Card Game Maker: A card game creation system for Unity

Rainer M. Vieira¹ João R. Bittencourt²

Universidade do Vale do Rio dos Sinos, Brazil

Abstract

The current article reports the development process of a card game creation system bundled with an engine to execute these games. To find out the project requirements, case studies of modern card games were used as guidelines to define common components in card games that can be used to the development of said system. With the data acquired on these studies, a two-layered architecture model had been defined, demonstrated by a class diagram. With the architecture proposed, a study to define the technology requirements to the project had been made, defining the system as a Unity extension developed in C#. When the Card Game Maker creational process is concluded, a complete game creation tool for card game creation is expected to be developed.

Keywords: Game Development, Card Games, Game Engine, Software Architecture, Unity

Author's contact:
¹ rainermv@gmail.com
² joaorb@unisinos.br

1. Introduction

Card games are part of a group of the oldest and most popular game genre there are, next to sports games and some board games. In classic card games, the diversity of possible combinations of elements (color, suit, number) allows for a variety of different games (Poker, Solitaire, Bridge), while some games - keeping the familiarity of card handling and reading - extend these elements allowing limitless combinations. For Nesi [2011], in the context of digital games, this game type has been having descending popularity relative to other genres, with the absence of technology being one of the reasons. To fulfill this technology requirement, it was decided to propose the creation of a tool that would enable a fast, simple and budgeted creation of digital card games, a game creation system named Card Game Maker (CGM).

To enable the execution of the games created by this tool, it must attach a game engine. There are some game engine options already on the market, but there are few specific designed for card games [Nesi 2011]. The full development of an engine at low level, on the other hand, has costs beyond the scope of the project. The solution found was to use an existing generic game engine and adapt it to run card games employing the specifications of the creation system. This engine, being a layer below the CGM, was named Card Game Maker Engine (CGME).

This article is organized into six Sections: in Section 1 the motivation for creating the CGM was presented; In Section 2 a brief review of related work will be done; in Section 3 the results of case studies that demonstrate the common elements in classic and modern card games will be reported; In Section 4 the CGME layer proposed model will be presented; In Section 5 the technological decisions for the implementation of the presented model will be presented; and finally, in Section 6 the concluding remarks will be made, also reporting future work.

2. Related Work

This article was created based on Nesi's Joker Engine [2011], reporting an analysis of classic card games, using this data as a basis for creating a model for card game engines and a development of a generic card games engine as an Unity [2005] extension.

There are other game engines developed for the purpose of card games, such as the Vassal Engine [2003], marketed as a "game engine for building and playing online adaptations of board games, tabletop games and card games. It allows users to play in real time over a live Internet connection, and also by email" [Vassal Engine 2003].

3. Case Studies

In order to extend the card games analysis made by Nesi [2011], a case study was made which would also include modern card games[1]. This study sought to do a detailed study of three modern card games (Magic: The Gathering [1993], Hearthstone [2014], and Munchkin [2001]) in order to find common features among these games that could assist in the development of the Card Game Maker.

After the study, it was observed that the recurring characteristics of these games can be organized into a set of components: Rules, Player, Turn, Area, Deck,

[1] Modern card games are card games that do not use common playing cards, using, instead, specific cards with elements developed for each particular game.

Phase, Action, Card and Resource. These components form the basis for the creation of classes for the CGM and CGME model.

4. Proposed Model

In order to create a proposed architecture that satisfies the needs of a modular software, optimized for card games, enabled to code-reusing that is platform and technology independent, it is necessary to make a layered division between the and Card Game Maker and its engine, the Card Game Maker engine.

The CGM, being a game creation system, must have the capability of allowing the creation of card games components and providing the data created by the user to the game engine, which in turn, should functionally run the game developed. The CGME, as an independent layer, should run card games without needing to know the upper layers of the system.

In this article, the proposed Class model will be restricted to the Card Game Maker Engine layer, and the definition of the upper layer will be detailed in a future time.

Basing this research on the concepts of Design Patterns [Gamma et al 2011], the common elements of classical and modern card games and expanding the Joker Engine [Nesi 2011] architecture it is possible to create an architectural model for the Card Game Maker engine.The diagram shown in Figure 1 presents the structural class diagram of the CGME that demonstrates the relationship between classes.

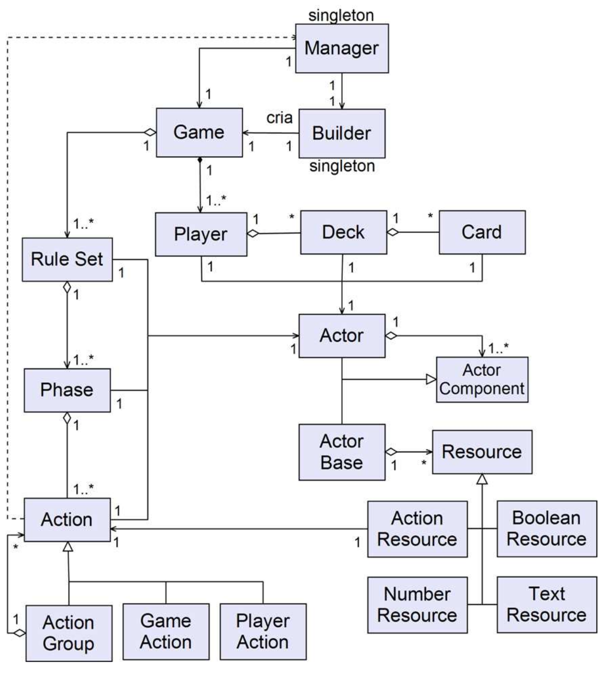

Figure 1 – CGME Class Diagram

The functions of each class within this diagram, their communication with other classes of this model and their relation to the components of card games described in Section 4 will be demonstrated in the following sub-Sections.

4.1 Manager

Manager is a “Singleton" class that acts as the engine module manager. Manager is initiated using the "Run" public method which starts the engine by using the class "Builder" to get a single instance of "Game". The Manager uses this instance to perform the sequence of phases and actions defined in the “Ruleset”. Manager is accessed by instances of "Action" to modify other components of the game (players, decks and cards).

4.2 Builder

Builder is an instance of the “Builder” design Pattern. When the "Build" method is called by the Manager, the Builder creates and returns the class "Game". Having settings that depend on the created game configuration, it is the only class that relies on the upper layer to work.

4.3 Game

Game is a class that represents physical card game set; it contains the rules and game elements. Game is created by "Builder" through the "Manager" and keeps a group of instances of "Ruleset" and also a group of instances of "Player".

4.4 Actor

Actor is an instance of the “Decorator” pattern. It represents a game component that may have extended functionality that can access and modify resources. Actor is a child of the “Component” class and can contain multiple instances of "Component".

As a “Decorator”, Actor delegates its methods to an instance of “Base Actor”, which in turn contains multiple instances of "Resource".
The Ruleset, Phase, Action, Player, Deck and Card classes have one instance of Actor each.

4.5 Resource

According to the definitions of card games, cards and other game components may contain resources. These resources are contained by a “Base Actor” class, are represented by the Resource class and contain the information necessary to the game logic, as well as the actions that can be performed by the corresponding actor. Resource is an abstract class so that multiple types of data can be used as resources.

The child classes of resources and the data they represent are: “TextResource” that represents a string, "NumberResource" that represents an integer value, "BooleanResource" that represents a "true or false" statement and "ActionResource" that represents an action.

4.6 Ruleset

The Ruleset class represents a set of rules of a card game. It contains a set of instances of "Phase" and a single instance of "Actor". When a game starts, only a single instance of "Ruleset" is executed.

4.7 Phase

Phase represents a card game phase; it has a set of instances of "Action" that run sequentially. Phase also contains a single instance of "Actor".

4.8 Action

Actions are the logical sequences representing card game actions. Action is an abstract class of an instance of the "Composite" design pattern and contains an abstract method that defines the action execution.

Action Group is a concrete class, child of Action, which comprises multiple instances of Action. In an Action Group, the execution method is delegated to its children.

Player Action is an abstract class, child of Action, which represents the actions made by players. Player Actions are only executed when there is a player input.

Game Action is an abstract class, child of Action, which represents the actions of the game rules. Unlike Player Actions, Game Actions do not depend on player input, and are executes automatically when the corresponding phase starts.

To be able to modify game components, Actions must use methods of the Singleton "Manager" class. Actions also have a single instance of "Actor".

4.9 Player

Player is a class that represents a player in a card game. Player contains a group of "Deck" instances and a single instance of "Actor".

4.10 Deck

Deck is a class that represents a set of cards and an area on a card game field. Deck has a list of instances of "Card" and a single instance of "Actor".

4.11 Card

Card is a class that represents a card in a card game. It contains only an instance of "Actor".

5. Technological decisions

Having defined the classes' architecture on Section 4, the technologies that will be used to implement the CGM and CGME remain undefined. For this technology to be chosen, there are some required characteristics that are essential. These requirements are: the ability of classes-reusing, ease of portability, speed of development and also the possibly to be acquired at a low cost.

5.1 Unity

Unity 4 [2014] is a powerful development platform that was used to create the Hearthstone card game by Blizzard Entertainment. Unity allows extensions to add features and to modify its interface via dynamic libraries (DLLs). This feature enables the creation of engines and genre-specific game creation systems as a complementary layer to Unity.

Unity is able to compile and export its games to various platforms (PC, Mac, Android, iOS, web, etc.) with minor adjustments in the controllers and minimal requirement to change the source code, satisfying the requirement of "portability".

A "Pro" Unity license is priced at $ 1,500.00, a value below other technologies available in the market. Its "Free" version contains all the features needed to fully develop the tool at no cost,_satisfying the requirement of "low cost".

Unity, in its basic package, has full 2D and 3D rendering, sound, Input modules and various other optional features and configurable modules. The Unity Editor also has an interface that includes a scene viewer and customizable windows via code (extensions). All these features help to meet the requirement of "agility".

The Unity editor supports two programming languages for creating scripts and libraries: JavaScript and C #.

Being object-oriented and able to develop the concepts of Design Patterns, the C# programming language is fully able to implement the architecture model defined in Section 4 and to be exported as a dynamic library, satisfying the requirement of "portability".

5.2Technology Definitions

Being observed the best options for the development of the project requirements, it was decided that the CGM and the CGME will be developed in C# and exported in DLLs as extensions of Unity. The CGM will use the extensibility of the Unity editor interface to support the creation of components of card games. These components, in turn, will be included in a game scene and will be run under the control of the CGME. A complete game developed by Unity with the Card Game Maker is able to be compiled for multiple platforms and released to the market as a standalone title.

6. Conclusion

In the development process of the technology described in this article process the market motivations that led to the need for the creation of a card game creation system were clarified, case studies were done to find the common components in modern card games, an architectural model classes was proposed to define the creation of an engine specific designed for card games and also the better technological decisions that met the system requirements.

Observing then the progress already made and the steps to be executed, it is possible to describe the strengths and weaknesses of the project.

The Card Game Maker, when fully functional, should be able to create classic and modern games much faster than existing technologies. Games created with the CGM can be executed without the need for other technologies. The CGM does not have costly requirements, so it can be sold with little or no cost, encouraging the development of card games to the market.

The Card Game Maker is dependent on the Unity engine and editor, so it cannot run by itself. Moreover, this dependence requires developers to have some knowledge of Unity.

Being Unity a generic game engine and the Card Game Maker Engine an extension of it, there is a worry that the performance of complex games in relation to a low-level engine specially developed for this type of game.

In order to conclude the development of the Card Game Maker, the implementation of the CGME classes is expected, followed by a second of architectural model creation for the main layer of the CGM and a second stage of development to implement it, finishing the integration with the Unity editor.

To conclude the project and to validate the CGM as a functional tool to assist in the development of digital card games, there will be a workshop in which a group of developers will be divided into groups to create games using the CGM tool, comparing them with groups that did not use it. If the proposed objectives are achieved, the CGM can be finally made available to the public as an extension of Unity.

References

GAMMA, E. et al., 2011 Design Patterns: Elements of Reusable Object-Oriented Software. Westford, Massachusetts: Addison-Wesley.

GREGORY, J., 2008 Game Engine Architecture. Natick, Mass.: A. K. Peters.

Hearthstone's official website. Blizzard Entertainment. Available from: http://us.battle.net/hearthstone/en/. [Accessed May 1, 2014].

Magic: the Gathering, rulebook summarized. Wizards of the Coast. Available from http://www.wizards.com/magic/rules/mtg9edrulebook_en.pdf. [Accessed April 25, 2014].

Munchkin Deluxe rulebook. Steve Jackson Games. Available from http://www.worldofmunchkin.com/rules/deluxe_rules.pdf[Accessed April 30, 2014].

NESI, L., 2011 Joker Engine: Um motor genérico para desenvolvimento de jogos de cartas.. São Leopoldo: UNISINOS, 2011. *Monograph (Undergraduate) - Graduation of Digital Games, Universidade do Vale do Rio dos Sinos, São Leopoldo.*

Unity, official website. Available from: http://unity3d.com/unity. [Accessed May 6, 2014].

Vassal Engine, official website. Available from: http://www.vassalengine.org/ [Accessed July 20, 2014].

WAWRO A., 2014. Interview with Jason Chayes and Bryan Chang, Hearthstone developers, digital card game created in Unity.. Available from: http://gamasutra.com/view/news/214930/QA_Hearthstone_heralds_new_challenges_for_Blizzard_on_mobile.php [Accessed June 12, 2014].