industry professionals.

2.2.1 The case for research in game engine architecture

Anderson, Engel, Comninos and McLoughlin have written a paper talking about the status and future of research into game engine architecture and design [Anderson et al., 2008]. They start by pointing out some relevant background information such as the lack of literature and research aimed specifically at the design and architecture of game engines, and go on to discuss a series of topics or questions that need answering in order to have a foundation on which to build future research into game engine design.

Games and game engines

The first topic discussed is the definition of a game engine and boundary between game and engine. They argue that both of these might be derived through analysis of several games and identifying common and unique components. In addition they discuss whether or not the custom tools that are used to create the games should be considered as part of the engine or not, but without reaching a conclusion.

They go on to discuss the game genres and how they currently affect the design and implementation of the game engines. They say that today the design and definition of a game engine is tightly coupled with the games using the engine, and that available game engines tend to be aimed at one particular type of game.

They argue that the lack of a taxonomy of game engine commonalities can be the reason that there exists no single game engine capable of being used across all the different game genres. Such an engine design could provide a number of benefits to the game development community. Development time and cost can be significantly reduced, and programmers and designers can more easily be transfered between different projects. They are convinced that general game engine components can be found by analysing game engines across different genres, but they also consider the possibility that this could exclude essential parts, making it less applicable and relevant when making games.

Game engine design

They also discuss design dependencies and how low-level issues affect top-level design and how the rapidly evolving technology behind computer games affect the architecture and design of the game engines available. They argue that this has not been considered in previous attempts at creating a general game engine architecture such the architecture presented by Folmer (See section 2.2.4), but they also mention examples of the opposite [Tulip et al., 2006]. They pose the question if any design methodology could be employed to minimize this effect.

The last topic discussed relates to existing and alternate design methods. Empirical observations suggest that many game engines grow and evolve over time, which suggests that overall engine design change along with the changes to engine components. They suggest that a *top-down* design and *bottom-up* implementation might be the way to go.

2.2.2 Game engine architecture

Game Engine Architecture is a book [Gregory, 2009] that covers several aspects of game engine architecture including a whole section dedicated to gameplay, a topic scarcely covered in previous books on the subject. In addition to the gameplay section the book features a section dedicated to describing low-level game engine systems such as memory management, game loop variants and human interface devices. Jason Gregory has also included a section called: *Graphics and Motion*, which deals with different approaches and principles relating to rendering, animation, collision detection and physics.

This book provides the best source of information we have found on all the different components that constitutes a modern game engines outside more documented areas such as graphics and physics, and for this reason it has been crucial to the process selecting candidate components for our game mechanics engine.

Games and game engines

The book starts with an introduction to games and game engines which includes a presentation of the history of game engines and a discussion on the conceptual line that separates the game from the engine, and how that line vary between different games and game engines.

Flexibility is a key factor in game engines since they must allow for the development of different types of games, but Jason Gregory argues that it is even more important for the architecture to support some form of *data-driven* development. This term is used to classify development of computer games using data files to specify parts of the game instead of source code, and Gregory argues that this quality is what distinguish a game engine from game software. The concept of data-driven development has evolved out of the need of game designers to extend and modify the game without including a programmer in the process.

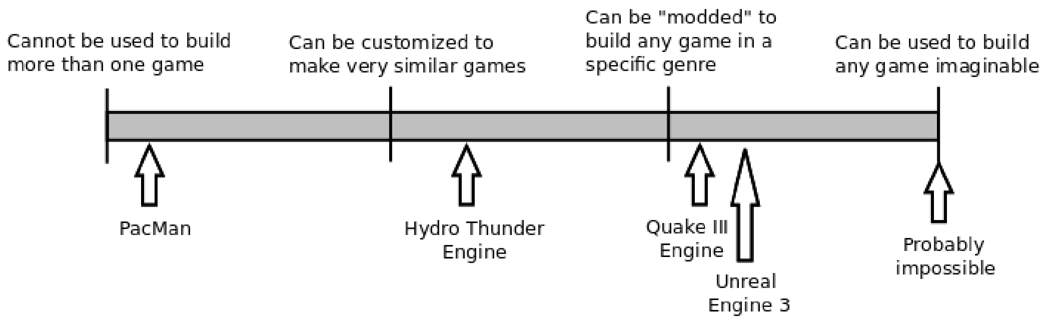

Cannot be used to build more than one game
Can be customized to make very similar games
Can be "modded" to build any game in a specific genre
Can be used to build any game imaginable
PacMan
Hydro Thunder Engine
Quake III Engine
Unreal Engine 3
Probably impossible

Figure 2.1: Game engine reusability gamut presented by Jason Gregory

The game engine introduction features a gamut which range existing engines depending on their reusability and flexibility, where complete reusability is thought to be impossible. There does not exist any general-purpose game engines that can run any game imaginable, and Gregory argues that it might not even be possible. Game engines are to some extent tied to one type of genre, and they typically need to be optimized for the intended hardware platform. in different genres.

Runtime engine architecture Jason Gregory presents a runtime engine architecture based around layers where the upper layers depends on the lower levels. The model incorporate low-level foundation such as hardware, operating system and third party software development kits (SDK) that can be considered outside the scope of thesis. The first game engine specific layer in his model is called *resources* and is responsible for loading the game assets such as textures and three-dimensional models. On top of this layer follows several other game engine components such as a low-level renderer, collision detection, physics, audio and the gameplay foundation, but the relationship between these components are not as strictly layered as before. Some of these components clearly work independently, while other components such as the gameplay foundation clearly depends on all other mentioned components. The top layer in the architecture is the game-specific layer which is dependent on all the layers below.

Tools and asset pipeline Games contain a varying degree of digital content such as three-dimensional models, images and sounds. The set of tools used to create all of this digital content vary between third-party tools and custom tools suchs as the game world builder and content importers and exporters. The game world builder can either be provided as a standalone application, or it might be integrated into the engine itself, providing in-game world creation.

The book also discuss some of the benefits of data-driven game engines and how this feature has evolved since the early days of computer games. The game world editor plays an important part in providing data in data-driven game engines.

Gameplay

The information provided in the book on gameplay starts with an abstract description on how it defines the game through a set of abstract mechanisms such as: *rules* and *objectives*. It is also mentioned briefly how gameplay has a close relationship with *narratology*, which is the study of narratives or story telling, but it is pointed out that a game need not include a story to be a game.

The gameplay theory presented also features a discussion on the anatomy of a game world, and how it is typically made up of both *static* and *dynamic* objects. The dynamic game elements tie into the gameplay system directly, while the static ones do not. Dynamic objects are typically more expensive in terms of processing than static objects, since pre-computing can be used to a larger degree when the object never changes during play.

The high-level flow of the game can be viewed as objectives integrated into the game world as either a sequence, a tree or a graph. The objectives are encoded with rules on how they should be solved, and the player will be awarded or penalized depending on the outcome. Story-driven games can also include support for in-game movies or *cutscenes*

Any gameplay system must have a game object system for managing the dynamic objects in the game, and all these objects can be modeled and simulated through the *game object model*, a specialized programming interface. This object model might vary between tool-side and runtime to allow suitable data in both cases.

Gameplay foundation systems

The game rules, objectives and dynamic world elements in a game are constructed with a collection of software components, which Jason Gregory calls the *gameplay foundation system*. He points out that in theory one could build a gameplay foundation system that is completely general, but in practice the system usually contain game- or genre- specific solutions. The gameplay foundation system, although implemented differently in each engine, usually consist of the following subsystems: *game object model*, *level management and streaming*, *real-time object model updating*, *messaging and event handling*, *scripting* and *objectives and game flow management*.

Game object model Out of all the components in the gameplay foundation system the game object model is probably the most complex. There are a number of ways of implementing the system but they all fit in one way or other on a list ranging from *object-centric* to *property-centric*. In an object-centric view each game object is represented as a single class instance which encapsulates the properties and behavior of the game object. In a property-centric view each game object is represented by a single identifier, and the properties aggregated from different property tables. The behavior in property-centric view is determined by the properties used by the game object.

The object model must be able uniquely identify all the objects in the model which requires some form of identifier to uniquely differentiate each of the objects. A hashed string identifier approach might be a better approach than using simple integers since it affords meaningful object names.

Game objects can support being spawned or destroyed dynamically, which can be tricky when the external resources needed have to be loaded into memory. Some engines do not allow dynamic spawning mid-game to avoid performance hits during play. Instead the dynamic game objects must be loaded along with the game world, which keeps the number of dynamic objects constant until the next game world is loaded.

The game object model is also responsible for the real-time update and simulation of the game objects which entail determining the update order. The simplest way would be to simply iterate the collection of game objects and call a virtual update method, but this approach is to simple for most cases. A batched approach where objects of the same type are processed together would solve some of the performance issues such as cache coherency and duplicate processing. Dependencies among game objects and game objects and low-level subsystems can be solved with *bucket* and *phased* updates respectively. In a bucket update approach the buckets are made up of the levels in object dependency forest where the roots are the objects with no object dependencies. A phased approach is used when several updates or *update phases* are needed to properly solve a low-level issues.

All game engines provide some way of querying the object model at least by the object identifier, but ideally the query support should be more advanced. There are any number of ways this could be useful such as; find all objects of certain type, find enemy players in the line of sight of the player and to find the objects in the path of a projectile in a certain order. The problem is that this kind of flexibility usually comes at the cost of performance, which makes game developers sacrifice flexibility for a few optimized genre-specific queries such as the case with the projectile.

In addition to these features, a game object model must provide the game objects with the ability to link with low-level game engine systems, allowing them to connect with the engine provided functionality such as graphics or physics. The types of objects available in the game object model should also be extendible so that new object types easily can be added, either through data or native code.

Loading and streaming Although not as complex as the game object model, loading and streaming the game world is challenging enough and has it's own set of pitfalls and design alternatives. The objects can be stored and loaded as *binary images* or binary copies of the tool-side objects, but this requires the runtime objects to have intimate knowledge of the tool-side object, right down to the endianness and bit layout. A slightly more flexible approach would be to use serialization into a readable format like XML, but this would increase the time required for loading during runtime. A third approach would be to use what he calls *spawners* which provide full decoupling from the tool-side object by storing just an identifier and an optional type identifier along with a key- value map providing the initial values of the object.

Loading the game world can also offer some challenges, since it is common today that the entire game world cannot fit into the available memory at once. There are alternatives to how to do this in practice, but the easiest would be to show a loading screen when loading new levels. This is the most games did it in the past, but today game designers try to avoid this complete stop in gameplay through the use of tricks such as small intermediate worlds used as loading locks or some other form of hidden loading.

Events and messages Events are essential in all computer games, and because of this game engines typically incorporate a system for passing events or messages between the game engine and game objects. A simple solution would be to just simple call a virtual method on each object, but this would require every game object to know about every game event since they would have to be inherited from a base class. A solution would be to build an event passing system that does not require the game objects to inherit from a base method. Some languages like C# have this built in in the form of the *delegate*.

Scripting A scripting language is a programming language that exist to permit user control to customize the behavior of a particular software application. There are a number of languages that can be used, ranging from *interpreted* to *compiled*, from *imperative* to *functional* and from *procedural* to *object-oriented* to mention some of the characteristics. Some of the game engines available today have created their own custom scripting language such as QuakeC and UnrealScript, while others use finished light-weight languages such as Lua or Python.

There are a number of ways to integrate a scripting environment with a game engine. *Scripted callbacks* are a method where the engine for the most is hard-coded, but certain selected areas allow customization through what Gregory refers to as *hook functions* and *callbacks*, which can be described as an empty function declarations, which will be invoked in fixed fashion, that can be implemented with the desired behavior. An alternative perspective is to consider these callbacks as *scripted event handlers* responding to certain events in the game or engine. The next alternative is to allow the scripting language not only to change behavior through callbacks, but also to extend the base class to create new types. In a component- or property-based object system it makes sense to make these customizable through script as well. The next alternative would be to have the entire game object model run in the script engine and allow access to low-level engine functionality when that is required. A last alternative mentioned is to flip the relationship and run the entire engine in the script engine and all the low-level engine functionality are accessed through libraries.

2.2.3 A software architecture for games

Michael Doherty has written an article on software architecture for games that has been published in his university's journal [Doherty, 2003]. The paper proposes a high-level general software architecture for games, with the hopes that this architecture can serve as a starting point for future research into the overall design of game engine architecture. He points to the fact that the game development industry are becoming more aware of the need for better development methodologies and software engineering techniques, and that existing game development literature focus on the low-level systems such as graphics and physics.

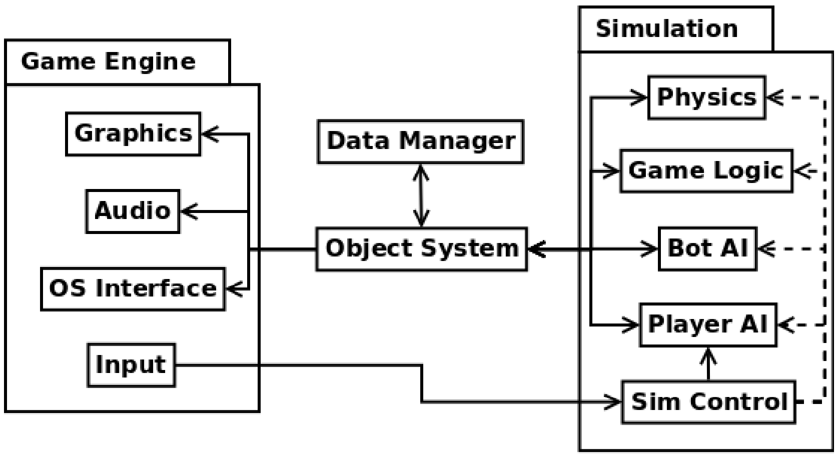


Figure 2.2: The top-level game architecture proposed by Michael Doherty

His architecture is made up of the following four components: *game engine*, *simulation*, *object system* and *data manager*. These main game engine components are discussed in the following sections.

Game engine

The game engine is defined as a component of engine modules responsible for interacting with the user and updating the object system. He mentions *graphics*, *audio*, *OS interface* and *input* as typical modules. In his engine design all of the modules has one way communication with the object system.

Simulation

The simulation component is responsible for simulating the virtual world and maintain the rules of the game. He divides the simulation component into the following modules: *simulation control*, *player AI*, *bot AI*, *game logic* and *physics*. The simulation control module receive input from the game engine and is responsible for forwarding the input information along with game time information to the rest of the modules in the simulation component. The AI modules are responsible for the internal decision process of the player and opponents. The game logic module is responsible for governing the rules of the game. The physics module controls the interaction of the objects in the game world. These modules do not interact directly, but rather through the object system.

Object system

The object system component is responsible for maintaining all the state information in the game world, and the type of data in the component is game specific. The article presents a design space for this component that range from object-centric to component-centric, and he presents research that have discuss the benefits and disadvantages of each of these approaches. He also argues that due to the similarities between game object systems and common database systems, the game industry could benefit from adopting theory and implementation techniques developed in the field of database systems.

Data manager

The data manager is not described in great detail, but it is responsible for providing save and load functionality to the engine. This is done by storing or restoring the relevant objects in the object system.

2.2.4 Component based game development - A solution to escalating costs and expanding deadlines?

Eelke Folmer has written an article [Folmer, 2007] that discuss decreasing the cost and time of creating games through the use of Commercial Of The Shelf components (COTS) or ready made components. Other than improving the development time and cost, he argues that these components will provide higher quality results than creating custom ones, and that by allowing COTS developers to focus on one specific area will advance the field at a faster rate than if they had to spread out their efforts to the entire engine.

His definition of a component makes no distinction between entire three-dimensional game engines and smaller components such as audio and networking, possibly to make sure that the architecture is applicable in both cases.

Layered component-based architecture

He present a reference architecture for games that have been inferred by combining the design from published and unpublished game architectures with a layered reference architecture for component-based development. He presents a layered reference architecture without any real detail. The reference architecture presented in the article was created to have a tool to find commonalities between game implementations or in other words point to potential areas of reuse. The reference architecture in the article is represented by four layers: *game interface layer*, *domain specific layer*, *infrastructure layer* and *platform software.*

The game interface layer is the top layer and contains the objects and components that are tied to a specific game such as the user interface, game logic, models and textures. The domain specific layer is made up of component that are specific to the domain of games e.g *graphics* and *physics.* The infrastructure layer contains general-purpose components that potentially could be used in any type of software e.g. *persistence* and *scripting.* The platform software layer is simply described as: "commonplace pieces of software that are brought in to underpin the game".

Areas of reuse

He identifies the components in the domain specific layer to hold the highest potential for reuse between different projects, but he points out that the components are designed for a particular game and might not fit need of every game. The tools such as exporter, importers and other third-party applications are also reused between different projects

Problems with components

The article also discuss the current problems with COTS development. The cost and dependency issues related to integrating third-party components have proven in the past to be outweighed by the ease of having everything provided ready for use in one single framework. The next problem is the increasing amount and complexity of the code needed to properly integrate the different components. Data synchronization between the different components can also quickly become a problem for the performance of the entire system. It can also lead to problems if the game architecture emerges rather than be designed intentionally, leading to a less than ideal design. Choosing the right component for the job can also be difficult since the choice typically needs to made early in the development cycle.

2.2.5 A flexible and expandable architecture for computer games

Jeff Plummer has written a master thesis that present an architecture for game software [Plummer, 2004] that govern how the subsystems interact without sacrificing the quality attributes such as flexibility and expandability. He use UML *use case* diagrams to analyze both Blizzard's Starcraft and Epic's Unreal Tournament to provide the functional requirements of these games. He discuss different possible solutions before settling on a design for the proposed architecture and developing a prototype for functional testing.

Game architecture

The game architecture proposed in the paper suggests that using a combination of *data-centered* and *system of systems* topology might be the best solution for games. In a data-centered topology the different subsystems of the game e.g. graphics all communicate directly with a central data component with shared game data. In a system of system topology the components are seen as complex and complete systems that provide a solution to a particular area of a game. When these two topologies are combined they make up an architecture of complex systems that interact with a central data component with shared game data. An overview of the presented architecture is given in figure 2.3.

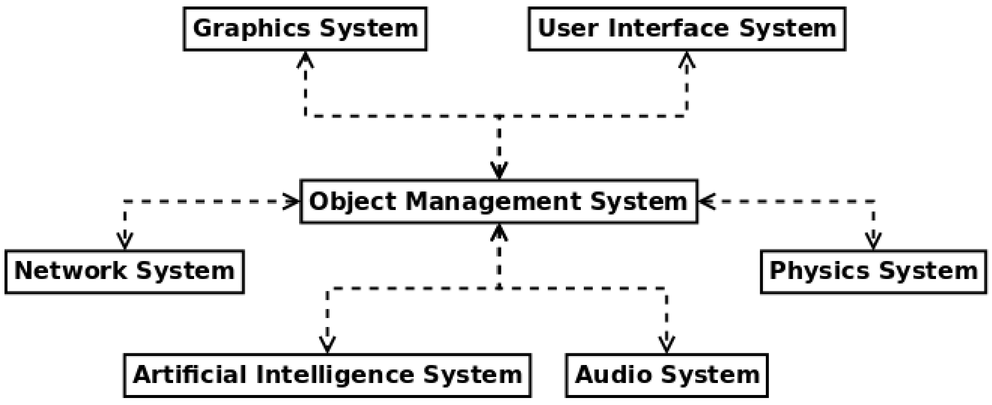


Figure 2.3: The top-level game architecture proposed by Jeff Plummer

The architecture allow domain specific components to connect with the central object management system in way that maintains the system-of-systems philosophy by allowing access only to the domain specific data.

This data-centered topology was chosen over a *layered* topology due the fact that the data-centered approach increase the attention to the individuality of the low-level components, which typically is not the case with a layered approach. Jeff Plummer argues that this will provide better support for COTS-based development (see section 2.2.4).

Jeff Plummer ha chosen a *repository* style of communication where the shared data is passive, and the clients or subsystems must query the data themselves. The alternative is the *blackboard* style of communication where the shared data lets the clients know when an update has occurred, but he argues that repository is the most logical, even though this would require an increase in the communication between the clients and the shared data.

Not a game engine

The goal of his thesis was not to create an architecture for game engines, but rather for games. He argues that the use of complete engine solutions, although an excellent example of code re-use, is hurting the industry. Game engines are designed with an initial game in mind, and as a result the design will not be flexible enough for all games. In addition the cost of licensing state-of- the-art engines are high, which means the game must sell big in order to recuperate the initial cost, and in the end this might make game publishers and developers focus on commercial viable projects rather than creative ones.

2.2.6 3D game engine design

David Eberly has written a popular book on three-dimensional game engine design [Eberly, 2001] that covers several of the aspects of three-dimensional game engine design from a practical point of view. The book has been included in the background information listing to represent the large number of available relating to the low-level systems of game engine development.

The book is written with a high level of detail and provides both the mathematics and the algorithms needed to implement just about all parts of a working three-dimensional game engine. He starts with a chapter covering the mathematic foundation before he moves on the domain-specific systems such as: graphics pipeline, scene representations, picking, collision detection, animation of characters, terrain and spatial sorting.

The book provides detailed descriptions on the low-level modules of three-dimensional game engines, but the overall architecture is not described in the same level of detail. The appendix includes a description of an object-oriented infrastructure that includes support for storing and loading game data, but the model does not describe how the modules in the engine interact with each other and the game data.

2.3 Other

This section contains the article that did not fit into the two other categories.

2.3.1 Game logic portability

BinSubaih, Maddock and Romano have written an article about game logic portability [BinSubaih et al., 2005] which is beneficial when using game engines for academic research. They point to the fact that although the game engines available are flexible, they all require game logic to be represented in a proprietary format such as *UnrealScript*.

They argue that separating the game logic from the game engine has a number of benefits. It could encourage more researchers to use game engines for their research, since a substitution would be easy to make, and it could increase re- usability between projects, since the engine would be familiar in any case. They also argue that the logic format could be standardised, and that the scalability of logic would be increased.

Game logic engine

The architecture separates game logic from game engine by using a mediator they have termed *events space*. This space is made up of a *rules-engine*, a *loader* and an *adapter*. The rules-based system were chosen for two reasons; portability and domain- independence, and they point to the fact that rule-based systems are well suited for encoding behavior of *Non Player Characters* (NPC). The adapter is responsible for communicating the game status between the engine and the rules-engine, and must therefor be able to speak the language of both parties. The loader is responsible for loading the game logic data and translating it to the language of the rules-engine. All the game objects in some way controlled by rule-based behavior must exist both in the game engine and in the rules-engine.

2.3.2 Game engine architecture and design

Game Engine Architecture and Design [Rollings and Morris, 2000] is a book that covers several aspects related to the process of designing and creating games. Judging by the name it should have been placed in section 2.2, but contradictory to the implication of title the topics in the book range from gameplay design and gameplay balancing to team management and general game development overview, and has little to do with game engine architecture and design.

Gameplay

Gameplay is introduced in the book as aims and objectives that if completed reward the player in one way or another. This view is relatively simple when compared to the definitions presented earlier, but this can be explained by the fact that the book is meant as a guide to game designers rather than to define what gameplay is. The gameplay discussion focus on *choices* and present several cases and principles that is meant to help make these choices interesting and in turn make the game fun. A big part of making the choices interesting is to balance the choices available in the game, and to avoid dominant or near dominant strategies.

Game architecture

The book also features information on how to turn the game design into a finished game. They discuss the benefits of using third-party libraries and art assets to decrease both the costs and development time of creating a game, and they discuss different game timing options.

They present an abstracted view of games that describe games using *game tokens* which they claim should be able to describe all type of games. These tokens are objects in the game e.g. game world and player, but they stay clear of calling them objects in order to avoid being associated with the object- oriented programming paradigm. The tokens all use what they call *interaction matrices* that contain the fixed possible events that occur different types of tokens interacts.

The structure of the tokens is stated to be mostly flat except for the abstract top level relationships such as: "..., every token has to operate within the game world in order to form a part of the game", and: "The player avatar is the representation of the player within the game world".

The book also features a section on how tokens and events of the game can be put represented as *finite-state machines*. Both the game tokens and the game itself can change behavior or state depending on the events occurring in the game. They extend the typical state machine with transition states and property information to allow timed transitions between finite states and more complex behavior respectively.

2.4 Theoretical foundation

The background material presented in the previous sections provide information on computer games and gameplay ranging from high-level philosophical observations of computer games to low-level game engine implementation details. The following sections will elevate the theoretical foundation for this thesis from this material. Section 2.5 discuss the issues.

2.4.1 General game logic

The game logic and rules have been described as algorithms by Jesper Juul 2.1.1 by the fact that they are both *finite*, *definite* and *effective*. As a consequence any game rule can be described using a programming language, but the general game data still need to be separated from the low-level systems in order for the game logic to be considered general.

The article on game logic portability by BinSubaih et. al. (see section 2.3.1) take an interesting approach to achieve game logic portability both between different releases of the same engine and different engines altogether. They separate the game logic from the game engine and let all the game logic be handled by a general game rules engine, and the outcome is synchronized with the underlying game engine.

2.4.2 Component-based development

A component-based development process has properties that could benefit the game development industry (see section 2.2.4 and 2.2.5) like the improved flexibility introduced by the innate separate nature of components. This view is shared by this thesis, but this thesis will develop this concept further by applying the development philosophy in the design and architecture of our game mechanics engine. Such an engine, if possible, could serve as a standard framework for integrating components.

2.4.3 Game object model and scripting environment

Both the game object model and the scripting environment are important topics in the context of this thesis since they are responsible providing the foundation for game logic and game rules.

The game object model contains all the game world data which has the consequence that it typically contains dependencies between game logic and low- level functionality and data. This corresponds directly with the primary research objective. The background information provides several alternative designs for the implementation of a game object model (see section 2.2.2 and 2.2.3) which will provide the foundation when finding a suitable candidate for the game object model in a game mechanics engine. The suitable candidates can be identified by the ability to separate pure game data from data required by low-level components which are liable to contain genre-specific solutions.

The scripting environment has no direct relationship with the primary research objective since most scripting languages have been made to be general i.e. they have no genre or game related low-level dependencies. The game object model in a game mechanics engine is general as well, so it could be required of the scripting environment to be able to extend the model to allow game-specific types. The background information contains a detailed description on game engine scripting environment (see section 2.2.2) which will provide the theoretical foundation for the choice of scripting architecture in our game mechanics engine.

2.4.4 Game engine design

The research in this thesis will base its approach in other areas of established game engine design as well the special topics already discussed.

The primary research objective requires detailed game engine descriptions that can provide the foundation for selecting genre-independent components. This is provided by the book on game engine architecture (see section2.2.2) which cover the components of a game engine from hardware to game logic. Of highest interest are the (other) components defined in the gameplay foundation layer and the game asset manager. Most of these components can be implemented so that they work independently of game and genre, and as such they are candidates to be included in a game mechanics engine.

Our prototype game mechanics engine created for the purpose of experimentation will be built according to the established theory in this paper, but the experiments will require low-level components as well in order to fulfill the first research objective. These components will be created in the easiest manner possible, but still according to established low-level game subsystem theory (see section 2.2.6).

2.5 Issues

This section discusses the issues with the existing theory and practice in relation to this thesis. Section 2.4 discuss the theoretical foundation for this thesis.

2.5.1 Engine-dependent game logic

The game logic framework in computer games today are game engine dependent (see section 2.3.1 and section 2.2.2). This has the consequence that the same logic would have to be implemented differently for each game engine, due to the differences in the design and implementation of the game logic framework. Another disadvantage of the engine-dependent game logic is that the possibilities in the architecture will be constrained by the same inflexibility as the underlying engine. Section 2.5.2 discuss this inflexibility in more detail.

Jesper Juul argues that the rules themselves can be considered as both absolute and definite (see section 2.1.1). A consequence of this is that they can be declared using a general purpose programming language, or even by incorporating a general rule framework, as proposed by BinSubaih, Maddock and Romano (see section 2.3.1). This thesis will attempt to combine the idea of general game logic into the existing theory on how to provide a game logic framework in a game engine.

2.5.2 Game engine problems

Game engines available on the market today are becoming more powerful and flexible with each new release, but the customizability is still constrained within the possibilities of the current low-level systems that constitutes the engine. These low-level systems have typically been designed and optimized for one particular game or genre, and as a result they are not as suitable for other types of games (see sections 2.2.2, 2.2.4, 2.2.1 and 2.2.5). Technical advances have improved this genre dependency in the low-level systems, but due to the vastly different requirements of the different types of games, it seems unlikely that one general *all-in-one* solution ever would be suitable for use in all game projects.

It is of course possible to replace the individual low-level components in the game engine, but as it has pointed out by Folmer (see section 2.2.4) and Plummer (see section 2.2.5), these replacements in the underlying architecture can be difficult to perform.

The article on written by Anderson et. al. on the future of research into game engine (see section 2.2.1) points out that even though there exists an empirical understanding of game engines as the software foundation for computer games, they have never accurately defined. They also point to the fact that there exists no all-purpose general game engine today, which is confirmed by the game engine reusability gamut presented by Gregory (see figure 2.1). These two existing issues with game engines, along with the others mentioned above, is what has motivated the search for an alternative in this thesis.

Anderson et. al. (see section 2.2.1) also discuss the possibility establishing whether or not there exist something that could be called a game engine that is independent of any game or genre. They suggest that a taxonomy of game engines from different genres is needed in order to find commonalities between the engines, and that this could lead to an all-purpose game engine. This approach is for the most part copied by this thesis, but there are some issues with the way they suggests the commonalities should be found.

They suggest that a taxonomy of game engines from different genres is needed in order to find commonalities between all the engine, and that this could lead to a genre-independent game engine. The use of the term taxonomy in this context is a bit confusing, but they seem to imply that an analysis is necessary, which would have several obvious practical issues.

They also point to the fact that even if commonalities are found and can be put together to form an engine, it remains to see if it will retain it's relevance to computer games by being able to keep essential game components intact, but they do not specify exactly what components this entail.

2.5.3 No common game patterns or structures

As established in this chapter, computer games are governed by a set of simple and typically easy-to-understand rules that control the outcome of the game (see section 2.1.1, 2.3.2 and 2.1.2). These rules exist at the core of all computer games and have been studied in a number of ways, but the definitions they provide do not reflect on implementation details, and as a result they offer little to construct reusable game object patterns.

Game rules and logic are tied to the underlying game engine components, which makes the process of specifying them change from platform to platform. With a stable game development environment, this would no longer be the case. Hopefully this will lead to reusable patterns emerging both in the structures and behavior of the objects in the game object model.

Chapter 3

Method

This chapter presents the theory behind this thesis in more detail. The first section discuss the reasoning behind both the idea and the design of a game mechanics engine, while the second section describes the experiments performed both to prove the thesis and to look for reusable gameplay patterns.

3.1 Game mechanics engine

This section will start by introducing the reader to the theory behind the concept of a game mechanics engine, and then the different design possibilities for realizing such an engine will be discussed.

3.1.1 The theory

The concept of a genre-independent game engine is by no means new, but like it was pointed out in the article on the future for research of game engine architecture presented in section 2.2.1, it remains to see if a genre-independent engine composed of common genre-independent components in game engines will retain the relationship with games.

This thesis has been formulated under the hypothesis that in order to maintain this relationship with games it is critical that the genre-independent engine successfully integrates a general foundation for game logic. The idea behind this reasoning has been taken from Jesper Juul's book where he describes computer games as made up of real rules and fictional worlds. This statement is elaborated further in section 2.1.1 on page 10, but the main idea is that at the core of every computer game is a set of absolute rules that govern the outcome of the game, and that these core rules can be viewed separately from the fictional or virtual world where the game takes place.

Juul defines these rules to be algorithmic since they fulfill certain requirements such as being both definite and finite, which means they could be declared using a general programming language. This view of general game logic is strengthened by research into general game logic engines (see section 2.3.1) and the abstract perspective of game logic as tokens (see section 2.3.2).

Separate the rules from the world

In order for the conceptual view of rules and worlds to make an impact outside the realm of theory, they must first be described from a practical perspective. The rules in a game are typically declared using a combination of game data in a game object model and some form of scripting language (see section 2.2.2) and 2.2.3). For the sake of the formulation of this thesis these two features will be referred to as the foundation for *gameplay mechanics.*

The problem with this design is that the general game data is intertwined with the low-level specific data, so to separate the real rules from the virtual world the primary challenge will be to find a game object model that can separate the general game data from the virtual world specific data. This corresponds with the primary research question in this thesis.

It is worth mentioning that separating the data would separate the scripted logic as well. The game logic code would be the code that operate solely on the *pure* game data, while a new layer of logic is introduced that operate on the virtual world data.

From object model and scripting environment to game mechanics engine

With the game logic cleanly abstracted from the virtual world, it can be combined with other game independent components commonly available in game engines. This relates to the topic of genre-independent engines in the article on the future of research on game engine architecture presented in section 2.2.1. They suggest that a genre-independent game engine might be found by analyzing games from different genres and finding common components. The approach taken in this thesis differ in that genre-independent game engine subsystems are extracted based on existing theory and logical reasoning alone e.g. a scripting environment can not be considered general and at same time be genre-dependent.

The different game engine designs and architectures presented in the previous chapter all have in common that they incorporate different subsystems that provides low-level functionality such as graphics and audio, and they all aim to game and genre independent. This means that one of these subsystem architectures as well can be integrated with the game mechanics foundation without sacrificing the game and genre independence.

Another obvious choice would be to integrate the other features of the game foundation layer (see section 2.2.2) such as events and message passing, game object queries and game data serialization. In addition the foundation can be expanded with support for data-driven development, which is considered important in game development.

The integration of these additional components would expand the conceptual gameplay mechanics foundation with features that go beyond the borders of gameplay into the game itself. The resulting game software architecture is in this paper referred to as a *game mechanics engine.*

3.1.2 The architecture and design alternatives

This section provides the theory behind the different design possibilities available for implementing our game mechanics engine. This theory is provided by the available background information on general game engine architecture and design as presented in chapter 2.

Game components

The low-level subsystems like graphics and audio will for the remainder of this thesis be referred to as *game components*. It has already been established in game engine architecture and design theory that it is important to keep the dependencies between low-level components to a minimum (see section 2.2.2, 2.2.3, 2.2.4 and 2.2.5).

The existing theory already provides solutions that keep the low-level game logic framework separate from the low-level components, so there is no need create a new architecture in order to comply to the primary research objective in this thesis. This section will discuss any benefits and disadvantages of the different approaches suggested in this background information to see if some solution might be more suitable than others.

The architecture presented by Gregory and Folmer are structured into layers, where the same-level components are both independent of each other and all components on the layers above, but dependent on components in the layers below. The two remaining architectures by Doherty and Plummer can both be considered as data-centered architectures since they both feature a central game data model which is directly connected to the other game components. All communication between the different components go through the central game data component.

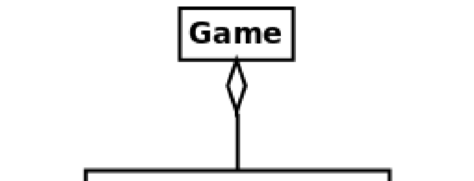


Figure 3.1: The top-level architecture in Microsoft XNA

The different top-level architectures proposed in the referred papers all have in common that they differ between different types of components i.e. they have different interfaces for the different types of components typically connected such as graphics and audio. An alternative inspired by the XNA framework by Microsoft [1] could be to separate only between the game components and the drawable game components. This arguably provides better separation from the low-level components than the architectures proposed in the background information, since the type of the component is not important, only how it connects to the engine.

The game object model

The discussion related to the game object model is an essential part of this thesis because it relates to the primary research objective. The game object model is typically the part of a game where the most dependencies between game logic and low-level functionality emerge, so it is essential for the success of this project that a suitable design can be found.

Design alternatives The background information provides different approaches to implementing a game object model, but not all of them are suitable for the purpose at hand. The design scope of the proposed object models vary between the different sources. Doherty propose a range from *object-centric* to *component-centric* (see section 2.2.3), while Gregory propose a range from *object-centric* to *property-centric* (see section 2.2.2).

An object-centric design would not be fitting, since this typically would require all the game object properties and data to be encapsulated in a single object, which would mean that the pure game data would have to be intermingled with low-level specific data. A general game object model based on an object- centric design could still be created, but it would require that each of the general object would be extended with the necessary low-level data for the game, which easily could lead

A property-centric design has the ability to split the game objects into pure game data properties and low-level specific properties, which would fit the requirement made by the primary research objective. A component-centric design would correspond nicely with the top-level game component design and at the same time fulfill the primary research objective. A component-centric design also has the benefit that pure game object data can be kept both in a game object component and the object itself.

Out of the two alternatives mentioned here the component-centric design seem to be the best choice. This would split the game object entity into *game objects* with pure game data and *game object components* with either game data or low-level data.

[1]http://create.msdn.com/en-US/ (last accessed December 15, 2010)

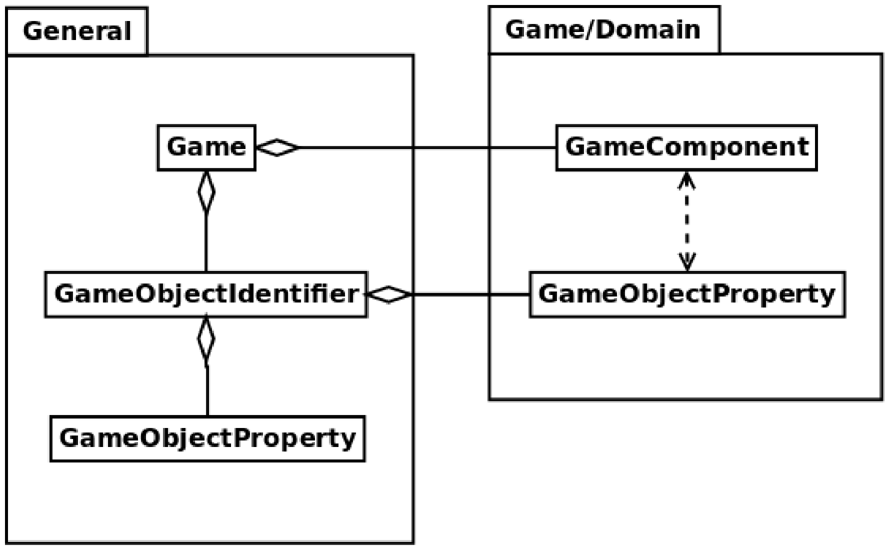


Figure 3.2: Top-level view of property-centric design

Extendible model The game object model in a game mechanics engine will contain no game- or genre-dependent objects, since this obviously would break the genre independence. Because of this it is required that the integrated game object model be extendible either by native code or scripting language.

Scripting

The scripting environment is an essential part of a game mechanics engine, and like with the game object model discussed in section 3.1.2 it is highly relevant to the primary research objective in this thesis.

There are a number of alternatives for a scripting environment both in terms of programming style, language selection and overall scripting architecture [Gregory, 2009] which provide the foundation for this discussion. The only real requirement for the scripting solution is that the scripting language must be able to specify general logic, but it would benefit the solution if it is capable of extending the game object model with new types.

Scripting architecture There are a number of alternatives on how to integrate a scripting environment with the engine to form a scripting architecture. All of the methods presented in background information should be able to describe general logic, so if the only desired feature of the scripting environment is to able to write simple logic then there should be no restrictions other use a language without low- level restrictions.

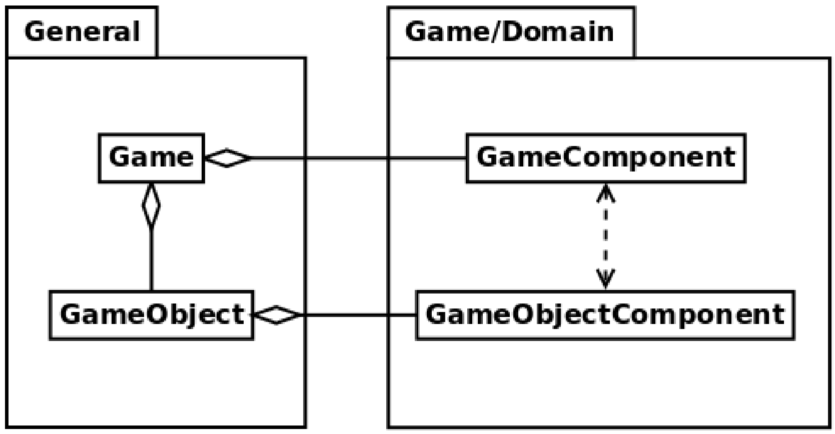


Figure 3.3: Top-level view of component-centric design

The game object model must be designed to be extendible by native code, but it can also allow the designers to extend the model using a scripting language. This optional requirement is only supported by the last three of the discussed alternatives in the background information.

Language selection The proposed languages range from compiled to interpreted, from functional to imperative and from procedural to object-oriented, and there are a number of languages available in each category. The different languages have different features and properties and as such the power of the different languages depends on the context.

There is no reason why several languages could be supported in the same environment. An example could be to use a declarative language like XML for declaring the game data which then could be combined with an object-oriented language for scripting the behavior. There are several of other combinations that could work equally well, or even better in some cases, which suggests that integrating a framework with support for multiple languages like Java [2] or Mono [3] could be a good idea.

Game logic and virtual world logic The scripting language would be used to customize the behavior of the entities in the game object model, but seeing as this model now is separated into pure game data and low-level data, so will the logic be separated in to pure game logic and low-level logic. In the terms this paper this will be referred to as *game object logic* and *game object component logic.*

[2]http://www.java.com/en/ (last accessed December 20, 2010)

[3]http://www.mono-project.com/ (last accessed December 20, 2010

Other features

The gameplay foundation layer as defined by Jason Gregory (see section 2.2.2) provides several other features that could be integrated into the concept of our game mechanics engine.

3.2 Game object patterns

This section will discuss game object model's preliminary building blocks and discuss the game object patterns types that will be included in the object model in one way or other. These patterns relate to the secondary research objective in this thesis.

3.2.1 Building blocks

The game object model in a game mechanics engine will need to include at least one base class from which other classes can inherit. The game object and game object component type is mandatory, but other types of base classes could be included as well. The following two paragraphs will discuss the two alternatives included in the prototype engine.

Game segment The game object model in the prototype game mechanics engine includes a base class for a game segment. These segments are meant as top-level game object containers that are meant to split the game into completely separate parts. The game segment is the parent container for all the game objects and game actions in that particular segment. Unlike the game object base class, the game segment base class does not support components, and it has a slightly different set of methods and events.

Game action Game actions was included in the game object model in the prototype to allow low-level game components to affect the game model in an alternative way to include a game object component.

3.2.2 Time in games

Time plays a part in just about all computer games, and even then time is involved in the actual real time it takes to play. Time in computer games can be viewed from at least three different perspectives. The real time is as mentioned the actual real time elapsed playing the game. The game time is the time elapsed as seen from the game updates, which can differ from the real time due to not having enough computing power or the game being paused. Fictional time is the time as experienced by the player in the game, which can be sped up or slowed down to fit the style of play.

3.2.3 User interface objects

Every game contain some form of user interface used for in-game menus, buttons, selection boxes and so on. The user interface architecture in a game typically also contain objects for creating a *heads-up-display* (HUD) for providing the player with relevant in-game feedback.

User interface objects like the menu and selection box are fairly well established so it should be relatively easy to integrate such mechanisms in the game object model. The other user interface related object mentioned might be more difficult to abstract in a meaningful way.

3.2.4 Component controller objects

The virtual world represented by the game components and game object components depend on the game rules, but in certain situations it might also be required to include *controller* objects in the object model. This would pollute the game object model with data outside the scope of pure game rules. An alternative would be to convert the controller objects to components, which would maintain the pure game objects.

3.2.5 Others

The secondary objective of this thesis is to see what kind of game object patterns emerge when creating games on a stable game development environment such as the game mechanics engine. See section 3.3 for details on the experiments.

3.3 Experiments

There has been conducted two separate experiments in this project in order to prove the theory established in this chapter. The two experiments where chosen to provide two widely different game environments in order to force multiple types of low-level components to be implemented.

The primary research objective was the only concern when settling on the manner in which these experiments should be conducted. The focus in these experiments have been to see if a game object model can be structured in way that separate the pure game data from the low-level game component data, which makes it critical to test the solution in an actual working game engine. If the primary objective was to identify patterns it would be beneficial to only simulate different game genres, making it possible to test more game logic.

3.3.1 The prototype architecture

The prototype game mechanics engine we have created to support the experiments has been built using Microsoft's XNA framework, which provide many of the features needed in the engine out of the box.

Architecture

The prototype is built according to the theory discussed in this chapter which involve amongst other things to provide low-level game component management. This has been implemented in the prototype according to the design presented in figure 3.1 with a central *Game* class that control the connected components. Figure 3.4 shows how this design is integrated chosen game object model style to form an architecture that separates the *general* from the *domain* and *game* specific.

Scripting The scripting environment in the prototype game mechanics engine can be considered as a script-driven architecture where the scripting environment is in control of running the entire game. The .Net [4] platform by Microsoft is used for all scripting, which means the prototype in theory supports development with several languages since it is all compiled down to a common *intermediate language*.

The architecture includes a game object model that split game objects into both game objects and game object components. These are implemented as standard managed classes so extending them should not be a problem.

[4]http://www.microsoft.com/net/ (last accessed December 20, 2010)

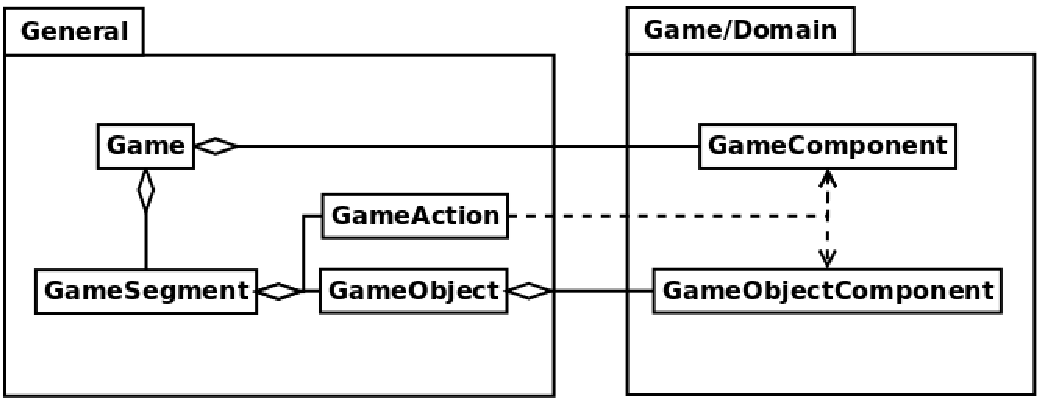


Figure 3.4: Top-level view of object model in the prototype

Content Pipeline A data-driven design has been pointed out as essential to the concept of game engines (see section 2.2.2). The content pipeline is the XNA term for the game asset pipeline. Figure 3.5 is flow chart describing how the XML game data with embedded game logic is transformed into a runtime assembly with the game logic and binary data file with the game data.

Other features

This section will discuss the other discussed features that been included in the game mechanics engine prototype.

The game resource loader is integrated into the XNA framework as the *content pipeline*. This provides a unified build pipeline for all game related content, and it provides runtime resource loading. The game scripts are compiled by the content pipeline into XNA's proprietary intermediate binary format and .Net assemblies.

A generic event handling mechanism has been integrated into .Net framework, so this feature is provided in the prototype without any effort at all. Event handlers are represented by the *delegate* which essentially is just a method reference.

Simple indexing techniques have been integrated into the game mechanics engine's central game class to provide the engine with simple game object query functionality. The implemented functionality index all the game objects and game object components by name and type and allows the clients to perform simple queries based on these tables. No support for more complex queries have been implemented into the prototype.

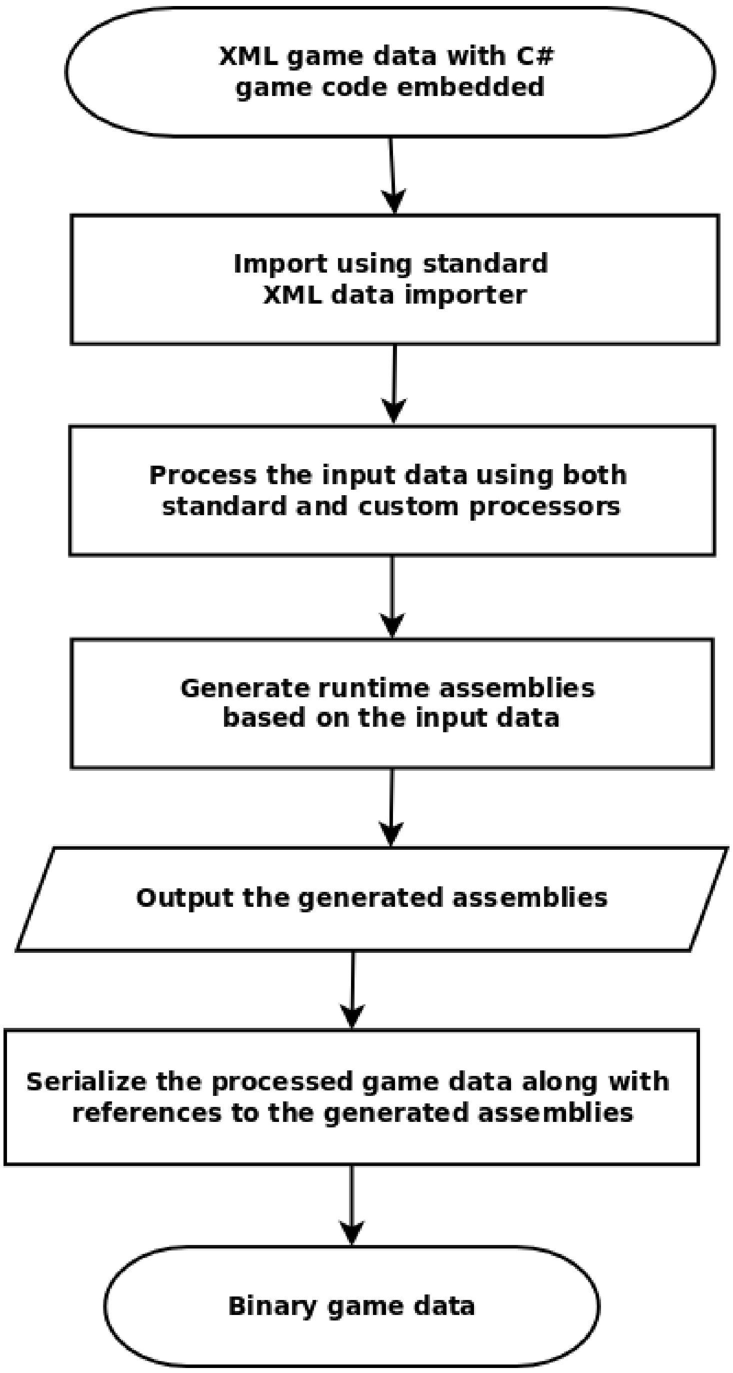
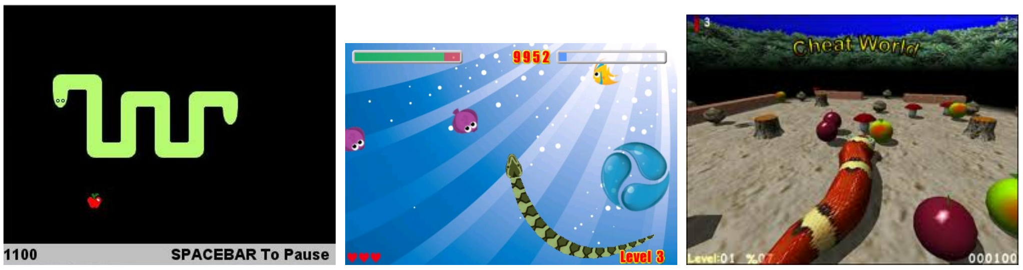

XML game data with C# game code embedded
Import using standard XML data importer
Process the input data using both standard and custom processors
Generate runtime assemblies based on the input data
Output the generated assemblies
Serialize the processed game data along with references to the generated assemblies
Binary game data

Figure 3.5: An overview of how XML game data with embedded C# game logic is transformed into binary data supported by the runtime architecture

3.3.2 Snake

The first game experiment in the project was chosen to be a remake, or rather a version, of the classic computer game *Snake*. This game has been created in a number of variant since it first appeared in arcades in the middle of the 1970's, and the reason for this is likely the same as the reason for choosing it as the first experiment; it is simple in terms of the game rules and it can be implemented with rudimentary graphics.



(a) 2D graphics, right turns (b) 2D graphics, non-right turns (c) 3D graphics, non-right turns

Figure 3.6: Existing Snake variants

Some of the different variants of the game Snake can be seen in figure 3.6. The first of these three is arguably the simplest variant i.e. two-dimensional graphics, right turn angles and simple keyboard control. The second alternative allows non-right turn angles, and probably an alternative input schema using the mouse. The third variant use three- dimensional graphics and most likely a third control schema optimized for three-dimensional gaming. The Snake variant created in this project will replicate the simplest of these variants i.e. use two-dimensional graphics, right turn angles and keyboard for input.

Rules of the game

The top-level objective in the game of Snake is to achieve the best score possible. The score points are gained by controlling the snake through a series of levels with a number of food sources in each level. For each food source consumed, the player receives a number of game points and the snake grows a number of world units. The player loose a life if he collides with either the world or itself, or if he/she attempt to turn in the snake into the current direction.

Requirements

The snake variant created for the first experiment will require a set of low-level game components and corresponding game object components that can provide the functionality needed to create the game as presented above. These components must all be implementable in way that is consistent with the design discussed in section 3.1.2.

Sprites *Sprites* are an easy and powerful approach to achieve a flexible two-dimensional graphics system, and they should be sufficient for providing all the graphics in the first experiment. The idea is to represent all graphics as two-dimensional images or texture and use an image combining methods such as *alpha blending* to merge the visible sprites into the resulting image on screen. Alpha blending use an additional channel in the sprite texture that specify the translucency of each of the texture elements or pixels, which also are known as *texels*.

In addition to just drawing an image on the screen, sprites are often used to render text by filling a single *sprite sheet* with all the letters in the alphabet and map each ascii code to a particular region on the sheet. Sprite sheets are smart solution where several textures are grouped into larger image to allow less frequent texture loads and at same time remove any texture size restrictions.

The sprite solution required for the experiment must be flexible enough to render everything from the snake itself to the text presented in the game, but must also be kept simple enough to be implementable in the time available.

World grid The virtual world in this first experiment is the field in which the player controls the snake. From a gameplay perspective this world is a two-dimensional grid of squared cells with fixed size that match the width of the snake.

Keyboard input The classic Snake game where controlled by just the arrow keys on the keyboard. Other control schemas are arguably more common today, but for the sake of this experiment only the keyboard will be required. The controls in Snake are simple in that only four actions are allowed; *turn left*, *turn up*, *turn right* and *turn down.* These actions should be mapped to the corresponding arrow key.

3.3.3 FPS

The second game experiment in the project is not a remake of a famous game as the previous experiment in section 3.3.2. Instead the second game, called *FPS*, was chosen to obtain different requirements for the low-level components in the game.

Rules of the game

The FPS game is actually more of a test-platform for common three-dimensional graphics functionality than a game. The amount of available time has forced us to focus solely on the primary research objective in this experiment and simply reuse as much of the established patterns in the first experiment as possible. This has the positive consequence that it shows in practice practice that patterns can be reused across different types of games, not just within a specific genre, as long as the engine is stable.

Requirements

There are no game to provide actual requirements for this experiment. Instead the requirements have been chosen manually to test essential three-dimensional functionality. There are plenty of resources available for common practices and techniques in three-dimensional game engine design, but one of the best is the book by David Eberly which has been briefly summarized in section 2.2.6.

Models As this second experiment is a three-dimensional game, it will need to be able to draw three-dimensional models to the screen. There are a number of ways to utilize the graphics pipeline to render three-dimensional objects, and some of these can be complex and hard to implement.

In the past all three-dimensional games used a *fixed* rendering pipeline, which means that a fixed set of operations were included for every pixel drawn to the screen. Today on the other hand, most games use a graphics pipeline that can be customized by specialized programs that run directly on the *Graphics Processing Unit* (GPU).

XNA, the chosen platform for our prototype game mechanics engine operate solely with a programmable graphics pipeline. This involves writing vertex programs and pixel programs to transform each vertex and render each pixel respectively, but a standard solution is included in the framework to simplify basic rendering.

Skybox A three-dimensional game with outdoors environments must be able to render a virtual sky. First-person perspective games with outdoor environments typically use some kind of sky box to render a virtual sky in the game world. This can be achieved by drawing a simple six-sided cube around the player at each frame create the illusion of a sky.

Spatial sorting Tree-dimensional games typically contain some form of spatial sorting system that control the movement of all the objects in the scene and provide functionality such as proximity testing and visibility testing. There are a number of different solutions for providing this functionality, ranging from completely open scenes, to scenes organized into complex structures such as a *binary-space-partitioning* structure [Eberly, 2001].

Mouse input The typical control schema for a first-person camera game is to use the keyboard for moving the character and the mouse for looking around. The mouse works differently from the keyboard since the mouse must convey not only the status of the buttons, but also the current position of the mouse.

Chapter 4

Results

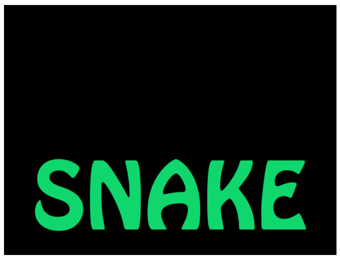
This chapter will present the reader to the results gained from the experiments, and then go on to discuss the results in the context of the issues identified in section 2.5.

4.1 Experiments

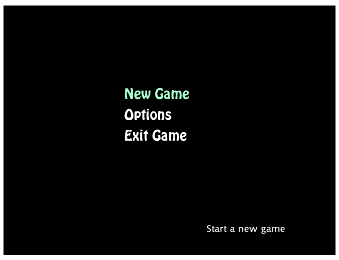
The primary goal for these experiments is to determine if the game object model can be separated properly according to the primary research objective in this thesis.

4.1.1 Snake

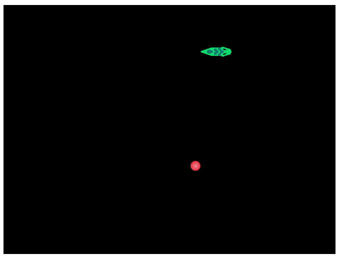
This game development experiment where the first of the two conducted, and as such it paved the way in terms of developing both the architecture of the prototype game mechanics engine and the game object patterns. The complete listing of the entire experiment has been included in the appendix on page 67. Figure 4.1 shows the game at different stages.



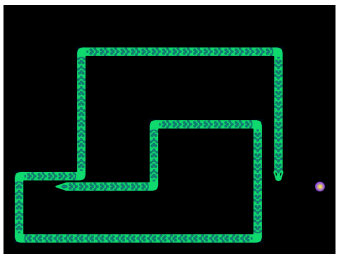
(a) Splash screen



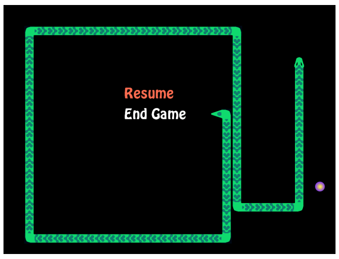
(b) Main menu



(c) After initial spawn



(d) After some eating



(e) In-game menu

Figure 4.1: Screen captures from our experiments with Snake

Components

This section discuss the different game components we created in order implement the first experiment, and the game object components that we created to provide the game components with appropriate data.

Sprite renderer The sprite renderer developed for this experiment provides just the basic functionality needed. Everything that should be rendered using the sprite renderer must be represented with one or more sprite operations. These operations provide all the necessary data to render an image, or even a text string at the desired position and orientation. The sprite renderer manages a list of all the sprite operations to be rendered in the next frame. The render operations are cleared after each frame and then repopulated by querying the game for all the game object component types that contain sprite data.

```
<Component Type="GamePipeline:Play.Snake.SnakeSpriteDataContent">
  <Name>SnakeSpriteData</Name>

  <RuntimeAssemblies>
    <Assembly>Game.Play.Snake</Assembly>
  <Assembly>Game.Play.Snake.SnakeSpriteData</Assembly>
  </RuntimeAssemblies>
  <PipelineAssemblies>
    <Assembly>
      Game.Content.Pipeline.Play.Snake.SnakeSpriteData
    </Assembly>
  </PipelineAssemblies>

  <TextureReference Type="XnaPipeline:ExternalReference[...]">
    <Reference>#External1</Reference>
  </TextureReference>
</Component>
```

Listing 4.1: The XML required to include a specialized snake sprite component in a world object

The sprite renderer game component as described above separate between regular sprites using only a single image and sprite operations that render text to the screen. Both the world objects in the first experiment require normal sprites to render their virtual representations in the game world, while the user interface require both images and text to properly render the controls. Listing 4.1 shows how a sprite data is added to the snake object.

Grid manager The grid manager developed for the experiment provides simple two-dimensional world management by arranging the world object based on the cells in the grid. The world object intersection tests are handled in a brute-force fashion in order to keep the game component simple.

Several different types of game object components where created to properly implement this functionality. The first one is meant to be used in the

```
<Component Type="Pipeline:Components.GameWorld.XnaWorldObjectGridDataContent">
  <Name>FoodWorldData</Name>
  <BaseTypeWriterName>XnaWorldObjectGridDataWriter</BaseTypeWriterName>
  <GeneratePipelineCode>true</GeneratePipelineCode>

  <RuntimeAssemblies>
    <Assembly>Game.Play.Food</Assembly>
  </RuntimeAssemblies>

  <Methods>
    <Method Type="Pipeline:CustomOverride">
      <Name>HandleIntersected</Name>
      <Source>
        // Remove the parent food object
        ((WorldObject)ParentObject).Remove();

        base.HandleIntersected(sender, eventArgs);
      </Source>
    </Method>
  </Methods>
</Component>
```

Listing 4.2: The XML required to create a new base object component type for world grid data that can be added to food objects

game world object, and it contains information on the width and height of the grid. The remaining ones are meant for the game objects in the game world, and they specify how the object is positioned in the world.

Keyboard manager Our keyboard manager component is a simple game component which sole purpose is to check the current state of the connected keyboard and fire game events based on the status. The game actions can connect to the different keys on the keyboard either by registering an event handler, or by querying the keyboard manager manually.

User interface data Our user interface data created for the first experiment contains logic and data that provide the game with basic user interface functionality. The components are based around a subset of the possible control hierarchy discussed in section 3.2. The game object component were implemented as active components, in that they do not require a central user interface manager game component in order to operate successfully.

Game object patterns

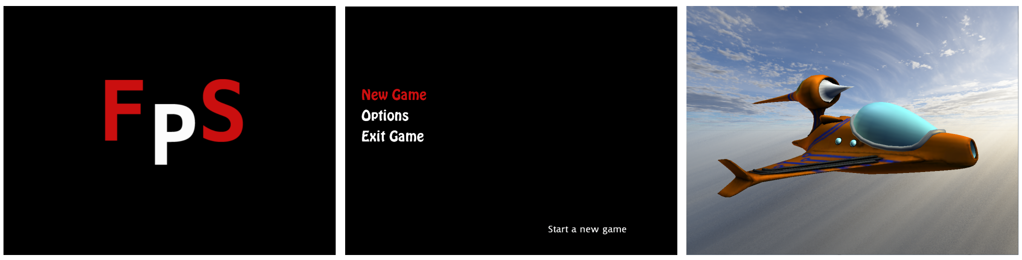
There are three game object types defined for this experiment; *Snake*, *World* and *Food*. Other than that, the game object base types are provided by the game mechanics engine.

The game use three game segments to structure the game into; *splash screen main menu* and *play*. The splash screen is controlled by a timed sequence structure that controls when to fade in and out the splash images. The main menu features a simple three-option game menu that provide the user with the ability to start a new game, edit the game options and to exit the game.

The play segment feature to controlling sequences and a representation of the game world along with the snake and the food objects in that world. The first controlling sequence is world sequence, which provide the progress through the levels (though only one were created). The second sequence is a random sequence that control which food object to spawn in the game world.

4.1.2 FPS

The second experiment as specified in section 3.3.3 is more focused towards testing new low-level components than to develop game object patterns. The overall structure of the game is reused from the first experiment so there are some noticeable similarities, but the game world itself is widely different. The entire experiment have been included in the appendix on page 108. Figure 4.2 presents some screen captures from the game.



(a) Splash screen (b) Main menu (c) The game

Figure 4.2: Screen captures from our experiments with FPS

Components

This experiment can reuse some of the components we created for the first experiment, since both the keyboard and a sprite renderer is required. This section will discuss the game components we developed especially for this experiment.

```
<Component Type="GamePipeline:Play.World.WorldModelDataContent">
  <Name>WorldModelData</Name>

  <RuntimeAssemblies>
    <Assembly>Game.Play.World.WorldModelData</Assembly>
  </RuntimeAssemblies>
  <PipelineAssemblies>
    <Assembly>Game.Content.Pipeline.Play.World.WorldModelData</Assembly>
  </PipelineAssemblies>

  <Properties>
    <Property>
      <Name>Rotation</Name>
      <TypeName>float</TypeName>
    </Property>
  </Properties>

  <Methods>
    <Method Type="Pipeline:CustomOverride">
      <Name>Update</Name>
      <Source>
        Rotation += (float)gameTime.ElapsedGameTime.TotalSeconds;
        WorldMatrix = Matrix.CreateRotationY(Rotation);
        base.Update(gameTime);
      </Source>
    </Method>
  </Methods>

  <ModelReference
    Type="XnaPipeline:ExternalReference[XnaGraphicsPipeline:NodeContent]">
    <Reference>#External1</Reference>
  </ModelReference>
</Component>
```

Listing 4.3: The XML required to add a rotating model to a game object

Mouse manager Any game using a first-person perspective would benefit from being able to provide player input by other means than just the keyboard. It is common in these types of game to use the mouse to control the view angle of the player camera, and the keyboard for moving the player in the world. The mouse manager is different from the keyboard manager in that it must provide values instead of just a simple boolean flag.

Scene manager Our scene manager implementation wrap all world object in an *axis-aligned bounding box*, which is a three-dimensional box with edges parallel to the world axis, but no actual collision detection have been included in the components update cycle.

Model renderer To render three-dimensional models can become quite complex, so the implementation chosen for our model renderer is implemented in the simplest manner possible.

Like with the sprite renderer, the model renderer use the discrete render operations to perform the rendering, but unlike the two-dimensional sprite operations, these three dimensional operations are considerably more complex. All operations in addition to the reference to the model to render, must contain the matrices that are used to position the model in the world, to provide the camera orientation and to provide the projection matrix that handle the perspective projection to the screen. Figure 4.3 contains the code necessary to create a rotating model in the scene.

Sky renderer The sky renderer we implemented for this experiment were implemented as a simple sky box renderer. It queries the game object model for a list of the supported object components and draws a simple textured cube around specified position. The box is rendered with depth testing turned off to allow subsequent render operations to draw over the sky, leaving just the visible sky when finished.

4.2 Research findings

This section will discuss the results presented in this chapter in relation to the issues identified in section 2.5.

4.2.1 General game logic

Both of the experiments were created successfully using the exact same game mechanics engine prototype which integrates the game object model as shown in figure 3.4. As can be observed in the results presented in this chapter and the appendices on page 67 and 108, the general purpose game objects have been combined with a number of different low-level systems through the use of game object components, and there has been no intermingling of the pure game data and the low-level specific data. Based on this it is reasonably safe to assume that the combination of applied game object model design and a general-purpose programming language can provide a general framework for game logic.

This expands on the theory presented by BinSubaih et. al. (see section 2.3.1), where a separate game logic engine is setup to handle the game rules, to integrate a general logic framework with existing theory on how to represent game logic in game engines.

4.2.2 A flexible game mechanics engine

The success of applying existing theory to establish a general game logic framework provides the foundation needed to establish our concept of a flexible and genre-independent game mechanics engine, and the prototype engine proves effectively that such an engine can in fact be created.

The engine is free of any low-level components that might include any game- or genre-dependencies, so the issues with inflexibility in the low-level components of the game engine (see section 2.5.2). As consequence of not having any low-level game components integrated, the engine has been designed from the beginning to allow for easy integration of any low- level functionality. This provides a solution to the other kind of engine inflexibility identified in section 2.5.2 since would make it considerably easier to change low-level components, depending on the needs of the game.

4.2.3 Reusable game object patterns

To be able to discuss the findings related to game object model patterns, we must first acknowledge the fact that the search for these patterns have been affected due time restrictions. The second experiment was originally intended to be a test-platform for typical game patterns in the *first-person-shooter* (FPS) genre, but instead the experiment was restricted to only test for proper separation from the low-level components, in order to be able to conclude the outcome of the primary research objective.

The game object model in the prototype where initially setup with game objects that fit with the game object categories discussed in section 3.2. No additional non-trivial game objects were created during the experiments except the ones already expected.

The entire game structure was reused between the experiments, allowing the second experiment to have working structure within minutes. This kind of reuse, although not what expected, can still provide some value.

Chapter 5

Discussion

This chapter starts with a discussion on how the findings in section 4.2 can provide possible benefits to the game development industry. The chapter is concluded with a section discussing the issues with the theory established in this thesis, and also a section discussion the ways in which it can be improved further.

5.1 Benefits

A genre-independent game mechanics engine as proposed in this thesis could have a number of benefits to the community.

5.1.1 Separate game from game engine

The article on the future of game engine design research presented in section 2.2.1 discuss the lack of proper definition that separates the game from the game engine. A game mechanics engine with a stable data interface would remove this ambiguity by accurately defining what is required to make a game using the engine.

5.1.2 Serious games

The genre-independent nature of the game mechanics engine would make it suitable for application in areas outside the normal computer game. The engine would provide many of the same benefits as the solution proposed by BinSubaih et. al in their article on portable game logic (see section 2.3.1), but without the need for a separate rules engine.

5.1.3 Third-party tools

A stable game mechanics engine would allow for the introduction of third-party tools specifically designed for creating games. Today a large part of the tools used to create a game is created by the same company that provided the game engine, which means that a large part of game developers' valuable time are spent creating the tools necessary to create games instead of actually creating them.

With third-party tools this situation would be greatly improved. Some tool development would still be required by all the component providers, since they also would be required to supply the tools to import the component data into the game asset pipeline, and any tool-side plug-ins required.

Third-party developers could with time be able to provide the game development industry with more sophisticated tools than they were able develop on their own before.

5.1.4 Game development standard

The game mechanics engine as proposed in this thesis could provide the foundation for a game development standard, or at least an open or in some collaborative project that could be maintained by all interested parties. The true potential of the concept can only be achieved through collaboration.

5.1.5 The game designer profession

The game designer creates games, but how he/she does it depends on lots of factors, one of them being the company's game development platform. With third- party tools this would no longer be the case, since the game designers will work with the same tools no matter what game project they are working on. This also has the direct effect that designers more easily can transfer between projects.

The game mechanics engine would also arguably allow the game designers to focus more on the game rules than before. This statement can be defended by the fact that the game rules framework would be stable, allowing the designers gain more experience as time goes by. In addition the game rules are placed at the center of the actual game specification, which could shift the focus away from the low-level functionality.

5.2 Issues

There are a number of potential issues with the proposed concept of our game mechanics engine.

5.2.1 Too simple?

The concept of a game mechanics engine is simple, but no mention of such an engine have been found during the background research for this thesis. It is likely that someone have thought of this already, so why then have the idea been discarded? One obvious reason could be that such an engine were thought not to offer enough features to be beneficial.

The term game engine are today almost synonymous with complex low-level solutions game related functionality such as graphics, but the game mechanics has been stripped of all such components, which could make it less attractive than the standard alternative.

5.2.2 Added complexity

One could also argue that the game mechanics solution would add complexity to an already complex solution, since the game mechanics engine would have to be able to connect with other existing low-level component to provide the functionality needed.

Another argument in the same category could be that the game mechanics engine would provide general solutions to situations where custom solutions would be better suited for some reason or other.

5.2.3 Not invented here

There are a number of game development companies out there that have implemented their own powerful game engine. The reason for developing own technology is probably one of three things. Either they cannot afford to licence a third-party engine, or they can afford it, but cannot find a suitable solution, or the simply want to develop their own.

In order for the solution proposed here to reap any of the potential benefits, it is essential that enough game developers embrace the idea. Without the support of the community, the solution would be just another game engine, and not the all-purpose solutions it is meant to be.

5.3 Future work

There are a number ways the theory presented in this thesis can be improved further. This last section will discuss some of these possible improvements to give an idea of how the theory can be developed further.

5.3.1 General query framework

The game object model in our game mechanics engine allow clients to query the model based on simple criteria such as identifier and type. It would be very helpful if this framework could be extended to allow for more complex queries (see section 2.2.2).

Ideally it should be possible to integrate a general solution for this as well, but this could prove to be difficult. The game object model queries can be dependent on low-level functionality, which makes it necessary to come up with a solution that provide a general abstraction to this dependency.

5.3.2 Network support

The need for network support in computers games are apparent since networked gaming today is ubiquitous. There are both theory and established practices that provide solutions for integrating network support in game engines. These solutions should provide the foundation needed to network replication support in the game mechanics engine proposed in this thesis.

5.3.3 Thread support

The current trend in CPU's development have been to increase the number of available cores in the CPU, which consequently makes it important for multiple threads to be integrated into the proposed design. The area of multiple threads in game engines have barely been researched, but the research on threads in general software should also be relevant.

5.3.4 Shared data support

The game mechanics engine as proposed in this thesis do not contain features for shared low-level data. This is handled in the current design by synchronizing the data between the game object components. It could be beneficial to integrate a solution that could handle the shared data automatically. Such a solution could also prove useful for integrating support for multiple threads, by integrating mechanisms that synchronize the data access between the different threads.

5.3.5 Reusable patterns

The search for reusable patterns were proved to require to much work to be executed properly in this thesis, but the possible benefits of such patterns have not diminished any because of this. Therefor, it would be desirable to investigate this area in the future.

Chapter 6

Conclusion

This concluding chapter will start by a recapitulation of the research problem that has motivated this thesis, and then go on present the main findings and a short summary of future work.

Game logic and game rules exists in all computer games and can be considered both absolute and algorithmic. Despite this general nature, the game logic are implemented differently in each of the game engine available today, due to differences in low-level engine design and the data interfaces supported. A solution have been proposed that integrate a general rules engine with existing game engines and synchronize the game state representation between the game engine and the rules engine, but this solution is aimed towards easier use of game engines in research and other serious application areas, rather than actual game development.

The game object model in the computer games today typically contain both game data and low-level data intermingled in the game object, which affects the game engine flexibility by making it difficult to exchange one low-level component with another. Solutions have been proposed that could solve this issue by keeping the low-level systems separate, but these efforts are aimed at providing standard way of structuring the game engines rather than to provide a genre-independent game engine.

6.1 Contribution

The main contribution of this thesis is to present a genre-independent game mechanics engine capable of providing general game logic, based completely on existing theory on game logic and game engine design.

6.1.1 General game logic

This thesis expands on the established theory and practices related to the game object model, to integrate the idea of general game logic. A component- or property-centric design have been proposed to deal with some of the issues related to object or class inheritance. This works by splitting the game object data into either components or properties based mainly on the context of the functionality they help provide, but simply by changing this method slightly to separate the general from the game and genre specific, the object model will be separated accordingly. This has the desirable consequence that a general purpose scripting or programming language could be used to specify general game logic using the general data in the object model.

This concept has been tested with two separate experiments using our game mechanics engine prototype, and the experiments where designed to require a variety of low-level game components to provide the necessary functionality. Both a two-dimensional world structured around a central grid and a open three- dimensional world were tested along with different input styles, and the architecture proved to be capable of all of these components without sacrificing the general nature of the core game data.

6.1.2 Genre-independent game mechanics engine

With the general game logic in place, it is possible to combine this architecture with other genre-independent game engine features to create the concept of the genre-independent game mechanics engine. The idea of general game logic is not new, and neither is the game object model design and general purpose scripting language used to achieve general game logic in a game engine. The actual implementation of a genre-independent game engine on the other hand have not been yet been realized, which is why our game mechanics engine can be considered the main contribution of this thesis.

Our conceptual game mechanics engine can not considered a game engine in the normal sense, since it is completely free of low-level components that might contain game- or genre-dependent solutions such as graphics and audio, and instead include only a general framework for connecting any such component. On the other hand, the game mechanics engine do contain other essential parts to game development such as the general game logic architecture and a stable data pipeline, which could outweigh the perceived disadvantage of not including low- level components if the game mechanics engine and component based philosophy were accepted and adopted by the game development community. If this were to happen, these components could be readily available in a number of variants, along third-party game development tools.

6.1.3 Reusable game object patterns

The search for reusable patterns in the game object model has suffered under the scope of the first two objectives in this thesis, but some simple conclusions can still be drawn from the experience of using our game mechanics engine prototype to perform the experiments in this thesis.

The overall structure of the game were reused between the two projects. This provided the second experiments with a working splash screen, a main menu and an empty structure for creating the actual game within minutes. Besides this, no non-trivial reusable patterns, outside the expected areas, were discovered. Despite of this, it is reasonable to assume that these patterns will emerge with a stable game development environment, at least as a set of personal tools for the individual designer.

6.2 The future

The future research into the proposed game mechanics engine will attempt to include the existing theory on networking and multiple threads in the design. In addition to these already researched areas, the theory will be expanded with solutions that provide a general query framework, along with a solution for shared low-level data.

Appendices

Appendix A

Snake experiment

This appendix contain all the XML source files created for the Snake experiment. The data files contains embedded C# logic that serves as the scripting language. The appendix is started by listing the base game object types in the game, followed by the segments and the game objects.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<XnaContent xmlns:Components="GaMEX.Components"
    xmlns:Pipeline="GaMEX.Content.Pipeline"
    xmlns:XnaPipeline="Microsoft.Xna.Framework.Content.Pipeline"
    xmlns:XnaGraphicsPipeline="Microsoft.Xna.Framework.Content.Pipeline.Graphics">
  <Asset Type="GaMEX.Content.Pipeline.GameWorld.WorldObjectContent">
    <Name>Snake</Name>
    <Namespace>Game.Play</Namespace>
    <BaseTypeWriterName>
      GaMEX.Content.Pipeline.Serialization.WorldObjectWriter, GaMEX.Content.Pipeline
    </BaseTypeWriterName>
    <GeneratePipelineCode>true</GeneratePipelineCode>

    <Enumerations>
      <!-- Enumeration over the different directions the snake can move -->
      <Enumeration>
        <Name>SnakeDirection</Name>
        <Values>
          <Value>Up</Value>
          <Value>Down</Value>
          <Value>Left</Value>
          <Value>Right</Value>
        </Values>
      </Enumeration>
    </Enumerations>

    <Properties>
      <!-- The current direction -->
      <Property>
        <Name>Direction</Name>
        <TypeName>SnakeObject.SnakeDirection</TypeName>
      </Property>

      <!-- The next direction to take -->
      <Property>
        <Name>NextDirection</Name>
        <TypeName>SnakeObject.SnakeDirection</TypeName>
      </Property>

      <!-- The length of the snake -->
      <Property>
        <Name>Length</Name>
        <TypeName>int</TypeName>
      </Property>

      <!-- The speed of the snake -->
      <Property>
        <Name>Speed</Name>
        <TypeName>int</TypeName>
      </Property>

      <!-- The number of ticks to grow the snake-->
      <Property>
        <Name>TicksToGrow</Name>
        <TypeName>int</TypeName>
      </Property>
    </Properties>

    <Events>
      <!-- Event fired when the snake starts growing -->
      <Event>
        <Name>StartedGrowing</Name>
        <DelegateTypeName>GaMEX.GameObjectEventHandler, GaMEX</DelegateTypeName>
      </Event>

      <!-- Event fired when the snake has stopped growing -->
      <Event>
        <Name>FinishedGrowing</Name>
        <DelegateTypeName>GaMEX.GameObjectEventHandler, GaMEX</DelegateTypeName>
      </Event>

      <!-- Event fired when the length of the worm is changed -->
      <Event>
        <Name>LengthChanged</Name>
        <DelegateTypeName>GaMEX.GameObjectEventHandler, GaMEX</DelegateTypeName>
      </Event>
    </Events>

    <Methods>
      <Method Type="Pipeline:CustomOverride">
        <Name>Initialize</Name>
        <Source>
          World world = ParentContainer as World;
          world.Loaded += delegate(object sender, GameEventArgs e) { Spawn(); };
          world.Unloaded += delegate(object sender, GameEventArgs e) { Remove(); };

          base.Initialize();
        </Source>
      </Method>

      <Method Type="Pipeline:CustomOverride">
        <Name>Spawn</Name>
        <Source>
          // Set the proper speed, register the tick event handler and start the timer
          GaMEX.Gameplay.Time.Timer timer =
            Game.GetObject&lt;GaMEX.Gameplay.Time.Timer&gt;("Game.Play.MoveTimer");
          timer.TicksPerSecond = Speed;
          timer.Tick += HandleTick;
          timer.Start();

          // Register the snake controls
          Game.GetActionTrigger("Game.Play.TurnUp").Triggered += HandleAction;
          Game.GetActionTrigger("Game.Play.TurnDown").Triggered += HandleAction;
          Game.GetActionTrigger("Game.Play.TurnLeft").Triggered += HandleAction;
          Game.GetActionTrigger("Game.Play.TurnRight").Triggered += HandleAction;

          base.Spawn();
        </Source>
      </Method>

      <Method Type="Pipeline:CustomOverride">
        <Name>Remove</Name>
        <Source>
          // Unregister the tick event handler and stop the timer
          GaMEX.Gameplay.Time.Timer timer =
            Game.GetObject&lt;GaMEX.Gameplay.Time.Timer&gt;("Game.Play.MoveTimer");
          timer.Tick -= HandleTick;
          timer.Stop();

          // Unregister the snake controls
          Game.GetActionTrigger("Game.Play.TurnUp").Triggered += HandleAction;
          Game.GetActionTrigger("Game.Play.TurnDown").Triggered += HandleAction;
          Game.GetActionTrigger("Game.Play.TurnLeft").Triggered += HandleAction;
          Game.GetActionTrigger("Game.Play.TurnRight").Triggered += HandleAction;

          base.Remove();
        </Source>
      </Method>

      <!-- Custom method for starting the process of growing a snake -->
      <Method Type="Pipeline:CustomMethod">
        <Name>Grow</Name>
        <Source>
          TicksToGrow += numTicks;

          if (StartedGrowing != null &amp;&amp; TicksToGrow == numTicks)
            StartedGrowing(this, null);
        </Source>
```

```
        <Parameters>
          <Parameter>
            <Name>numTicks</Name>
            <TypeName>int</TypeName>
          </Parameter>
        </Parameters>
      </Method>

      <Method Type="Pipeline:CustomEventHandler">
        <Name>HandleAction</Name>
        <Source>
          // Set the preliminary next direction based on the action triggered.
          if (sender == Game.GetActionTrigger("Game.Play.TurnUp"))
            NextDirection = SnakeDirection.Up;
          else if (sender == Game.GetActionTrigger("Game.Play.TurnDown"))
            NextDirection = SnakeDirection.Down;
          else if (sender == Game.GetActionTrigger("Game.Play.TurnLeft"))
            NextDirection = SnakeDirection.Left;
          else if (sender == Game.GetActionTrigger("Game.Play.TurnRight"))
            NextDirection = SnakeDirection.Right;
        </Source>
        <EventOwnerTypeName>GaMEX.GameActionTrigger, GaMEX</EventOwnerTypeName>
        <EventName>Triggered</EventName>
      </Method>

      <Method Type="Pipeline:CustomEventHandler">
        <Name>HandleTick</Name>
        <Source>
          // Update the direction
          Direction = NextDirection;

          // Grow the snake if currently growing
          if (TicksToGrow > 0)
          {
            TicksToGrow -= 1;

            Length += 1;
            if (LengthChanged != null)
              LengthChanged(this, null);
          }

          // Fire event if finished growing
          if (TicksToGrow == 0 &amp;&amp; FinishedGrowing != null)
            FinishedGrowing(this, null);
        </Source>
        <EventOwnerTypeName>GaMEX.Gameplay.Time.Timer, GaMEX</EventOwnerTypeName>
        <EventName>Tick</EventName>
      </Method>
    </Methods>

    <Components>
      <!-- Base component for all snake world data components -->
      <Component Type="Pipeline:Components.GameWorld.XnaWorldObjectGridDataContent">
        <Name>SnakeWorldData</Name>
        <BaseTypeWriterName>
          GaMEX.Content.Pipeline.Serialization.Components.XnaWorldObjectGridDataWriter
,
          GaMEX.Content.Pipeline
        </BaseTypeWriterName>
        <GeneratePipelineCode>true</GeneratePipelineCode>

        <RuntimeAssemblies>
          <Assembly>Game.Play.Snake</Assembly>
          <Assembly>Game.Play.Food</Assembly>
        </RuntimeAssemblies>

        <Properties>
          <!-- Property for storing the last position of the tail -->
          <Property>
            <Name>LastTailPosition</Name>
            <TypeName>XnaGridPosition</TypeName>
          </Property>
        </Properties>

        <Methods>
          <Method Type="Pipeline:CustomEventHandler">
            <Name>HandleTick</Name>
            <Source>
              // Save tail position
              LastTailPosition = ((XnaGridChain)Structure).Positions.Last.Value;

              // Move the snake according to the current direction
              SnakeObject parentSnake = ParentObject as SnakeObject;
              if (parentSnake.Direction == SnakeObject.SnakeDirection.Left)
                Structure.Move(-1, 0);
              else if (parentSnake.Direction == SnakeObject.SnakeDirection.Right)
                Structure.Move(1, 0);
              else if (parentSnake.Direction == SnakeObject.SnakeDirection.Up)
                Structure.Move(0, -1);
              else if (parentSnake.Direction == SnakeObject.SnakeDirection.Down)
                Structure.Move(0, 1);
            </Source>
            <EventOwnerTypeName>GaMEX.Gameplay.Time.Timer, GaMEX</EventOwnerTypeName>
            <EventName>Tick</EventName>
          </Method>

          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              // Add the new tail if the length of the snake changes
              ((SnakeObject)ParentObject).LengthChanged +=
                delegate(GameObject sender, GameObjectEventArgs e)
                {
                  ((XnaGridChain)Structure).Positions.AddLast(LastTailPosition);
                };

              base.Initialize();
            </Source>
          </Method>

          <Method Type="Pipeline:CustomOverride">
            <Name>Enable</Name>
            <Source>
              // Register the tick event handler
              Game.GetObject&lt;GaMEX.Gameplay.Time.Timer&gt;
                ("Game.Play.MoveTimer").Tick += HandleTick;

              base.Enable();
            </Source>
          </Method>

          <Method Type="Pipeline:CustomOverride">
            <Name>Disable</Name>
            <Source>
              // Unregister the tick event handler
              Game.GetObject&lt;GaMEX.Gameplay.Time.Timer&gt;
                ("Game.Play.MoveTimer").Tick -= HandleTick;

              base.Disable();
            </Source>
          </Method>

          <Method Type="Pipeline:CustomOverride">
            <!-- Handle intersection -->
            <Name>HandleIntersected</Name>
            <Source>
              foreach (XnaGridIntersection intersection in Intersections)
              {
                // Remove the snake if it has hit itself
                if (intersection.Second.ParentObject == this)
                  ((WorldObject)ParentObject).Remove();
                // Grow the snake if hit food
                else if (intersection.Second.ParentObject.GetType() ==
                    typeof(FoodObject))
                  ((SnakeObject)ParentObject).Grow(10);
              }
              base.HandleIntersected(sender, eventArgs);
            </Source>
          </Method>
        </Methods>
      </Component>

      <!-- Base component for all snake sprite data components -->
      <Component Type="Pipeline:Components.Graphics.XnaSpriteDataContent">
        <Name>SnakeSpriteData</Name>
        <BaseTypeWriterName>
          GaMEX.Content.Pipeline.Serialization.Components.XnaSpriteDataWriter,
          GaMEX.Content.Pipeline
        </BaseTypeWriterName>
        <GeneratePipelineCode>true</GeneratePipelineCode>

        <RuntimeAssemblies>
          <Assembly>Game.Play.Snake</Assembly>
        </RuntimeAssemblies>

        <Properties>
          <Property>
            <Name>GridData</Name>
            <TypeName>XnaWorldObjectGridData</TypeName>
          </Property>
        </Properties>

        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              GridData = ParentObject.GetComponent("SnakeWorldData") as
                XnaWorldObjectGridData;

              // Add necessary sprite operations when the parent snake is spawned
              ((SnakeObject)ParentObject).Spawned +=
                delegate(GameObject sender, GameObjectEventArgs e)
                {
                  int numPositions =
                    ((XnaGridChain)GridData.Structure).Positions.Count;
                  while (numPositions-- > 0)
                  {
                    XnaSpriteOperation operation = AddOperation();
                    operation.Origin = new Vector2(16, 16);
                  }
                };

              // Remove all sprite operations when parent snake
              ((SnakeObject)ParentObject).Removed +=
                delegate(GameObject sender, GameObjectEventArgs e)
                {
                  Operations.Clear();
                };

              // Add new sprite operation when the length of the parent snake changes
              // (it cannot shrink)
              ((SnakeObject)ParentObject).LengthChanged +=
                delegate(GameObject sender, GameObjectEventArgs e)
                {
                  XnaSpriteOperation operation = AddOperation();
                  operation.Origin = new Vector2(16, 16);
                };
              base.Initialize();
            </Source>
          </Method>

          <Method Type="Pipeline:CustomOverride">
            <Name>OnUpdate</Name>
            <Source>

              // This would probably be better of in a custom "native" game object
              // component. This code is responsible for updating the sprite
              // operations based on the orientation of the world data.

              LinkedListNode&lt;XnaGridPosition&gt; positionNode =
                ((XnaGridChain)GridData.Structure).Positions.First;

              SnakeObject parentSnake = (SnakeObject)ParentObject;

              int operationIndex = 0;
              while (positionNode != null)
              {
                XnaSpriteOperation operation = Operations[operationIndex++];
                if (positionNode.Previous == null)
                {
                  operation.SourceRectangle = new Rectangle(0, 0, 32, 32);

                  if (parentSnake.Direction == SnakeObject.SnakeDirection.Left)
                    operation.Rotation = -(float)Math.PI/2.0f;
                  else if (parentSnake.Direction == SnakeObject.SnakeDirection.Right)
                    operation.Rotation = (float)Math.PI/2.0f;
                  else if (parentSnake.Direction == SnakeObject.SnakeDirection.Down)
                    operation.Rotation = (float)Math.PI;
                  else
                    operation.Rotation = 0.0f;
                }
                else
                {
                  if (positionNode.Next == null)
                    operation.SourceRectangle = new Rectangle(0, 128, 32, 32);
                  else if (positionNode.Previous.Value.X == positionNode.Next.Value.X
||
                            positionNode.Previous.Value.Y == positionNode.Next.Value.Y
)
                    operation.SourceRectangle = new Rectangle(0, 32, 32, 32);
                  else if ((positionNode.Previous.Value.X &lt;
                              positionNode.Next.Value.X &amp;&amp;
                            positionNode.Previous.Value.Y &lt;
                              positionNode.Next.Value.Y &amp;&amp;
                            positionNode.Value.Y &lt;
                              positionNode.Next.Value.Y) ||
                           (positionNode.Previous.Value.X &lt;
                              positionNode.Next.Value.X &amp;&amp;
                            positionNode.Previous.Value.Y &gt;
                              positionNode.Next.Value.Y &amp;&amp;
                            positionNode.Value.X &lt;
                              positionNode.Next.Value.X) ||
                           (positionNode.Previous.Value.X &gt;
                              positionNode.Next.Value.X &amp;&amp;
                            positionNode.Previous.Value.Y &gt;
                              positionNode.Next.Value.Y &amp;&amp;
                            positionNode.Value.Y &gt;
                              positionNode.Next.Value.Y) ||
                           (positionNode.Previous.Value.X &gt;
                              positionNode.Next.Value.X &amp;&amp;
                            positionNode.Previous.Value.Y &lt;
                              positionNode.Next.Value.Y &amp;&amp;
                            positionNode.Value.X &gt;
                              positionNode.Next.Value.X))
                    operation.SourceRectangle = new Rectangle(0, 96, 32, 32);
```

```
                  else
                    operation.SourceRectangle = new Rectangle(0, 64, 32, 32);


                  if (positionNode.Value.X &lt; positionNode.Previous.Value.X)
                    operation.Rotation = (float)Math.PI/2.0f;
                  else if (positionNode.Value.X &gt; positionNode.Previous.Value.X)
                    operation.Rotation = -(float)Math.PI/2.0f;
                  else if (positionNode.Value.Y &lt; positionNode.Previous.Value.Y)
                    operation.Rotation = (float)Math.PI;
                  else
                    operation.Rotation = 0.0f;
                }
                operation.DestinationRectangle =
                  GridData.GetCellPosition(positionNode.Value, 16);

                positionNode = positionNode.Next;
              }

              base.OnUpdate(gameTime);
            </Source>
          </Method>
        </Methods>
      </Component>
    </Components>
  </Asset>
</XnaContent>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<XnaContent xmlns:Pipeline="GaMEX.Content.Pipeline">
  <Asset Type="Pipeline:GameWorld.WorldContent">
    <Name>World</Name>
    <Namespace>Game.Play</Namespace>
    <BaseTypeWriterName>
      GaMEX.Content.Pipeline.Serialization.WorldWriter, GaMEX.Content.Pipeline
    </BaseTypeWriterName>
    <GeneratePipelineCode>true</GeneratePipelineCode>

    <Methods>
      <Method Type="Pipeline:CustomOverride">
        <Name>Initialize</Name>
        <Source>
          SequenceStep parentStep = ParentContainer as SequenceStep;

          // Load the world when the parent sequence step is entered.
          parentStep.Entered += delegate(GameObject sender, GameObjectEventArgs e)
          {
            Load();
          };

          // Unload the world when the parent sequence step is exited.
          parentStep.Exited += delegate(GameObject sender, GameObjectEventArgs e)
          {
            Unload();
          };

          base.Initialize();
        </Source>
      </Method>
      <Method Type="Pipeline:CustomOverride">
        <Name>Load</Name>
        <Source>
          base.Load();
        </Source>
      </Method>
    </Methods>

    <Components>
      <!-- Specifies the dimension and size of the world grid -->
      <Component Type="Pipeline:Components.GameWorld.XnaWorldGridDataContent">
        <Name>WorldData</Name>
        <Grid>
          <Width>32</Width>
          <Height>24</Height>
          <CellSize>32</CellSize>
        </Grid>
      </Component>
    </Components>
  </Asset>
</XnaContent>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<XnaContent xmlns:Components="GaMEX.Components"
            xmlns:Pipeline="GaMEX.Content.Pipeline">
  <Asset Type="Pipeline:GameWorld.WorldObjectContent">
    <Name>Food</Name>
    <Namespace>Game.Play</Namespace>
    <BaseTypeWriterName>
      GaMEX.Content.Pipeline.Serialization.WorldObjectWriter, GaMEX.Content.Pipeline
    </BaseTypeWriterName>
    <GeneratePipelineCode>true</GeneratePipelineCode>

    <Properties>
      <!-- The value of the food object -->
      <Property>
        <Name>Value</Name>
        <TypeName>int</TypeName>
      </Property>

      <!-- The name of the sequence step controlling when the food object
           should be available-->
      <Property>
        <Name>FoodSequenceStepName</Name>
        <TypeName>string</TypeName>
      </Property>
    </Properties>

    <Methods>
      <Method Type="Pipeline:CustomOverride">
        <Name>Initialize</Name>
        <Source>
          if (!string.IsNullOrEmpty(FoodSequenceStepName))
          {
            // Spawn the food object when the sequence step is entered
            Game.GetObject&lt;SequenceStep&gt;(FoodSequenceStepName).Entered +=
              delegate(GameObject sender, GameObjectEventArgs e) { Spawn(); };

            // Remove the food object when the sequence step is exited
            Game.GetObject&lt;SequenceStep&gt;(FoodSequenceStepName).Exited +=
              delegate(GameObject sender, GameObjectEventArgs e) { Remove(); };
          }

          base.Initialize();
        </Source>
      </Method>
    </Methods>

    <Components>
      <Component Type="Pipeline:Components.GameWorld.XnaWorldObjectGridDataContent">
        <Name>FoodWorldData</Name>
        <BaseTypeWriterName>
          GaMEX.Content.Pipeline.Serialization.Components.XnaWorldObjectGridDataWriter
,
          GaMEX.Content.Pipeline
        </BaseTypeWriterName>
        <GeneratePipelineCode>true</GeneratePipelineCode>

        <RuntimeAssemblies>
          <Assembly>Game.Play.Food</Assembly>
        </RuntimeAssemblies>

        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>HandleIntersected</Name>
            <Source>
              // Remove the parent food object
              ((WorldObject)ParentObject).Remove();
              Game.GetObject&lt;SequenceStep&gt;(
                ((FoodObject)ParentObject).FoodSequenceStepName).Exit();
              base.HandleIntersected(sender, eventArgs);
            </Source>
          </Method>
        </Methods>
      </Component>

      <Component Type="Pipeline:Components.Graphics.XnaSpriteDataContent">
        <Name>FoodSpriteData</Name>
        <BaseTypeWriterName>
          GaMEX.Content.Pipeline.Serialization.Components.XnaSpriteDataWriter,
          GaMEX.Content.Pipeline
        </BaseTypeWriterName>
        <GeneratePipelineCode>true</GeneratePipelineCode>

        <RuntimeAssemblies>
          <Assembly>Game.Play.Food</Assembly>
        </RuntimeAssemblies>

        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              // Add the required sprite operation when the parent food object is
              // spawned
              ((WorldObject)ParentObject).Spawned +=
                delegate(GameObject sender, GameObjectEventArgs e)
                {
                  XnaWorldObjectGridData gridData =
                    ParentObject.GetComponent("FoodWorldData") as
                      XnaWorldObjectGridData;
                  XnaSpriteOperation operation = AddOperation();
                  operation.SourceRectangle = new Rectangle(
                    (((FoodObject)ParentObject).Value - 1) * 32, 0, 32, 32);
                  operation.DestinationRectangle = gridData.GetCellPosition(
                    ((XnaGridCell)gridData.Structure).Position);
                };

              // Remove the sprite operation
              ((WorldObject)ParentObject).Removed +=
                delegate(GameObject sender, GameObjectEventArgs e)
                {
                  Operations.Clear();
                };

              base.Initialize();
            </Source>
          </Method>
        </Methods>
      </Component>
    </Components>
  </Asset>
</XnaContent>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<XnaContent xmlns:Pipeline="GaMEX.Content.Pipeline"
            xmlns:UserInterface="GaMEX.Content.Pipeline.UserInterface"
            xmlns:Gameplay="GaMEX.Content.Pipeline.Gameplay"
            xmlns:Components="GaMEX.Content.Pipeline.Components">
  <Asset Type="Pipeline:GameSegmentContent">
    <Name>SplashScreen</Name>

    <Methods>

      <Method Type="Pipeline:CustomOverride">
        <Name>Activate</Name>
        <Source>
          Game.GetActionTrigger("Game.SplashScreen.SkipSplash").Triggered +=
            delegate(object sender, GameEventArgs e)
              { Game.LoadSegment("Game.MainMenu"); };

          ((Sequence)Game.GetObject("Game.SplashScreen.SplashSequence")).Completed +=
            delegate(GameObject sender, GameObjectEventArgs args)
              { Game.LoadSegment("Game.MainMenu"); };

          base.Activate();
        </Source>
      </Method>

    </Methods>

    <Actions>

      <Action>
        <Name>SkipSplash</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Activate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Space].StateChanged +=
                  HandleKeyPressed;
            </Source>
          </Method>

          <Method Type="Pipeline:CustomOverride">
            <Name>Deactivate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Space].StateChanged -=
                  HandleKeyPressed;
            </Source>
          </Method>

          <Method Type="Pipeline:CustomEventHandler">
            <Name>HandleKeyPressed</Name>
            <Source>
              if (args.Key.State == ButtonState.Pressed)
                Trigger();
            </Source>
            <EventOwnerTypeName>
              GaMEX.Components.Input.XnaKey, GaMEX
            </EventOwnerTypeName>
            <EventName>StateChanged</EventName>
          </Method>
        </Methods>
      </Action>

    </Actions>

    <GameObjects>

      <GameObject Type="Gameplay:SequenceContent">
```

```
        <Name>SplashSequence</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              ((GameSegment)ParentContainer).Activated +=
                delegate(object sender, GameEventArgs args) { Start(); };

              ((TimedLinearInterpolation)Game.GetObject(
                "Game.SplashScreen.SplashFade")).Completed +=
                  delegate(GameObject sender, GameObjectEventArgs e) { Step(); };

                base.Initialize();
            </Source>
          </Method>
        </Methods>
        <Steps>
          <Step Type="Gameplay:SequenceStepContent">
            <Name>Step1</Name>
          </Step>
          <Step Type="Gameplay:TimedSequenceStepContent">
            <Name>Step2</Name>
            <Duration>PT4S</Duration>
          </Step>
          <Step Type="Gameplay:SequenceStepContent">
            <Name>Step3</Name>
          </Step>
          <Step Type="Gameplay:SequenceStepContent">
            <Name>Step4</Name>
          </Step>
          <Step Type="Gameplay:TimedSequenceStepContent">
            <Name>Step5</Name>
            <Duration>PT4S</Duration>
          </Step>
          <Step Type="Gameplay:SequenceStepContent">
            <Name>Step6</Name>
          </Step>
        </Steps>
      </GameObject>

      <GameObject Type="Gameplay:Math.TimedLinearInterpolationContent">
        <Name>SplashFade</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              ((SequenceStep)Game.GetObject(
                "Game.SplashScreen.SplashSequence.Step1")).Entered +=
                  delegate(GameObject sender, GameObjectEventArgs e)
                  {
                    FromValue = 0;
                    ToValue = 255;
                    Restart();
                  };

              ((SequenceStep)Game.GetObject(
                "Game.SplashScreen.SplashSequence.Step3")).Entered +=
                  delegate(GameObject sender, GameObjectEventArgs e)
                  {
                    FromValue = 255;
                    ToValue = 0;
                    Restart();
                  };

              ((SequenceStep)Game.GetObject(
                "Game.SplashScreen.SplashSequence.Step4")).Entered +=
                  delegate(GameObject sender, GameObjectEventArgs e)
                  {
                    FromValue = 0;
                    ToValue = 255;
                    Restart();
                  };

              ((SequenceStep)Game.GetObject(
                "Game.SplashScreen.SplashSequence.Step6")).Entered +=
                  delegate(GameObject sender, GameObjectEventArgs e)
                  {
                    FromValue = 255;
                    ToValue = 0;
                    Restart();
                  };

              base.Initialize();
            </Source>
          </Method>
        </Methods>
        <FromValue>0</FromValue>
        <ToValue>255</ToValue>
        <Duration>PT2S</Duration>
      </GameObject>

      <GameObject Type="UserInterface:ImageContent">

        <Name>SplashImage1</Name>

        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              ((SequenceStep)Game.GetObject(
                "Game.SplashScreen.SplashSequence.Step1")).Entered +=
                  delegate(GameObject sender, GameObjectEventArgs e) { Enable(); };

              ((SequenceStep)Game.GetObject(
                "Game.SplashScreen.SplashSequence.Step3")).Exited +=
                  delegate(GameObject sender, GameObjectEventArgs e) { Disable(); };

              base.Initialize();
            </Source>
          </Method>
        </Methods>

        <Components>
          <Component Type="Components:UserInterface.XnaImageDataContent">
            <Name>ImageData1</Name>
            <Methods>
              <Method Type="Pipeline:CustomOverride">
                <Name>Update</Name>
                <Source>
                  Style.Color = new Color(
                    Style.Color.R, Style.Color.G, Style.Color.B,
                    (byte)((LinearInterpolation)Game.GetObject(
                      "Game.SplashScreen.SplashFade")).Value);

                  UpdateControl();

                  base.Update(gameTime);
                </Source>
              </Method>
            </Methods>
            <Style>
              <HorizontalAlignment>Stretch</HorizontalAlignment>
              <VerticalAlignment>Stretch</VerticalAlignment>
              <Padding>0 0 0 0</Padding>
              <Margin>0 0 0 0</Margin>
            </Style>
          </Component>
        </Components>
        <ImageFilename>Components/UserInterface/SnakeSplashLogo.png</ImageFilename>

      </GameObject>



      <GameObject Type="UserInterface:ImageContent">

        <Name>SplashImage2</Name>

        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              ((SequenceStep)Game.GetObject(
                "Game.SplashScreen.SplashSequence.Step4")).Entered +=
                  delegate(GameObject sender, GameObjectEventArgs e) { Enable(); };
              ((SequenceStep)Game.GetObject(
                "Game.SplashScreen.SplashSequence.Step6")).Exited +=
                  delegate(GameObject sender, GameObjectEventArgs e) { Disable(); };

              base.Initialize();
            </Source>
          </Method>
        </Methods>

        <Components>
          <Component Type="Components:UserInterface.XnaImageDataContent">
            <Name>ImageData2</Name>
            <Methods>
              <Method Type="Pipeline:CustomOverride">
                <Name>Update</Name>
                <Source>
                  Style.Color = new Color(
                    Style.Color.R, Style.Color.G, Style.Color.B,
                    (byte)((LinearInterpolation)Game.GetObject(
                      "Game.SplashScreen.SplashFade")).Value);

                  UpdateControl();

                  base.Update(gameTime);
                </Source>
              </Method>
            </Methods>
            <Style>
              <Width>600</Width>
              <Height>200</Height>
              <HorizontalAlignment>Right</HorizontalAlignment>
              <VerticalAlignment>Center</VerticalAlignment>
              <Padding>0 0 0 0</Padding>
              <Margin>0 0 10 0</Margin>
            </Style>
          </Component>
        </Components>

        <ImageFilename>Components/UserInterface/LaViMaSplashLogo.png</ImageFilename>

      </GameObject>
    </GameObjects>
  </Asset>
</XnaContent>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<XnaContent xmlns:Pipeline="GaMEX.Content.Pipeline"
            xmlns:UserInterface="GaMEX.Content.Pipeline.UserInterface"
            xmlns:Components="GaMEX.Content.Pipeline.Components">
  <Asset Type="Pipeline:GameSegmentContent">
    <Name>MainMenu</Name>

    <Methods>

      <Method Type="Pipeline:CustomOverride">
        <Name>Initialize</Name>
        <Source>
          base.Initialize(game);

          ((TextControl)Game.GetObject("Game.MainMenu.DescriptionOutput")).Enable();
          ((Menu)this.Game.GetObject("Game.MainMenu.Menu")).Activate();

          ((MenuItem)Game.GetObject("Game.MainMenu.Menu.NewGame")).Selected +=
          delegate(GameObject sender, GameObjectEventArgs e)
          {
          this.Game.LoadSegment("Game.Play");
          };

          ((MenuItem)Game.GetObject("Game.MainMenu.Menu.Exit")).Selected +=
          delegate(GameObject sender, GameObjectEventArgs e)
          {
          this.Game.Exit();
          };
        </Source>
      </Method>

    </Methods>

    <Actions>

      <Action>
        <Name>Select</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Activate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Enter].StateChanged +=
                  HandleKeyPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomOverride">
            <Name>Deactivate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Enter].StateChanged -=
                  HandleKeyPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomEventHandler">
            <Name>HandleKeyPressed</Name>
            <Source>
              if (args.Key.State == ButtonState.Pressed)
                Trigger();
            </Source>
            <EventOwnerTypeName>
              GaMEX.Components.Input.XnaKey, GaMEX
            </EventOwnerTypeName>
            <EventName>StateChanged</EventName>
          </Method>
        </Methods>
      </Action>

      <Action>
```

```
        <Name>NavigateUp</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Activate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Up].StateChanged +=
                  HandleKeyPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomOverride">
            <Name>Deactivate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Up].StateChanged -=
                  HandleKeyPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomEventHandler">
            <Name>HandleKeyPressed</Name>
            <Source>
              if (args.Key.State == ButtonState.Pressed)
                Trigger();
            </Source>
            <EventOwnerTypeName>
              GaMEX.Components.Input.XnaKey, GaMEX
            </EventOwnerTypeName>
            <EventName>StateChanged</EventName>
          </Method>
        </Methods>
      </Action>

      <Action>
        <Name>NavigateDown</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Activate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Down].StateChanged +=
                  HandleKeyPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomOverride">
            <Name>Deactivate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Down].StateChanged -=
                  HandleKeyPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomEventHandler">
            <Name>HandleKeyPressed</Name>
            <Source>
              if (args.Key.State == ButtonState.Pressed)
                Trigger();
            </Source>
            <EventOwnerTypeName>
              GaMEX.Components.Input.XnaKey, GaMEX
            </EventOwnerTypeName>
            <EventName>StateChanged</EventName>
          </Method>
        </Methods>
      </Action>

    </Actions>

    <GameObjects>
      <GameObject Type="UserInterface:MenuContent">
```

```
        <Name>Menu</Name>

        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              Game.GetActionTrigger("Game.MainMenu.Select").Triggered +=
                delegate(object sender, GameEventArgs e)
                {
                  MenuItems[ActiveIndex].Select();
                };

              Game.GetActionTrigger("Game.MainMenu.NavigateUp").Triggered +=
                delegate(object sender, GameEventArgs e)
                {
                  if (ActiveIndex == 0)
                    ActiveIndex = MenuItems.Count - 1;
                  else
                    ActiveIndex -= 1;
                };

              Game.GetActionTrigger("Game.MainMenu.NavigateDown").Triggered +=
                delegate(object sender, GameEventArgs e)
                {
                  if (ActiveIndex == MenuItems.Count - 1)
                    ActiveIndex = 0;
                  else
                    ActiveIndex += 1;
                };

              base.Initialize();
            </Source>
          </Method>
        </Methods>

        <Components>
          <Component Type="Components:UserInterface.XnaMenuDataContent">
            <Name>MenuControlData</Name>
            <Style>
              <Width>300</Width>
              <Height>300</Height>
              <HorizontalAlignment>Center</HorizontalAlignment>
              <VerticalAlignment>Center</VerticalAlignment>
              <Padding>10 10 10 10</Padding>
              <Margin>0 0 0 0</Margin>
            </Style>
          </Component>
        </Components>

        <MenuItems>
          <MenuItem Type="UserInterface:MenuItemContent">
            <Name>NewGame</Name>
            <Components>
              <Component Type="Components:UserInterface.XnaMenuItemDataContent">
                <Name>NewGameMenuItemData</Name>
                <Style>
                  <Color>FFFFFFFF</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 0 0 0</Margin>
                </Style>
                <ActiveStyle>
                  <Color>FFAAFFCC</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 0 0 0</Margin>
                </ActiveStyle>
                <FontFilename>
                  Components/UserInterface/MenuFont.spritefont
                </FontFilename>
              </Component>
            </Components>
            <Text>New Game</Text>
            <Description>Start a new game</Description>
          </MenuItem>

          <MenuItem Type="UserInterface:MenuItemContent">
            <Name>Options</Name>
            <Components>
              <Component Type="Components:UserInterface.XnaMenuItemDataContent">
                <Name>OptionsMenuItemData</Name>
                <Style>
                  <Color>FFFFFFFF</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 64 0 0</Margin>
                </Style>
                <ActiveStyle>
                  <Color>FFAAFFCC</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 64 0 0</Margin>
                </ActiveStyle>
                <FontFilename>
                  Components/UserInterface/MenuFont.spritefont
                </FontFilename>
              </Component>
            </Components>
            <Text>Options</Text>
            <Description>Configure game options</Description>
          </MenuItem>

          <MenuItem Type="UserInterface:MenuItemContent">
            <Name>Exit</Name>
            <Components>
              <Component Type="Components:UserInterface.XnaMenuItemDataContent">
                <Name>ExitMenuItemData</Name>
                <Style>
                  <Color>FFFFFFFF</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 128 0 0</Margin>
                </Style>
                <ActiveStyle>
                  <Color>FFAAFFCC</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 128 0 0</Margin>
                </ActiveStyle>
                <FontFilename>
                  Components/UserInterface/MenuFont.spritefont
                </FontFilename>
              </Component>
            </Components>
            <Text>Exit Game</Text>
            <Description>Exit the game</Description>
          </MenuItem>
        </MenuItems>

      </GameObject>

      <GameObject Type="UserInterface:TextControlContent">
```

```
        <Name>DescriptionOutput</Name>

        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              Menu menu = Game.GetObject("Game.MainMenu.Menu") as Menu;
              Text = menu.ActiveItem.Description;
              menu.ActiveIndexChanged +=
                delegate(GameObject sender, GameObjectEventArgs e)
                {
                  Text = menu.ActiveItem.Description;
                };

              base.Initialize();
            </Source>
          </Method>

        </Methods>

        <Components>
          <Component Type="Components:UserInterface.XnaTextDataContent">
            <Name>DescriptionTextData</Name>
            <Style>
              <Width>400</Width>
              <Height>100</Height>
              <HorizontalAlignment>Right</HorizontalAlignment>
              <VerticalAlignment>Bottom</VerticalAlignment>
            </Style>
            <FontFilename>Components/UserInterface/TextFont.spritefont</FontFilename>
          </Component>
        </Components>
      </GameObject>
    </GameObjects>
  </Asset>
</XnaContent>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<XnaContent xmlns:Pipeline="GaMEX.Content.Pipeline"
            xmlns:UserInterface="GaMEX.Content.Pipeline.UserInterface"
            xmlns:Gameplay="GaMEX.Content.Pipeline.Gameplay"
            xmlns:Components="GaMEX.Content.Pipeline.Components">
  <Asset Type="Pipeline:GameSegmentContent">
    <Name>Play</Name>

    <Methods>
      <Method Type="Pipeline:CustomOverride">
        <Name>Activate</Name>
        <Source>
          ((Sequence)Game.GetObject("Game.Play.WorldSequence")).Start();

          base.Activate();
        </Source>
      </Method>
    </Methods>

    <Actions>

      <!-- Snake control -->

      <Action>
        <Name>TurnLeft</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Left].StateChanged +=
                  delegate(object sender, XnaKeyEventArgs e)
                  {
                    if (e.Key.State == ButtonState.Pressed)
                      Trigger();
                  };
            </Source>
          </Method>
        </Methods>
      </Action>

      <Action>
        <Name>TurnRight</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Right].StateChanged +=
                delegate(object sender, XnaKeyEventArgs e)
                {
                  if (e.Key.State == ButtonState.Pressed)
                    Trigger();
                };
            </Source>
          </Method>
        </Methods>
      </Action>

      <Action>
        <Name>TurnUp</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Up].StateChanged +=
                  delegate(object sender, XnaKeyEventArgs e)
                  {
```

```
                  if (e.Key.State == ButtonState.Pressed)
                    Trigger();
                };
        </Source>
      </Method>
    </Methods>
  </Action>

  <Action>
    <Name>TurnDown</Name>
    <Methods>
      <Method Type="Pipeline:CustomOverride">
        <Name>Initialize</Name>
        <Source>
          ((XnaKeyboardManager)Game.GetComponent(
            typeof(XnaKeyboardManager))).Buttons[Keys.Down].StateChanged +=
              delegate(object sender, XnaKeyEventArgs e)
              {
                if (e.Key.State == ButtonState.Pressed)
                  Trigger();
              };
        </Source>
      </Method>
    </Methods>
  </Action>

  <!-- Menu control -->

  <Action>
    <Name>ToggleMenu</Name>
    <Methods>
      <Method Type="Pipeline:CustomOverride">
        <Name>Activate</Name>
        <Source>
          ((XnaKeyboardManager)Game.GetComponent(
            typeof(XnaKeyboardManager))).Buttons[Keys.Escape].StateChanged +=
              HandleKeyPressed;
        </Source>
      </Method>
      <Method Type="Pipeline:CustomOverride">
        <Name>Deactivate</Name>
        <Source>
          ((XnaKeyboardManager)Game.GetComponent(
            typeof(XnaKeyboardManager))).Buttons[Keys.Escape].StateChanged -=
              HandleKeyPressed;
        </Source>
      </Method>
      <Method Type="Pipeline:CustomEventHandler">
        <Name>HandleKeyPressed</Name>
        <Source>
          if (args.Key.State == ButtonState.Pressed)
            Trigger();
        </Source>
        <EventOwnerTypeName>
          GaMEX.Components.Input.XnaKey, GaMEX
        </EventOwnerTypeName>
        <EventName>StateChanged</EventName>
      </Method>
    </Methods>
  </Action>


  <Action>
    <Name>Select</Name>
    <Methods>
      <Method Type="Pipeline:CustomOverride">
        <Name>Activate</Name>
        <Source>
          ((XnaKeyboardManager)Game.GetComponent(
            typeof(XnaKeyboardManager))).Buttons[Keys.Enter].StateChanged +=
              HandleButtonPressed;
        </Source>
      </Method>
      <Method Type="Pipeline:CustomOverride">
        <Name>Deactivate</Name>
        <Source>
          ((XnaKeyboardManager)Game.GetComponent(
            typeof(XnaKeyboardManager))).Buttons[Keys.Enter].StateChanged -=
              HandleButtonPressed;
        </Source>
      </Method>
      <Method Type="Pipeline:CustomEventHandler">
        <Name>HandleButtonPressed</Name>
        <Source>
          if (args.Key.State == ButtonState.Pressed)
            Trigger();
        </Source>
        <EventOwnerTypeName>
          GaMEX.Components.Input.XnaKey, GaMEX
        </EventOwnerTypeName>
        <EventName>StateChanged</EventName>
      </Method>
    </Methods>
  </Action>

  <Action>
    <Name>NavigateUp</Name>
    <Methods>
      <Method Type="Pipeline:CustomOverride">
        <Name>Activate</Name>
        <Source>
          ((XnaKeyboardManager)Game.GetComponent(
            typeof(XnaKeyboardManager))).Buttons[Keys.Up].StateChanged +=
              HandleKeyPressed;
        </Source>
      </Method>
      <Method Type="Pipeline:CustomOverride">
        <Name>Deactivate</Name>
        <Source>
          ((XnaKeyboardManager)Game.GetComponent(
            typeof(XnaKeyboardManager))).Buttons[Keys.Up].StateChanged -=
              HandleKeyPressed;
        </Source>
      </Method>
      <Method Type="Pipeline:CustomEventHandler">
        <Name>HandleKeyPressed</Name>
        <Source>
          if (args.Key.State == ButtonState.Pressed)
            Trigger();
        </Source>
        <EventOwnerTypeName>
          GaMEX.Components.Input.XnaKey, GaMEX
        </EventOwnerTypeName>
        <EventName>StateChanged</EventName>
      </Method>
    </Methods>
  </Action>

  <Action>
    <Name>NavigateDown</Name>
    <Methods>
      <Method Type="Pipeline:CustomOverride">
        <Name>Activate</Name>
        <Source>
          ((XnaKeyboardManager)Game.GetComponent(
            typeof(XnaKeyboardManager))).Buttons[Keys.Down].StateChanged +=
              HandleKeyPressed;
        </Source>
      </Method>
      <Method Type="Pipeline:CustomOverride">
        <Name>Deactivate</Name>
        <Source>
          ((XnaKeyboardManager)Game.GetComponent(
            typeof(XnaKeyboardManager))).Buttons[Keys.Down].StateChanged -=
              HandleKeyPressed;
        </Source>
      </Method>
      <Method Type="Pipeline:CustomEventHandler">
        <Name>HandleKeyPressed</Name>
        <Source>
          if (args.Key.State == ButtonState.Pressed)
            Trigger();
        </Source>
        <EventOwnerTypeName>
          GaMEX.Components.Input.XnaKey, GaMEX
        </EventOwnerTypeName>
        <EventName>StateChanged</EventName>
      </Method>
    </Methods>
  </Action>

</Actions>

<GameObjects>
  <GameObject Type="UserInterface:MenuContent">
    <Name>InGameMenu</Name>

    <Methods>
      <Method Type="Pipeline:CustomOverride">
        <Name>Initialize</Name>
        <Source>
          Game.GetActionTrigger("Game.Play.ToggleMenu").Triggered +=
            delegate(object sender, GameEventArgs e)
            {
              if (IsEnabled)
                Deactivate(true);
              else
                Activate();
            };

          Game.GetObject&lt;MenuItem&gt;(
            "Game.Play.InGameMenu.Resume").Selected +=
              delegate(GameObject sender, GameObjectEventArgs e)
              {
                Deactivate(true);
              };

          Game.GetObject&lt;MenuItem&gt;(
            "Game.Play.InGameMenu.EndGame").Selected +=
              delegate(GameObject sender, GameObjectEventArgs e)
              {
                Game.GetObject&lt;Sequence&gt;("Game.Play.WorldSequence").Stop();
                Deactivate(true);

                Game.LoadSegment("Game.MainMenu");
              };

          base.Initialize();
        </Source>
      </Method>
      <Method Type="Pipeline:CustomOverride">
        <Name>Activate</Name>
        <Source>
          Game.GetActionTrigger("Game.Play.Select").Triggered += HandleSelect;
          Game.GetActionTrigger("Game.Play.NavigateUp").Triggered +=
            HandleNavigateUp;
          Game.GetActionTrigger("Game.Play.NavigateDown").Triggered +=
            HandleNavigateDown;

          base.Activate();
        </Source>
      </Method>
      <Method Type="Pipeline:CustomOverride">
        <Name>Deactivate</Name>
        <Source>
          Game.GetActionTrigger("Game.Play.Select").Triggered -= HandleSelect;
          Game.GetActionTrigger("Game.Play.NavigateUp").Triggered -=
            HandleNavigateUp;
          Game.GetActionTrigger("Game.Play.NavigateDown").Triggered -=
            HandleNavigateDown;

          base.Deactivate(disable);
        </Source>
        <ParameterTypeNames>
          <ParameterTypeName>System.Boolean</ParameterTypeName>
        </ParameterTypeNames>
      </Method>
      <Method Type="Pipeline:CustomEventHandler">
        <Name>HandleSelect</Name>
        <Source>
          MenuItems[ActiveIndex].Select();
        </Source>
        <EventOwnerTypeName>GaMEX.GameActionTrigger, GaMEX</EventOwnerTypeName>
        <EventName>Triggered</EventName>
      </Method>
      <Method Type="Pipeline:CustomEventHandler">
        <Name>HandleNavigateUp</Name>
        <Source>
          if (ActiveIndex == 0)
            ActiveIndex = MenuItems.Count - 1;
          else
            ActiveIndex -= 1;
        </Source>
        <EventOwnerTypeName>GaMEX.GameActionTrigger, GaMEX</EventOwnerTypeName>
        <EventName>Triggered</EventName>
      </Method>
      <Method Type="Pipeline:CustomEventHandler">
        <Name>HandleNavigateDown</Name>
        <Source>
          if (ActiveIndex == MenuItems.Count - 1)
            ActiveIndex = 0;
          else
            ActiveIndex += 1;
        </Source>
        <EventOwnerTypeName>GaMEX.GameActionTrigger, GaMEX</EventOwnerTypeName>
        <EventName>Triggered</EventName>
      </Method>
    </Methods>

    <Components>
      <Component Type="Components:UserInterface.XnaControlDataContent">
        <Name>InGameMenuControlData</Name>
        <Style>
          <Width>300</Width>
          <Height>300</Height>
          <HorizontalAlignment>Center</HorizontalAlignment>
          <VerticalAlignment>Center</VerticalAlignment>
          <Padding>10 10 10 10</Padding>
          <Margin>0 0 0 0</Margin>
        </Style>
      </Component>
    </Components>

    <MenuItems>

      <MenuItem Type="UserInterface:MenuItemContent">
```

```
            <Name>Resume</Name>
            <Components>
              <Component Type="Components:UserInterface.XnaMenuItemDataContent">
                <Name>ResumeItemData</Name>
                <Style>
                  <Color>FFFFFFFF</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 0 0 0</Margin>
                </Style>
                <ActiveStyle>
                  <Color>FFF96B4F</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 0 0 0</Margin>
                </ActiveStyle>
                <FontFilename>
                  Components/UserInterface/MenuFont.spritefont
                </FontFilename>
              </Component>
            </Components>
            <Text>Resume</Text>
          </MenuItem>

          <MenuItem Type="UserInterface:MenuItemContent">
            <Name>EndGame</Name>
            <Components>
              <Component Type="Components:UserInterface.XnaMenuItemDataContent">
                <Name>EndGameItemData</Name>
                <Style>
                  <Color>FFFFFFFF</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 64 0 0</Margin>
                </Style>
                <ActiveStyle>
                  <Color>FFF96B4F</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 64 0 0</Margin>
                </ActiveStyle>
                <FontFilename>
                  Components/UserInterface/MenuFont.spritefont
                </FontFilename>
              </Component>
            </Components>
            <Text>End Game</Text>
          </MenuItem>
        </MenuItems>
      </GameObject>

      <GameObject Type="Gameplay:Time.TimerContent">
        <Name>PlayTimer</Name>
        <TicksPerSecond>1.0</TicksPerSecond>
      </GameObject>

      <GameObject Type="Gameplay:Time.TimerContent">
        <Name>MoveTimer</Name>
      </GameObject>

      <GameObject Type="Gameplay:SequenceContent">
        <Name>WorldSequence</Name>

        <Steps>
          <Step Type="Gameplay:ContainerSequenceStepContent">
```

```
            <Name>Step1</Name>

            <ContentNames>
              <ContentName>Game.Play.WorldSequence.Step1.World</ContentName>
              <ContentName>Game.Play.WorldSequence.Step1.FoodSequence</ContentName>
            </ContentNames>
          </Step>
        </Steps>
      </GameObject>
    </GameObjects>
  </Asset>
</XnaContent>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<XnaContent xmlns:Pipeline="GaMEX.Content.Pipeline">
  <Asset Type="Pipeline:Gameplay.RandomSequenceContent">
    <Name>FoodSequence</Name>
    <Namespace>Game.Play.WorldSequence.Step1</Namespace>

    <Methods>
      <Method Type="Pipeline:CustomOverride">
        <Name>Initialize</Name>
        <Source>
          Game.GetObject&lt;World&gt;(
            "Game.Play.WorldSequence.Step1.World").Loaded +=
              delegate(object sender, GameEventArgs e) { Start(); };

          base.Initialize();
        </Source>
      </Method>
    </Methods>

    <Steps>
      <Step Type="Pipeline:Gameplay.TimedSequenceStepContent">
        <Name>Step1</Name>
        <Duration>PT20S</Duration>
      </Step>

      <Step Type="Pipeline:Gameplay.TimedSequenceStepContent">
        <Name>Step2</Name>
        <Duration>PT20S</Duration>
      </Step>

      <Step Type="Pipeline:Gameplay.TimedSequenceStepContent">
        <Name>Step3</Name>
        <Duration>PT20S</Duration>
      </Step>

      <Step Type="Pipeline:Gameplay.TimedSequenceStepContent">
        <Name>Step4</Name>
        <Duration>PT20S</Duration>
      </Step>

      <Step Type="Pipeline:Gameplay.TimedSequenceStepContent">
        <Name>Step5</Name>
        <Duration>PT20S</Duration>
      </Step>

      <Step Type="Pipeline:Gameplay.TimedSequenceStepContent">
        <Name>Step6</Name>
        <Duration>PT15S</Duration>
      </Step>

      <Step Type="Pipeline:Gameplay.TimedSequenceStepContent">
        <Name>Step7</Name>
        <Duration>PT15S</Duration>
      </Step>

      <Step Type="Pipeline:Gameplay.TimedSequenceStepContent">
        <Name>Step8</Name>
        <Duration>PT15S</Duration>
      </Step>

      <Step Type="Pipeline:Gameplay.TimedSequenceStepContent">
        <Name>Step9</Name>
        <Duration>PT15S</Duration>
      </Step>

      <Step Type="Pipeline:Gameplay.TimedSequenceStepContent">
        <Name>Step10</Name>
        <Duration>PT12S</Duration>
      </Step>
      <Step Type="Pipeline:Gameplay.TimedSequenceStepContent">
        <Name>Step11</Name>
        <Duration>PT12S</Duration>
      </Step>

      <Step Type="Pipeline:Gameplay.TimedSequenceStepContent">
        <Name>Step12</Name>
        <Duration>PT12S</Duration>
      </Step>

      <Step Type="Pipeline:Gameplay.TimedSequenceStepContent">
        <Name>Step13</Name>
        <Duration>PT10S</Duration>
      </Step>

      <Step Type="Pipeline:Gameplay.TimedSequenceStepContent">
        <Name>Step14</Name>
        <Duration>PT10S</Duration>
      </Step>
    </Steps>
  </Asset>
</XnaContent>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<XnaContent xmlns:Components="GaMEX.Components"
    xmlns:Pipeline="GaMEX.Content.Pipeline"
    xmlns:GamePipeline="Game.Content.Pipeline"
    xmlns:XnaPipeline="Microsoft.Xna.Framework.Content.Pipeline"
    xmlns:XnaGraphicsPipeline="Microsoft.Xna.Framework.Content.Pipeline.Graphics">
  <Asset Type="GamePipeline:Play.WorldContent">
    <Name>World</Name>
    <Namespace>Game.Play.WorldSequence.Step1</Namespace>
    <BaseContentName>Game.Play.World</BaseContentName>
    <RuntimeAssemblies>
      <Assembly Type="System.String">Game.Play.World</Assembly>
    </RuntimeAssemblies>
    <PipelineAssemblies>
      <Assembly Type="System.String">Game.Content.Pipeline.Play.World</Assembly>
    </PipelineAssemblies>

    <Methods>
      <Method Type="Pipeline:CustomOverride">
        <Name>Load</Name>
        <Source>
          base.Load();


        </Source>
      </Method>
    </Methods>

    <WorldObjects>
      <WorldObject Type="GamePipeline:Play.SnakeContent">
        <Name>Snake</Name>
        <RuntimeAssemblies>
          <Assembly>Game.Play.Snake</Assembly>
        </RuntimeAssemblies>
        <PipelineAssemblies>
          <Assembly>Game.Content.Pipeline.Play.Snake</Assembly>
        </PipelineAssemblies>

        <Components>
          <Component Type="GamePipeline:Play.Snake.SnakeWorldDataContent">
            <Name>SnakeWorldData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Snake</Assembly>
              <Assembly>Game.Play.Snake.SnakeWorldData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Snake.SnakeWorldData</Assembly>
            </PipelineAssemblies>

            <Methods>
              <Method Type="Pipeline:CustomOverride">
                <Name>Initialize</Name>
                <Source>
                  ((SnakeObject)ParentObject).Spawned +=
                    delegate(GameObject sender, GameObjectEventArgs e)
                    {
                      ((XnaGridChain)Structure).SetPositions(
                        new XnaGridPosition(15, 9),
                        new XnaGridPosition(16, 9),
                        new XnaGridPosition(17, 9));
                    };

                  base.Initialize();
                </Source>
              </Method>
            </Methods>

            <Structure Type="Components:GameWorld.XnaGridChain" />
          </Component>

          <Component Type="GamePipeline:Play.Snake.SnakeSpriteDataContent">
            <Name>SnakeSpriteData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Snake</Assembly>
              <Assembly>Game.Play.Snake.SnakeSpriteData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Snake.SnakeSpriteData</Assembly>
            </PipelineAssemblies>

            <TextureReference Type=
                "XnaPipeline:ExternalReference[XnaGraphicsPipeline:TextureContent]">
              <Reference>#External1</Reference>
            </TextureReference>
          </Component>
        </Components>

        <Direction>Left</Direction>
        <Length>1</Length>
        <Speed>6</Speed>
      </WorldObject>

      <WorldObject Type="GamePipeline:Play.FoodContent">
        <Name>Food1</Name>

        <RuntimeAssemblies>
          <Assembly>Game.Play.Food</Assembly>
        </RuntimeAssemblies>
        <PipelineAssemblies>
          <Assembly>Game.Content.Pipeline.Play.Food</Assembly>
        </PipelineAssemblies>

        <Components>
          <Component Type="GamePipeline:Play.Food.FoodWorldDataContent">
            <Name>FoodWorldData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodWorldData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodWorldData</Assembly>
            </PipelineAssemblies>

            <Structure Type="Components:GameWorld.XnaGridCell">
              <Position>18 15</Position>
            </Structure>
          </Component>

          <Component Type="GamePipeline:Play.Food.FoodSpriteDataContent">
            <Name>FoodSpriteData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodSpriteData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodSpriteData</Assembly>
            </PipelineAssemblies>

            <TextureReference Type="
                XnaPipeline:ExternalReference[XnaGraphicsPipeline:TextureContent]">
              <Reference>#External2</Reference>
            </TextureReference>
          </Component>
        </Components>

        <Value>1</Value>
        <FoodSequenceStepName>
          Game.Play.WorldSequence.Step1.FoodSequence.Step1
        </FoodSequenceStepName>
      </WorldObject>

      <WorldObject Type="GamePipeline:Play.FoodContent">
        <Name>Food2</Name>

        <RuntimeAssemblies>
          <Assembly>Game.Play.Food</Assembly>
        </RuntimeAssemblies>
        <PipelineAssemblies>
          <Assembly>Game.Content.Pipeline.Play.Food</Assembly>
        </PipelineAssemblies>

        <Components>
          <Component Type="GamePipeline:Play.Food.FoodWorldDataContent">
            <Name>FoodWorldData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodWorldData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodWorldData</Assembly>
            </PipelineAssemblies>

            <Structure Type="Components:GameWorld.XnaGridCell">
              <Position>19 16</Position>
            </Structure>
          </Component>

          <Component Type="GamePipeline:Play.Food.FoodSpriteDataContent">
            <Name>FoodSpriteData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodSpriteData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodSpriteData</Assembly>
            </PipelineAssemblies>

            <TextureReference Type=
                "XnaPipeline:ExternalReference[XnaGraphicsPipeline:TextureContent]">
              <Reference>#External2</Reference>
            </TextureReference>
          </Component>
        </Components>

        <Value>1</Value>
        <FoodSequenceStepName>
          Game.Play.WorldSequence.Step1.FoodSequence.Step2
        </FoodSequenceStepName>
      </WorldObject>

      <WorldObject Type="GamePipeline:Play.FoodContent">
        <Name>Food3</Name>

        <RuntimeAssemblies>
          <Assembly>Game.Play.Food</Assembly>
        </RuntimeAssemblies>
        <PipelineAssemblies>
          <Assembly>Game.Content.Pipeline.Play.Food</Assembly>
        </PipelineAssemblies>

        <Components>
          <Component Type="GamePipeline:Play.Food.FoodWorldDataContent">
            <Name>FoodWorldData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodWorldData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodWorldData</Assembly>
            </PipelineAssemblies>

            <Structure Type="Components:GameWorld.XnaGridCell">
              <Position>14 15</Position>
            </Structure>
          </Component>

          <Component Type="GamePipeline:Play.Food.FoodSpriteDataContent">
            <Name>FoodSpriteData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodSpriteData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodSpriteData</Assembly>
            </PipelineAssemblies>

            <TextureReference Type="
              XnaPipeline:ExternalReference[XnaGraphicsPipeline:TextureContent]">
              <Reference>#External2</Reference>
            </TextureReference>
          </Component>
        </Components>

        <Value>1</Value>
        <FoodSequenceStepName>
          Game.Play.WorldSequence.Step1.FoodSequence.Step3
        </FoodSequenceStepName>
      </WorldObject>

      <WorldObject Type="GamePipeline:Play.FoodContent">
        <Name>Food4</Name>

        <RuntimeAssemblies>
          <Assembly>Game.Play.Food</Assembly>
        </RuntimeAssemblies>
        <PipelineAssemblies>
          <Assembly>Game.Content.Pipeline.Play.Food</Assembly>
        </PipelineAssemblies>

        <Components>
          <Component Type="GamePipeline:Play.Food.FoodWorldDataContent">
            <Name>FoodWorldData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodWorldData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodWorldData</Assembly>
            </PipelineAssemblies>

            <Structure Type="Components:GameWorld.XnaGridCell">
              <Position>17 16</Position>
            </Structure>
          </Component>

          <Component Type="GamePipeline:Play.Food.FoodSpriteDataContent">
            <Name>FoodSpriteData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodSpriteData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodSpriteData</Assembly>
            </PipelineAssemblies>
```

```
            <TextureReference Type=
                "XnaPipeline:ExternalReference[XnaGraphicsPipeline:TextureContent]">
              <Reference>#External2</Reference>
            </TextureReference>
          </Component>
        </Components>

        <Value>1</Value>
        <FoodSequenceStepName>
          Game.Play.WorldSequence.Step1.FoodSequence.Step4
        </FoodSequenceStepName>
      </WorldObject>

      <WorldObject Type="GamePipeline:Play.FoodContent">
        <Name>Food5</Name>

        <RuntimeAssemblies>
          <Assembly>Game.Play.Food</Assembly>
        </RuntimeAssemblies>
        <PipelineAssemblies>
          <Assembly>Game.Content.Pipeline.Play.Food</Assembly>
        </PipelineAssemblies>

        <Components>
          <Component Type="GamePipeline:Play.Food.FoodWorldDataContent">
            <Name>FoodWorldData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodWorldData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodWorldData</Assembly>
            </PipelineAssemblies>

            <Structure Type="Components:GameWorld.XnaGridCell">
              <Position>16 15</Position>
            </Structure>
          </Component>

          <Component Type="GamePipeline:Play.Food.FoodSpriteDataContent">
            <Name>FoodSpriteData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodSpriteData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodSpriteData</Assembly>
            </PipelineAssemblies>

            <TextureReference Type=
                "XnaPipeline:ExternalReference[XnaGraphicsPipeline:TextureContent]">
              <Reference>#External2</Reference>
            </TextureReference>
          </Component>
        </Components>

        <Value>1</Value>
        <FoodSequenceStepName>
          Game.Play.WorldSequence.Step1.FoodSequence.Step5
        </FoodSequenceStepName>
      </WorldObject>

      <WorldObject Type="GamePipeline:Play.FoodContent">
        <Name>Food6</Name>

        <RuntimeAssemblies>
          <Assembly>Game.Play.Food</Assembly>
        </RuntimeAssemblies>
        <PipelineAssemblies>
          <Assembly>Game.Content.Pipeline.Play.Food</Assembly>
        </PipelineAssemblies>

        <Components>
          <Component Type="GamePipeline:Play.Food.FoodWorldDataContent">
            <Name>FoodWorldData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodWorldData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodWorldData</Assembly>
            </PipelineAssemblies>

            <Structure Type="Components:GameWorld.XnaGridCell">
              <Position>22 21</Position>
            </Structure>
          </Component>

          <Component Type="GamePipeline:Play.Food.FoodSpriteDataContent">
            <Name>FoodSpriteData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodSpriteData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodSpriteData</Assembly>
            </PipelineAssemblies>

            <TextureReference Type=
                "XnaPipeline:ExternalReference[XnaGraphicsPipeline:TextureContent]">
              <Reference>#External2</Reference>
            </TextureReference>
          </Component>
        </Components>

        <Value>2</Value>
        <FoodSequenceStepName>
          Game.Play.WorldSequence.Step1.FoodSequence.Step6
        </FoodSequenceStepName>
      </WorldObject>

      <WorldObject Type="GamePipeline:Play.FoodContent">
        <Name>Food7</Name>

        <RuntimeAssemblies>
          <Assembly>Game.Play.Food</Assembly>
        </RuntimeAssemblies>
        <PipelineAssemblies>
          <Assembly>Game.Content.Pipeline.Play.Food</Assembly>
        </PipelineAssemblies>

        <Components>
          <Component Type="GamePipeline:Play.Food.FoodWorldDataContent">
            <Name>FoodWorldData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodWorldData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodWorldData</Assembly>
            </PipelineAssemblies>

            <Structure Type="Components:GameWorld.XnaGridCell">
              <Position>21 13</Position>
            </Structure>
          </Component>
          <Component Type="GamePipeline:Play.Food.FoodSpriteDataContent">
            <Name>FoodSpriteData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodSpriteData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodSpriteData</Assembly>
            </PipelineAssemblies>

            <TextureReference Type=
                "XnaPipeline:ExternalReference[XnaGraphicsPipeline:TextureContent]">
              <Reference>#External2</Reference>
            </TextureReference>
          </Component>
        </Components>

        <Value>2</Value>
        <FoodSequenceStepName>
          Game.Play.WorldSequence.Step1.FoodSequence.Step7
        </FoodSequenceStepName>
      </WorldObject>

      <WorldObject Type="GamePipeline:Play.FoodContent">
        <Name>Food8</Name>

        <RuntimeAssemblies>
          <Assembly>Game.Play.Food</Assembly>
        </RuntimeAssemblies>
        <PipelineAssemblies>
          <Assembly>Game.Content.Pipeline.Play.Food</Assembly>
        </PipelineAssemblies>

        <Components>
          <Component Type="GamePipeline:Play.Food.FoodWorldDataContent">
            <Name>FoodWorldData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodWorldData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodWorldData</Assembly>
            </PipelineAssemblies>

            <Structure Type="Components:GameWorld.XnaGridCell">
              <Position>14 12</Position>
            </Structure>
          </Component>

          <Component Type="GamePipeline:Play.Food.FoodSpriteDataContent">
            <Name>FoodSpriteData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodSpriteData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodSpriteData</Assembly>
            </PipelineAssemblies>

            <TextureReference Type=
                "XnaPipeline:ExternalReference[XnaGraphicsPipeline:TextureContent]">
              <Reference>#External2</Reference>
            </TextureReference>
          </Component>
        </Components>

        <Value>2</Value>
        <FoodSequenceStepName>
          Game.Play.WorldSequence.Step1.FoodSequence.Step8
        </FoodSequenceStepName>
      </WorldObject>

      <WorldObject Type="GamePipeline:Play.FoodContent">
        <Name>Food9</Name>

        <RuntimeAssemblies>
          <Assembly>Game.Play.Food</Assembly>
        </RuntimeAssemblies>
        <PipelineAssemblies>
          <Assembly>Game.Content.Pipeline.Play.Food</Assembly>
        </PipelineAssemblies>

        <Components>
          <Component Type="GamePipeline:Play.Food.FoodWorldDataContent">
            <Name>FoodWorldData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodWorldData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodWorldData</Assembly>
            </PipelineAssemblies>

            <Structure Type="Components:GameWorld.XnaGridCell">
              <Position>13 20</Position>
            </Structure>
          </Component>

          <Component Type="GamePipeline:Play.Food.FoodSpriteDataContent">
            <Name>FoodSpriteData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodSpriteData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodSpriteData</Assembly>
            </PipelineAssemblies>

            <TextureReference Type=
                "XnaPipeline:ExternalReference[XnaGraphicsPipeline:TextureContent]">
              <Reference>#External2</Reference>
            </TextureReference>
          </Component>
        </Components>

        <Value>2</Value>
        <FoodSequenceStepName>
          Game.Play.WorldSequence.Step1.FoodSequence.Step9
        </FoodSequenceStepName>
      </WorldObject>

      <WorldObject Type="GamePipeline:Play.FoodContent">
        <Name>Food10</Name>

        <RuntimeAssemblies>
          <Assembly>Game.Play.Food</Assembly>
        </RuntimeAssemblies>
        <PipelineAssemblies>
          <Assembly>Game.Content.Pipeline.Play.Food</Assembly>
        </PipelineAssemblies>

        <Components>
          <Component Type="GamePipeline:Play.Food.FoodWorldDataContent">
            <Name>FoodWorldData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodWorldData</Assembly>
            </RuntimeAssemblies>
```

```
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodWorldData</Assembly>
            </PipelineAssemblies>

            <Structure Type="Components:GameWorld.XnaGridCell">
              <Position>26 17</Position>
            </Structure>
          </Component>

          <Component Type="GamePipeline:Play.Food.FoodSpriteDataContent">
            <Name>FoodSpriteData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodSpriteData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodSpriteData</Assembly>
            </PipelineAssemblies>

            <TextureReference Type=
                "XnaPipeline:ExternalReference[XnaGraphicsPipeline:TextureContent]">
              <Reference>#External2</Reference>
            </TextureReference>
          </Component>
        </Components>

        <Value>3</Value>
        <FoodSequenceStepName>
          Game.Play.WorldSequence.Step1.FoodSequence.Step10
        </FoodSequenceStepName>
      </WorldObject>

      <WorldObject Type="GamePipeline:Play.FoodContent">
        <Name>Food11</Name>

        <RuntimeAssemblies>
          <Assembly>Game.Play.Food</Assembly>
        </RuntimeAssemblies>
        <PipelineAssemblies>
          <Assembly>Game.Content.Pipeline.Play.Food</Assembly>
        </PipelineAssemblies>

        <Components>
          <Component Type="GamePipeline:Play.Food.FoodWorldDataContent">
            <Name>FoodWorldData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodWorldData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodWorldData</Assembly>
            </PipelineAssemblies>

            <Structure Type="Components:GameWorld.XnaGridCell">
              <Position>7 5</Position>
            </Structure>
          </Component>

          <Component Type="GamePipeline:Play.Food.FoodSpriteDataContent">
            <Name>FoodSpriteData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodSpriteData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodSpriteData</Assembly>
            </PipelineAssemblies>

            <TextureReference Type=
                "XnaPipeline:ExternalReference[XnaGraphicsPipeline:TextureContent]">
              <Reference>#External2</Reference>
            </TextureReference>
          </Component>
        </Components>

        <Value>3</Value>
        <FoodSequenceStepName>
          Game.Play.WorldSequence.Step1.FoodSequence.Step11
        </FoodSequenceStepName>
      </WorldObject>

      <WorldObject Type="GamePipeline:Play.FoodContent">
        <Name>Food12</Name>

        <RuntimeAssemblies>
          <Assembly>Game.Play.Food</Assembly>
        </RuntimeAssemblies>
        <PipelineAssemblies>
          <Assembly>Game.Content.Pipeline.Play.Food</Assembly>
        </PipelineAssemblies>

        <Components>
          <Component Type="GamePipeline:Play.Food.FoodWorldDataContent">
            <Name>FoodWorldData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodWorldData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodWorldData</Assembly>
            </PipelineAssemblies>

            <Structure Type="Components:GameWorld.XnaGridCell">
              <Position>13 22</Position>
            </Structure>
          </Component>

          <Component Type="GamePipeline:Play.Food.FoodSpriteDataContent">
            <Name>FoodSpriteData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodSpriteData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodSpriteData</Assembly>
            </PipelineAssemblies>

            <TextureReference Type=
                "XnaPipeline:ExternalReference[XnaGraphicsPipeline:TextureContent]">
              <Reference>#External2</Reference>
            </TextureReference>
          </Component>
        </Components>

        <Value>3</Value>
        <FoodSequenceStepName>
          Game.Play.WorldSequence.Step1.FoodSequence.Step12
        </FoodSequenceStepName>
      </WorldObject>

      <WorldObject Type="GamePipeline:Play.FoodContent">
        <Name>Food13</Name>

        <RuntimeAssemblies>
          <Assembly>Game.Play.Food</Assembly>
        </RuntimeAssemblies>
        <PipelineAssemblies>
          <Assembly>Game.Content.Pipeline.Play.Food</Assembly>
        </PipelineAssemblies>

        <Components>
          <Component Type="GamePipeline:Play.Food.FoodWorldDataContent">
            <Name>FoodWorldData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodWorldData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodWorldData</Assembly>
            </PipelineAssemblies>

            <Structure Type="Components:GameWorld.XnaGridCell">
              <Position>30 17</Position>
            </Structure>
          </Component>

          <Component Type="GamePipeline:Play.Food.FoodSpriteDataContent">
            <Name>FoodSpriteData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodSpriteData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodSpriteData</Assembly>
            </PipelineAssemblies>

            <TextureReference Type=
                "XnaPipeline:ExternalReference[XnaGraphicsPipeline:TextureContent]">
              <Reference>#External2</Reference>
            </TextureReference>
          </Component>
        </Components>

        <Value>4</Value>
        <FoodSequenceStepName>
          Game.Play.WorldSequence.Step1.FoodSequence.Step13
        </FoodSequenceStepName>
      </WorldObject>

      <WorldObject Type="GamePipeline:Play.FoodContent">
        <Name>Food14</Name>

        <RuntimeAssemblies>
          <Assembly>Game.Play.Food</Assembly>
        </RuntimeAssemblies>
        <PipelineAssemblies>
          <Assembly>Game.Content.Pipeline.Play.Food</Assembly>
        </PipelineAssemblies>

        <Components>
          <Component Type="GamePipeline:Play.Food.FoodWorldDataContent">
            <Name>FoodWorldData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodWorldData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodWorldData</Assembly>
            </PipelineAssemblies>

            <Structure Type="Components:GameWorld.XnaGridCell">
              <Position>3 20</Position>
            </Structure>
          </Component>

          <Component Type="GamePipeline:Play.Food.FoodSpriteDataContent">
            <Name>FoodSpriteData</Name>
            <RuntimeAssemblies>
              <Assembly>Game.Play.Food.FoodSpriteData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Food.FoodSpriteData</Assembly>
            </PipelineAssemblies>

            <TextureReference Type=
                "XnaPipeline:ExternalReference[XnaGraphicsPipeline:TextureContent]">
              <Reference>#External2</Reference>
            </TextureReference>
          </Component>
        </Components>

        <Value>4</Value>
        <FoodSequenceStepName>
          Game.Play.WorldSequence.Step1.FoodSequence.Step14
        </FoodSequenceStepName>
      </WorldObject>
    </WorldObjects>
  </Asset>

  <ExternalReferences>
    <ExternalReference ID="#External1"
        TargetType="XnaGraphicsPipeline:TextureContent">
      Components/Graphics/SnakeSprite.png
    </ExternalReference>

    <ExternalReference ID="#External2"
        TargetType="XnaGraphicsPipeline:TextureContent">
      Components/Graphics/FoodSprite.png
    </ExternalReference>
  </ExternalReferences>
</XnaContent>
```

Appendix B

FPS experiment

This appendix contain all the XML source files created for the FPS experiment. Like with the previous experiment, the data files contains embedded C# logic that serves as the scripting language. The same structure is used in this appendix as well by starting the listing the base game object types in the game, and go on to the segments and the game objects.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<XnaContent xmlns:Pipeline="GaMEX.Content.Pipeline"
    xmlns:XnaPipeline="Microsoft.Xna.Framework.Content.Pipeline"
    mlns:XnaGraphicsPipeline="Microsoft.Xna.Framework.Content.Pipeline.Graphics">
  <Asset Type="GaMEX.Content.Pipeline.GameWorld.WorldObjectContent">
    <Name>Player</Name>
    <Namespace>Game.Play</Namespace>
    <BaseTypeWriterName>
      GaMEX.Content.Pipeline.Serialization.WorldObjectWriter, GaMEX.Content.Pipeline
    </BaseTypeWriterName>
    <GeneratePipelineCode>true</GeneratePipelineCode>

    <Methods>
      <Method Type="Pipeline:CustomOverride">
        <Name>Initialize</Name>
        <Source>
          World world = ParentContainer as World;
          world.Loaded += delegate(object sender, GameEventArgs e)
          {
            Spawn();
          };

          world.Unloaded += delegate(object sender, GameEventArgs e)
          {
            Remove();
          };

          base.Initialize();
        </Source>
      </Method>


    </Methods>
    <Components>
      <Component Type="Pipeline:Components.Graphics.XnaFirstPersonCameraDataContent">
        <Name>PlayerCameraData</Name>
        <BaseTypeWriterName>
          GaMEX.Content.Pipeline.Serialization.Components.XnaFirstPersonCameraDataWrit
er,
          GaMEX.Content.Pipeline
        </BaseTypeWriterName>
        <GeneratePipelineCode>true</GeneratePipelineCode>

        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              base.Initialize();
            </Source>
          </Method>
        </Methods>
      </Component>
    </Components>

  </Asset>
  <ExternalReferences>

  </ExternalReferences>
</XnaContent>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<XnaContent xmlns:Pipeline="GaMEX.Content.Pipeline"
    xmlns:XnaGraphicsPipeline="Microsoft.Xna.Framework.Content.Pipeline.Graphics">
  <Asset Type="Pipeline:GameWorld.WorldContent">
    <Name>World</Name>
    <Namespace>Game.Play</Namespace>
    <BaseTypeWriterName>
      GaMEX.Content.Pipeline.Serialization.WorldWriter, GaMEX.Content.Pipeline
    </BaseTypeWriterName>
    <GeneratePipelineCode>true</GeneratePipelineCode>

    <Properties>
      <Property>
        <Name>PlayerName</Name>
        <TypeName>string</TypeName>
      </Property>
    </Properties>

    <Methods>
      <Method Type="Pipeline:CustomOverride">
        <Name>Initialize</Name>
        <Source>
          SequenceStep parentStep = ParentContainer as SequenceStep;

          parentStep.Entered +=
            delegate(GameObject sender, GameObjectEventArgs e) { Load(); };
          parentStep.Exited +=
            delegate(GameObject sender, GameObjectEventArgs e) { Unload(); };

          base.Initialize();
        </Source>
      </Method>
    </Methods>

    <Components>
      <Component Type="Pipeline:Components.Graphics.XnaModelDataContent">
        <Name>WorldModelData</Name>
        <BaseTypeWriterName>
          GaMEX.Content.Pipeline.Serialization.Components.XnaModelDataWriter,
          GaMEX.Content.Pipeline
        </BaseTypeWriterName>
        <GeneratePipelineCode>true</GeneratePipelineCode>

        <RuntimeAssemblies>
          <Assembly>Game.Play.World</Assembly>
        </RuntimeAssemblies>

        <Properties>
          <Property>
            <Name>PlayerCameraData</Name>
            <TypeName>XnaFirstPersonCameraData</TypeName>
          </Property>

          <Property>
            <Name>PlayerWorldData</Name>
            <TypeName>XnaWorldObjectSceneData</TypeName>
          </Property>
        </Properties>

        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              GameObject player = Game.GetObject(
                ((Game.Play.WorldObject)ParentObject).PlayerName);

              PlayerCameraData =
                (XnaFirstPersonCameraData)player.GetComponent("PlayerCameraData");
              PlayerWorldData =
                (XnaWorldObjectSceneData)player.GetComponent("PlayerWorldData");

              base.Initialize();
            </Source>
          </Method>

          <Method Type="Pipeline:CustomOverride">
            <Name>Update</Name>
            <Source>
              WorldMatrix = WorldMatrix * Matrix.Invert(PlayerWorldData.WorldMatrix);
              ViewMatrix = PlayerCameraData.ViewMatrix;
              ProjectionMatrix = PlayerCameraData.ProjectionMatrix;

              base.Update(gameTime);
            </Source>
          </Method>
        </Methods>
      </Component>
    </Components>
  </Asset>
</XnaContent>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<XnaContent xmlns:Pipeline="GaMEX.Content.Pipeline"
            xmlns:UserInterface="GaMEX.Content.Pipeline.UserInterface"
            xmlns:Gameplay="GaMEX.Content.Pipeline.Gameplay"
            xmlns:Components="GaMEX.Content.Pipeline.Components">
  <Asset Type="Pipeline:GameSegmentContent">
    <Name>SplashScreen</Name>

    <Methods>
      <Method Type="Pipeline:CustomOverride">
        <Name>Activate</Name>
        <Source>
          Game.GetActionTrigger("Game.SplashScreen.SkipSplash").Triggered +=
            delegate(object sender, GameEventArgs e)
              { Game.LoadSegment("Game.MainMenu"); };

          ((Sequence)Game.GetObject("Game.SplashScreen.SplashSequence")).Completed +=
            delegate(GameObject sender, GameObjectEventArgs args)
              { Game.LoadSegment("Game.MainMenu"); };

          base.Activate();
        </Source>
      </Method>
    </Methods>

    <Actions>
      <Action>
        <Name>SkipSplash</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Activate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Space].StateChanged +=
                  HandleKeyPressed;

              ((XnaMouseManager)Game.GetComponent(
                typeof(XnaMouseManager))).LeftButton.StateChanged +=
                  HandleMouseButtonPressed;
            </Source>
          </Method>

          <Method Type="Pipeline:CustomOverride">
            <Name>Deactivate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Space].StateChanged -=
                  HandleKeyPressed;

              ((XnaMouseManager)Game.GetComponent(
                typeof(XnaMouseManager))).LeftButton.StateChanged -=
                  HandleMouseButtonPressed;
            </Source>
          </Method>

          <Method Type="Pipeline:CustomEventHandler">
            <Name>HandleKeyPressed</Name>
            <Source>
              if (args.Key.State == ButtonState.Pressed)
                Trigger();
            </Source>
            <EventOwnerTypeName>
              GaMEX.Components.Input.XnaKey, GaMEX
            </EventOwnerTypeName>
            <EventName>StateChanged</EventName>
          </Method>

          <Method Type="Pipeline:CustomEventHandler">
            <Name>HandleMouseButtonPressed</Name>
            <Source>
              if (args.Button.State == ButtonState.Pressed)
                Trigger();
            </Source>
            <EventOwnerTypeName>
              GaMEX.Components.Input.XnaMouseButton, GaMEX
            </EventOwnerTypeName>
            <EventName>StateChanged</EventName>
          </Method>
        </Methods>
      </Action>
    </Actions>

    <GameObjects>
      <GameObject Type="Gameplay:SequenceContent">
        <Name>SplashSequence</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              ((GameSegment)ParentContainer).Activated +=
                delegate(object sender, GameEventArgs args) { Start(); };

              ((TimedLinearInterpolation)Game.GetObject(
                "Game.SplashScreen.SplashFade")).Completed +=
                  delegate(GameObject sender, GameObjectEventArgs e) { Step(); };

              base.Initialize();
            </Source>
          </Method>
        </Methods>
        <Steps>
          <Step Type="Gameplay:SequenceStepContent">
            <Name>Step1</Name>
          </Step>
          <Step Type="Gameplay:TimedSequenceStepContent">
            <Name>Step2</Name>
            <Duration>PT4S</Duration>
          </Step>
          <Step Type="Gameplay:SequenceStepContent">
            <Name>Step3</Name>
          </Step>
          <Step Type="Gameplay:SequenceStepContent">
            <Name>Step4</Name>
          </Step>
          <Step Type="Gameplay:TimedSequenceStepContent">
            <Name>Step5</Name>
            <Duration>PT4S</Duration>
          </Step>
          <Step Type="Gameplay:SequenceStepContent">
            <Name>Step6</Name>
          </Step>
        </Steps>
      </GameObject>

      <GameObject Type="Gameplay:Math.TimedLinearInterpolationContent">
        <Name>SplashFade</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              ((SequenceStep)Game.GetObject(
                "Game.SplashScreen.SplashSequence.Step1")).Entered +=
                  delegate(GameObject sender, GameObjectEventArgs e)
                  {
                    FromValue = 0;
                    ToValue = 255;
                    Restart();
                  };
              ((SequenceStep)Game.GetObject(
                "Game.SplashScreen.SplashSequence.Step3")).Entered +=
                  delegate(GameObject sender, GameObjectEventArgs e)
                  {
                    FromValue = 255;
                    ToValue = 0;
                    Restart();
                  };

              ((SequenceStep)Game.GetObject(
                "Game.SplashScreen.SplashSequence.Step4")).Entered +=
                  delegate(GameObject sender, GameObjectEventArgs e)
                  {
                    FromValue = 0;
                    ToValue = 255;
                    Restart();
                  };

              ((SequenceStep)Game.GetObject(
                "Game.SplashScreen.SplashSequence.Step6")).Entered +=
                  delegate(GameObject sender, GameObjectEventArgs e)
                  {
                    FromValue = 255;
                    ToValue = 0;
                    Restart();
                  };

              base.Initialize();
            </Source>
          </Method>
        </Methods>
        <FromValue>0</FromValue>
        <ToValue>255</ToValue>
        <Duration>PT2S</Duration>
      </GameObject>

      <GameObject Type="UserInterface:ImageContent">
        <Name>SplashImage1</Name>

        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              ((SequenceStep)Game.GetObject(
                "Game.SplashScreen.SplashSequence.Step1")).Entered +=
                  delegate(GameObject sender, GameObjectEventArgs e) { Enable(); };

              ((SequenceStep)Game.GetObject(
                "Game.SplashScreen.SplashSequence.Step3")).Exited +=
                  delegate(GameObject sender, GameObjectEventArgs e) { Disable(); };

              base.Initialize();
            </Source>
          </Method>
        </Methods>

        <Components>
          <Component Type="Components:UserInterface.XnaImageDataContent">
            <Name>ImageData1</Name>
            <Methods>
              <Method Type="Pipeline:CustomOverride">
                <Name>Update</Name>
                <Source>
                  Style.Color = new Color(
                    Style.Color.R, Style.Color.G, Style.Color.B,
                    (byte)((LinearInterpolation)Game.GetObject(
                      "Game.SplashScreen.SplashFade")).Value);
                  UpdateControl();

                  base.Update(gameTime);
                </Source>
              </Method>
            </Methods>
            <Style>
              <HorizontalAlignment>Stretch</HorizontalAlignment>
              <VerticalAlignment>Stretch</VerticalAlignment>
              <Padding>0 0 0 0</Padding>
              <Margin>0 0 0 0</Margin>
            </Style>
          </Component>
        </Components>

        <ImageFilename>Components/UserInterface/FPSSplashLogo.png</ImageFilename>
      </GameObject>



      <GameObject Type="UserInterface:ImageContent">
        <Name>SplashImage2</Name>

        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              ((SequenceStep)Game.GetObject(
                "Game.SplashScreen.SplashSequence.Step4")).Entered +=
                  delegate(GameObject sender, GameObjectEventArgs e) { Enable(); };

              ((SequenceStep)Game.GetObject(
                "Game.SplashScreen.SplashSequence.Step6")).Exited +=
                  delegate(GameObject sender, GameObjectEventArgs e) { Disable(); };

              base.Initialize();
            </Source>
          </Method>
        </Methods>

        <Components>
          <Component Type="Components:UserInterface.XnaImageDataContent">
            <Name>ImageData2</Name>
            <Methods>
              <Method Type="Pipeline:CustomOverride">
                <Name>Update</Name>
                <Source>
                  Style.Color = new Color(
                    Style.Color.R, Style.Color.G, Style.Color.B,
                    (byte)((LinearInterpolation)Game.GetObject(
                      "Game.SplashScreen.SplashFade")).Value);

                  UpdateControl();

                  base.Update(gameTime);
                </Source>
              </Method>
            </Methods>
            <Style>
              <Width>600</Width>
              <Height>200</Height>
              <HorizontalAlignment>Right</HorizontalAlignment>
              <VerticalAlignment>Center</VerticalAlignment>
              <Padding>0 0 0 0</Padding>
              <Margin>0 0 10 0</Margin>
            </Style>
          </Component>
        </Components>
        <ImageFilename>Components/UserInterface/LaViMaSplashLogo.png</ImageFilename>
      </GameObject>
    </GameObjects>
  </Asset>
</XnaContent>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<XnaContent xmlns:Pipeline="GaMEX.Content.Pipeline"
            xmlns:UserInterface="GaMEX.Content.Pipeline.UserInterface"
            xmlns:Components="GaMEX.Content.Pipeline.Components">
  <Asset Type="Pipeline:GameSegmentContent">
    <Name>MainMenu</Name>

    <Methods>

      <Method Type="Pipeline:CustomOverride">
        <Name>Initialize</Name>
        <Source>
          base.Initialize(game);

          ((TextControl)Game.GetObject("Game.MainMenu.DescriptionOutput")).Enable();
          ((Menu)this.Game.GetObject("Game.MainMenu.Menu")).Activate();

          ((MenuItem)Game.GetObject("Game.MainMenu.Menu.NewGame")).Selected +=
          delegate(GameObject sender, GameObjectEventArgs e)
          {
          this.Game.LoadSegment("Game.Play");
          };

          ((MenuItem)Game.GetObject("Game.MainMenu.Menu.Exit")).Selected +=
          delegate(GameObject sender, GameObjectEventArgs e)
          {
          this.Game.Exit();
          };
        </Source>
      </Method>

    </Methods>

    <Actions>

      <Action>
        <Name>Select</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Activate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Enter].StateChanged +=
                  HandleKeyPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomOverride">
            <Name>Deactivate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Enter].StateChanged -=
                  HandleKeyPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomEventHandler">
            <Name>HandleKeyPressed</Name>
            <Source>
              if (args.Key.State == ButtonState.Pressed)
                Trigger();
            </Source>
            <EventOwnerTypeName>
              GaMEX.Components.Input.XnaKey, GaMEX
            </EventOwnerTypeName>
            <EventName>StateChanged</EventName>
          </Method>
        </Methods>
      </Action>

      <Action>
```

```
        <Name>NavigateUp</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Activate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Up].StateChanged +=
                  HandleKeyPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomOverride">
            <Name>Deactivate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Up].StateChanged -=
                  HandleKeyPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomEventHandler">
            <Name>HandleKeyPressed</Name>
            <Source>
              if (args.Key.State == ButtonState.Pressed)
                Trigger();
            </Source>
            <EventOwnerTypeName>
              GaMEX.Components.Input.XnaKey, GaMEX
            </EventOwnerTypeName>
            <EventName>StateChanged</EventName>
          </Method>
        </Methods>
      </Action>

      <Action>
        <Name>NavigateDown</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Activate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Down].StateChanged +=
                  HandleKeyPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomOverride">
            <Name>Deactivate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Down].StateChanged -=
                  HandleKeyPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomEventHandler">
            <Name>HandleKeyPressed</Name>
            <Source>
              if (args.Key.State == ButtonState.Pressed)
                Trigger();
            </Source>
            <EventOwnerTypeName>
              GaMEX.Components.Input.XnaKey, GaMEX
            </EventOwnerTypeName>
            <EventName>StateChanged</EventName>
          </Method>
        </Methods>
      </Action>
    </Actions>

    <GameObjects>
      <GameObject Type="UserInterface:MenuContent">
        <Name>Menu</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              Game.GetActionTrigger("Game.MainMenu.Select").Triggered +=
              delegate(object sender, GameEventArgs e)
              {
              MenuItems[ActiveIndex].Select();
              };

              Game.GetActionTrigger("Game.MainMenu.NavigateUp").Triggered +=
              delegate(object sender, GameEventArgs e)
              {
              if (ActiveIndex == 0)
              ActiveIndex = MenuItems.Count - 1;
              else
              ActiveIndex -= 1;
              };

              Game.GetActionTrigger("Game.MainMenu.NavigateDown").Triggered +=
              delegate(object sender, GameEventArgs e)
              {
              if (ActiveIndex == MenuItems.Count - 1)
              ActiveIndex = 0;
              else
              ActiveIndex += 1;
              };

              base.Initialize();
            </Source>
          </Method>
        </Methods>

        <Components>
          <Component Type="Components:UserInterface.XnaMenuDataContent">
            <Name>MenuControlData</Name>
            <Style>
              <Width>300</Width>
              <Height>300</Height>
              <HorizontalAlignment>Left</HorizontalAlignment>
              <VerticalAlignment>Center</VerticalAlignment>
              <Padding>10 10 10 10</Padding>
              <Margin>40 0 0 0</Margin>
            </Style>
          </Component>
        </Components>

        <MenuItems>

          <MenuItem Type="UserInterface:MenuItemContent">
            <Name>NewGame</Name>
            <Components>
              <Component Type="Components:UserInterface.XnaMenuItemDataContent">
                <Name>NewGameMenuItemData</Name>
                <Style>
                  <Color>FFFFFFFF</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 0 0 0</Margin>
                </Style>
                <ActiveStyle>
                  <Color>FFCC0F0F</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 0 0 0</Margin>
                </ActiveStyle>
                <FontFilename>
                  Components/UserInterface/MenuFont.spritefont
                </FontFilename>
              </Component>
            </Components>
            <Text>New Game</Text>
            <Description>Start a new game</Description>
          </MenuItem>

          <MenuItem Type="UserInterface:MenuItemContent">
            <Name>Options</Name>
            <Components>
              <Component Type="Components:UserInterface.XnaMenuItemDataContent">
                <Name>OptionsMenuItemData</Name>
                <Style>
                  <Color>FFFFFFFF</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 64 0 0</Margin>
                </Style>
                <ActiveStyle>
                  <Color>FFCC0F0F</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 64 0 0</Margin>
                </ActiveStyle>
                <FontFilename>
                  Components/UserInterface/MenuFont.spritefont
                </FontFilename>
              </Component>
            </Components>
            <Text>Options</Text>
            <Description>Configure game options</Description>
          </MenuItem>

          <MenuItem Type="UserInterface:MenuItemContent">
            <Name>Exit</Name>
            <Components>
              <Component Type="Components:UserInterface.XnaMenuItemDataContent">
                <Name>ExitMenuItemData</Name>
                <Style>
                  <Color>FFFFFFFF</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 128 0 0</Margin>
                </Style>
                <ActiveStyle>
                  <Color>FFCC0F0F</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 128 0 0</Margin>
                </ActiveStyle>
                <FontFilename>
                  Components/UserInterface/MenuFont.spritefont
                </FontFilename>
              </Component>
            </Components>
            <Text>Exit Game</Text>
            <Description>Exit the game</Description>
          </MenuItem>
        </MenuItems>
      </GameObject>

      <GameObject Type="UserInterface:TextControlContent">
        <Name>DescriptionOutput</Name>
```

```
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              Menu menu = Game.GetObject("Game.MainMenu.Menu") as Menu;
              Text = menu.ActiveItem.Description;
              menu.ActiveIndexChanged +=
                delegate(GameObject sender, GameObjectEventArgs e)
                {
                  Text = menu.ActiveItem.Description;
                };

              base.Initialize();
            </Source>
          </Method>

        </Methods>

        <Components>
          <Component Type="Components:UserInterface.XnaTextDataContent">
            <Name>DescriptionTextData</Name>
            <Style>
              <Width>400</Width>
              <Height>100</Height>
              <HorizontalAlignment>Right</HorizontalAlignment>
              <VerticalAlignment>Bottom</VerticalAlignment>
            </Style>
            <FontFilename>
              Components/UserInterface/TextFont.spritefont
            </FontFilename>
          </Component>
        </Components>
      </GameObject>
    </GameObjects>
  </Asset>
</XnaContent>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<XnaContent xmlns:Pipeline="GaMEX.Content.Pipeline"
            xmlns:UserInterface="GaMEX.Content.Pipeline.UserInterface"
            xmlns:Gameplay="GaMEX.Content.Pipeline.Gameplay"
            xmlns:Components="GaMEX.Content.Pipeline.Components">
  <Asset Type="Pipeline:GameSegmentContent">
    <Name>Play</Name>

    <Methods>
      <Method Type="Pipeline:CustomOverride">
        <Name>Activate</Name>
        <Source>
          ((Sequence)Game.GetObject("Game.Play.WorldSequence")).Start();

          base.Activate();
        </Source>
      </Method>
    </Methods>

    <Actions>

      <!-- Player control -->

      <!-- Menu control -->

      <Action>
        <Name>ToggleMenu</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Activate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Escape].StateChanged +=
                  HandleKeyPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomOverride">
            <Name>Deactivate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Escape].StateChanged -=
                  HandleKeyPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomEventHandler">
            <Name>HandleKeyPressed</Name>
            <Source>
              if (args.Key.State == ButtonState.Pressed)
                Trigger();
            </Source>
            <EventOwnerTypeName>
              GaMEX.Components.Input.XnaKey, GaMEX
            </EventOwnerTypeName>
            <EventName>StateChanged</EventName>
          </Method>
        </Methods>
      </Action>

      <Action>
        <Name>Select</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Activate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Enter].StateChanged +=
                  HandleKeyPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomOverride">
            <Name>Deactivate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Enter].StateChanged -=
                  HandleKeyPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomEventHandler">
            <Name>HandleKeyPressed</Name>
            <Source>
              if (args.Key.State == ButtonState.Pressed)
                Trigger();
            </Source>
            <EventOwnerTypeName>
              GaMEX.Components.Input.XnaKey, GaMEX
            </EventOwnerTypeName>
            <EventName>StateChanged</EventName>
          </Method>
        </Methods>
      </Action>

      <Action>
        <Name>NavigateUp</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Activate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Up].StateChanged +=
                  HandleKeyPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomOverride">
            <Name>Deactivate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Up].StateChanged -=
                  HandleKeyPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomEventHandler">
            <Name>HandleKeyPressed</Name>
            <Source>
              if (args.Key.State == ButtonState.Pressed)
                Trigger();
            </Source>
            <EventOwnerTypeName>
              GaMEX.Components.Input.XnaKey, GaMEX
            </EventOwnerTypeName>
            <EventName>StateChanged</EventName>
          </Method>
        </Methods>
      </Action>

      <Action>
        <Name>NavigateDown</Name>
        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Activate</Name>
            <Source>
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Down].StateChanged +=
                  HandleButtonPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomOverride">
            <Name>Deactivate</Name>
            <Source>
```

```
              ((XnaKeyboardManager)Game.GetComponent(
                typeof(XnaKeyboardManager))).Buttons[Keys.Down].StateChanged -=
                  HandleButtonPressed;
            </Source>
          </Method>
          <Method Type="Pipeline:CustomEventHandler">
            <Name>HandleButtonPressed</Name>
            <Source>
              if (args.Key.State == ButtonState.Pressed)
                Trigger();
            </Source>
            <EventOwnerTypeName>
              GaMEX.Components.Input.XnaKey, GaMEX
            </EventOwnerTypeName>
            <EventName>StateChanged</EventName>
          </Method>
        </Methods>
      </Action>
    </Actions>

    <GameObjects>
      <GameObject Type="UserInterface:MenuContent">
        <Name>InGameMenu</Name>

        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Initialize</Name>
            <Source>
              Game.GetActionTrigger("Game.Play.ToggleMenu").Triggered +=
              delegate(object sender, GameEventArgs e)
              {
              if (IsEnabled)
              Deactivate(true);
              else
              Activate();
              };

              Game.GetObject&lt;MenuItem&gt;(
                "Game.Play.InGameMenu.Resume").Selected +=
                  delegate(GameObject sender, GameObjectEventArgs e)
                  {
                    Deactivate(true);
                  };

              Game.GetObject&lt;MenuItem&gt;(
                "Game.Play.InGameMenu.EndGame").Selected +=
                  delegate(GameObject sender, GameObjectEventArgs e)
                  {
                    Game.GetObject&lt;Sequence&gt;("Game.Play.WorldSequence").Stop();
                    Deactivate(true);

                    Game.LoadSegment("Game.MainMenu");
                  };

              base.Initialize();
            </Source>
          </Method>
          <Method Type="Pipeline:CustomOverride">
            <Name>Activate</Name>
            <Source>
              Game.GetActionTrigger("Game.Play.Select").Triggered += HandleSelect;
              Game.GetActionTrigger("Game.Play.NavigateUp").Triggered +=
                HandleNavigateUp;
              Game.GetActionTrigger("Game.Play.NavigateDown").Triggered +=
                HandleNavigateDown;

              base.Activate();
            </Source>
          </Method>
          <Method Type="Pipeline:CustomOverride">
            <Name>Deactivate</Name>
            <Source>
              Game.GetActionTrigger("Game.Play.Select").Triggered -= HandleSelect;
              Game.GetActionTrigger("Game.Play.NavigateUp").Triggered -=
                HandleNavigateUp;
              Game.GetActionTrigger("Game.Play.NavigateDown").Triggered -=
                HandleNavigateDown;

              base.Deactivate(disable);
            </Source>
            <ParameterTypeNames>
              <ParameterTypeName>System.Boolean</ParameterTypeName>
            </ParameterTypeNames>
          </Method>
          <Method Type="Pipeline:CustomEventHandler">
            <Name>HandleSelect</Name>
            <Source>
              MenuItems[ActiveIndex].Select();
            </Source>
            <EventOwnerTypeName>GaMEX.GameActionTrigger, GaMEX</EventOwnerTypeName>
            <EventName>Triggered</EventName>
          </Method>
          <Method Type="Pipeline:CustomEventHandler">
            <Name>HandleNavigateUp</Name>
            <Source>
              if (ActiveIndex == 0)
                ActiveIndex = MenuItems.Count - 1;
              else
                ActiveIndex -= 1;
            </Source>
            <EventOwnerTypeName>GaMEX.GameActionTrigger, GaMEX</EventOwnerTypeName>
            <EventName>Triggered</EventName>
          </Method>
          <Method Type="Pipeline:CustomEventHandler">
            <Name>HandleNavigateDown</Name>
            <Source>
              if (ActiveIndex == MenuItems.Count - 1)
                ActiveIndex = 0;
              else
                ActiveIndex += 1;
            </Source>
            <EventOwnerTypeName>GaMEX.GameActionTrigger, GaMEX</EventOwnerTypeName>
            <EventName>Triggered</EventName>
          </Method>
        </Methods>

        <Components>
          <Component Type="Components:UserInterface.XnaControlDataContent">
            <Name>InGameMenuControlData</Name>
            <Style>
              <Width>300</Width>
              <Height>300</Height>
              <HorizontalAlignment>Center</HorizontalAlignment>
              <VerticalAlignment>Center</VerticalAlignment>
              <Padding>10 10 10 10</Padding>
              <Margin>0 0 0 0</Margin>
            </Style>
          </Component>
        </Components>

        <MenuItems>

          <MenuItem Type="UserInterface:MenuItemContent">
            <Name>Resume</Name>
            <Components>
              <Component Type="Components:UserInterface.XnaMenuItemDataContent">
                <Name>ResumeItemData</Name>
                <Style>
                  <Color>FFFFFFFF</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 0 0 0</Margin>
                </Style>
                <ActiveStyle>
                  <Color>FFAAFFCC</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 0 0 0</Margin>
                </ActiveStyle>
                <FontFilename>
                  Components/UserInterface/MenuFont.spritefont
                </FontFilename>
              </Component>
            </Components>
            <Text>Resume</Text>
          </MenuItem>

          <MenuItem Type="UserInterface:MenuItemContent">
            <Name>EndGame</Name>
            <Components>
              <Component Type="Components:UserInterface.XnaMenuItemDataContent">
                <Name>EndGameItemData</Name>
                <Style>
                  <Color>FFFFFFFF</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 64 0 0</Margin>
                </Style>
                <ActiveStyle>
                  <Color>FFAAFFCC</Color>
                  <HorizontalAlignment>Left</HorizontalAlignment>
                  <VerticalAlignment>Top</VerticalAlignment>
                  <Padding>0 0 0 0</Padding>
                  <Margin>0 64 0 0</Margin>
                </ActiveStyle>
                <FontFilename>
                  Components/UserInterface/MenuFont.spritefont
                </FontFilename>
              </Component>
            </Components>
            <Text>End Game</Text>
          </MenuItem>

        </MenuItems>

      </GameObject>

      <GameObject Type="Gameplay:Time.TimerContent">
        <Name>PlayTimer</Name>
      </GameObject>

      <GameObject Type="Gameplay:SequenceContent">
        <Name>WorldSequence</Name>
        <Steps>
          <Step Type="Gameplay:ContainerSequenceStepContent">
            <Name>Step1</Name>

            <ContentNames>
              <ContentName>Game.Play.WorldSequence.Step1.World1</ContentName>
            </ContentNames>
          </Step>
        </Steps>
      </GameObject>
    </GameObjects>
```

```
  </Asset>
</XnaContent>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<XnaContent xmlns:Pipeline="GaMEX.Content.Pipeline"
    xmlns:GamePipeline="Game.Content.Pipeline"
    xmlns:XnaPipeline="Microsoft.Xna.Framework.Content.Pipeline"
    xmlns:XnaGraphicsPipeline="Microsoft.Xna.Framework.Content.Pipeline.Graphics">
  <Asset Type="GamePipeline:Play.WorldContent">
    <Name>World</Name>
    <Namespace>Game.Play.WorldSequence.Step1</Namespace>

    <RuntimeAssemblies>
      <Assembly Type="System.String">Game.Play.World</Assembly>
    </RuntimeAssemblies>
    <PipelineAssemblies>
      <Assembly Type="System.String">Game.Content.Pipeline.Play.World</Assembly>
    </PipelineAssemblies>

    <Methods>
      <Method Type="Pipeline:CustomOverride">
        <Name>Load</Name>
        <Source>
          base.Load();
        </Source>
      </Method>
    </Methods>

    <Components>
      <Component Type="Pipeline:Components.Graphics.XnaSkyBoxDataContent">
        <Name>SkyData</Name>
        <TextureFrontReference>
          <Reference>#External2</Reference>
        </TextureFrontReference>
        <TextureBackReference>
          <Reference>#External3</Reference>
        </TextureBackReference>
        <TextureLeftReference>
          <Reference>#External4</Reference>
        </TextureLeftReference>
        <TextureRightReference>
          <Reference>#External5</Reference>
        </TextureRightReference>
        <TextureTopReference>
          <Reference>#External6</Reference>
        </TextureTopReference>
        <TextureBottomReference>
          <Reference>#External7</Reference>
        </TextureBottomReference>
      </Component>

      <Component Type="GamePipeline:Play.World.WorldModelDataContent">
        <Name>WorldModelData</Name>

         <RuntimeAssemblies>
          <Assembly>Game.Play.World.WorldModelData</Assembly>
        </RuntimeAssemblies>
        <PipelineAssemblies>
          <Assembly>Game.Content.Pipeline.Play.World.WorldModelData</Assembly>
        </PipelineAssemblies>

        <Properties>
          <Property>
            <Name>Rotation</Name>
            <TypeName>float</TypeName>
          </Property>
        </Properties>

        <Methods>
          <Method Type="Pipeline:CustomOverride">
            <Name>Update</Name>
            <Source>
```

```
              Rotation += (float)gameTime.ElapsedGameTime.TotalSeconds;
              WorldMatrix = Matrix.CreateRotationY(Rotation);
              base.Update(gameTime);
            </Source>
          </Method>
        </Methods>

        <ModelReference
            Type="XnaPipeline:ExternalReference[XnaGraphicsPipeline:NodeContent]">
          <Reference>#External1</Reference>
        </ModelReference>
      </Component>
    </Components>

    <WorldObjects>
      <WorldObject Type="GamePipeline:Play.PlayerContent">
        <Name>Player</Name>

        <RuntimeAssemblies>
          <Assembly>Game.Play.Player</Assembly>
        </RuntimeAssemblies>
        <PipelineAssemblies>
          <Assembly>Game.Content.Pipeline.Play.Player</Assembly>
        </PipelineAssemblies>

        <Components>
          <Component Type="GamePipeline:Play.Player.PlayerCameraDataContent">
            <Name>PlayerCameraData</Name>

            <RuntimeAssemblies>
              <Assembly>Game.Play.Player.PlayerCameraData</Assembly>
            </RuntimeAssemblies>
            <PipelineAssemblies>
              <Assembly>Game.Content.Pipeline.Play.Player.PlayerCameraData</Assembly>
            </PipelineAssemblies>

          </Component>

          <Component
              Type="Pipeline:Components.GameWorld.XnaWorldObjectSceneDataContent">
            <Name>PlayerWorldData</Name>

            <Methods>
              <Method Type="Pipeline:CustomOverride">
                <Name>Initialize</Name>
                <Source>
                  SetPosition(new Vector3(0.0f, 100.0f, 2500.0f));
                  base.Initialize();
                </Source>
              </Method>
            </Methods>
          </Component>
        </Components>
      </WorldObject>
    </WorldObjects>

    <PlayerName>Game.Play.WorldSequence.Step1.World.Player</PlayerName>
  </Asset>

  <ExternalReferences>
    <ExternalReference ID="#External1"
        TargetType="XnaGraphicsPipeline:NodeContent">
      Components/Graphics/Ship.fbx
    </ExternalReference>
    <ExternalReference ID="#External2"
        TargetType="XnaGraphicsPipeline:TextureContent">
      Components/Graphics/CloudsFront.png
    </ExternalReference>
    <ExternalReference ID="#External3"
```

```
        TargetType="XnaGraphicsPipeline:TextureContent">
      Components/Graphics/CloudsBack.png
    </ExternalReference>
    <ExternalReference ID="#External4"
        TargetType="XnaGraphicsPipeline:TextureContent">
      Components/Graphics/CloudsLeft.png
    </ExternalReference>
    <ExternalReference ID="#External5"
        TargetType="XnaGraphicsPipeline:TextureContent">
      Components/Graphics/CloudsRight.png
    </ExternalReference>
    <ExternalReference ID="#External6"
        TargetType="XnaGraphicsPipeline:TextureContent">
      Components/Graphics/CloudsTop.png
    </ExternalReference>
    <ExternalReference ID="#External7"
        TargetType="XnaGraphicsPipeline:TextureContent">
      Components/Graphics/CloudsBottom.png
    </ExternalReference>
  </ExternalReferences>
</XnaContent>
```

Bibliography

Eike Falk Anderson, Steffen Engel, Peter Comninos, and Leigh McLoughlin. The case for research in game engine architecture. In *Future Play '08: Proceedings of the 2008 Conference on Future Play*, pages 228–231, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-218-4. doi: http://doi.acm.org/10.1145/1496984.1497031.

Ahmed BinSubaih, Steve Maddock, and Daniela Romano. Game logic portability. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, ACE '05, pages 458–461, New York, NY, USA, 2005. ACM.

Michael Doherty. A software architecture for games. *University of the Pacific Department of Computer Science Research and Project Journal*, 2003.

David H. Eberly. *3D Game Engine Design*. Academic Press, 2001.

Eelke Folmer. Component based game development - a solution to escalating costs and expanding deadlines? *Lecture Notes in Computer Science*, 4608: 66 – 73, 2007.

Jason Gregory. *Game Engine Architecture*. A K Peters, 2009.

Jesper Juul. *Half-Real: Video Games between Real rules and Fictional Worlds*. The MIT Press, 2005.

Lars Konzack. Computer game criticism: A method for computer game analysis. In *Game Developer Conference*, 2002.

Microsoft. Xna 4.0. AppHub `http://create.msdn.com/en-US/`, 2010. Last accessed: December 15, 2010.

Jeff Plummer. A flexible and expandable architecture for computer games. Master's thesis, 2004.

Andrew Rollings and Dave Morris. *Game Architecture and Design*. 2000.

James Tulip, James Bekkema, and Keith Nesbitt. Multi-threaded game engine design. In *IE '06: Proceedings of the 3rd Australasian conference on Interactive entertainment*, pages 9–14, Murdoch University, Australia, Australia, 2006. Murdoch University. ISBN 86905-902-5.