Pumpkin Seed Framework

GSIP Brief

UAT

Jake O'Connor

11.27.2021

Table of Contents

Introduction

Background

The Pumpkin Seed Framework is a game development toolkit meant to jumpstart the development of turn-based games. Based in part on the work of linguist James Paul Gee in his book on *Unified Discourse Analysis: Language, Reality, Virtual Worlds and Video Games*, the Pumpkin Seed Framework aims to abstract the design of turn-based games down into their shared component elements in order to provide tools that help to jumpstart the development process of other turn-based games. Using this base level of abstraction, the Pumpkin Seed Framework can serve as the foundation for turn-based games regardless of their myriad genre, topic, or design.

Prior Art

There are dozens of commercial products available which seek to provide some jumping-off point for game developers looking to make turn-based games. Nearly every major asset store, such as Unity, Unreal, or Itch.io, have some form of package for sale. The downside to most of these products, in my opinion, is that their implementation is too rigid in either constraining game design or game engine. Several high-quality packages exist for creating RPGs, JRPGs, Tactical RPGs, and Strategy games, but more often than not they are highly specific to their target genre, meaning that developers must switch between multiple varied solutions when working on different turn-based games or fight against their chosen framework in order to bend the rules of design constraints.

Turn Based Strategy Framework

Turn Based Strategy Framework by Crooked Head, available on the Unity Asset Store, is a toolkit and template for users to implement turn-based strategy games and tactical RPGs within the Unity game engine (Crooked Head). This product has been used in the creation of several indie games available on Steam, and has features ranging from map generation to pathfinding to combat execution. This product is similar to the Pumpkin Seed Framework in that it assists users in creating customizable turn-based gameplay, but there as some major differences. *Turn Based Strategy Framework* is designed and developed to work exclusively within the Unity game engine, due to its reliance on Unity features such as GameObjects and Monobehaviors it cannot be used within any other engine. This product also highly constrains a game's design to the prescribed genres, leading any developer who wants to use it for something else to have to either fight against the toolkit or modify it heavily.

Turn-Based RPG Battle Engine 2D

Turn-Based RPG Battle Engine 2D by ByteVeil, available on the Unity Asset Store, is a toolkit and template for users to implement 2D turn-based RPG battles within Unity games (ByteVeil). This product provides extensible unit statistic and ability logic that can be modified using the Unity game engine's inspector windows, aiming to provide a drop-in solution for standard RPG combat. Like the Pumpkin Seed Framework, *Turn-Based RPG Battle Engine 2D* provides tools to execute and resolve turn-based combat and is meant to be extensible for the user. Unlike the Pumpkin Seed Framework, this toolkit is highly reliant on the Unity game engine to function, constrains game design with built-in assumptions about game mechanics, and can only be used for combat resolution.

ORK Framework

ORK Framework by Gaming is Love, available both through the ORK website and the Unity Asset Store, is a fully featured game framework and toolkit for developing RPGs of all types, including turn-based, active time, and real time (Gamingislove). This product features dozens of tools and samples usable to create myriad games, including multiple different combat systems, AI, equipment and crafting systems, menus, and a dialog system. *ORK Framework* is in the same category of game development toolkits as the Pumpkin Seed Framework, but their similarities end there. Where *ORK Framework* serves as a full end-to-end solution for RPG development within Unity, and does a very good job of it, the Pumpkin Seed Framework serves as a modular and abstract foundation to develop upon.

RPG Maker Series

The RPG Maker series of tools developed by Gotcha Gotcha Games Inc, available on myriad stores and platforms from Steam on PC to the PlayStation 1, are standalone game toolkits which enable the creation and release of JRPG games (RPG Maker…). This product has many iterations, but all provide for full featured game development and include everything from battle systems to map building to animation and scripting systems. *RPG Maker* could not be more different from the Pumpkin Seed Framework, while both aim to support the development of turn-based games the *RPG Maker* series constrains developers into a rigid style and set of mechanics that are supported by the engine, unless the developer does a lot of work to unwind and script their way around assumptions built into the engine.

Target Market Analysis

The Pumpkin Seed Framework is being developed as an in-house solution for indie game development at Sad Pumpkin Games in order to speed up iteration time on new game designs. As such, there is not exactly a target market for the innovation, nor a plan to market it to a larger audience. Since the Pumpkin Seed Framework is not a marketable product per se it will be available for free in both source and package form for public use in any environment which supports the Common Language Runtime (CLR).

Innovation Claim

The Pumpkin Seed Framework is innovative in that it is a design- and platform-agnostic toolset for use in the development of turn-based games of any type. Unlike other products currently available on the market, the Pumpkin Seed Framework aims to have no inherent biases towards specific genres, styles, or mechanics and is built in such a way that it can be used on most major development platforms. It can be used to develop the combat engine for a Tactical RPG within Unity, a basic checkers app written for the web in ASP.NET, or anything in between.

Creation Procedure

Evaluation Factors

The Pumpkin Seed Framework aims to be genre-, style-, and platform-agnostic, and be highly modular, so the following questions will be used to identify the successful completion of the project along those criteria:

1. Does the Pumpkin Seed Framework support a significant number of modern game development platforms, such that it can be considered adequately platform-agnostic?
2. Is the Pumpkin Seed Framework flexible enough, and contain few enough biases, that it can be considered adequately genre-agnostic?
3. Does the Pumpkin Seed Framework limit style and mechanical constraints put onto developers such that it can be considered adequately style-agnostic?
4. Is the Pumpkin Seed Framework modular enough to support myriad turn-based game designs without significant hurdles for developers?
5. Does the Pumpkin Seed Framework support abstracted turn-based rules resolution?
6. Does the Pumpkin Seed Framework support common JRPG mechanics?
7. Does the Pumpkin Seed Framework support common Tactical RPG mechanics?
8. Can simple turn-based games be created with the Pumpkin Seed Framework without significant hurdles, rewriting, and excess code?

Medium and Methods

Software Design

The Pumpkin Seed Framework is designed using many common software engineering patterns, especially those used commonly in the games industry, so that it is easier to integrate with existing workflows. The most important one of these engineering patterns is the Façade Pattern, in which a simplified interface is used to interact in predefined ways with more complex objects. The Pumpkin Seed Framework heavily makes use of the Façade Pattern in its interaction with game-specific logic, the framework itself only interacts with abstract interfaces that guarantee specific types of behavior but not what that specific behavior is. The next most important of these engineering patterns is the Strategy Pattern, in which objects that implement disparate algorithms can be easily swapped out with one another. The Pumpkin Seed Framework makes use of this pattern in how the individual components of the framework are created, which allows users to swap in different versions, either included with the toolkit or of their own game-specific construction, in order to modify the functionality of the framework.

Technology

The Pumpkin Seed Framework is developed in C# using the .NET Core framework. .NET Core is a highly portable CLR framework that can be used alongside nearly any engine or platform. As a result the Pumpkin Seed Framework can be seamlessly used natively within other .NET products like Unity, ASP.NET, WPF, and Stride, the most popular C++ game engine Unreal (via a CLR plugin), or a Java application (via a Java/COM bridge).

Creation Specifics

Entities

Within the design and implementation of the Pumpkin Seed Framework, all objects that can have their state mutated by events that transpire within gameplay are referred to as Entities; Entities are objects that can be *affected*. These objects can be anything from locations to items to characters, depending on the needs of the specific game design. In an example prototype JRPG using the Pumpkin Seed Framework, all characters were Entities. In an example prototype Checkers client using the framework, both the players and the pieces themselves were Entities.

Actors

Within the design and implementation of the Pumpkin Seed Framework, all objects that can *affect* the state of the game are referred to as Actors. These objects are the ones which receive turn prompts from the framework and are asked to take actions to affect change upon the game state. Within the JRPG prototype, all characters were Actors in addition to being Entities. Within the Checkers client, only the players themselves were Actors. This separation between Entity and Actor allows for more modularity in the implementation and design of game-specific details.

Actions

Within the design and implementation of the Pumpkin Seed Framework, commands which enact change upon the game state are referred to as Actions. Actions are derived from the active Actor and the current game state, calculated after the previous Action's effects have been resolved within the framework and the game state has been updated. Actors are prompted with all of their Actions, both those which are currently available and those which are disallowed, so that whatever controls them can make an informed decision. Once an Action is selected, the game enacts the changes defined within it onto the game state and Entities being tracked.

Initiative

Within the design and implementation of the Pumpkin Seed Framework, the order in which Actors are prompted for their Action selection is referred to as Initiative. The Initiative system used within the framework is highly customizable based on the specific game design, ranging from something as simple as basic turn-taking, such as that in Checkers, to something much more complex and malleable like the combat ordering system of a JRPG.

Outcomes

Prototype

The initial prototype of the Pumpkin Seed Framework, at that point called simply called CombatEngine, was completed in November 2020 alongside a JRPG project called *Thirty Day Hero*. This version of the Pumpkin Seed Framework was built on the same foundational design patterns as mentioned above, but was less modular and abstract. At this point the Pumpkin Seed Framework was being created specifically for *Thirty Day Hero,* a JRPG with lite Roguelike mechanics, so the framework made a lot of assumptions about the existence of characters, classes, abilities, and equipment. This sort of game-specific logic is meant to be built atop the Pumpkin Seed Framework's more abstract facades, but at this point both the facades and game-specific logic were built into the same project and a lot of assumptions were being made.

Evaluation Assessment

1. **Does the Pumpkin Seed Framework support a significant number of modern game development platforms, such that it can be considered adequately platform-agnostic?**
   Yes. Being developed in C# on .NET Core, the Pumpkin Seed Framework already has a significant theoretical range. .NET can be used either directly or through optional plugins on many major game engines, as well as frameworks not traditionally considered for game development. The Unity, Godot, CryEngine and Stride game engines all support C# natively and cover a large portion of the game engines used in both amateur and professional development. Unreal, the largest C++ based game engine, supports C# through a plugin, further expanding the potential usability of the Pumpkin Seed Framework.
2. **Is the Pumpkin Seed Framework flexible enough, and contain few enough biases, that it can be considered adequately genre-agnostic?**

Yes. Though the initial prototype of the Pumpkin Seed Framework has inherent biases towards the JRPG genre, as it was developed alongside one, the current version of the framework has been significantly abstracted out to the point that it can provably support three different disparate game genres. The implementation of both Checkers and a Tactical RPG prototype resulted in a lot of assumptions being identified within the Pumpkin Seed Framework and subsequently corrected. The framework should now be abstract enough that it can support nearly any turn-based game without significant hurdles in place for developers.

3. **Does the Pumpkin Seed Framework limit style and mechanical constraints put onto developers such that it can be considered adequately style-agnostic?**

   Yes. The Pumpkin Seed Framework contains very little concrete implementation aside from the basic scaffolding of the turn-based rules evaluation, meaning that style and mechanics are entirely up to the developer making use of the framework.

4. **Is the Pumpkin Seed Framework modular enough to support myriad turn-based game designs without significant hurdles for developers?**

   Yes. While the first non-JRPG game implementation using the Pumpkin Seed Framework revealed significant hurdles and assumptions embedded in the framework, all identified issues have been resolved with the latest updates to the framework. The JRPG *Thirty Day Hero* was implemented smoothly with the framework, with little to no hurdles encountered. The Checkers game *English Draughts* exposed significant hurdles and assumptions built into the framework, which were subsequently corrected through multiple updates to the Pumpkin Seed Framework. An unnamed Tactical RPG prototype currently being worked on exposed further hurdles within the Pumpkin Seed Framework, most notably the inability to perform two-stage actions such as move-and-then-attack, which have been corrected in the latest version of the framework.

5. **Does the Pumpkin Seed Framework support abstracted turn-based rules resolution?**

   Yes. The Pumpkin Seed Framework supports a range of turn-based rules interactions. The framework operates as a rules-layer and supports ordered actors being prompted to select from generic actions implemented by the game-layer. These actors and actions can be implemented in whichever way the game-layer decides, meaning game-specific actions and their effects can be anything from moving a checkers piece to casting a spell and anything in between.

6. **Does the Pumpkin Seed Framework support common JRPG mechanics?**

   Yes. As it was originally designed to support the implementation of a JRPG, the Pumpkin Seed Framework is adept at supporting the common mechanics of a JRPG. This includes customizable initiative systems, highly adaptable actor and action interfaces, customizable multi-stage turn resolution, and many features adjustable or extendable through game-specific layers.

7. **Does the Pumpkin Seed Framework support common Tactical RPG mechanics?**

   Yes. Tactical RPG mechanics are often very similar to JRPG mechanics, which the Pumpkin Seed Framework supports (see above), but for multi-stage turn resolution and the addition of map-based mechanics. Alongside the implementation of a basic Checkers app, the Pumpkin Seed Framework was adapted to allow for outside information to be more available to the action selection pipeline, which for a Tactical RPG can be used to identify the position, facing, and proximity of actors and objects on the map.

8. **Can simple turn-based games be created with the Pumpkin Seed Framework without significant hurdles, rewriting, and excess code?**

   Yes. While not true at first, the Pumpkin Seed Framework is now abstract enough and contains few enough assumptions that simple games such as Checkers can be implemented without using excess code or requiring any rewriting. In a simple game such as Checkers, each player could be implemented as an actor, with all their available moves being represented as actions.

Completion Assessment

Overall, the completion of the Pumpkin Seed Framework went smoothly but slowly. On paper, the specific design aspects seemed very manageable to implement completely into the framework. When it came to actually implementing that design it also stayed manageable. The problem came in the testing aspect of the Pumpkin Seed Framework. As the framework in-and-of-itself doesn't exactly *do* anything, testing new features required the separate implementation of multiple sample games with different rulesets. This was the largest hurdle in the development of the Pumpkin Seed Framework, and each phase of its development was tied to one or more of those separate prototypes.

The first phase of development, back when the Pumpkin Seed Framework was just a glimmer of an idea, was pinned to the implementation of a web-based JRPG. This resulted in the framework's initial prototype functionally skewing more towards a system which supports a JRPG, not a general solution for any type of turn-based game. At that point, it hadn't been identified as a GSIP or thoroughly planned out, so these kinds of trends were not unexpected.

The second phase of development happened after the Pumpkin Seed Framework became a GSIP. This phase necessitated a large amount of documentation and an almost total rewrite of the functional code in order to strip out lingering aspects of a JRPG's game design. The second phase of development was pinned to the remastering of phase one's web-based game into the Unity engine. The design-specific code was stripped out into modules that could be included into the Unity game, and the remaining design-agnostic code was rewritten to be highly modular and allow for significant customization by the design-specific code.

The third phase of development happened much later. After the Pumpkin Seed Framework had proven sufficient as both a design-specific rules engine and a design-agnostic framework, it was necessary to create as different a project as possible from the prior work. Checkers was identified as one of the simplest turn-based games to implement, and was implemented atop the Pumpkin Seed Framework.

This phase further identified areas of the framework which, while not design-specific, were inextricably linked to the complexity of a JRPG, something which was both unnecessary and unwieldy when used to implement a simple game of Checkers. During this phase, more of the framework's components were stripped out and turned into modular pieces that could be opted-into by the design-specific implementation. One of the major aspects changed during this phase was the separation between an Entity and an Actor. Previously, all objects within the game design of prototypes using the framework could take Actions, but in Checkers only the player themselves can take actions, which then affect the status of individual pieces.

This frequent revision of the Pumpkin Seed Framework was significant work, but resulted in a much better end product that is sufficiently capable of supporting turn-based games at varying sizes and with myriad designs. Were this framework planned as a GSIP from the start it would have been much simpler to identify earlier on the aspects that need to be made modular in order to maintain a design-agnostic approach.

References

Abstract Factory. Refactoring.Guru. (n.d.). https://refactoring.guru/design-patterns/abstract-factory.

Adapter. Refactoring.Guru. (n.d.). https://refactoring.guru/design-patterns/adapter.

Arktentrion. (n.d.). 2D RPG Kit. Packs | Unity Asset Store. https://assetstore.unity.com/packages/templates/packs/2d-rpg-kit-163910.

ByteVeil. (n.d.). Turn-based RPG Battle Engine 2D. Game Toolkits | Unity Asset Store. https://assetstore.unity.com/packages/tools/game-toolkits/turn-based-rpg-battle-engine-2d-135496.

Chain of Responsibility. Refactoring.Guru. (n.d.). https://refactoring.guru/design-patterns/chain-of-responsibility.

Crooked Head. (n.d.). Turn Based Strategy Framework. Systems | Unity Asset Store. https://assetstore.unity.com/packages/templates/systems/turn-based-strategy-framework-50282.

Dextero Solutions. (n.d.). RPG All-in-One. Systems | Unity Asset Store. https://assetstore.unity.com/packages/templates/systems/rpg-all-in-one-53542.

EviLA's Unity Assets. (n.d.). EviLA's RPG Pack For Invector. Systems | Unity Asset Store. https://assetstore.unity.com/packages/templates/systems/evila-s-rpg-pack-for-invector-102817.

Facade. Refactoring.Guru. (n.d.). https://refactoring.guru/design-patterns/facade.

Gamingislove. (2019, January 30). Features. ORK Framework - The complete RPG Engine for Unity. http://orkframework.com/features/.

Gee, J. P. (2015). Unified discourse analysis: Language, reality, virtual worlds, and video games. Routledge.

Prototype. Refactoring.Guru. (n.d.). https://refactoring.guru/design-patterns/prototype.

RPG Maker MZ: RPG Maker: Make Your Own Video Games! RPG Maker MZ | RPG Maker | Make Your Own Video Games! (n.d.). https://www.rpgmakerweb.com/products/rpg-maker-mz.

Strategy. Refactoring.Guru. (n.d.). https://refactoring.guru/design-patterns/strategy.