

Western Washington University – CS Department  
CSCI 497M/597M Virtual Worlds  
Final Report – Spring 2018

# Virtual Worlds Crafting System

*Nicholas Majeske, Daniel Lorigan, Brandon Tran, Ian Crowe*

# Abstract

This project aims to emulate the general task of crafting as seen in many real-world domains. With this system, we hope to alleviate issues that many of these domains encounter in crafting by providing an alternative environment in which these crafting operations may performed. Our approach to solving this problem is to design a virtual crafting system in the Unity game engine that will allow users to interact with and perform crafting operations in a virtual world. At this time, we have a base-line system allowing user interaction with virtual items and the ability to perform crafting operations on those items, and tools for developing customized items and crafting recipes.

## Introduction

### Motivation

Most virtual worlds/environments have some form of crafting. Although we often see crafting in gaming environments, at some level, these crafting systems are reflective of the crafting/manufacturing that persists in the real world. In gaming environments, crafting systems facilitate a myriad of player needs and may be thought of as a component aimed at player satisfaction. Similarly, we are motivated by supplying the user with a robust crafting system that will meet their needs but aim to emulate real-world scenarios to facilitate solving real-world problems as they pertain to crafting. To be specific, many domains/businesses have a form of crafting that is part of their product development. A crafting system flexible enough that it may be used to facilitate those core crafting processes would be useful to many such domains.

### Problem

Many real-world domains involve crafting that is core to their processes and these processes are often expensive and/or dangerous. And currently, there does not exist a system to facilitate the practice of these crafting processes in a safe and inexpensive way.

### Challenges

People, and the domains they work in, are very different. Because if this, developing a crafting system that adequately meets the needs and preferences of thousands, possibly millions, of users is a very complex task. We want to meet and exceed the goals of our crafting system while maintaining a balance that achieves maximal user satisfaction. Specifically, we want our crafting system to excel user productivity while maintaining an intuitive, unintimidating, interactive environment. With all of that being said, this problem is difficult to solve because there are many dimensions to user satisfaction which will be pivotal to the success of our system.

### Objective

Implementing a virtual crafting system will require four individual parts: an Inventory GUI, a Crafting GUI, a Chest GUI, and an Item/Recipe Database. The Inventory GUI will allow the user to interact with and manage their virtual items. The Crafting GUI will allow the user to order materials and perform the crafting of objects. The Chest GUI will act as a storage device for materials to later be used for crafting objects. And the Item/Recipe Database will contain all items and crafting recipes of the virtual world. Together, these four parts will allow the user to combine basic material objects into crafted objects.

### Potential Impact

Depending on the level of generality we are able to achieve with our crafting system, there may be many positive real-world impacts of such a system. For example, given diagrams, manufacturing materials, tools, and environments, our system may be used to train new employees of manufacturing companies without the risk associated with those work environments. This usability extends to many fields where risk is involved including, health-care, culinary, automotive, and any other work environment that follows the general concept of crafting products. We believe the use of our system could keep employees safe while protecting businesses from costly accidents.

## Related Work

### Key Concepts

* **Crafting**
  + Crafting, in a general sense, refers to the process of creating new, often more elaborate, objects from one or many more basic objects. In this system we aim to develop a system that achieves this most general, highly extensible, form of crafting.
* **Event Driven Architecture (EDA)**
  + An EDA may be defined as: “A software architecture promoting the production, consumption of, and reaction to events” [1]. With the high degree of interaction between the user and this crafting system, the design of our system will take from many EDA principles.
* **Graphical User Interface (GUI)**
  + A GUI is a component of EDAs that generally enables user interaction with software components. More precisely, a GUI may defined as: “a computer program that enables a person to communicate with a computer through the use of symbols, visual metaphors, and pointing devices” [2]. Our system uses many different GUIs to enable users to effectively interact with our software to perform an array of crafting processes.

### Related Work

There exist many different crafting systems to date each of which designed to meet their own unique set of requirements. In our approach, we aim to create a crafting system under the requirement that it is as general and as extensible as possible. The visual and functional design of our system is largely influenced by that of the Minecraft crafting system [4]. We use the same concept of crafting objects by meeting the correct ordering of items in a crafting workspace. Our crafting system differs from this work in that we extend to the user the ability to create and customize their own crafting system by creating their own items and recipes.

### Key Technologies

**Unity Game Engine:**

The Unity Game Engine provides an intuitive virtual development environment needed to effectively develop our crafting system. We initially considered using the Unreal Game Engine, but chose the Unity Game Engine for our crafting system to ensure a much easier integration with other systems being developed in this Virtual Worlds class.

**Unity Canvas Component - UI**

The Unity canvas game object provides all of the standard formatting features needed for GUI implementations while providing the parent-child relation of the Hierarchy necessary to adequately structure them. We chose to use this feature in our GUI design over the GUI Box feature that was previously considered because it provided a more flexible, intuitive, and extensible interface for development.

### Related Class Projects

## Builder System – Users will need a tool to manage in-game objects that they create using the Builder System. The GUI components of our crafting system used for item interaction can be shared with the Builder System to meet this user requirement.

## Scripting System – The Scripting System will allow users to attach behaviors to in-game items and that system may need an intuitive way of combining items, and their behaviors, into new items with new behaviors. In this use case, the concept of crafting can be shared between our Crafting System and the Scripting System.

* Workflow Specification Language – The Workflow Specification Language system allows users to compose higher-level tasks from multiple lower-level tasks, which can be thought of as a type of crafting. With this in mind, our systems can share this process of crafting by allowing users to create “task recipes”, combine the appropriate “task items”, and create their very own workflows.

## Architecture

### Requirements or Use Cases

1. Create extensible item and recipe classes.
2. Implement crafting API that creates item(s) from item(s) and a recipe.
3. Create a Graphical User Interface (GUI).
   * Implement a GUI for user inventory and item management.
   * Implement a GUI for item storage.
   * Implement a GUI for crafting.
     + Crafting workspace as a 3X3 grid interfaced with crafting API.
     + Crafting button and output slot for crafted items.
4. Create an SQLite database for storing items and crafting recipes.
5. Create back-end tools for creating custom items and crafting recipe sets.
   * Interface a GUI with database for inserting custom items and recipes.
   * Add all item or recipe class attributes as customizable components of the GUI.
   * Implement user specification of extended item or recipe class.

### Architecture or Design Space

### Architecture

Our crafting system consists of 4 major components: A set of GUIs, an item/recipe SQLite database, an extensible crafting API, and a back-end item and recipe creating tool. All components are implemented in the Model-View-Controller (MVC) design pattern.

* Chest, Inventory, and Crafting GUIs.

These GUIs allow users to interact with, organize, and craft items in an intuitive, event driven space. Each GUI component operates from its own set of items/recipes that are initialized from their specific tables of the database. All three GUIs are connected in that they are “aware” of each other and allow the user to move items from one GUI to another, but are otherwise mutually exclusive.

* Item/recipe SQLite database

The item and recipe SQLite database adds persistence to user created items and recipes. This database supports basic operations such as the creation new tables, insertion of items and recipes, and retrieval of items and recipes. Our database is connected to each component although GUIs use one-time interactions with the database to “checkout” their appropriate copy of items and recipes.

* Crafting API

The crafting API allows the user to crafting new item(s) from existing item(s) and a given recipe. Our crafting API connects to the crafting GUI so that crafting may be performed in a graphical way, but more robustly, allows users to specify their own GUI (or not GUI at all) and supports extensions for item and recipe classes.

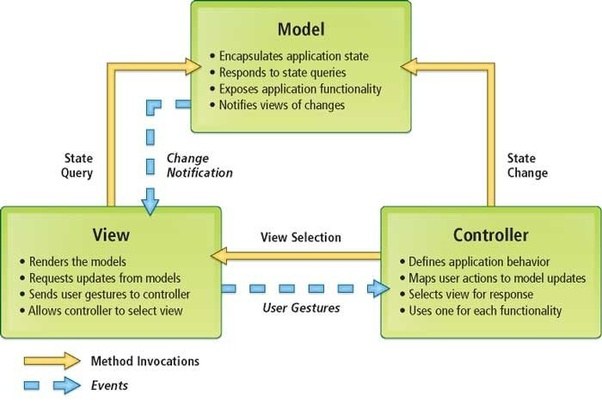
* Back-end item and recipe creation tool

The back-end item and recipe creation tool allows users to create their own custom item and recipe sets. This tool places all attributes of the given item/recipe into the default inspector GUI as editable fields, instantiates an item/recipe object of those specified attributes, and inserts the resulting item/recipe object into the user specified database table. This tool has a unidirectional connection to the item/recipe database.

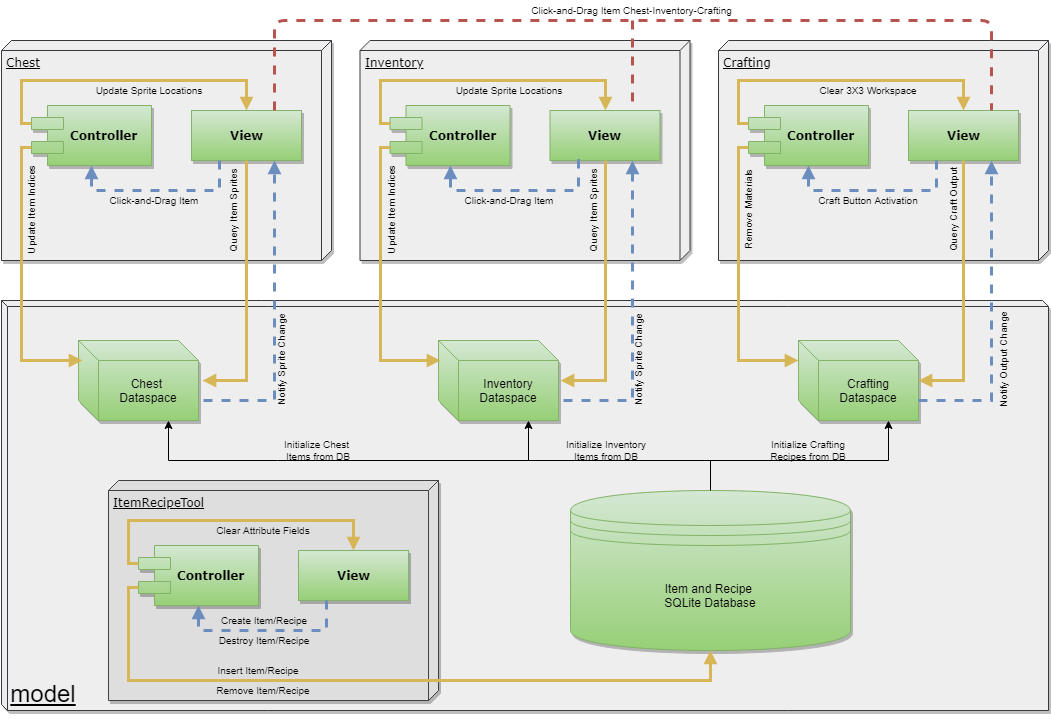
### Design Space

Our approach did not initially elicit additional design dimensions. The GUIs and back-end tools were each implemented from the initial design of our system and without need of refactoring. Our initial design of the database (a pseudo database where item and recipes were stored as json files) was eventually redesigned as a SQLite database resulting in two design dimensions being considered. Lastly, our crafting API elicited many design dimensions and our team struggled to find an optimal balance between generality and initial usability. In making our final crafting API design decision, our team spent two lectures carefully considering a myriad of approaches. We considered all methods being of ‘generic’ type, requiring users core methods, designing the API to be item/recipe class structure invariant, and so on. We finally settled on implementing base item/recipe classes and a base crafting API allowed users to start using the system without any development while facilitating customization by support for extension and API overrides.

With the high degree of user interaction and event driven nature of many of today’s crafting systems, our team decided to implement this crafting system using the Model-View-Controller (MVC) design pattern. The model of our crafting system includes the inventory, chest, crafting dataspaces, and the item/ recipe database. The view components of our system include the inventory, chest, crafting, ItemRecipeTool GUIs. Finally, each view was implemented with its own controller: inventory, chest, and crafting controllers interface their GUIs to their underlying dataspaces, while the ItemRecipeTool controller interfaces its GUI to the item/recipe database. Figure~1 graphically conceptualizes the MVC design pattern and Figure~2 shows a mock-up of our crafting system as an MVC design.



1. *Model-View-Controller (MVC) Design Pattern.*



1. *Model-View-Controller (MVC) design of our crafting system.*

### Tasks

1. Create an Inventory, Chest, and Crafting GUI to allow user-item interaction.
   1. Implement inventory GUI: Nicholas Majeske.
   2. Implement chest GUI: Brandon Tran
   3. Implement crafting GUI: Daniel Lorigan.
   4. Integrate inventory, chest and crafting GUIs for synchronous usage: Daniel Lorigan and Brandon Tran.
2. Create a database to store and organize user defined item and recipe sets.
   1. Design and implement item/recipe object structure and requirements: Ian Crowe, Daniel Lorigan
   2. Design and implement item/recipe database: Brandon Tran, Ian Crowe.
   3. Design and implement item/recipe back-end tools: Nicholas Majeske
3. Create an extensible crafting API interfaced with crafting GUI.
   * Implement crafting workspace design and functionality: Daniel Lorigan.

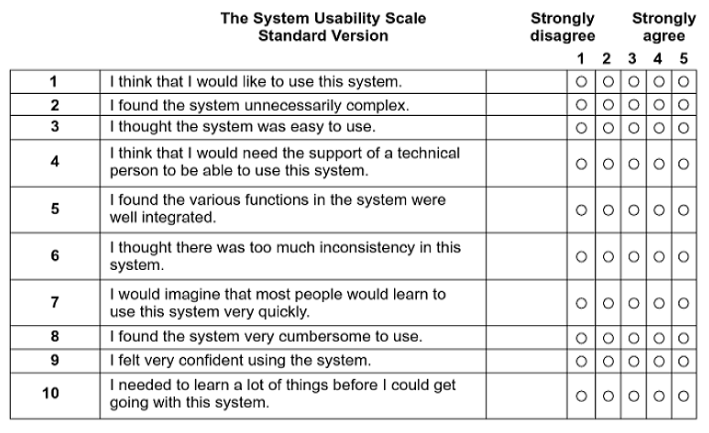
## Testing

### Methodology

Our team has tested the Crafting System code throughout the development process by continually exploring its interaction space using the Unity “game” emulator. In the following scenarios, we tested our system to ensure that it behaved as expected and that those behaviors met our functional requirements.

1. We first evaluated the behavior of our system for basic user interaction with our View component. Tested scenarios in this category included:
   1. User moving items within current view (inventory, chest, and crafting GUIs).
   2. User moving items between views (chest->inventory, inventory->crafting, etc…).
   3. User executing functional units of the view (crafting button properly invokes controller, ItemRecipeTool buttons refresh attribute fields when clicked).
2. Secondly, we evaluated the behavior of our system for proper updates in the database (Model state changes) with our Controller component. Tested scenarios in this category included:
   1. The user creates a new Item or Recipe database using the ItemRecipeTool by creating an Item or Recipe for a non-existent database ID (new database file created with new Item/Recipe as its first).
   2. The user adds a new Item or Recipe to an existing database using the ItemRecipeTool by creating an Item or Recipe for an existing database ID (new Item/Recipe appended to existing set of Items/Recipes in database file).
3. Lastly, we evaluated the behavior of our system for proper View updates for Model state changes. Tested scenarios in this category included:
   1. The user places the correct Items in the crafting workspace for the selected Recipe and clicks the “craft” button (“material” items removed from inventory model and replaced by “crafted” item, View removes “material” item sprites and adds “crafted” item sprite).

Additionally, our team evaluated the usability of this Crafting System by inviting students to use the system in the Unity game emulator. These students were given 3-5 minutes to explore and interact with the crafting system and all its features. After their session, the participants were asked to rank their level of agreement with the following 10 points.



### Results and Analysis

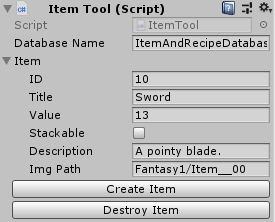
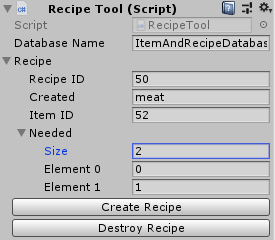
The following screen shots are taken from the Unity game emulator and represent the current state of our crafting system. Figure-1 shows the complete view of our 3 different GUIs including inventory, the chest, and crafting GUIs. In Figure-2 is an example of using our crafting GUI to craft an iron sword using the material items “stick” and “iron ingot”. Finally, in Figure-3 is our ItemRecipeTool back-end tool for created new item and recipe tables as well as adding item and recipes to existing tables in the item/recipe database.



1. *The full view of our inventory, chest, and crafting GUIs.*



1. *An example of crafting in our system. Iron sword Recipe (1 stick and 1 iron).*



1. Our ItemRecipeTool back-end component.

The following table shows the results of our survey where we asked 5 students to interact with out crafting system and take a survey. The left box shows the survey question asked and the right 5 boxes indicate the proportion of participants that gave a particular ranking. Note that 1 indicates *strongly disagree*, 5 indicates *strongly agree*, and 3 indicates *neither strongly agree or disagree.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| System Usability Question | 1 | 2 | 3 | 4 | 5 |
| I think that I would like to use this system. |  |  | 20% | 20% | 60% |
| I found the system unnecessarily complex. | 80% | 20% |  |  |  |
| I thought the system was easy to use. |  |  |  | 60% | 40% |
| I think that I would need the support of a technical person to be able to use this system. | 60% |  | 20% | 20% |  |
| I found the various functions in the system were well integrated. |  |  | 20% | 60% | 20% |
| I thought there was too much inconsistency in this system. | 80% | 20% |  |  |  |
| I would imagine that most people would learn to use this system very quickly. |  |  |  | 60% | 20% |
| I found the system very cumbersome to use. | 80% | 20% |  |  |  |
| I felt very confident using the system. | 20% | 20% | 20% |  | 40% |
| I needed to learn a lot of things before I could get going with this system. | 40% |  | 40% | 20% |  |

1. *Ranking proportions in our usability survey.*

On review of this survey, our team felt there were two major take-aways. First, our system does well at being functionally consistent. Specifically, once users understood the controls of one component, they could effectively use the entire system since all components operate on the same set of controls. Second, our system does poorly at introducing itself and teaching new users how to start. Specifically, it was difficult for users to begin interacting with the system and they often needed direction to get started. We felt that this was because our system is lacking a tutorial or any dialogue to help introduce new users. Our team realized that this feature comes standard for many systems and will be implementing a tutorial feature in the future.

## Conclusions

### Summary

Our attempt at implementing an extensible crafting system in the Unity game engine was a success. We created an intuitive system that allows users to create their very own item sets and crafting recipes. More importantly, we effectively implemented our core crafting system API, items, and recipes as fully extensible C# classes. This is vital since it will allow users to define their own logical routines for determining the satisfaction of recipe requirements for their own custom system. Additionally, we were able to make all of this compatible with an SQLite back-end database and supply back-end tools for creating and storing custom item and recipe instances through. And finally, we created a set of GUI components to facilitate the process of crafting for less technically users while maintaining a level of design modularity that will allow more technical users to integrate their very own set of GUI components.

### Future Work

Our crafting system needs to be made compatible with the other projects including, but not limited to, Builder, Scripting, Workflow Specification Language, and Network Systems. Additionally, with strong base-line system in place, we want to add more features to our crafting system. Some thoughts on this would be to add leveling and perk systems that would benefit the user’s crafting in some capacity. Another idea, which is already in progress, is to add the requirement of events as part of a crafting recipe. For example, baking a turkey may require an oven and the event of that oven being at 350 degrees for a duration of 4 hours.

### Project Value

We selected this project because each of us has experienced a crafting system in some capacity, enjoyed what those systems offer as a component of virtual worlds/environments, and wanted to learn more about these systems. Additionally, these experiences gave each of us a firm grasp on the fundamentals of these systems which in turn made us confidence in our ability to build such a system. Overall, we feel that developing this system has thoroughly introduced us to software development in the Unity game engine and this experience will be integral to future work in this environment.

# Bios

* **Student Nicholas, Majeske** – Majeske is a graduate student of the Computer Science department of Western Washington University. He is an active research associate of professors Brian Hutchinson, Filip Jagodzinksi, and Tanzima Islam focusing on the discovery of volatile protein mutations. Majeske’s areas of focus as a graduate student include Bioinformatics, Machine Learning and High Performance Computing.

# References

[1] "Event-driven Architecture." Wikipedia. April 30, 2018. Accessed May 18, 2018. <https://en.wikipedia.org/wiki/Event-driven_architecture>.

[2] Levy, Steven. "Graphical User Interface." Encyclopædia Britannica. March 29, 2018. Accessed May 18, 2018. <https://www.britannica.com/technology/graphical-user-interface>.

[3] Minecraft Wiki. "Crafting." Minecraft Wiki. May 10, 2018. Accessed May 18, 2018. <https://minecraft.gamepedia.com/Crafting>.

[4] "Model-View-Controller." About Processes and Threads (Windows). Accessed May 18, 2018. https://msdn.microsoft.com/en-us/library/ff649643.aspx.

## Appendix A – Deliverables Manifest

**Inventory GUI:**

A GUI that allows the user to interact with and manage their current set of virtual items. Used in conjunction with Crafting and Chest GUIs.

**Crafting GUI:**

A GUI that allows the user to combine basic virtual items into more complex ones through a crafting operation. Used in conjunction with Inventory GUI.

**Chest GUI:**

A GUI that allow the user to acquire virtual items (crafting materials, etc…). Used in conjunction with Inventory GUI.

**Item/Recipe Database:**

Provides back-end support for the storage and reference for virtual items and recipes for crafting. Use in conjunction with Inventory, Crafting and Chest GUI.