October 2018

Ben-tomo! A Cooperative VR Cooking Game

David Yuhua Tang
Worcester Polytechnic Institute

Jimmy Ngoc Tran
Worcester Polytechnic Institute

Sean Robert Briggs
Worcester Polytechnic Institute

Ben-tomo!

A Cooperative VR Cooking Game

October 4th, 2018

Team Members

Sean Briggs, *srbriggs@wpi.edu*
David Tang, *dytang@wpi.edu*
Jimmy Tran, *jntran@wpi.edu*

WPI Advisors

Gillian Smith, *gmsmith@wpi.edu*

Collaborators

Osaka University; Takemura Labs; Cybermedia Department

Worcester Polytechnic Institute

This report is submitted to Worcester Polytechnic Institute faculty in partial fulfillment of the Degree of Bachelor of Science.

I. Abstract

Ben-tomo! is a cooperative multiplayer VR cooking experience, in which 2 players must work together maneuvering around progressively wackier kitchens to prepare bentos, Japanese lunch boxes, before their customers lose interest. Using the VR facilities provided by Takemura Labs at Osaka University, we were able to create a compelling experience for both HTC Vive and Oculus Rift users that encourages cooperation and excitement between players.

II. Acknowledgements

Our amazing advisors, Gillian Smith and Jennifer DeWinter.

Takemura Labs at Osaka University, for being gracious enough to offer us their facilities.

Ae-Sensei, for always being helpful in and out of the lab.

Jason-Sensei, for encouraging us to dream big and enriching our experience in Japan.

All of our testers, who were with us through the good and the bad times.

III. Table of Contents

IV. Table of Figures

Chapter 5

V. Table of Code

Chapter 4

1. Introduction

Ben-tomo! is a multiplayer Virtual Reality (further referred to as VR for brevity) cooking game, which simulates the experience of rushing to complete orders in a Japanese kitchen. The title “Ben-tomo” is a pun on the word bento, a kind of Japanese packed lunch, and “tomo”, Japanese for “friend”, but also means wisdom or knowledge. In it two players, both in VR headsets, play as the chefs Ben and Tomo, collaborating to complete meal tickets in a Japanese bento-box shop.

Ben-tomo! is a level based game written in Unity. It is cross-compatible with both the Oculus Rift and HTC Vive headsets. In each level players are given a set of ingredients and tools, which can be used to make dishes. By completing dishes within a given time, players can get higher scores. By reaching a target score players can proceed to the next level. As the levels progress, the cooking implements graduate from realistic to absurd, such as kitchen knife windmills as a replacement for cutting boards.

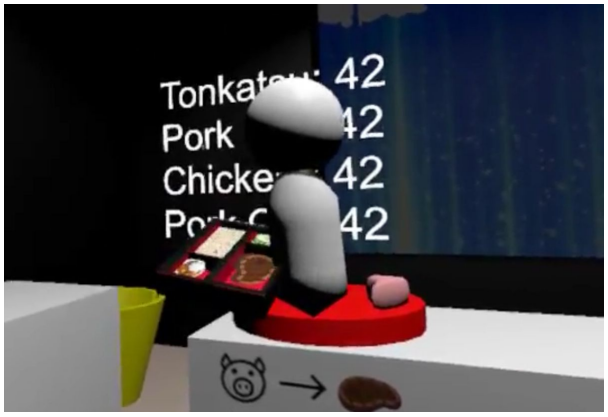


Figure 1.1: Delivering a finished plate

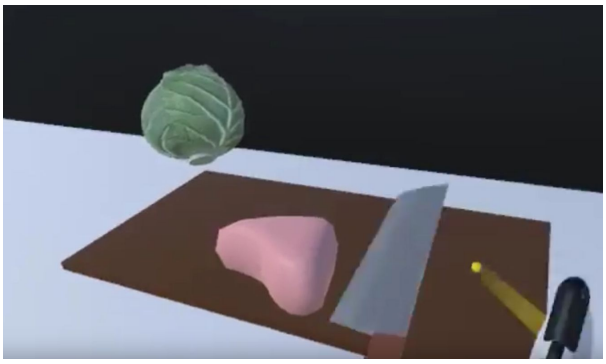


Figure 1.2: Chopping chicken

An important component of the game is cooperation between the two players. Cooperation is necessary to proceed in the game efficiently, similar to how chefs in a kitchen work together. The game was designed and balanced to encourage this by utilizing mechanics and levels that would encourage interaction, such as impeding movement to encourage passing ingredients.

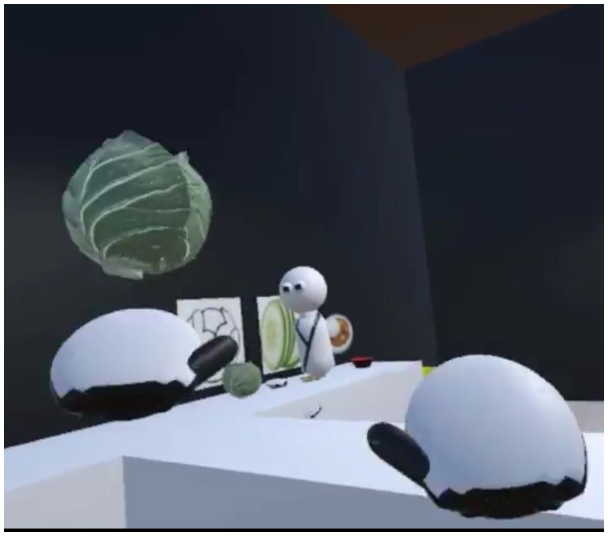


Figure 1.3: Throwing/Catching a Cabbage

It was important that *Ben-tomo!* contained references to the culture of Japan. To ensure our game was culturally appropriate and properly represented the food of Japan, we went around different locations in Japan, trying the food, and talking with restaurant owners about their business.

Our game seeks for the players to meet certain experience goals, further detailed in sections 3A and 3B. In summary, we would like players to enjoy a humorous experience in a Japanese kitchen environment. By progressing through series of levels introducing new

mechanics and more outlandish machines, players should be entertained both by the light-hearted gameplay and learn a bit about Japanese cooking and food through internal subtleties, such as rice fed chicken.

The biggest problems we encountered were on the logistics side. While we were capable of implementing more levels and mechanics, we had not planned it out too well within the scope of our project, and underestimated the amount of time we spent bugfixing. We learned the importance of early implementation and testing, as well as inherent dangers to insufficient testing in both VR and multiplayer settings. Further details on problems and their resolutions can be found in the Postmortem, section 7.

2. Background Research

2.A. Japanese Cooking

2.A.1. Food

Food is a major part of the local and tourist culture of Japan and incorporating that would assist in creating an immersive game. To keep true to the culture, we had to figure out what dishes are popular and how to properly prepare them in a way that is true to the Japanese tradition of food preparation. Considering that there are numerous dishes available, we decided to take the more popular dishes that also had simple preparation techniques in order to implement them into the game. We had to be careful that the preparation techniques can be translated into the game properly so that the techniques fit with the style of gameplay we are looking for.

Research was conducted on their traditional cutlery as a chef's pride is partially in their utensils. In Japan, traditional chefs rarely use a "kitchen knife", usually known as a jack-of-all-trades knife that is used for everything, and would generally use specialized knives created for very limited purposes. They have different knives for cutting fish, cutting meats, filleting fish, carving fish, etc.

Take for instance the Yanagi style knife. These styles of knives are traditionally single-beveled and long and are created for cutting through raw fish and seafood. Meanwhile a Deba knife would be for gutting and filleting a fish. Thus, depending on what the restaurant theme is and what ingredient they are cutting, different knives, ergo different models, must be created in order to keep true to the culture.

Sushi, although very popular in many countries, are not done in a traditional sense. Traditionally, sushi is considered a bar-type food where the sushi chef will be standing in front of you making the sushi and conversing, similar to how bartenders function in bars. (Bolois 2016) However, an increasingly popular way of eating sushi is kaitenzushi which translates to rotating sushi. This type of restaurant would have multiple chefs stand in the middle of a moving conveyor belt preparing various sushi dishes that are then placed on the belt for customers to grab.

Kaiseki is a traditional style of eating where the patron would generally have 14-18 courses for their meal. Multi-course meals started from the Japanese tradition of kaiseki and expanded to other nations as people visited Japan to explore their dining culture. We would have liked to incorporate this into our game in a later level to let players better appreciate the kaiseki tradition.

2.A.2. Restaurants

Japan, while having many restaurants, does not have many that serve numerous types of dishes. Most restaurants will specialize in one type of dish requiring customers to commute to various restaurants depending on their desires. This happens for a variety of reasons, including smaller menus, ease of choice, and specialized chefs. (Restaurant Engine) When the restaurant only needs to focus on one type of dish, they can develop it to be higher quality while reducing the cost of the dish. In order to preserve that tradition, our game will divide the different types of dishes into different groups of levels we call worlds, which would serve traditional foods in their respective traditional settings.

2.B. Similar Games - Comparisons

We performed research on what makes a cooking game enjoyable and how levels are designed around it. Similar games to ours include *Overcooked*, *Diner Duo*, and *ChefU*. Features in common between these games are appealing graphics, light hearted gameplay (e.g. does not punish you for dropping food on the ground) and various unique actions in order to prepare the dishes.

The overall gameplay is similar to that of the symmetrical coop game series *Overcooked,* in that players receive tickets, create the dishes using plates from a dispenser, ingredients processed using various machines, and delivering it through a specific exit. A trash can is also available for mistakes. Our game has a greater emphasis on the simultaneous, cooperative experience through use of machines that require two players to operate; the restricted fields of vision of VR compared to *Overcooked*'s top-down perspective encourage communication between players, and the relatively slower movement while holding items encourages passing them between players. This feature was added in *Overcooked 2*, which released shortly after our game's conceptualization. The interactions with food items are also in VR and require hand motions rather than a simple button press.

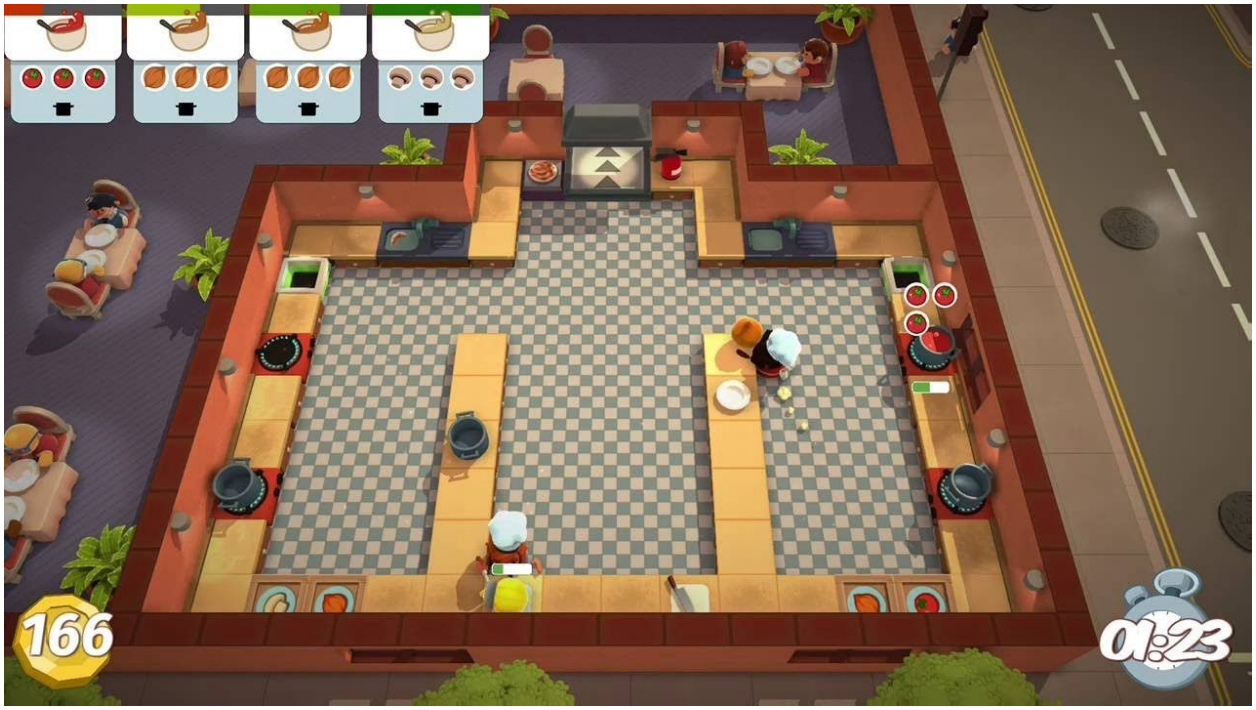


Figure 2.1: Overcooked (Source: https://i.ytimg.com/vi/JcT4DaHlcJc/maxresdefault.jpg)

Overcooked distinguishes its levels through uses of various themes and arrangements of the cooking items, allowing for variation in gameplay and comic relief from the environment. These themes are also used to house gimmicks specific to the theme. For example, a kitchen in a representation of California occasionally features an earthquake, in which a large gap separates the kitchen in 2 for a duration, and the players must work around this timing and plan ahead. Our game distinguishes the groups of levels (called "Worlds" in our game) by using a category of food, such as "Bento", "Ramen", or "Sushi". The machines and arrangement of the kitchen relate to the tools that would be used to prepare such food, and how the food is served in such an establishment. The Bento kitchen may have a 3D printer to dispense the bento-boxes, while a Sushi kitchen would have a sink to wash the dirty plates coming in. By restricting certain machines and recipes to specific worlds, we would allow the player to master skills associated with them. It would also allow them to be able to easily distinguish between the levels to choose levels they enjoy more, or change worlds if there is a mechanic they are currently having difficulty with.

Diner Duo is an asymmetrical mixed VR game (one player on computer, other with headset) revolving around a chef/waiter pair. One player prepares the food while the other delivers it to the customer. The asymmetric viewpoints gives the waiter (on the computer) more awareness than the cook, resulting in more one-sided communication regarding the state of the restaurant. The communication in this game also encourages role-playing to heighten the immersion.

Figures 2.2: The Chef's location

(Source: https://i.ytimg.com/vi/4_qjEXrAIZc/maxresdefault.jpg)

Figure 2.3: The Diner

(Source: https://i.imgur.com/FXO7bXf.jpg)

While *Diner Duo* does not change the arrangement of the kitchen, it slowly increases the menu options and the speed at which they must be completed. Since there is typically at least two orders are active at one time, it encourages the use of both hands to assemble two dishes simultaneously. The waiter later becomes required to get the drinks for customers as well, resulting in differences in the times of when they are available. As a cooperative game, similarity in task volume is important to ensure that each player feels that their part is significant. As our game is symmetrical it requires players to communicate amongst themselves to divide tasks in a way that they feel is appropriate for each player, so asymmetry in player skill will have a lesser effect on which player is waiting for the other.

Lastly, *ChefU* is a single player VR chef simulator. Based on the reviews on Steam (mostly negative), it is not as positively reviewed as the other two games but we can learn from the key contentions that players have regarding the game. The first is that the finished dishes can be arranged however they want without an effect on scoring, resulting in an unattractive final dish, such as the one shown on the right below.

Figure 2.4: Freeform cutting

(Source: https://steamcdn-a.akamaihd.net/steam/apps/680640/ss_641d8146836812143fa25f3968b1e7c4dfe3fc27.1920x1080.jpg?t=1507670426)

Figure 2.5: An unappealing dish

(Source: https://imgur.com/1XsDuTt)

While the graphics are detailed, many players have difficulty running the game without lag. The mechanics of chopping and cooking are fairly complex and graphically attractive, for example the insides of vegetables having textures with the seeds etc properly located regardless of whether it's cut lengthwise, diagonal, etc. but the physics engine seems to have trouble with the creation of irregularly cut objects. To alleviate the above problems, our game uses a more simplistic mechanics for the processing of ingredients where an ingredient will result in the same processed ingredient, and use simple polygonal graphics to reduce computational strain. Further information on the choice of art is detailed in Section 5.

Research into level difficulty scaling within these games shows that the first level is always one or two dishes using 3 or fewer unique ingredients. *Overcooked 2*'s first stage transitions the player(s) from a single ingredient dish (chopped lettuce) to a two ingredient dish (chopped lettuce and tomato slices). *Diner Duo*'s first recipe is a simple burger, a bottom bun, a cooked meat patty, and a top bun. *ChefU*'s first dish is bread, butter, and cheese. A simple first

recipe would allow players to familiarize themselves with the basic mechanics of the game without the overhead of time pressure or memorization. It also provides leniency for players to make mistakes and find out what works and what doesn't. As our game is symmetrical the tutorial levels are most similar to Overcooked, with a few key differences. As our game is in VR, players may not necessarily see what the other player has done, and therefore are not guaranteed to have learned how to perform a task without performing it themselves. As a result our first level requires the use of all of the most basic mechanics and are to complete individually what both players would later do together. The tools needed for completion are in two parallel sets, allowing the two players to face each other and learn by watching. In the other cooking games dishes are introduced one or two at a time giving players time to learn and remember these dishes when playing in consecutive sessions. When there are two they typically involve similar ingredients with a minor change, for example using pork instead of chicken.

As our game functions similarly to a cooperative puzzle game like *Portal 2*, certain characteristics can be drawn from the genre. Mechanics are introduced one at a time and actions that can be performed relating to that mechanic are introduced in successive levels. As our game processes ingredients in a limited number of ways based on machines available, we sought to introduce at most one machine per level that functions differently from something that existed in a previous level. For example the first level would feature ingredients directly on a countertop and diagrams depicting steps that can be taken. In the second level the fridge/pantry would serve as a source of ingredients while the ticket machine would display information on what to do. At the end of each set, a "Mastery Level" does not introduce new mechanics (it can still introduce

new arrangements of existing mechanics) but instead allows the player to demonstrate their mastery of the existing mechanics, and the ability to execute a plan with the other player. These levels result in a small difficulty spike, as it is assumed that the players have learned all existing mechanics, and serve as a way to enforce it.

To maintain a lighthearted atmosphere for a low stress cooperative experience, players in our game (and similar ones) are allowed to perform various actions regardless of impracticality or lack of realism. In all the cooking games listed, there is no penalty for using food that was picked up off of the ground, nor any penalty for throwing food items at people. In *ChefU* the player can choose to break plates and glasses by throwing them at the ground, or drink alcohol from the bottles and glasses. In *Portal 2* players were placed near humorously named machines (some of which are deadly), and can choose to help or destroy their fellow player and themselves, while the narrator GLaDOS demonstrates amusement or frustration at the players' antics. Our game seeks to maintain a similar atmosphere, partially through the use of various wildly unrealistic machines to prepare food, such as throwing food through a wheel of spinning kitchen knives, or an impractical means of moving items by throwing them to the other player; while still maintaining a coherent system within the game, such as physical accuracy in the case of *Portal 2*, for our game accuracy of the recipes and ingredients.

Another game worth noting is *Cooking Mama* for the Wii.This game is the archetypical minigame based cooking game series that helped to define the genre. While featuring a "co-op" mode, it seemed to be more competitive, giving both players the same tasks while scoring them.

Taking advantage of the Wii's sensors, the player moved the Wii-mote's controller similarly to how one would move the tools in an actual kitchen.

Before settling on our current design, our original design sought to use this minigame-based gameplay. However, we found that by having a fully cooperative game, the minigames would often result in one or more players waiting for another, which could be a potential source of irritation. Our move away from standard minigames and redesigning machines to require two players helped reach our experience goals described in section 3.A. and 3.B. much better. This and other problems with the original design are described in more detail in section 7.B.

2.C. What is Humor?

The definition of humor is still unclear, the English word "humor" itself having not been introduced until relatively recently. The term originated from the pseudo-medical analysis of the balance of the four humors in the human body, which were believed to regulate emotion. Over time the term's meaning shifted to one referencing an imbalance in the humors, particularly what we now would call "funny".

The concept of comedy dates back as far as ancient Greek comedies, from which it derives its name. These comedies functioned as political satire in "old comedy", which functioned as a veiled political attacks, or of a deprecative nature in "new comedy", depicting the

absurdity in life similar to a modern sitcom (Winkler 2001). Standard analysis of humor theories by D.H. Monro divides into three groups, superiority, relief, and incongruity.

The superiority theory has it that humor is derived from obtaining a sense of superiority. The ancient Greek comedies functioned to give superiority, either by mocking major political figures or a fictional representation of an everyman, so it is natural that Plato and Aristotle fall into this category. Whereas tragedy is the fall of a person usually better than average, comedy involved the deprecation of someone usually less than average, granting them a feeling of superiority. Thomas Hobbes was a strong proponent of this theory, describing it as "eminency in ourselves, by comparison with the infirmity of others, or with our own formerly" (2001).

Relief theory as proposed by Herbert Spencer and Sigmund Freud follows a tension-relief cycle. Tension-relief has been historically used to be pleasing to an audience, due to the pleasurable experience of the relief of stress. and can be encountered in music, literature, film, and video games. In regard to humor, it is the build-up of stress, and subsequent release. For example In games the desire to seek this release of stress is what motivates them to build up tension, as normally tension is a negative experience (Rose, 2010). However this theory mainly refers to the use of laughter to relieve stress, and thus does not take into account the difference between humorous and non-humorous laughter.

The last theory is the incongruity theory, which occurs when something encountered is beyond expectations. In addition to the superiority theory, Aristotle wrote on the incongruity

theory in *The Rhetoric* (III, 2), setting up an expectation and delivering "a twist". The surprise must also "fit the facts", in that it must be coherent and capable of resolution within the medium. Immanuel Kant's description is similar to the relief theory, in that it is "an affection arising from the sudden transformation of a strained expression into nothing" (1951).

Henry Bergson's essay "Laughter" (1980) describes how humor can be derived from the interaction of humans and their environment, and that humor is the "mechanical encrusted upon the living" (p. 84). It describes recognition of superiority over the subhuman, and things that would subvert this superiority can be found humorous. These kinds of bodily humor can be found in slapstick silent films, like those portrayed by Charlie Chaplin.

Our game uses components of all of these schools of humor. By first introducing natural, intuitive machines to teach the mechanics, it also sets up an expectation which is subverted by the addition of wacky machines, such as the windmill of knives. These machines will still "fit the facts", as they are instruments that can and will be used in a kitchen, and would work physically, albeit poorly. As levels progress, players would develop another expectation a new machine to be introduced, resulting in a tension-relief cycle between completing a level while wondering what comes next, and encountering and learning how to use a new, absurd machine. The arrangements of the kitchen allows for Bergson's approach to humor to take effect, by allowing players to be defeated by a simple wall or corner, as failure to transport items properly will result in them spilling on the floor. Lastly, players may derive comedy from messing up (self deprecation) and seeing the other player mess up (superiority). There is yet another less well

studied source of humor: wit. However what people find funny varies from culture to culture, and use of it, such as in puns, often require the use of language which would detract from our intended goal of universality, and was thus not included into our game.

3. Design and Gameplay

3.A. Intended Gameplay Experience

The intended gameplay should match the design philosophies detailed in Section 3.B. The two players should feel coordinated, cheerful, and committed. They should become connected in a way to the other player, being able to work in tandem, each getting a sense of what the other player is doing without necessarily speaking or looking, and be able to tell what each other needs to have done with simple communication, like “chop cabbage”, or “food gonna burn”. They should experience a small amount of stress trying to make the dishes in time, but they should not be frustrated if they don’t. They should be entertained by the wacky machines in the levels, and find enjoyment from their novel workings and both succeeding and failing in their use.

Rather than overwhelming players with complicated recipes and quantity of tickets as in other cooking games, the players will have more varied steps to make the recipes instead. The game will be level based, divided into sets of levels called “worlds”, increasing in difficulty throughout each world so the players can learn and master various techniques before moving on to the next. Each level can be divided into three main components: the early-game, the mid-game, and the end-game.

When a stage begins, there exists a short duration for players to acclimate themselves to the VR environment, test the controls, and examine the machines/stations. The players can test what throws can be made, and practice throwing items accurately. When they are satisfied, the host player can start the level. Tickets will come in slowly first (one or two at a time), allowing the players to learn and become more confident in their abilities to use the stations, and begin communicating to allow for more effective preparation, ex. throwing a dish to another player instead of moving over to a location.

By the mid-game, the players will receive multiple tickets simultaneously, and can choose which ones to do first. As the two players become more efficient, they can start doing multiple dishes simultaneously, which should require constant communication to prevent repetition of the same tasks. As the difficulty increases, the players should start to feel a bit stressed.

By the end-game, players should be in the "flow" state. The two players should have settled into a pattern, to the point that spoken communication is not as important, and instead have a sense of what the other player is and will be doing. Communication that does occur should be concise, ex. "cut", "wash", etc. The number of tickets starts dying down, and along with it the feeling of relief from stress. However, if they were unable finish the remaining tickets within the time, the players should believe in their ability to perform better if they were to try again.

As levels progress, players will be introduced to new machines, which range from the ordinary to the extreme (windmill of knives, boiling oil faucets, etc). These machines will rely on the players throwing items through the machines, where it is much more efficient if the other player is on the other side to catch them. This should increase the cooperation and reliance on each other, as well as provide comic relief with how strange the machines are. Players can master new techniques with different machines. For example a "stove" made from a giant fire cooks food by throwing food through it. Through practice, players can throw it through slow enough that it is given time to cook, but fast enough not to fall into the fire. The levels are described in more detail in section 3.C.

3.B. Core Design Philosophies and Mechanics to Meet Them

3.B.1. Variations in Interaction

By using VR, the players are more immersed in the environment, and the range of actions they can perform is significantly increased. The interactions players can take should feel natural, as if they were actually in the room. For example, a player needs to press and hold the grab button with their hand on a knife to pick it up, then move the knife onto a choppable food item to cut it. moving a hand up and down to simulate chopping. Objects can be thrown by pressing and holding the grab button, swinging the arm, and releasing a button.

All interactions outside of menus involve pressing the grab button to "grab" a virtual object, and placing it or moving it in a convenient way. By slowly transitioning the player from

ordinary, intuitive machines to “wacky” machines that are based around throwing, the players will become more accustomed to throwing items into the machines, and thus to the other player.

3.B.2. Player difficulty places them in the “flow”

According to Mihaly Csikszentmihalyi, the players will experience different emotional and cognitive effects depending on the difficulty of the game versus their own skill level. Simply put, when the skill level of the player is low but the task is more demanding, the player will enter a state of anxiety. On the flip side, a high skill level combined with a low task demand will cause the player to feel bored. Finding the right balance to enter the states of flow is crucial to ensure the player has the correct effects on their cognitive and emotional beings.

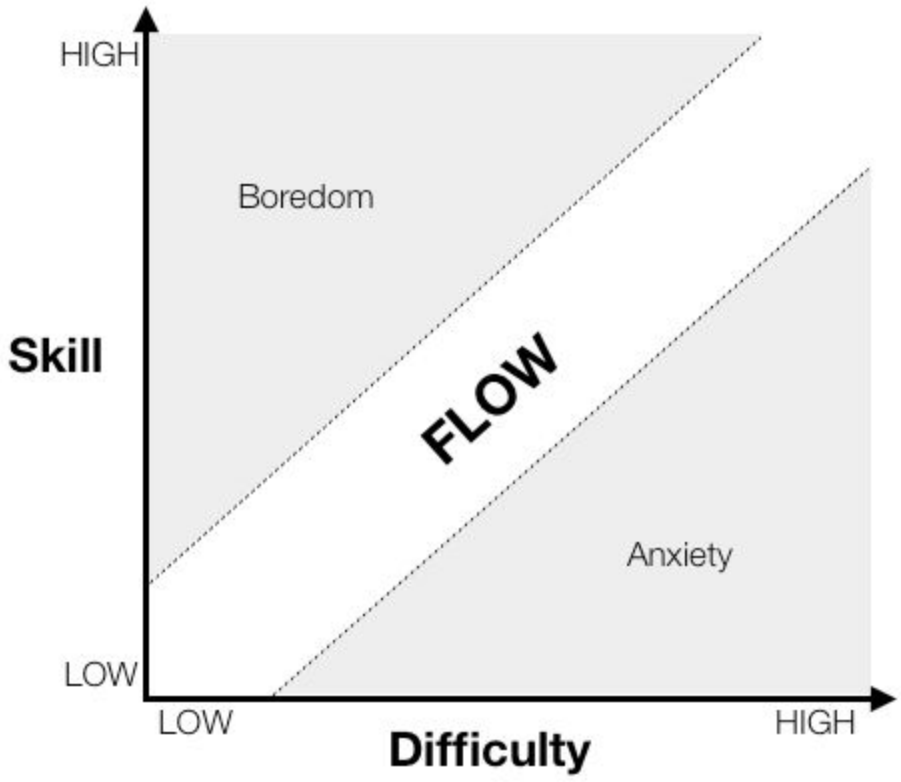


Figure 3.1: Difficulty Flow Chart

By ensuring players learn new skills before proceeding to new levels, players should not end up stuck due to not learning a specific ability or mechanic. By only teaching new mechanics once they master all prerequisites, for example first teach that a plate can be washed by holding it under a sink faucet, then teach that objects can be thrown, before teaching that a plate can be thrown through the sink and it will be washed.

The machines and level layouts will feature components which can be learned for higher cooking efficiency. For example, on level 6 a wall is positioned in such a way that by making a good throw of a meat item from the spawner, it will bounce off the wall and onto the stoves, which are obstructed normally. Similarly, rice can be placed directly into the rice cooker, but can also be thrown into a "basketball hoop" over it that drops the rice into the machine, allowing rice to be positioned accurately from a further distance.

3.B.3. Cooperation

By focusing one of the core mechanics on the primitive cooperative game "Catch", both players must coordinate timing and positioning for the right objects to go through the right machine. Although food can be picked up off of the ground, it is more efficient if it is caught. The first levels are designed so both players can learn the basic mechanics of food preparation: chopping and cooking. However as levels progress the kitchens will become more difficult to navigate, but feature positions where items can be thrown to transfer them to the other side. Information will be restricted due to the first person view of VR, so information like the

incoming tickets will only be visible from certain positions. By communicating this information players will interact more with each other. Some machines may require the use of two players, or be greatly expedited if two players are participating.

However, there are some inherent design problems that can occur within cooperative games (Kumar 2012). An imbalance in skills can result in a less enjoyable experience for one or both players. While the levels are designed to have an "optimal" distribution of tasks, the players can still choose how the tasks are being distributed. For example, if a player tends to get nauseous if they move around in-game too much, they can be distributed the more stationary tasks like cooking, chopping, and throwing. Though the tutorial level is currently set to occur when the two players connected, it was intended to initialize before connection to allow a beginner player to learn the basic movement mechanics without the pressure of a more experienced player waiting. The next level has two sets of machines facing each other, allowing a newer player to learn by watching. The second level does not require cooperation, so the new player has the opportunity to practice at their own pace.

Another problem that could occur is competitiveness between players, which we experienced ourselves during testing. While the players found it enjoyable to do an offense-defense style with one player trying to cook dishes and the other obstructing them, they were aware that there was no benefit to doing so. As the levels progress and the players' roles become more specialized, it would become less enjoyable to attempt to create dishes alone, thus disincentivizing this competitive aspect. Lastly, though they can proceed to the next level

without reaching the target score, which they are unlikely to reach if competing, the score serves as an incentive for the players to cooperate and provide a sense of accomplishment for when they do.

3.B.4. Comic Relief

Comic relief and humor in digital mediums is often attributed to "slapstick" humor. In a study on the most popular humor videos made using game engines, the key attributes found were "variety" and "novelty" (Švelch 2014). The "wacky machines" that we implemented meet both these criteria, and as described in Section 2.D, novelty is an important source of humor. VR is an inherently stressful environment, and by featuring humor through wacky machines the amount of stress in the player can be reduced allowing for more enjoyment and longer play sessions (Mills et al.).

3.B.5. Japanese Style

Japanese cooking varies differently from "Western" cooking, using a variety of ingredients prepared based on the seasons, set up to have visual presentation, etc. By implementing certain characteristics of Japanese cooking (including ingredients and meals) the game can create a "Japanese" feel. Rather than have the customers actively waiting outside the window, the ticket system is often used in Japanese restaurants for a smaller number of cooks to serve more customers, something we tried to emulate in the game.

Our use of wacky machines may seem to contradict with Japanese style cooking, as Japanese chefs obviously don't throw food over large fires to cook them. However we put a greater emphasis on the other components, aesthetically (the food items, bowls, and even the chefs) and other subtle ways: the "body" of the VR avatar allows bowing or other body gestures to be more apparent, the ticket system, and bento-boxes using compartments for food items.

3.B.6. Universality

Though our game is set in Japan, players should be able to play the game regardless of cultural background. There are, however, a few long sections of text in the game. These are either technical terms (IP Address, etc) which shouldn't have significant gameplay effect; recipe names, which are primarily for recognition and could be replaced with an image of the finished dish, if needed; and a few tutorial messages; all of the above are primarily stopgap methods for unimplemented features or information. The remainder are short 1-word information like "Go", which should be fairly recognizable with rudimentary English knowledge.

The names of the two chefs, Ben and Tomo, are of different national origins, Ben is used in multiple European languages, including English, French, and German. Tomo is in the common Japanese names Tomoko and Tomomi, which feature the root words 智 and 朋 meaning wisdom and friend, respectively (Behind the Name).

3.C. Level Design

3.C.1. Design Principles

When designing the levels, certain design principles were set.

3.C.1.a. Only use intended features

The levels are not to encourage the use of unintended features, like walking through walls. All levels must be completable without such features. In addition, levels should not accomodate use of unintended features, if possible. For example, level 1 can no longer reach the target score if a player places a bento-box outside the map. Most walls are thick enough to prevent players from pulling themselves or items through them.

3.C.1.b. Players should learn the mechanics in the levels

Each level in the first world must teach at least one new mechanic (for our current game, all levels). The levels should not teach too many new mechanics simultaneously. Only teach mechanics that will be used later, and don't use mechanics that haven't been taught. The level design should take advantage of the mechanics taught, and use of the mechanics (excluding unintended features) should be the optimal solution.

3.C.1.c. Players should communicate

All levels after the first few must encourage transfer of ingredients between players by throwing or placement. Should visual communication be necessary, players should be able to see each other by taking a few steps from their any position on the level.

3.C.2.d. Levels should accommodate for imbalance of skill

Tasks should be distributed roughly evenly, but not identical tasks. Players should be free to distribute these tasks, though certain distributions may be more optimal than others.

3.C.2. Level Layout

Research into similar cooking games like Overcooked often have an introductory sequence, where the primary mechanics are taught, then for each "world" or set of stages, a new gimmick is revealed and variations allow players to master it. We originally intended for multiple worlds to exist. The gimmick for the Bento world is simplicity to teach the basic game mechanics, as Bento plates do not need to be washed and are simply 3D-printed (spawned in the scene). The next set of levels, the Sushi world, would have introduced dish washing (throwing plates through a sink), and featured conveyor belts prominently, replacing most countertops and transporting items placed around the kitchen or into the dish-out window. The orders would be large numbers of simple recipes (raw fish type on rice, then cut), similar to a conveyor belt sushi restaurant. The third world, the Ramen world, would have featured noodlemaking using different types of dough, different thickness of noodles, and different sauces, all placed in a bowl, where the order of ingredients matters. The worlds after the Bento world would have been parallel,

allowing the players to choose which they wanted to do first, and have their own unique mechanics that would not require completion of the other to learn.

Drafts of the level designs and screenshots of their actual implementation can be found in Appendix 9.A. The first "level" the player will encounter is the main menu. Within the main menu the players can see their own IP address, and can enter the IP address of another player on a large number pad to join their session. They can also press the other "Go" button to host their own game. Currently the buttons on the main menu are situated a distance away from the player, requiring them to move a short distance, however the tutorial for movement does not start until they enter a game. We intended to remedy this by featuring the tutorial before the main menu, but did not due to time constraints.

3.C.2.a. Level 0: Tutorial Level

The tutorial level (Appendix 9.A.3.) starts the player in an enclosed hallway, with a few vertical bars placed. We originally planned for the player's avatar's hands to "close" when the trigger button is pressed, but this could not be implemented in the given time. The intention is that players will attempt to grab the only visible objects, the poles. It is also instinctive to grab nearby objects in an enclosed space for support in movement. By "grabbing" these poles and moving their arm, the player will pull themselves relative to this pivot point. The poles after the first are alternating on the left and right sides to encourage the use of alternating pulling with both left and right hands.

After traversing forward using this method a few meters, the player comes across an arrow pointing up, with another vertical bar. The intention was for players to use their instinct on how vertical bars are climbed in real-life (grab with one hand, grab with the other, pull self up, release first hand and grab again) to teach the controls. If players decide instead to attempt to pull themselves through the wall, they will learn that they will be "bounced" back out. Holograms illustrating both horizontal movement and vertical movement, "grabbing the air" and pulling, is depicted with text. This was included due to uncertainty in whether the intended teaching method would work (grab bars when in an enclosed space) and a limited testing window. This however does take away from the intention of universality.

Upon climbing the pole, the player is to "jump" off the pole into another hallway, with an arrow pointing forward. This passageway does not feature any vertical bars, and is intended to teach the player that the player can grab and pull themselves off of any point in space. At the end of this short passageway, there is a vertical drop to show the player that, if they have not encountered it already, they will fall when not grabbing the air.

At the bottom of the drop, they are placed in a small hallway, with a "cabbage" sign visible below a "window". There is a cabbage in a small hole in the ground, and a hologram that travels from the cabbage to the window, instructing the player on what they are supposed to do. The cabbage was placed in this hole to teach the player to pull themselves downward to pick it up, instead of simply bending down in real life, as they would hit the floor. There is a solid bar over the cabbage, partly to teach that items can be brought through others while held, but

primarily to circumvent falling into the hole and avoiding learning this lesson. The player pick up the cabbage using the same grab button they use for movement. The roof of this segment is intentionally low, to prevent successful throwing of the cabbage into the window. The remainder of this segment is intended to teach the player how to transport an item. It can be thrown, then picked up again at the landing spot, but the item will roll if it is round (like the cabbage). It can also be held in one hand while the player moves with the other. A horizontal bar is placed at one side to encourage this method. The player places the cabbage in the window, which is intended to teach that the window is a place where items are delivered (in the actual levels, the bentos), and when placed, the tutorial is complete, and will automatically transfer them to the level-select menu (Appendix 9.A.2).

3.C.2.b. Level 1: Open for Business!

The first "real" level, level 1 is intended to teach the players (as there should now be two) the basic mechanics of the game. The kitchen layout can be seen in Appendix 9.A.4.b. When asking each other and figuring out the surroundings of the kitchen, they should soon figure out that only one player (the server) can see the balloon that says "Push me down to start!" upon doing so, the "orders", displayed as text near the dish-out window, will appear. These dishes are untimed, giving the players the time to further examine their surroundings to figure out what the recipe names mean.

On the other side of the room, a large sign displaying the recipe name, and components. "Tonkatsu", the recipe for this level, is composed of fried pork, rice, and chopped cabbage. The

players may notice a few ingredients directly below this large sign with icons behind them, however the icons do not match those on the recipe. The pork shows a pig, the rice a picture of the plant, and the cabbage is whole. Within the center of the kitchen are a few machines, the rice cooker, two cutting boards, and a stove, in addition to two bento-boxes sitting on the table. In front of each machine, there is a diagram of icon, arrow, icon. The icon on the back end of the arrow is that of the raw ingredient below the sign, and the icon in front is that of the processed ingredient in the recipe. This is intended to teach the players that by using these machines, the players can process the ingredients into the final ones used in the recipes. The bento-boxes do not have icons, with the implication that they will not process ingredients.

When players grab the ingredients, similarly to the tutorial, and transport them a short distance using one of the methods they should have learned in the tutorial, another ingredient will appear in its place. This should indicate to the player that areas with the icon will spawn a new ingredient of that type when one is taken away. In contrast, if the player moves a bento-box, no more would spawn, as there is no icon.

As the cabbage was in the tutorial, let us suppose the players use the cutting board first. There is a large, flat surface (the board) and a knife. Using existing schema for how a cutting board works, a player would place an object onto the board, and move the knife onto the object. The “chopped” item would then spawn, which, given the icon diagram, players should assume is the processed ingredient shown. In this level, the cabbage turns into a chopped cabbage. The raw pork and rice do not have similar diagrams on the cutting board, and attempting to cut them will

have no effect. Note that in the level there are duplicates of the cutting board and stovetop facing each other, to encourage both players to attempt the actions simultaneously, and to be able to see each other perform the actions in the event that one player learns how it works first.

The stovetop works similarly, processing the depicted ingredient, raw pork, into fried pork. When a player places raw pork over the stove top, a short bar appears next to the pork. Once the bar reaches the end, the item will be transformed into a different one, fried pork. The players may notice that the fried pork also has a bar, and decide to leave it on the stove. Upon completion of this bar, a dark colored item will appear in its place, and emit smoke particles. Using existing schema for cooking food, the players should realize that the food is now burnt. This burnt item, in addition to the rice and cabbage, does not have a loading bar, implying that nothing will happen when placed on the stove. Though we had a 3D model for a stove, it was not implemented, and were thus concerns whether players could recognize that the red cylinder was supposed to be a stove.

Lastly, the rice cooker appears similarly to one in real-life, with the exception of the large funnel on top and a small output on bottom. Given the schema on how funnels are used, it is an affordance for the player to place the rice on top. There is currently no indication that the rice is being processed, but we intended to add an indicator for how much time is left until it is done. Once it is, three rice balls will appear on the exit output if it is unobstructed. The rice balls will roll down the short ramp into a pot in front of the rice cooker, allowing large quantities of rice to be stored while not clogging the exit ramp. Given how the rice will cook and store itself without

burning like on the stove, players should learn that while the stove must be actively watched or checked, the rice cooker can be left alone, and produce multiple units of rice.

Upon obtaining the processed ingredients, players should now try to figure out how to serve them. This leads to the only item they have not touched: the bento-boxes. By placing a processed ingredient above a bento-box, it will be sucked into it, appearing as an icon in one of the slots. The players may notice that only one of each type of ingredient (meat, vegetable, rice, etc) can be placed into a single bento. Once one of each ingredient is stored, the players should attempt to place it in the window, similar to how it was done in the tutorial. The ticket menu is placed next to this window to reinforce this concept. Like the machines, the bento-boxes are placed at opposite sides of the counter to encourage both players to attempt the tasks. Whenever a bento-box is served, the "score" displayed would increase by a fixed amount. Once both of the bento-boxes in the scene are served, a "level end" box will appear. Currently the box is haphazardly placed, due to insufficient testing, but may possibly be repositioned to where the exit window is instead, as at least one player should be facing that direction when the level is complete.

The level end box has three "buttons": Retry, Menu, and Next. Retry will restart the current level, menu will take them to a level select menu, and next will take them to the next level. These buttons are contrary to existing preconceptions, as players must hold their hand inside the button for a duration instead of pushing it as all previous buttons had worked. This is intended to be changed in a later iteration. The level end box will also display a "current score"

and “target score”, though the calculations can be inconsistent and strange. The player can still proceed to the next level regardless of if the score is reached, but the level will be considered “complete” only if the target score is reached or the timer runs out. The score is intended to be changed in a later update.

3.C.2.c Level 2: No Time to Waste!

The second level is arranged similarly to the first, as seen in Appendix 9.A.5.b. however, rather than a divided center, the center is open with the machines on the sides. While the first level allowed players to learn by watching, the second encourages speaking instead, as the players will be facing opposite directions when using the machines. Before the level starts, the players may notice a new recipe, “Karaage”, and a new ingredient, raw chicken. Based on the first level, players should examine the diagrams to learn how to turn raw chicken into the fried chicken icon shown in the recipe chart. They should see that the chicken must first be chopped into chopped chicken, before it can be cooked. This also teaches the player that certain ingredients can take multiple steps to process. Though chicken is currently the only two-step ingredient implemented, more were intended to be added in later Worlds, including ones in the reverse order (chopping after cooking). There are two cutting boards on one side and two stoves on the other, but the icon diagrams are only shown once each. Simple experimentation reveals that regardless of item diagrams, the same machine will have the same functionality.

When the players start the level by the server player pushing down on the balloon, the recipes will show up again similarly to the first level. Unlike in the first level, the numbers

shown next to the recipe name will tick down at a rate of 1 per second, implying that something will happen when it reaches 0. If it does, the recipe will disappear, and no score will be added. This will give the player incentive to complete dishes faster, and the two different dishes shown will discourage making dishes in advance.

The preparation of ingredients should follow a similar pattern to the first level. However, the open space in the center is intended to increase the frequency in which a bento-box is dropped. If a bento-box lands upside-down, the contents will spill out and must be inserted again. An keen player will use this to their advantage by turning a bento-box upside down if the ingredients inside the bento-box need to be changed. As the first level's bento-boxes cannot have excess or invalid ingredients, the addition of the chicken ingredient encourages the use of this feature.

It is also to be noted that in this level, the bento-boxes will spawn to replace bento-boxes taken. This is to allow the player to prepare more bentos than the two in the first level, and also as it is representative of how bento-boxes will be obtained in later levels. The bento-box is also placed in such a way that it is more likely to "accidentally" scoop rice into the bento-box from the rice cooker's pot. This is a practical and efficient way to add rice to a bento-box, which can help players serve bentos faster. A trash can is placed very close by to increase the likelihood of "accidentally" dropping an item inside. An item dropped in a trash can will be removed from the scene and no longer usable. More trash cans are placed toward the stove, as the "burnt" item is the only item that serves no beneficial purpose. Originally there was only one trash can, but

based on user testing players would only use them if they were nearby, and not specifically go to put items inside.

As all dishes require rice, and the rice cooker can only cook one raw rice item at a time, players from playtesting had a tendency to forget to place new rice when the first finished cooking. This is an intended gameplay experience in multitasking, and remembering to refill the rice cooker as needed, as all dishes (in that level and all future ones) include rice as an ingredient.

As the players are able to "fail" the level by not reaching the target score within the given time, they can choose to retry the level to improve their skills, or simply proceed to the next level whether they succeeded or not.

3.C.2.d. Level 3: Keep Calm and Curry On

Level 3 (Appendix 9.A.6.b) follows a similar layout to levels 2 and 1 again, with the dish-out window on one side, ingredients on the other, and countertops on the sides. The machines are located in the center of the room on the same, small table, parallel to the exit window.

In contrast to Level 2, Level 3 only has one of each machine, and the machines are placed close together, requiring players to coordinate use of the machines to use them efficiently. The lack of empty space on the table deters the players from stockpiling large amounts of ingredients

and dishes preemptively, and encourages the players to fetch ingredients only when necessary. As the table also serves as a separator between the ingredients and the dish-out, the players must be able to perform at least one of three tasks: increase their mobility to go over or around the table quickly, be able to throw a dish a short distance accurately without flipping over, or being able to transfer ingredients quickly between players. All of these are important skills, but mastery of one can suffice for this level.

To further discourage stockpiling of ingredients for completing recipes, two new recipes including two new ingredients were added. Pork and chicken curry both use one each of the new "pickled vegetables" and "curry" ingredients. Unlike the other ingredients, the icons for these are the same as those on the recipe, meaning they can be served directly. This also means that by assembling the dish at the location of the ingredients instead, players effectively have an infinite stockpile of those ingredients.

A trash can and the pot of the rice cooker were placed toward the center to prevent easy access to all ingredients or machines, respectively, from a stationary position. This would encourage a more mobile gameplay, only getting and processing the ingredients that are currently needed. By having the players moving rather than stationary, it would also increase the amount of communication needed, as the actions a player can take will vary depending on their location, and the other player would not know which of those can be taken at any given time due to the restricted view of VR.

3.C.2.e Level 4: A Big Missed Steak

Level 4 is where the arrangement of the kitchen starts to vary. As can be seen from Appendix 9.A.7.b., the room is significantly longer than the previous square kitchens, and features two kitchen counters in the center, resulting in an “S” shape.

Just as the previous level had a single divider to increase one of mobility, throwing, or transferring, the two dividers requires them to be capable of both mobility and throwing, or throwing accurately from long distances. Due to the difficulty of accurate throwing and the two counters obstructing the players’ vision of the floor, the latter was intended to be less of a focus.

As the S shape of the kitchen essentially divides the room into three compartments, of which the one closest to the window holds little of value to the players, the optimal positional setup is intended to have one player at the far compartment chopping ingredients and giving bento-boxes, rice and meat to the other player, who is in the center compartment cooking the meat and rice, giving cabbages to the first player, assembling the dishes and throwing them into the dish-out. The components are spaced much further apart in the far compartment than in the middle one, resulting in that player needing to move faster, and ideally improving in the process. This mobility vs. throwing accuracy separation of tasks can be seen in some of the later levels.

Due to difficulty increase in level layout, only one new recipe was added: gyukatsu. Unlike tonkatsu and karaage, gyukatsu uses pickled vegetables instead of chopped cabbage,

resulting in memory playing a minor role, and further deters stockpiling of ingredients. As always, new recipes are added onto the large recipe chart, visible from anywhere in the kitchen.

3.C.2.f. Level 5: The N-egg-st Generation

Level 5 (Appendix 9.A.8.b) was originally designed to introduce the "Chicken" machine, where a player would feed rice and receive eggs over time in return, similar to rice cooker. It was also intended to roughly approximate how eggs are farmed in Japan: chickens are placed in enclosed spaces, are given rice to eat, and eggs laid will drop down along a sloped surface for farmers to collect. Unlike the rice cooker, the eggs would be received one at a time over a longer duration instead of all at once. The eggs would "break" (turn into a broken egg ingredient) upon impact with a solid surface and above a certain velocity. The eggs would turn into different cooked ingredients: soft-boiled if cooked while in egg, hard-boiled if cooked longer, scrambled egg if cooked after breaking.

The two stoves in level 5 were intended to accommodate the increased amount of cookable ingredients, and new recipes would have been added including the different egg types. Otherwise the general idea for the level is similar to level 4, with a few minor differences. Instead of the vegetables located near the middle compartment, all ingredients are at the far compartment. The bento-box spawner is located in the center to reduce the work the player in the far compartment would do, as they were originally intended to carefully move eggs if they were not meant to be broken. The location of the spawner and the empty countertop space near it would allow dishes to be assembled there, before being thrown through the dish-out.

Similar the Gyukatsu recipe, the new Beef Curry recipe does not follow the same pattern as the ones Pork Curry and Chicken Curry share; Beef Curry does no feature pickled vegetables. This is the last recipe introduced in the current version of the game. As humans can remember seven chunks of memory plus or minus two on average according to Miller's Law (Miller, 1956), our target recipe count was within the range of five to nine. As the egg recipes (three) were cut, the six recipes are what the players are tasked with remembering for the rest of the levels. Each world after the first would have had a few more recipes toward the later levels, but the target range for each was around ten.

Instead of the new egg recipes, this level introduces an increased frequency of dishes to allow for a higher possible maximum score, and a higher skill cap needed to reach it. As the kitchens get progressively more difficult, a higher potential for more score can give players a feeling of improvement, as otherwise the score players would get would start decreasing significantly.

3.C.2.g. Level 6: Spin for the Win

This level (Appendix 9.A.9.b) is the first fully divided level, featuring ceiling height walls toward the center. It introduces two new materials, a "slippery" material which lacks friction, allowing objects to slide down sloped surfaces, and a transparent glass material that functions like a wall but can be seen through. There are two recipe charts in this level, one located on either side of the divide.

This level introduces the first wacky machine, the fan of knives (aka a “chopper”), which replaces the cutting board in functionality. A crank is located near the fan, and by grabbing the crank and pulling down (on the side closer to the player) the fan will start spinning. By throwing choppable ingredients like cabbages and raw chicken into a spinning fan, they will turn into their chopped counterparts. The fan is placed high up at an angle on one side of the kitchen, visible on the other side, where the ingredients are located, over the top of a glass wall. Underneath the fan is a ramp of the slippery material, allowing the items to slide to a nearby countertop. Should a player fail to throw food over the wall, the beveled top of the glass will reduce the chance of an item landing on top of the wall, and a trash can in front of the glass will catch some failed throws.

The crank for this level is located next to the chopper, and is placed above eye level (for someone of ordinary height). Thus if the player does not know what the crank does, by pulling it they will see that the fan starts spinning. The current crank model is just a textureless cube, but would be replaced by something more intuitive in further development, with arrow(s) to denote which way it should be turned.

The bento-box spawner is placed high in the air, directly over a trash can. The bento-boxes will continuously spawn and fall into the trash. A player needs to have good timing to be able to grab a bento-box out of the air, as grabbing too early or late will result in grabbing the air.

As can be seen in the second picture of Appendix 9.A.9.b, there is a large pipelike structure above the rice cooker with an opening on top, shaped vaguely like a basketball hoop, and has the icon for raw rice in front. This functions similarly to a basketball hoop, in that a player throws rice into the "hoop", and it will land squarely in the rice cooker. As mentioned previously, it is difficult to throw rice into the rice cooker, and this hoop allows the rice to be put in from a further distance. There is a gap between the pipe and the rice cooker to allow for the removal of invalid ingredients, or simply placing the rice in normally.

Between the rice cooker's pipe and the wall there is a thin vertical gap, behind which are the stoves. The angle of this gap prevents a player near the ingredients from throwing them to the other player or onto the stoves directly, but the nearby wall allows the food to be bounced off, either to the other player or onto a stove. There are three stoves arranged in series to increase the chance of food landing on them. A trash can placed a short distance near the wall will catch some failed throws.

The location of that wall means the location tickets would normally be displayed is visually obstructed. Instead, it is located further away from the window and is thus only visible on the other side of the divide from the window. As that side does not have easy access to the window, players are intended to split up, with one player on either side of the divider. One player reads tickets and communicates them to the other player, as well as giving them the ingredients and bento-boxes. The other player places meat on a stove (if it did not land on it), takes them off

before burning, and cranks the chopper as needed. They then assemble the dishes and serve them through the dish-out window.

The difficulty gradient discussed in Section 4.3.B has been adjust in this level, leading to a slightly easier beginning with a spike in the middle, fading out toward the end. The easier beginning would allow the players to be less surprised at a sudden change in how often they receive tickets.

3.C.2.h. Level 7: Up the Ante

As can be seen in Appendix 9.A.10.b, this level is also divided, but instead of a glass wall, the wall is solid save for a small gap in the center over a large fire. Each side of the divider has a short ramp with a 90 degree bend leading to the other side, allowing items to be transferred between sides without being able to see through and making it difficult to throw items the wrong way on the ramp. Each side has a "basketball hoop" as well, and ingredients that go in them can move to a machine.

The most eye-catching component of the level is the large fire in the middle. Unlike the other levels with a light placed on the ceiling, the only light source in this room is from the fire, giving it more visual impact. It is noteworthy that it appears to be located in a larger version of the trash can seen in previous levels, and dropping items inside reveals that the fire can function as a trash can. There are no "normal" trash cans in this level to emphasize this use.

If a player moves a meat item over the fire, they should notice that it will cook rapidly, about 10 times the speed of a normal stove. If the meat finishes cooking while being held, it will drop out of the player's hand, usually into the fire, which would destroy it. Cooking meat becomes a lot more risk-reward. Throwing the meat through quickly means it probably won't land in the fire, but will also be cooked less. Slowly, and it may land inside, requiring the players to try again. Holding it over the fire and then throwing it over requires the players to be nearby, taking up valuable time to move, while still risking dropping the food inside.

As throwing the food through the fire onto the ground requires the players to pick it up, the efficiency of cooking is greatly increased if both players are nearby and catch the thrown meat, resulting in a minigame similar to "hot potato". Requiring the active participation of both players also necessitates the coordination of their activities to reduce the time spent waiting for the other player, thus helping to further increase the player's communication throughout the level. It is notable that a single player can cook meat safely by holding the meat over a bento-box, as it will snap into the bento-box when it finishes cooking, thus avoiding being destroyed or burnt. As this is an unintended feature, the bento-box spawner was placed further away from the fire to prevent this from happening by accident

As can be seen in the second image of Appendix 9.A.10.b, a "basketball hoop" with icons of cabbage and chicken is located on the side with the meat and vegetable ingredients. By throwing ingredients in the hoop, they will slide down a long narrow ramp until it reaches the knife fan on the other side, as can be seen in the third Appendix image. There is a crank located

on the same side as that hoop, so the player throwing the food must remember to crank the chopper, otherwise the food will end up unchopped on the other side. Due to a lack of auditory feedback, the players may benefit from informing the other when chopping the food.

There is also another "basketball hoop" on the other side, which can also be seen in the third image. By throwing the rice from the nearby spawner, the rice will land in the rice cooker on the other side. A slippery top is placed on top of the wall below this hoop, to missed throws to slide back to the player. As the rice is only visible and accessible from the other side, the players may benefit from informing each other when rice is being cooked, or the rice cooker is empty.

As the bento-box spawner and the knife fan are on one side of the divider while the rice cooker and uncooked ingredients are on the other, and cooked meats can and up on either side of the wall, players will need to transfer ingredients and other items between each other often. The two ramps with bends are the safest choices, but may takes some time to slide down the ramp or end up with the items located in an inconvenient position. The players can also throw items into the "basketball hoops", but this may be more difficult and certain items like bento-boxes may not fit, while still ending up in inconvenient positions (except for items that need to use the corresponding machine). Players can throw items through the fire, but there is a high risk of burning meat or dropping the item into the fire, which would be problematic for a finished bento. Lastly the players could go around the wall, but would take some time to move.

3.C.2.i. Level 8 (Bonus Level)

The final level allows players to practice everything they learned over a much larger scale kitchen, as seen in Appendix 9.A.11.b. Once again the kitchen is divided, though the division is not as clear, as the players will be moving in a larger area.

On one end there is a “basketball court”, featuring cabbages and rice on one end and two “basketball hoops” on the other. The ground in between is green an bouncy, meaning both players and items will bounce off this floor. Due to a lack of testing on later levels, it is unclear whether this would cause nausea in players. If it does not cause significant issues, the bouncy material would be used in later levels to bounce items off walls, ceilings, or other surfaces.

One of the hoops drops the items down a ramp, which lead to a chopper, and the ramp circles back to the basketball hoop. The crank for the chopper is located on the other side of the divider, which should be operated by the other player. The other hoop leads to the rice cooker. The gap between the pipe and the rice cooker also serves as a small window in which items can be transferred, although this may be difficult if the player throwing is standing too far.

Near the “basketball court” is the pickled vegetables and curry spawners. A short distance in front of it is a long, gently sloping slippery ramp between the side of the kitchen and a glass wall. Items placed on the ramp will slide to the other side of the kitchen, but if thrown onto the ramp too hard the items may slide off the side, not reaching the intended destination. The glass

wall allows players to better see when this occurs, as well as see the other player without being able to throw the items straight to them.

Partway down the ramp, the large fire is located a short distance away, which functions similarly to the one in the previous level. A small wall was placed on one side to reduce the effectiveness of a single player throwing it to themself, to better utilize the cooperative aspect. Due to the larger kitchen, there is also less need for a trash can as clutter will be spread throughout, so this fire still functions as the only trash can.

As the ramp continues, it passes through a hallway with glass on one side. On the other side of the glass are the bento-box spawner and meat spawners with some countertops and the crank for the chopper. The glass wall is intended to be climbed over regularly whenever the player needs to deliver a dish, retrieve the curry or pickled vegetables off the ramp, or when meat needs to be cooked. Players will need to practice their vertical mobility as taught in the tutorial. This area also has the clearest view of the tickets, timer and score, so placing a player in the nearby area would be important to know what dishes need to be made. It is notable that a player in this area could stick their hand through a glass wall to place a food item in the chopper, but this is not factored into the target score for the level.

As the “bonus level”, this level is intended to be optional, in that later worlds would not require the completion of this level to unlock. This would allow players to choose to test and show off their skills with a high score, or to try levels in new worlds introducing new mechanics.

3.C.3. Scoring

As more recipes were intended to be set up, a formula was created to standardize the points given for submitting a finished dish, and the time allotted for its creation. Currently it just so happens that all dishes are 120 seconds, but the formulas are as follows:

Dish score = (base score per dish: 20) + 5*(number of ingredients) + 5*(number of ingredients that need to be chopped) + **Σ**(cook time of ingredients, excluding rice)

Time allotted, in seconds =2*Max(60, Round to nearest multiple of 15((base time: 15) + 5*(number of ingredients) + 10*(number of ingredients that need to be chopped) + **Σ**(cook time of ingredients, excluding rice)))

Figure 3.2: Formulas for dish scoring

Let's use Tonkatsu as an example. The score would be the base score (20) plus 5 times the number of ingredients (3), plus 5 times the ingredients that need to be chopped (cabbage, 1) and the sum of the cook time for the ingredients (just pork, 10s), totalling to 20+15+5+10 = 50. The time given is base score (15) + 5*ingredients (3) + 10*chopped(1) + cooktime(10) rounded to the nearest multiple of 15 that is at least 60 (60), then multiply by two, giving 120s to complete the dish. In comparison, Karaage requires one more chopped ingredient and a five second longer cook time for meat, yielding ten more points with the same timeframe.

New ingredients like the egg would function similarly, with the 5 points and seconds per ingredient and 5 points and 10 seconds per action that needs to be taken, like breaking the egg. If the ingredient functions peculiarly and difficult, such as eggs which would cook differently before and after breaking, and break when thrown and not caught, extra points and time would be added on depending on the difficulty of the ingredient.

Due to the difficulty gradient system, the game will select different combinations of dishes such that their "difficulty" values have a certain sum depending on the game timer. The approximate values in these formulas would help ensure that the actual difficulty remains consistent. If all dishes counted as the same "difficulty", in later worlds the players could be given a number of simple dishes (cut raw fish on rice, 35 points) one iteration, but then given the same number of complex dishes in another iteration. For example the most difficult dish we conceptualized was fugu, a poisonous fish which is a delicacy in Japan. It must be skinned which can be represented by cutting exactly once: 10 points, washed: 5 points, carefully gut the fish which can be represented by touching the knife to one subsection of the fish while touching certain others would poison the ingredient: 15 points, then cut normally: 5 points, then boiled for 10sec: 20 points, and add decorative ingredients: 5 points, and arrange carefully on a special plate: 15 points, but would result in an instant game over if served while poisonous: 20 points, totalling 115 points each after adding the base dish score.

The target score of each level was roughly obtained from the score taken from the level designer, who attempted the level solo, but was aware of the nuances of how the level runs and was experienced with the game's mechanics. The later levels were designed to discourage a one-man playstyle, so an amount of points were added on as well. Due to the lack of testing on later levels, the scores may not be a good fit difficulty-wise, but as the current model had a few issues tracking and recording the score, the score has no significant impact on the game.

4. Implementation and Technology

4.A. The Game Engine

This game was built for Windows on Unity 2018.2.3. This engine best fit our needs, as we had the most experience with this engine cumulatively and it gave us access to tools which expedited development. Unreal, the other main popular engine for VR development, seemed like the wrong choice, as Unity was more suited to our particular kind of game, reliant on physics and scripting of individual objects, especially with the VR Toolkit asset package which acts as the base for the bulk of our game's code. Additionally, none of us had prior experience working with networking or VR in Unreal, and decided that learning would be more of a time investment than we would get in return.

The Unity game engine abstracts objects in the game world as positional data (its position, orientation, and size) and a set of `MonoBehaviours`, components that define the object in both data and function. These `GameObjects` are essentially instances of a set of all of these `MonoBehaviours`, and the values of these `MonoBehaviours` can be modified individually for each `GameObject`. Specific combinations of `MonoBehaviours` with default values called "prefabs" can be saved in a file and instantiated at runtime. For example, a prefab for a "cabbage" `GameObject` has an "`Ingredient`" `MonoBehaviour` that holds an `IngredientType` variable, and a "`Rigidbody`" `MonoBehaviour` that applies physics forces such as gravity to the object.

4.A.1. VRTK

VR Toolkit, henceforth VRTK, is an asset package freely available on the Unity Asset Store which handles item interactions in VR across most major VR headsets. This package allowed us to build for both Vive and Oculus players simultaneously, and therefore let us make the best use of our lab space, which had one of each. `GameObjects` with the `VRTK_InteractableObject` component, and with that component set to be grabbable, can be manipulated naturally via the VR controllers. Because our game heavily involves picking up, putting down, and throwing objects with relative precision, this package was a perfect match. VRTK also provides a plethora of "Control" objects, such as buttons, levers, and wheels, which can be manipulated with a similar mechanism as objects in the world.

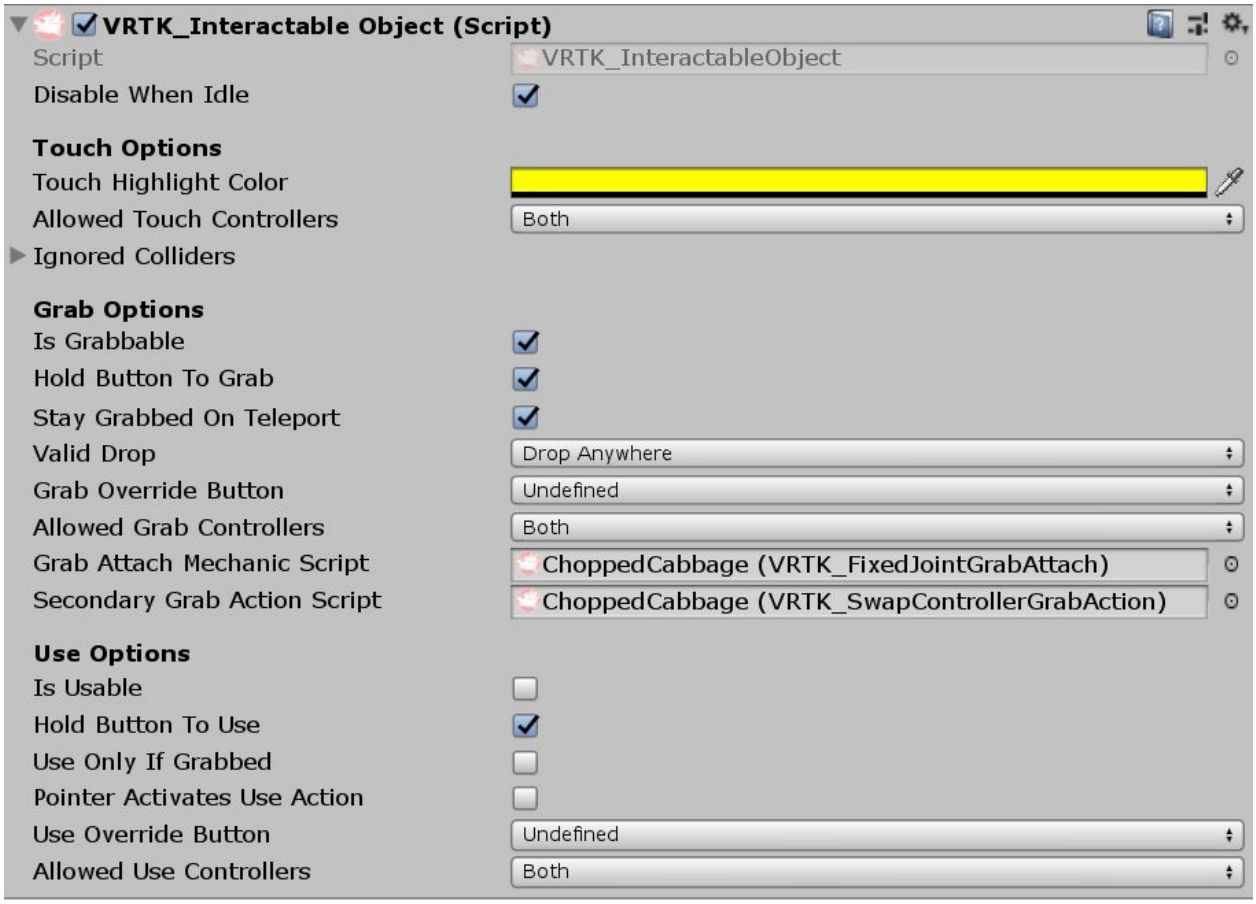


Figure 4.1 : The VRTK_InteractableObject component, as seen in the Editor

The central component of VRTK is the `VRTK_InteractableObject` component, mentioned above. By default, this component allows players to pick them up with the grab button on their controller and "use" them by pulling the trigger. Both of these features have callbacks associated with them such as `InteractableObjectGrabbed` and `InteractableObjectUngrabbed`, which other scripts can add their own listeners to. The component also allows for other types of grabbing behaviors by extending the `VRTK_BaseGrabAttach` component, or using the extensions provided by VRTK. For

example, should the given object be grabbed by the point nearest the controller (the default behavior), be grabbed by the handle (in the case of knives or rolling pins), or be made a child of the controller object's transform (in the case of plates which would otherwise be affected by the weight of their contents)? The core logic of our game is built upon this component, as it averts what would otherwise be a longer development process.

4.A.2. UNet

UNet is Unity's built-in networking API, which was used to facilitate the multiplayer portion of our game. GameObjects with the NetworkTransform component have their positions, velocities, and rotations synced across both clients. The basic server and client connection scripts are also provided by this API. UNet also has the concept of "authority". While the networked components like `NetworkTransform` do sync the object's properties, authority determines which client is used as the basis for synchronizing these values. While this might have been a problem when syncing between two clients trying to manipulate the same items, this was quickly averted with the `GrabbingMutex` script, which gives "authority" to whichever client last grabbed an object (further detailed in section 4.B.1).

4.B. Code Structure

The systems which define our levels were built with Unity's Component system in mind. When objects would have shared features, instead of giving them components which extend the

same base class, the base class can be given as one component and the differing features as different components. For example, Pork and Cabbage both have the `Ingredient` component, which stores what kind of Ingredient they are. However, the former can be cooked on a `Stovetop`, while the latter can be chopped on a `CuttingBoard`. Instead of creating two new classes to facilitate this (i.e. a `CookableIngredient` or a `CuttableIngredient`), they are both given the `Ingredient` component, and Pork is given a `Stovetoppable` component while Cabbage has a `Choppable` component. This prevents problems later down the line when more combinations are needed, like a `CookableCuttableIngredient` or a `CookableCuttableBoilableStirrableIngredient`. Objects are defined by what components they have, and the existence of these components is what other scripts use to detect them. Ingredients, for example have their interactions defined by components which inherit from `IMachinable`, like `Stovetoppable` and `Choppable`. The `Stovetop` only looks for `Stovetoppable` objects, instead of looking for `Ingredients` and checking if they have an `onStovetop` field, or the like.

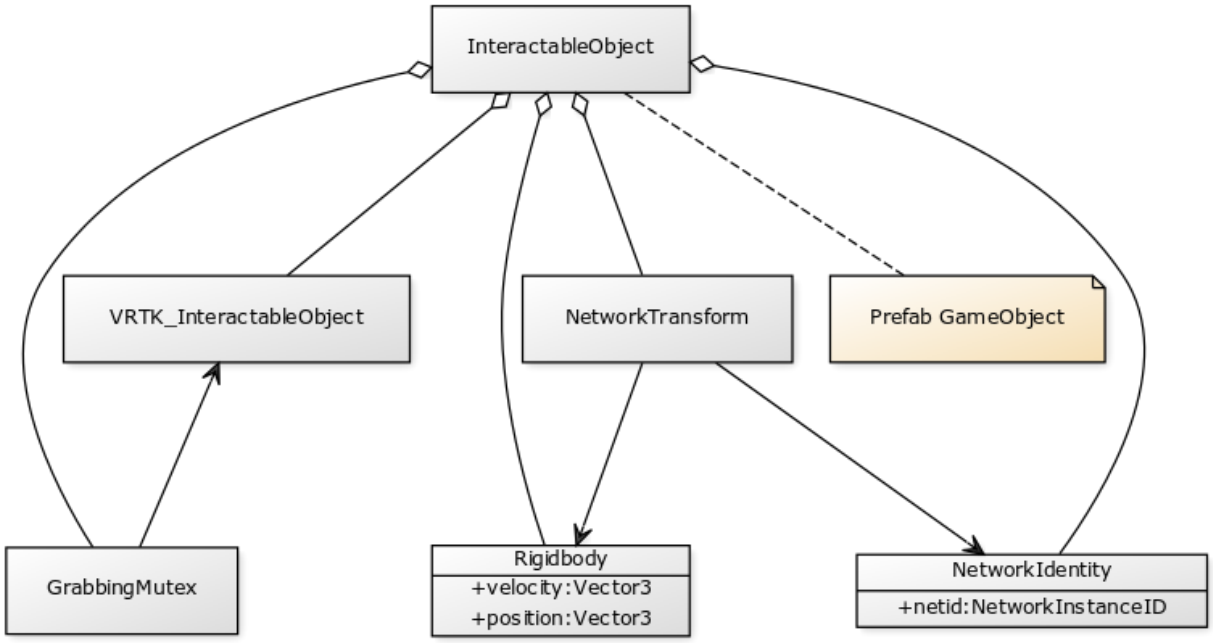


Figure 4.2: Composition Diagram of the shared Components for all interactable prefabs

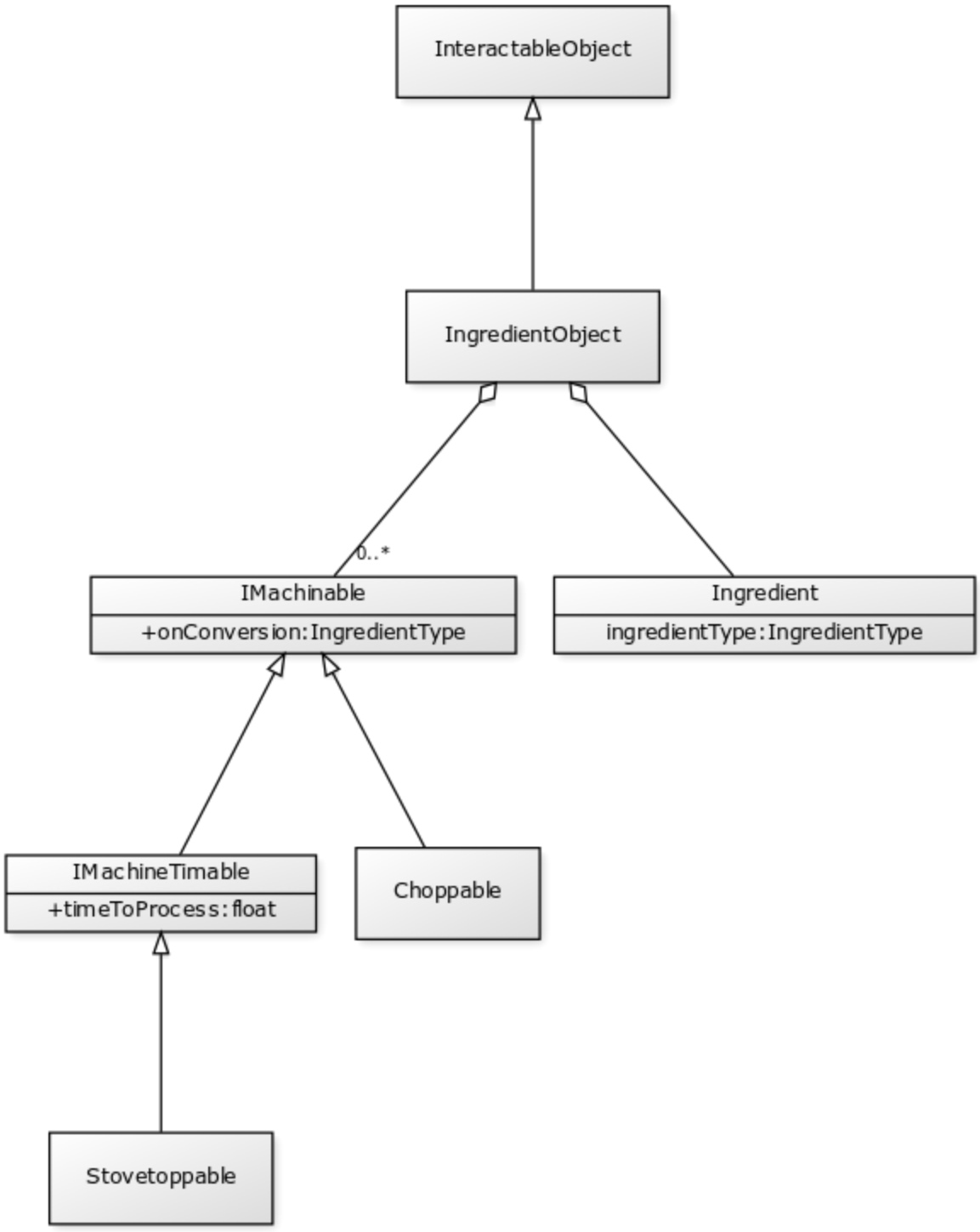


Figure 4.3 : Composition Diagram of the shared Components for all Ingredient prefabs

The `Ingredient` component is simply a container for its `IngredientType`, an enumeration containing every type of ingredient. It does not contain any other information, like an object's mesh, collider, or how it interacts with the cooking implements in the scene, which is instead stored in a prefab, an object template stored as a game asset file. Instead of searching the file directory for this information, the references were stored to these prefabs in the `AproFoodRepo` object, named after the supermarket "Apro" a short ways from our house in Japan. This object is a singleton with a static method `getPrefab`, which takes in the desired `IngredientType` and returns a reference to the corresponding prefab, making `Ingredients` easier to define and change. The scope of all of our Ingredients could be redefined by simply adding to the `IngredientType` enum. Instead of keeping that information solely in a folder as prefabs, we stored it as a list of names, which proved much easier to maintain due to the one-to-one mapping of `IngredientTypes` to `Ingredient` prefabs making it easy to spot discrepancies. Furthermore, changing the value of an enum is generally easier than changing the value of a prefab in Unity's inspector. While Unity does allow you to drag and drop prefab references into public component fields, it proved unwieldy compared to the quick, sorted dropdown menu for enums.

One of our larger hurdles was the netcode we needed. We initially planned on using the UNet Matchmaking system, which connects players via Unity's Relay Server, so much of our game was implemented with that system in mind. However, asplaytesting sessions became longer, there were often automatic disconnections for no reason listed other than "Timeout". We later discovered that the Relay Server has a hard bandwidth limit of 4KB per session on Unity

Personal Edition, which was too small for our needs. Instead, we created a VR-compatible interface for users to enter the IP address of the other, connecting them directly.

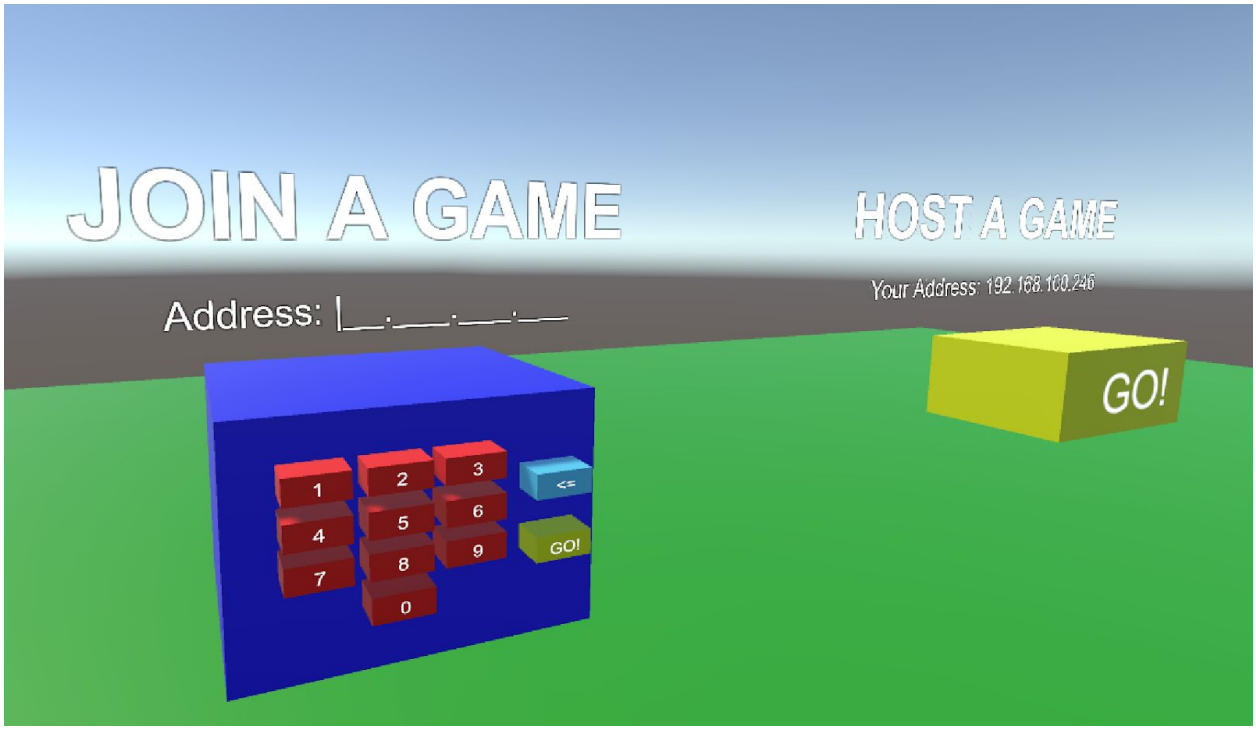


Figure 4.4 : Clients use the keypad on the left to enter the server IP, displayed on the right here and in the level (not pictured)

4.B.1. Networking Scripts

In order to minimize the amount of VR scripts being run on one machine, the player's headset and controllers are only actually in one scene. Instead, we simply emulate the other player's avatar with the `AvatarVisualizer` script, and its corresponding implementation as a prefab. The `AvatarVisualizer` is simply a component which copies the orientation of the

headset and controllers to the corresponding “head” and “hand” models on each client. Because the effects that each player has on both servers are already synchronized, this is the only remaining data that needed to be synchronized between players.

As mentioned earlier in section 4.A.2, UNet uses the concept of “authority” when synchronizing objects between clients. Whichever client has authority over a networked `GameObject` has primary control over its properties, the remaining clients copy those values to their own instances of the `GameObject`. Every `Ingredient` and `Plate` in our game is an instance of a networked `GameObject`, and those change hands very frequently, which begs the question: which client should have authority? We decided that authority should go to whichever client was holding the `GameObject`, since that client would be the one manipulating it, which brings us to our next problem, the edge case when two players would try to hold something at the same time. The solution to both of these issues was the `GrabbingMutex` script. UNet provides a C# attribute [`SyncVar`] for primitive data types, which keeps the value of a given variable constant across clients. `GrabbingMutex` has two public boolean fields, `isHeld`, which is a `SyncVar`, and `isHeldLocal`, which is not. Upon being grabbed, both of these values are set to true by that client. However, only `isHeld` is set to true on the other clients. Therefore, an object can be held by a client if and only if `isHeld` is false or `isHeldLocal` is true.

```
[SyncVar] public bool isHeld;
public bool isHeldLocal;
```

```
void OnGrab(){
      isHeld = isHeldLocal = true;
}
bool CanBeGrabbed(){
      return !isHeld || isHeldLocal;
}
```

Code Figure 4.1 : Excerpt from the `GrabbingMutex` script

The final, but most important, networking script is the `ChefController`. This is the sole script through which clients can send `Commands` to the server. Every public function in `ChefController` is a `Command`, a function which is always executed exclusively on the server. The `ChefController` has a static property "`local`", which returns the instance of `ChefController` in the scene which represents the local player, and the only direct line of communication to the server. With this pseudo-singleton, the rest of the objects in the scene can also affect the server, routed through this script.

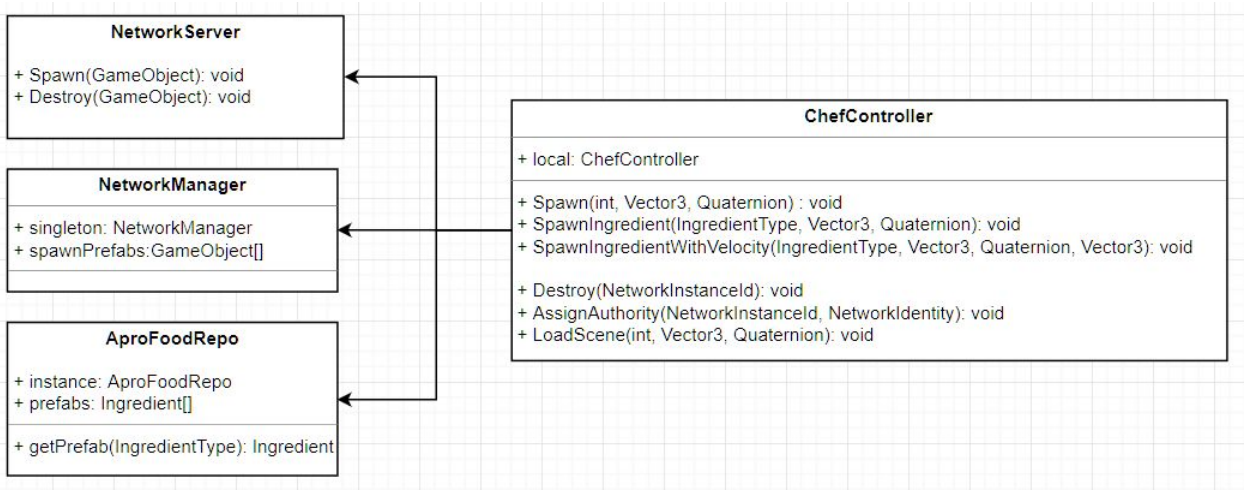


Figure 4.5: A UML Diagram of `ChefController` and its neighbors

The script has six public functions, three of which are used for spawning in objects on the network, accordingly named in sequence: `Cmd_Spawn`, `Cmd_SpawnIngredient`, and `Cmd_SpawnIngredientWithVelocity`. All of these functions use the `NetworkServer` static class to create objects across all clients.

`Cmd_Destroy` is similar to those three, but instead destroys the `GameObject` with the provided unique `NetworkInstanceId` on the server. `Cmd_AssignAuthority` also takes a `NetworkInstanceId` as well as a `NetworkConnection`, and gives the client with the given connection authority over the corresponding `NetworkInstanceID`. This is primarily called by `GrabbingMutex` to give clients authority over the objects they are holding.

4.B.2. May I Speak With the Manager?

The game itself is run through the aptly named `GameManager` script. This script controls the game flow, as well as the difficulty curve. Every 30±5 seconds, the `GameManager` spawns a burst of tickets for the players to complete, according to the current "`difficulty`" value, a number from 0 to 255. In order to easily manipulate difficulty curves for testing we made use of Unity's very convenient gradient editing tool. The `difficulty` value at any point in the game is simply the red value at the corresponding point on the gradient.

$$Difficulty = Gradient(\frac{t_{elapsed}}{t_{maximum}})$$

Formula for calculating difficulty

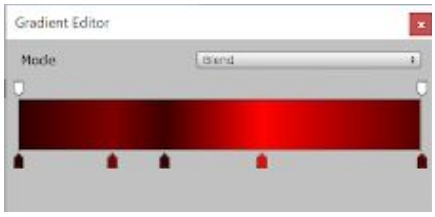


Figure 4.6: The Gradient Editor, depicting the difficulty curve from level 1

We used the Gradient Editor in this way to save time, both by utilizing one of the only in-engine methods to store an array of linear interpolation functions and by creating an intuitive visualization of the difficulty progression in a given level. The gradient shown above is made of four color interpolation functions, functions which represent a transition between two color values over time. Each notch on the bottom portion of the gradient represents a end-point of the previous interpolation and the start-point of the next one. We used a color gradient instead of an array of values representing each point because using visual cues like the differences between colors is more intuitive than trying to interpret raw numbers.

Next is the `TicketManager`. While the `GameManager` assigns the tickets, the `TicketManager` is what stores the recipes for each ticket, and evaluates whether or not submitted plates correspond to a ticket currently being served. A `Ticket` is simply a struct containing a list of the required `Ingredients` as well as the provided time to complete the ticket and its difficulty, which doubles as a measurement of score upon completing the ticket.

5. Art

The art style of this game is low-poly, for a few reasons. Chiefly, our team had no experienced artists and little real art practice, so the simpler the models, the better. Secondly, VR can be fairly CPU-intensive, so simple models and simple artwork allow this game to run on more systems.

Our art design choices come from having easily recognizable objects and ingredients. This is to help ensure that players can easily distinguish between different game objects as they are running around the level. As such some items are in higher details, for example the cabbage and pickled vegetables. However, to avoid a hyperrealistic feel, some assets were made to be more cartoonish as seen by the pork and avatar models.

5.A. Here be Art

A lot of the art started off as concept art in a notebook. These sketches are then used as references while modeling. The software used was primarily Autodesk Maya, and Adobe Photoshop due to familiarity with these programs. As mentioned above, these models are

low-poly in order to keep the scope manageable. Furthermore, only color was applied to the renders as opposed to textures as that would have taken a significant amount of more time for the stand-in artist to learn and implement. Some examples of the concept art include:

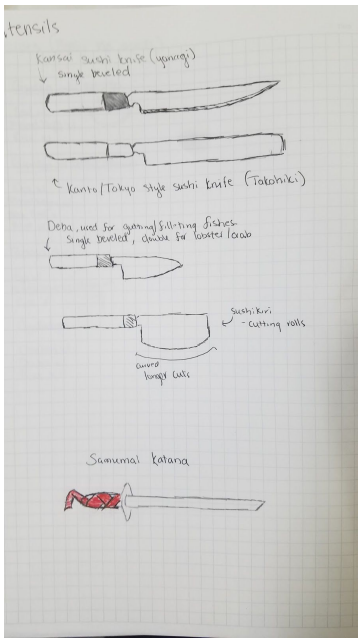


Figure 5.1 : Different Knives

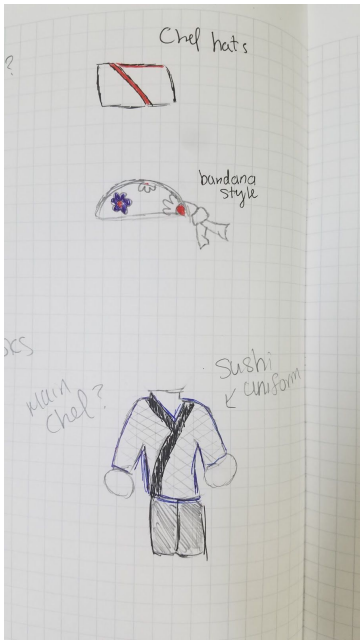


Figure 5.2: Uniform for the player avatars

Maya was used due to its easy accessibility to students compared to other 3D modeling software that required payment. In addition, the software can export models into popular formats such as .obj and .fbx that work well with Unity, the game engine that we are using for our project. Maya was used to model and color every custom object in our game including kitchenware, appliances, ingredients, and the player avatar, outside of a few items which were a primitive shape like a cube or cylinder, or were a single, solid color.

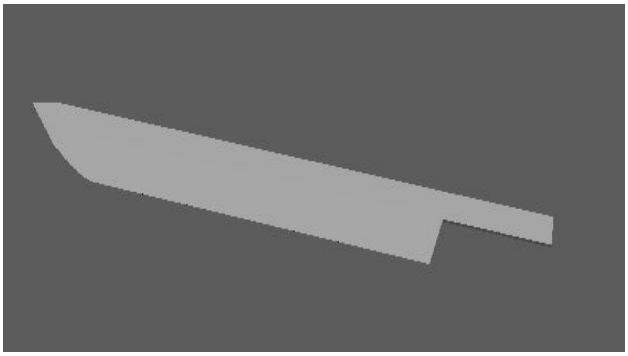


Figure 5.3: Untextured Knife Model

To easily color the models, UV maps were used, as Maya provided several useful features. The software would automatically create a UV map of the model that would then be exported into a .png for editing in Adobe Photoshop. UV mapping is a process in which a 2D map is projected onto a 3D object. This allows the artist to give the model more life by using textures by allowing it to have colors or the illusion of a textured feel. For example, the knife above would have a UV map shown below.

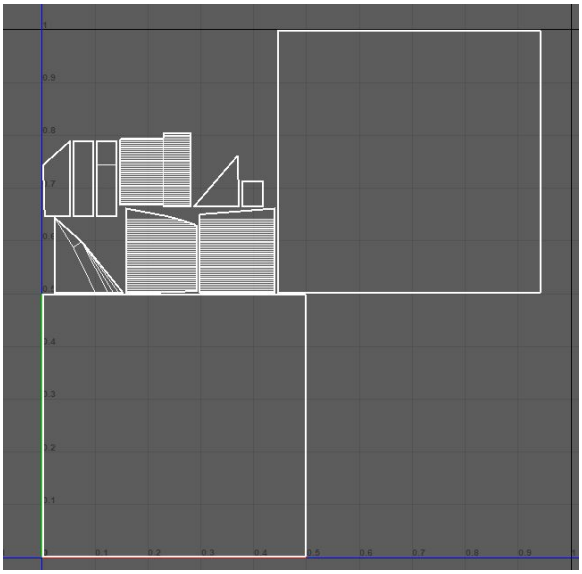


Figure 5.4: UV Map of Knife

Once this map is exported to a .png file, the next step would be to load it into Adobe Photoshop in order to put color on it. If exported correctly, one would see the UV outlines on top of a transparent background. The artists' job here would be to determine which set of UV outlines corresponds to what portions of the model in order to color it correctly. Generally, the next step would be to create a new layer below the UV map and color accordingly. The next step would be to remove the transparency in the image and then loaded into Maya to check for mistakes or places to tinker. The editing process is repeated until satisfaction.

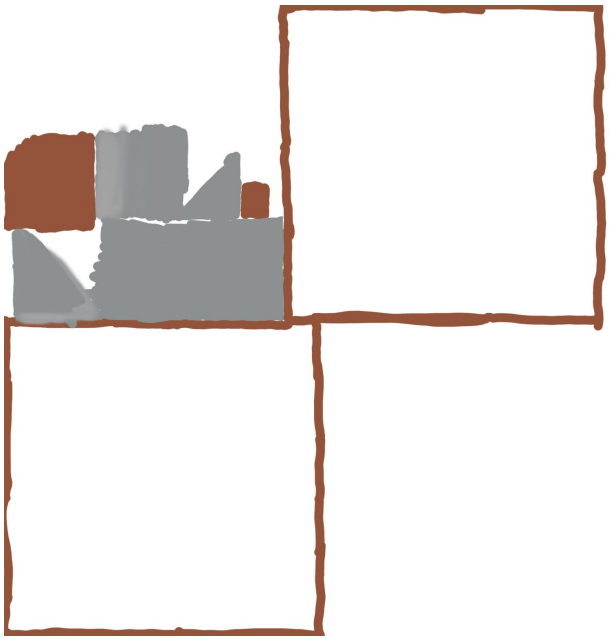


Figure 5.5: Colored UV Map

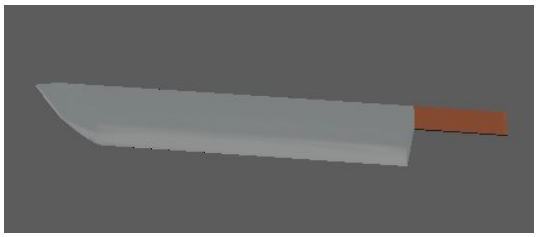


Figure 5.6: Final Knife Model

After the textured model is deemed satisfactory it would be exported as an .obj or .fbx and loaded into Unity which has built-in support for both of these formats. One issue, however, is that Unity does not like editing materials that are not created by the software itself. Thus the next step would be to create our own material and apply our new UV map in the engine itself.

In order to understand what kind of art we would like in our game, research was done on popular casual games. Casual games belong to a genre that generally provides simpler rules and reduces demand on time required to learn skills necessary skills. Many of these games, including *Overcooked*, *Jackbox Party Packs*, and *Wii Sports*, and others generally will have cartoon-ish artwork with more simpler models. These were found to be more associated with easier to play games versus more detailed art-styles which normally represented longer and harder games.

6. Testing

6.A. Early Testing

In the early phases of testing, the game had yet to be feature-complete, and had some networking bugs that weren't caught until later. As a result, opinions of the game were much more negative at this time. They were also more critical, which helped us track down and eliminate some of the harder to detect bugs. Players enjoyed our early playtesting sessions, but also feeling confused, frustrated, and occasionally nauseated, all of which conflicted with our experience goals. Point for point, this confusion stemmed from lack of clear instruction either in control schemes or game mechanics, the frustration from errors preventing the game from working as intended, and the nausea from these same errors slowing down the computer and causing network desynchronization. The error that caused the largest problem was object collision. Players would routinely find themselves bumping into ingredients that had fallen onto the floor. In VR, having the player haphazardly changing position is a surefire way to cause nausea and discomfort, and given the number of items that can be in a scene, this was a recurrent

problem and a contributing factor to the differences in later testing. All these technical issues combined created one major design issue that affected the core of our game: Players were not communicating with each other. We hypothesized two reasons for this behavior. Firstly, the game's objectives weren't clear enough, which might have had players mainly communicate with the experimenters, estranging them from each other as a result. Secondly, it also wasn't necessary for them to communicate in order to complete their goals. While it might have saved time, the balance of the game was still off at that time, so players may not have seen any value in coordinating with each other.

Despite the myriad issues in these early phases, opinions of the game and its potential were generally positive. The object interaction mechanics were engaging enough to carry the experience on its own. Players would often have fun passing ingredients between each other, or trying to use kitchen tools from across the room. They also enjoyed the main gameplay, preparing ingredients as fast as possible to assemble meal tickets. Once they understood what was going on, players would quickly try to complete them, though usually on their own instead of with their teammate in the earlier levels. By level 4, the arrangement of the kitchen made it significantly more difficult to complete as a single player, and players began passing ingredients and items between each other, which was the goal of that level. Despite this, a few players would try to actively obstruct the other players, but competition is an inherent problem with cooperative game design as was described in section 3.B.3.

Overall, the early testing phase was indicative of the direction the project would go in. The focus was on fixing the fundamental issues brought up in our responses and building on what our testers found enjoyable. We started analyzing the problem points and found it to be very important to convey the most basic mechanics. The new tutorial stage would teach basic movement and how to go vertically, as well as pick up items off the ground without bending down much in real life. In the early stages the recipe guides and machine I/O diagrams informed players of the general flow of how the recipes could be made, and use their intuition on how each machine worked. The bugs that made players nauseous were quickly remedied as well. After these changes were made, we could then continue our testing, now able to put more weight in the results.

6.B. Later Testing

Now that the large problems were out of the way, we could continue testing. In this stage, players were communicating more than they were before. The feelings mentioned in the early phases, confusion, frustration, and nausea, had largely been resolved. All the levels were fully playable, and players found them and the mechanics therein enjoyable. Because the major issues were resolved, testers were now more able to review the elements of the intended gameplay instead of the bugs, and players thoroughly enjoyed the provided experience. Not only that, but all of our primary experience goals were met: players were physically engaged, planning ahead, and actively communicating. Unfortunately, while the game was feature-complete at this time, testers could only remain in VR for about 30 minutes, making later levels harder to test without

skipping over tutorial content. As a result, while opinions of these levels were still good, we have less information about the enjoyability of the last four levels than we do about the first four.

Our testing phase was a critical time for our project, when we could see our game through the perspective of our users. It revealed several problems in our project which we could not have found otherwise, in both our code and our design. From our results in later testing compared to our earlier testing, it's fair to say that the game had significant improvement between the two phases. Players went from barely talking in the early levels to working together to try new things and better ways of cooperating.

6.C. Results

After each playtesting session, we had our playtesters take a short survey about their experience, what they liked, what they didn't, and what they would improve or add. Many of these suggestions made it into the game, and seemed to greatly boost opinions of the game as testing continued.

6.C.1 Feedback and Suggestions

The first major addition was that of a recipe list. Initially, the only information players had on what they were making was the name of the final product, which didn't make clear the process of making those ingredients or even what those ingredients were. These suggestions resulted in the creation of a large recipe list to clarify the individual components of each meal, and the smaller diagrams which displayed the inputs and outputs for each machines.

The second change was the switch from the grip button to the trigger button. Players noted that it felt awkward to use the grip button for long periods of time, because of its position near the base of their grip. As a result, we decided to switch to the triggers, located by the pointer and middle fingers, which in turn led to the experience being much easier on the player's hands, allowing for a much more enjoyable experience.

We also created a tutorial to teach players the nuances of movement and interaction. The tutorial took the form of a miniature obstacle course, with clearly indicated directions, instructions, and examples of proper movement. Players who went through the tutorial were often more efficient in their movements. Earlier players were often more cautious, moving only small distances at a time, but after the tutorial was added, players felt more comfortable making drastic and quick motions, often launching themselves across the room in order to complete their objective faster.

6.C.2 What They Didn't Like

No great game comes without faults. According to the survey results, there are a few key things and a few quality of life features that the players did not approve of. These small issues include unlimited spawning of cabbages that flooded the game room (fixed in later iterations), accidentally teleporting, and the recipes not being clear.

The foremost complaint was the movement system. Prior to changing the movement button to the trigger, it used the grip buttons which people felt were unintuitive; people were used to using the trigger instead. The movement system was buggy at first and caused issues with players being able to move at a fast pace inducing nausea. Furthermore they would like to be taught how to move in a tutorial which was later implemented. Since the button to start the process of moving is the same button used for grabbing objects in the world, people were confused initially in how to navigate with items other than throwing them, causing frustration due to consistent accidental dropping of ingredients. The ingredients would drop whenever the ingredient touches a solid object whether or not the player is still holding onto the ingredient. Not only did this cause inconvenience for the players, it also occasionally rendered the ingredient unusable as the player can no longer pick it up. When players walked on top of ingredients, their cameras would shake violently causing nausea.

Another major complaint was that the game was not fully working during the initial testing phase. Machines would only work or ingredients would only spawn on either the client or the server but not both. Completed dishes could not be submitted even if it's correct unless it was submitted on the server side. In addition, the players could also take things out of the hands of their partner. These issues were frustrating to the players who were already confused on what they were doing and believed that they did something wrong.

Things were confusing for the players at first in terms of what they should be doing in the game, how they should cook the ingredients, and how they need to put it in the bento box as it

was originally a snapzone that the player had to manually place the ingredients in. This was changed so that it was more convenient for the player by only checking for collision with ingredients and placed into its respective spots. The later testers no longer had these complaints due to graphics being displaying which ingredients are what and which ingredients makes up a recipe.

6.C.3. What They Liked

Regardless of the errors, players found the game to be enjoyable. After the buttons were changed to the triggers from the grip buttons, all the players found the game to be significantly more intuitive (3 out of 4 instead of the previous 1 or 2 out of 4). When the collision with dropped items was fixed, the average nausea level dropped from 2.75 to 1.2 on a scale of 1 to 4. With more intuitive controls and without major sources of nausea, the players become significantly more involved, making much larger movements in the real world in an attempt to perform better and move faster in-game.

The players particularly enjoyed that the interaction of objects with the world, particularly the more open and intuitive actions players can take, such as with the chopping board/cutting knife, that by placing an object onto the board, they can pick up and move the knife onto the object to cut it. They also like throwing items to players and around the kitchen, regardless of whether or not the item ended up anywhere useful.

Most of all, players enjoyed the wacky machines. Though the issues with the two currently implemented were only fixed toward the end of testing, players liked the outlandish visual design mixed with the simplicity of use, using only the two primary mechanics in the game: grabbing and throwing. The most desired feature to be added was more wacky machines, followed by more food types.

7. Post Mortem

7.A. What Went Right

Despite lacking in terms of game length, numerous aspects of this project went well in both team dynamics and the actual project in question. The game is fully functional with levels following the flow system described earlier in (Figure 3.1) that introduces the player slowly to new mechanics and how the game is run through the introductory tutorial levels. Our framework is robust and very extendable for programmers who are interested in developing their own recipes and levels, as detailed in section 4.B. Members of our team have grown significantly in our respective inter-personal and professional skills. However, most importantly, those who have played our final game were observed to have positive feedback and enjoying their gameplay experience, a significant change from the first round of testing, as shown in section 6.B.

Technically, Sean has grown proficient in the use of VR Toolkit and UNet, the two APIs central to the game's function. UNet's authority-based synchronization proved unwieldy

initially, but he came to understand it over the course of the project. Likewise, despite VRTK's occasionally wanting documentation, he became proficient in its uses and workings.

Professionally, Sean learned the dangers of over-optimization, planning too far ahead led to wasted effort that would have been better spent elsewhere. He did successfully create an extensible backbone for the game's primary functions. On paper, this was a good idea, as this would expedite development of the later worlds, levels, and machines we had planned. In practice, due to only having one world and eight levels, many of those extensions were only used sparingly, or not at all.

David has gained more familiarity with the Unity Engine, particularly in how 3D objects interact. He has also increased his understanding over how interaction in VR is performed and how to design around it. Having learned good code design practices but lacking in opportunities to use them, David gained increased confidence in his programming capabilities. David gained an increased appreciation for planning out code design beforehand.

Having had difficulties with communication in a team, particularly when ideas differ, David improved his abilities to voice his opinions and to provide criticism in a positive way. By creating a set of design goals early on and making changes as needed, conflicts can be resolved by analyzing the different options' effects and how they best meet the goals.

David improved his ability to design levels and features by analyzing them from a players' perspective, and examine which features would best meet the experience goals, and how to adjust features based on user feedback. By using concepts of signifiers, affordances, and schema, the machines for processing ingredients were made to be more intuitive without requiring the use of language. By searching for potential confusion in the game's parts, the players can be given the solution visibly (the recipes) or can be learned through experiments and accidents (items fall out of upside-down bentos).

Lastly, this was David's first foray into multiplayer game development. He gained an increased understanding of issues associated with networking, and how to troubleshoot and fix them. He also improved his ability to design cooperative gameplay, particularly in perceived equality despite asymmetry of skill, distribution of information to encourage communication, regular gameplay interaction that is greatly expedited with two players.

Jimmy has grown in a few technical areas, including developing new skills and familiarities with C#, Unity, and 3D modeling, primarily Autodesk Maya. His greatest improvements lie within Unity's component system and using Maya as they were his most used assets within this project. The main skills learned in Maya was the ability to use the tools to create 3D models from scratch at a more efficient rate than prior. UV Mapping was something he had trouble with prior to the project as it was difficult to understand well especially with more complex models. With the aid of colleagues and numerous tutorials on Youtube, it became increasingly easier to understand and implement properly as well as efficiently into the project.

In terms of inter-personal and professional skills, Jimmy has grown to become a better leader and delegator within the team, ensuring other team members have a goal to work on, reducing down-time. In order to do that well, he enhanced his communication skills to more accurately reflect his desires and delegations without sounding like a boss, but rather a leader. It is of utmost importance that the team members do not feel a sense of an exacting overtone, demanding that they do a task, but rather a collegial connotation, where he will work together in order to achieve the task at hand.

Jimmy has also learned that sometimes it may be better to use assets on the asset store as a surrogate for implementation of the art in order to provide a full visual experience rather than have few completed assets. Meanwhile the assets that are completed are higher in quality, contrasting them with the rest of the incomplete asset creates a dissonance in immersion between the player and the game, due to the inconsistency in artwork.

7.B. What Went Wrong

Unfortunately, while we did many things right, we also encountered many obstacles. First and foremost, our scope was significantly higher than our allocated resources had allowed. Secondly, the time for us to finalize our design took longer than expected due to the original design lacking direction and holding unclear experience goals, cutting our development time shorter. Lastly, some core features of the game were not able to become implemented with the

desired quality due to the time constraints. Finally, we had devoted too much time to the framework rather than to the design and implementation of the actual gameplay.

Originally our team had scoped for three full-time programmers to simultaneously work on the project. However, we had forgotten the fact that art assets were crucial to a game of this nature. Due to this mental lapse, one of our members had to become a part-time programmer and spend most of their time creating the art assets. Unfortunately, time was further restrained by the artist having to primarily learn the skills necessary to implement the art assets. As our allocated time was coming to an end, art asset production was halted in exchange for an all-hands on deck approach to ensuring we have fully functional, playable levels. As thus, the art assets are not as consistent in relation to each other as desired.

Implementation needs to start much earlier in the development cycle. The implementation stage was too far into the development cycle that it did not afford us enough time to properly implement the features developed into our game. Our team had believed that every feature was working well that it would be flawlessly integrated into the game but was proven tremendously wrong. The majority of the time dedicated to implementation of levels ended up being debugging of features and of the framework errors that were not apparent until it was being pieced together.

Before the design evolved into its current iteration, it was originally a 2-4 player game where only one person is in a VR while everybody else is using a keyboard or controller. However, there were several problems we came across with this design. It became excruciatingly difficult to find a good balance in gameplay between the VR player and the non-VR players (will be referred to as KBP or Keyboard Players for brevity). As the KBP would vary in number we

felt that there was no good way to split up the game's tasks that would be enjoyable with the differing number of players. For instance, the VR-player were to be the one in charge of cutting proteins and vegetables and plating while the KBP would cook. Doing a test run of this iteration of balance showed that the VR player did not have much to do while the KBP had a significant amount to achieve especially if they were not accompanied by other KBP. Through preliminary testing, we also realized that by having the KBP play different minigames than the VR player, it resulted in little interaction between the KBP and the VR player, which would be no different than if they were playing entirely different games. The variable number of KBP would be a nightmare to test, as each variation in player number (2-4) would need to be thoroughly tested, and we did not have the time. It was also found that the VR player's role would not be more significant than that of a wacky machine. The VR player would only do a simple task when an item needs to be chopped, which is only once or twice per ticket, and only the KBP would have control over when this occurs.

The intended minigames were also no different from existing ones, and would essentially be KBP playing *Overcooked* while the VR player played *Beat Saber*. The resulting game would have brought nothing new to what already existed, and would not have done any of those things better than their source inspiration.

Lastly, the original concept was lacking in design direction. The experience goals for the players were muddled and unclear, and the only one seemed to be that it was to be "fun". As a

result, concepts for features that had dubious benefit were proposed and added on, exacerbating the problem.

One of the core features of a game is the audio. Audio is important for immersion into the game yet was left out of the final product. The decision to do so is because there was not the time to balance out sound effects and music in the game to an acceptable level. In order to reach this acceptable level, testing would need to be assured that specific sound effects and music would match the tones and experience goals that were planned. In addition, the SFX had to be immersive and not annoy the players with repetitiveness or unpleasant noises. Having sound in the game can help players with spatial awareness; a stove sizzling allowing a player to more easily locate it. Also due to the lack of sound, there is minimal auditory feedback in our game to alert the players of changes to the world state, such as successfully submitting a dish, an ingredient that has finished cooking or burnt, new tickets appearing, and the location of the level end screen. This along with numerous other features would have been implemented if our limited time and resources had been more merciful.

7.C. What We Would Change

7.C.1. Implementation

There are many things that were planned to be added but have not been implemented. One significant design choice was for the game to be universally available. In order to accomplish that goal, the game would have need to consist of almost no text at all, in essence purely graphical. An example is the recipe board in every level where the ingredients were

portrayed by graphical icons. Unfortunately, the name of the dish is still written in english. The change that would fit the design goal would be to have the dish name be changed into an easily recognizable icon. However, this can prove to be difficult when dishes are very similar. For instance, gyudon and katsudon when in a bowl can look identical. Currently, in the tutorial the movement controls are written alongside a hologram. A more desired design choice would be to show the scheme of "pulling yourself" without having to state it in words.

In addition to making the tutorial universally available, it is annoying for players to need to replay the tutorial level everytime they would restart the game. To remedy this, a persistent variable would need to be implemented that would check whether or not the tutorial level has been completed before on the local machine. From here, the player would just teleport to the world select upon booting up the game rather than starting at the tutorial.

To further make *Ben-tomo!* into more of a "game" than a simulation, a high score feature could be added that will be persistent through all playthroughs. All of the features necessary to implement this are already in place, save a persistent file to read from. Adding high scores would motivate players to figure out better strategies or improve their own gameplay in order to achieve higher scores. This does require tweaking how the scoring system works from being a set score per dish to something tangible such as points based on time remaining to give smaller improvements more visibility.

More worlds would be implemented such as a kaitenzushi world where players would have to create sushi dishes at an incredibly high rate. Other worlds include a ramen world where the player would have to mimic the production of making egg noodles and broth, and a gyudon level where order of placing ingredients matter. The aesthetic of each world would match those of an actual Japanese kitchen serving that kind of food, and certain characteristics like the typical size and shape would be factored in.

To add a sense of progression for the players, a level unlock feature would be implemented so that players cannot simply start at the later levels as in the current design. This would help the players feel as though they are making more progress due to having to have succeeded in completing the prior levels, as well as a way to ensure that the previous lessons were learned. This feature would be as simple as meeting specific criteria such as reaching a specified score, beating a certain time, or delivering a certain percentage of the dishes.

More wacky machines (some of which are shown in Appendix 9.B) would be implemented to give players more unique ways to cooking their ingredients. Many of these machines did not find a spot in the implemented levels as they either did not fit the intended design of the respective levels or due to time constraints on building more levels that would include them. Some, such as the chicken that lays eggs, were not fully implemented as the mechanics of how eggs would work was not fully fleshed out.

Finally, the ability for players to better communicate with one another is another feature that would greatly increase the quality of life for the players. In order to keep universality, voice chat would not be implemented but rather being able to point (using a virtual laser-like line) to objects and having the other player see it. Currently only the local player could see the laser pointer when it is being used. Furthermore, in order to convey emotions, an emoticon wheel could be implemented for players to communicate emotions with their partner.

7.C.2. Testing

The amount of testing that occurred was inadequate. Not enough testing was done during the development phases in order to figure out what is intuitive for players or what is detrimental to their gameplay experience. In addition, the lack of repeat testers meant it was more difficult to reach the later stages in time. There were two full-scaled testing schedules but unfortunately in the first set of testing there were too many bugs for the players to be able to proceed. The second testing set had players spending most of their time learning the controls and how the machines worked. Ideally there would be a third testing session where players can attempt to play through all of the levels without a time restriction to reach the later stages.

7.C.3. Immersion

To improve immersion, the visual effects and audio effects must be at a balance in order for the player's mind to believe it is inside the reality that is the game world. As such it is crucial for the visual art to be consistent all throughout the game world and the audio to be natural and

immersive. The current game lacks in an universal art style and lacks audio entirely. In a future iteration both of these issues will need to be addressed.

To help ensure a higher quality and to keep a sense of uniformity in the art style, an artist would be recruited into the team that will focus primarily on the creation and implementation of art. This will ensure that the art styles stay consistent with each other and that more assets will be completed thereby increasing the immersion into the world. As such, many placeholder icons and textures will be replaced with finished versions. An alternative solution would be to use assets found on the store and credit the original authors.

Audio can be incredibly tricky to implement well in a game. It is crucial that the audio provides feedback in such a way that it does not detract from the immersion. Furthermore it cannot be annoying to the player (except in the case that it is an experience goal) otherwise it will cause discomfort and frustration which defeats our desired experience goals. As such, more time would be allocated to researching to implement a soundscape that would immerse the player into the game world.

8. References

Behind the Name. (n.d.). *Benjamin*. [online] Available at: https://www.behindthename.com/name/benjamin/ [Accessed 14 Oct. 2018].

Behind the Name. (n.d.). *Japanese Names*. [online] Available at: https://www.behindthename.com/names/usage/japanese/ [Accessed 14 Oct. 2018].

Bergson, Henri (1980). "Laughter." Trans. Wylie Sypher, in Comedy, eds. Wylie Sypher. Baltimore: Johns Hopkins University Press.

Bolois, J. (2015). *The Great Divide: How Sushi Culture Differs in America Versus Japan*. [online] First We Feast. Available at: https://firstwefeast.com/eat/2015/10/differences-between-sushi-in-america-and-japan [Accessed 21 Sep. 2018].

Cooking Mama Ltd. (2008). *Cooking Mama: World Kitchen*. [Wii].

Ghost Town Games, Ltd. (2018). *Overcooked 2*. Team 17 Digital Ltd. [Computer software].

Hobbes, Thomas. (1840). *Human Nature, in The English Works of Thomas Hobbes of Malmesbury, Volume IV*, ed. William Molesworth, London: Bohn.

Kant, Immanuel. (1951). *Critique of Judgment*. J. H. Bernard, Trans. New York: Hafner.

Kumar, Matthew. (2012). *5 Problems with co-op game design (and possible solutions)*. [online] Available at: https://www.gamasutra.com/view/news/181576/5_problems_with_coop_game_design_and_possible_solutions.php/ [Accessed 14 Oct. 2018].

Lockem Reality. (2017). *ChefU*. [Computer software].

Miller, G. A. (1956). "The magical number seven, plus or minus two: Some limits on our capacity for processing information". *Psychological Review*. **63** (2): 81–97. doi:10.1037/h0043158. PMID 13310704.

Mills, Harry, Natalie Reiss, and Mark Dombeck. (n.d.). *Distraction and Humor in Stress Reduction*. [online] Available at https://www.mentalhelp.net/articles/distraction-and-humor-in-stress-reduction/ [Accessed 14 Oct. 2018].

Restaurant Engine. (n.d.). *Why Shorter Restaurant Menus Are Gaining In Popularity*. [online] Available at: https://restaurantengine.com/shorter-restaurant-menus-gaining-popularity/ [Accessed 21 Sep. 2018].

Rose, Nelson. (2010). "Addressing Conflicts: Tension and Release in Games". [online] Avalailable at: https://www.gamasutra.com/view/feature/134313/addressing_conflict_tension_and_.php/ [Accessed 14 Oct. 2018].

Švelch, J. Comedy of Contingency: Making Physical Humor in Video Game Spaces. International Journal of Communication 8 (2014), 2530--2552.

Valve. (2011). *Portal 2*. [Xbox 360].

VRTK - Virtual Reality Toolkit. (n.d.). *VRTK - Virtual Reality Toolkit*. [online] Available at: https://vrtoolkit.readme.io/ [Accessed 9 Oct. 2018].

Whirlybird Games. (2016). *The Diner Duo*. [Computer Software].

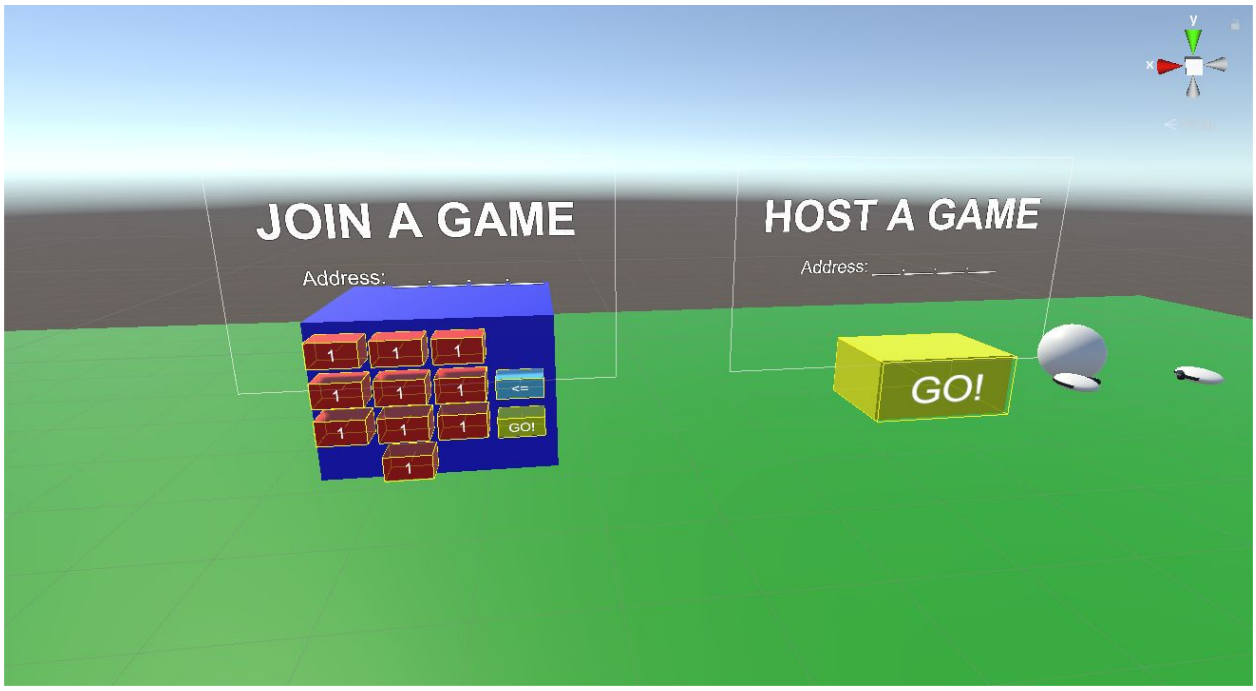
Winkler, Martin M. (2001), Classical Myth & Culture in the Cinema, p. 173. [online] Available at: https://books.google.com/books?id=f4vCQYhKCvcC&pg=RA2-PT112/ [Accessed 13 Oct. 2018].

Unity User Manual (2018.2). (n.d.). *Multiplayer and Networking*. [online] Available at: https://docs.unity3d.com/Manual/UNet.html [Accessed 9 Oct. 2018].

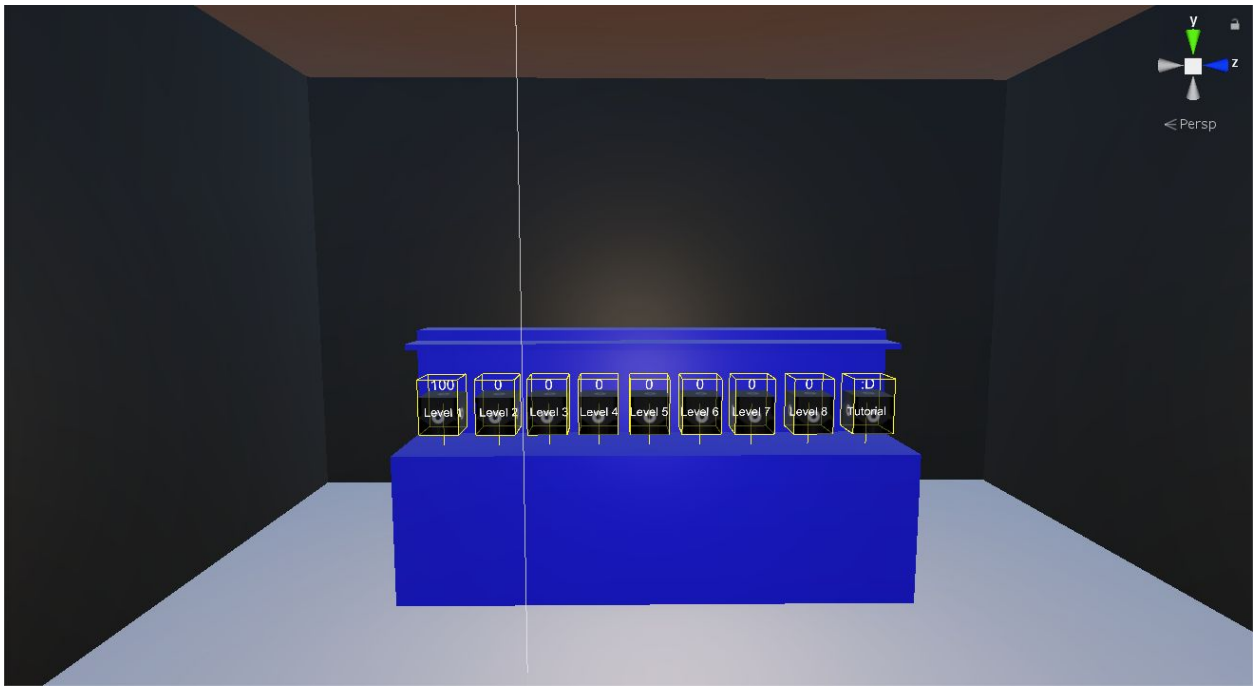
9. Appendices

9.A. Level Designs and Layouts

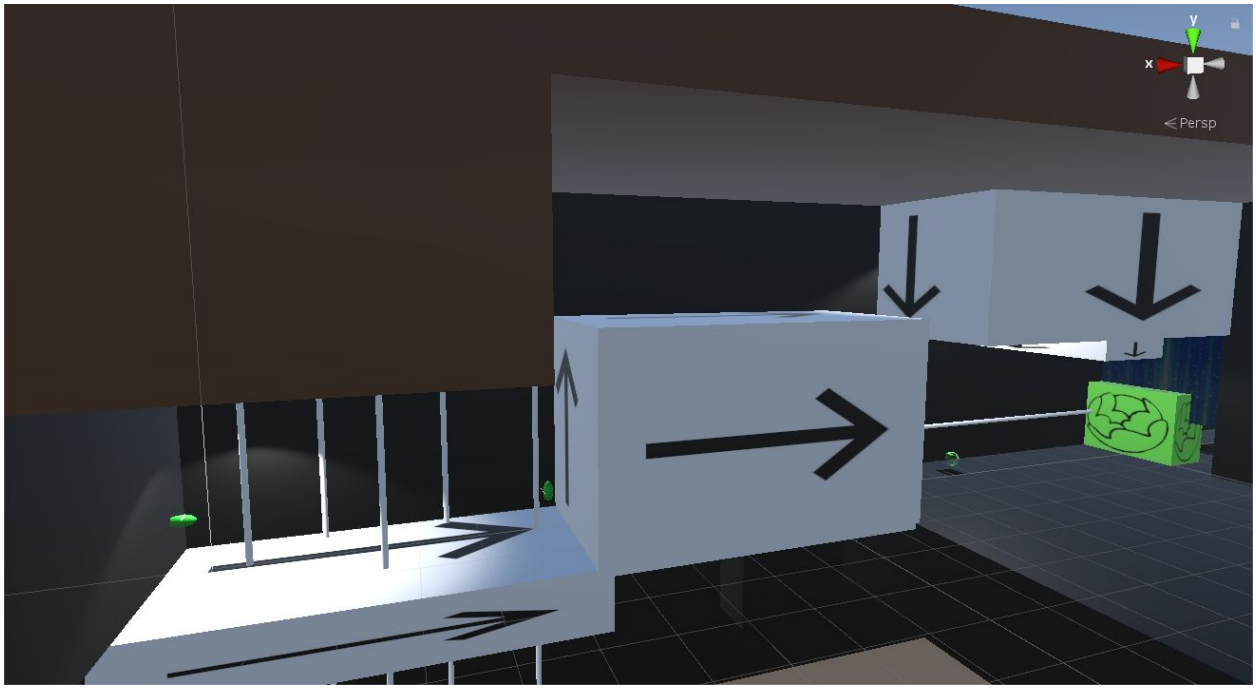
9.A.1. Main Menu Scene



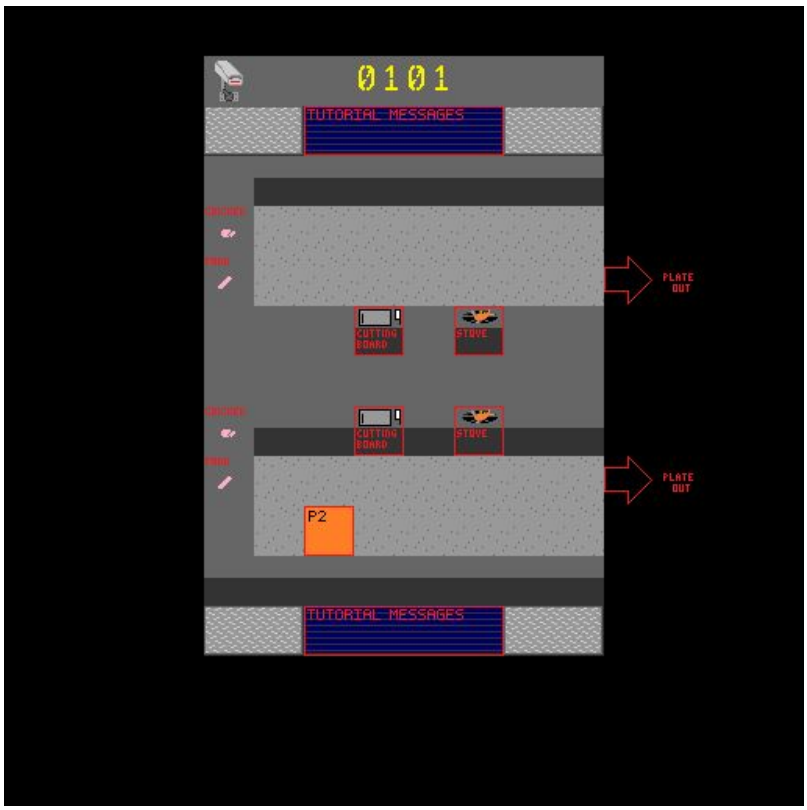
9.A.2. Level Select Scene



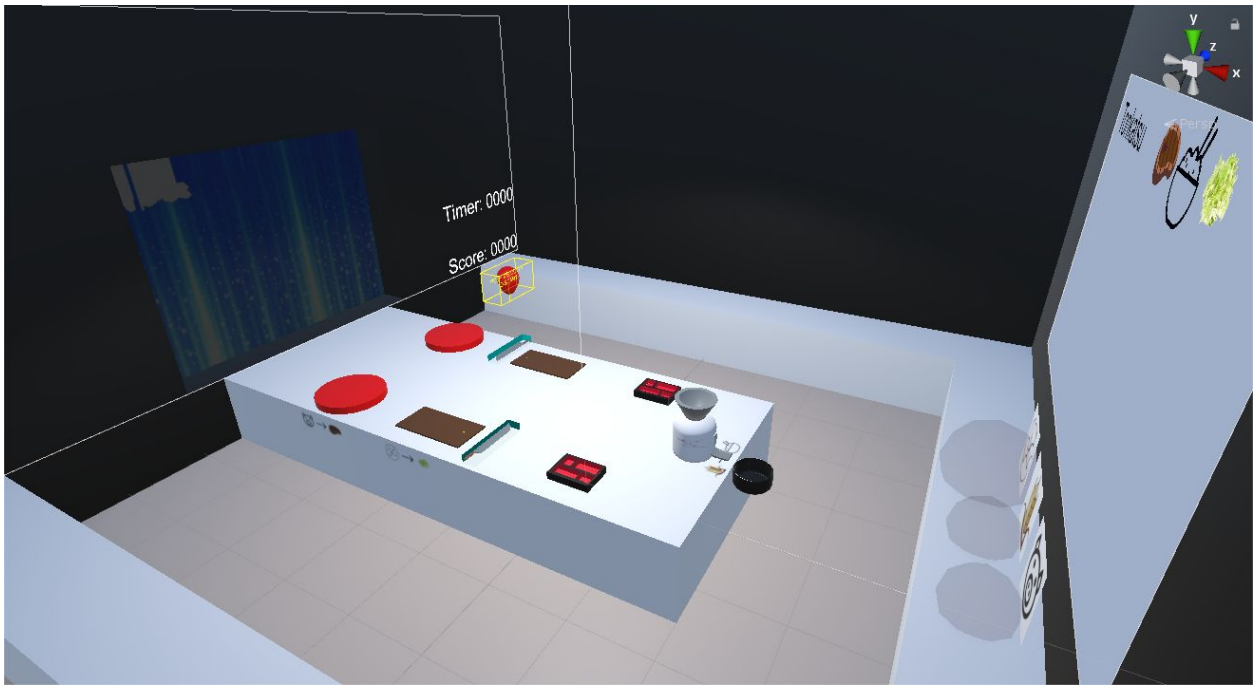
9.A.3. Tutorial Level



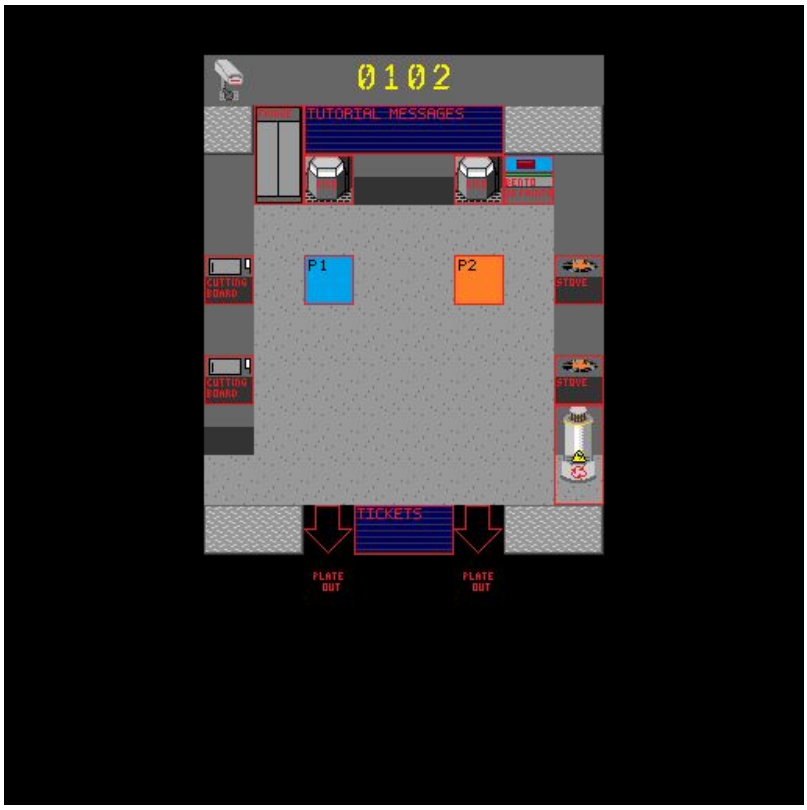
9.A.4.a. Level 1-1 Preliminary Design



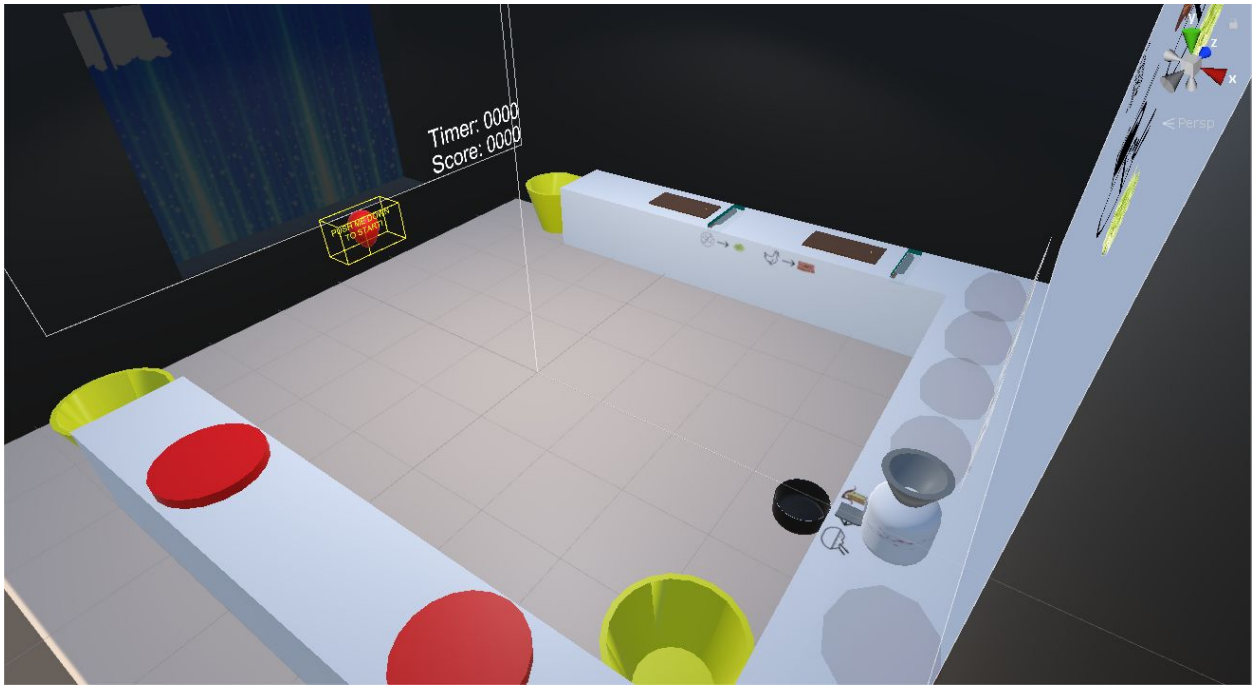
9.A.4.b. Level 1-1 Final Design



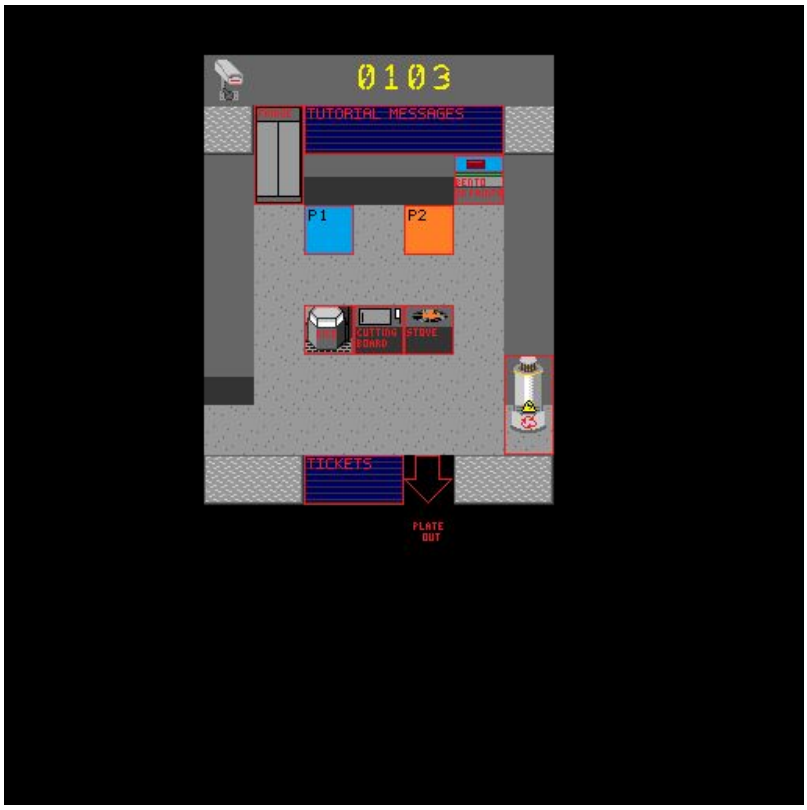
9.A.5.a. Level 1-2 Preliminary Design



9.A.5.b. Level 1-2 Final Design

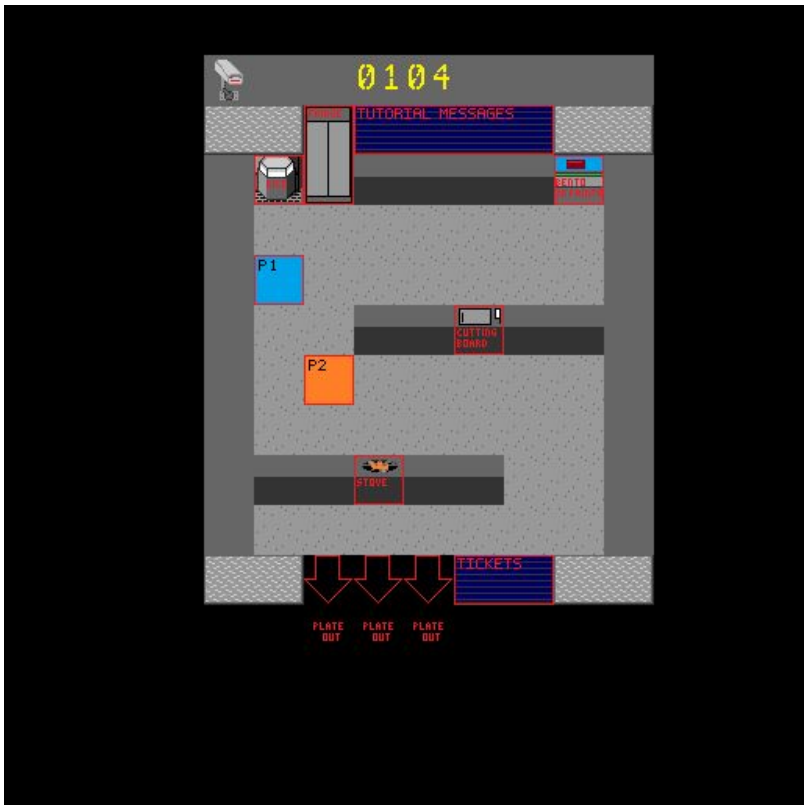


9.A.6.a. Level 1-3 Preliminary Design

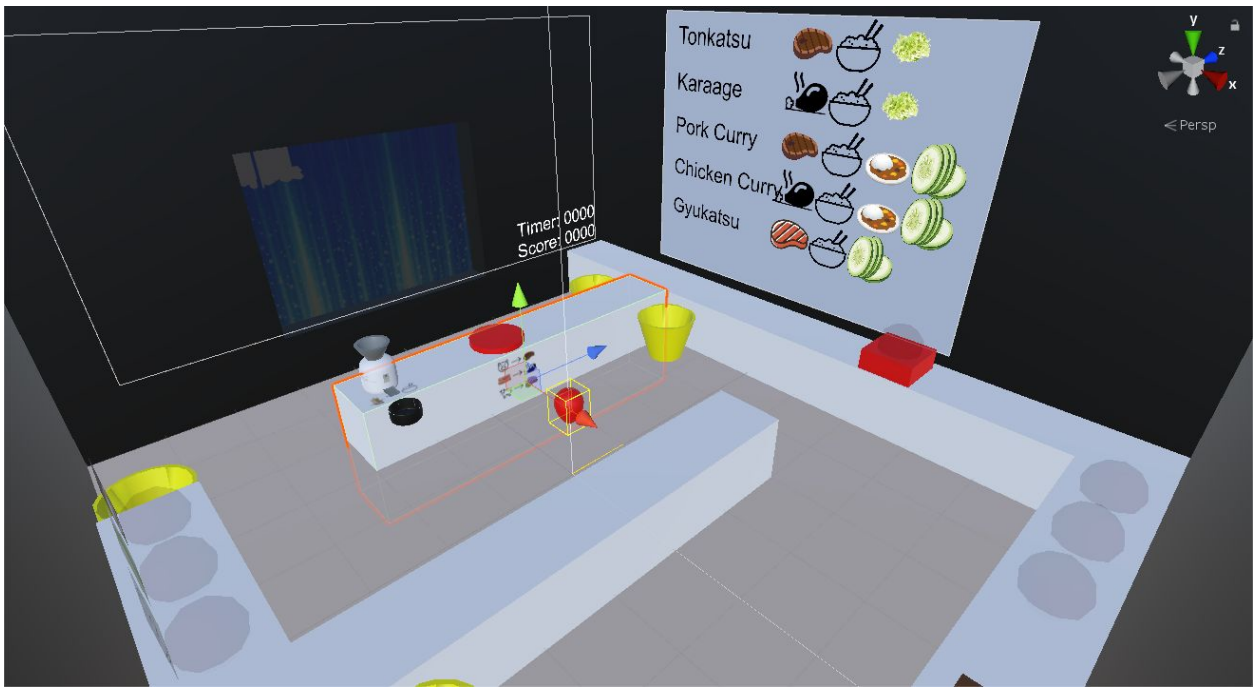


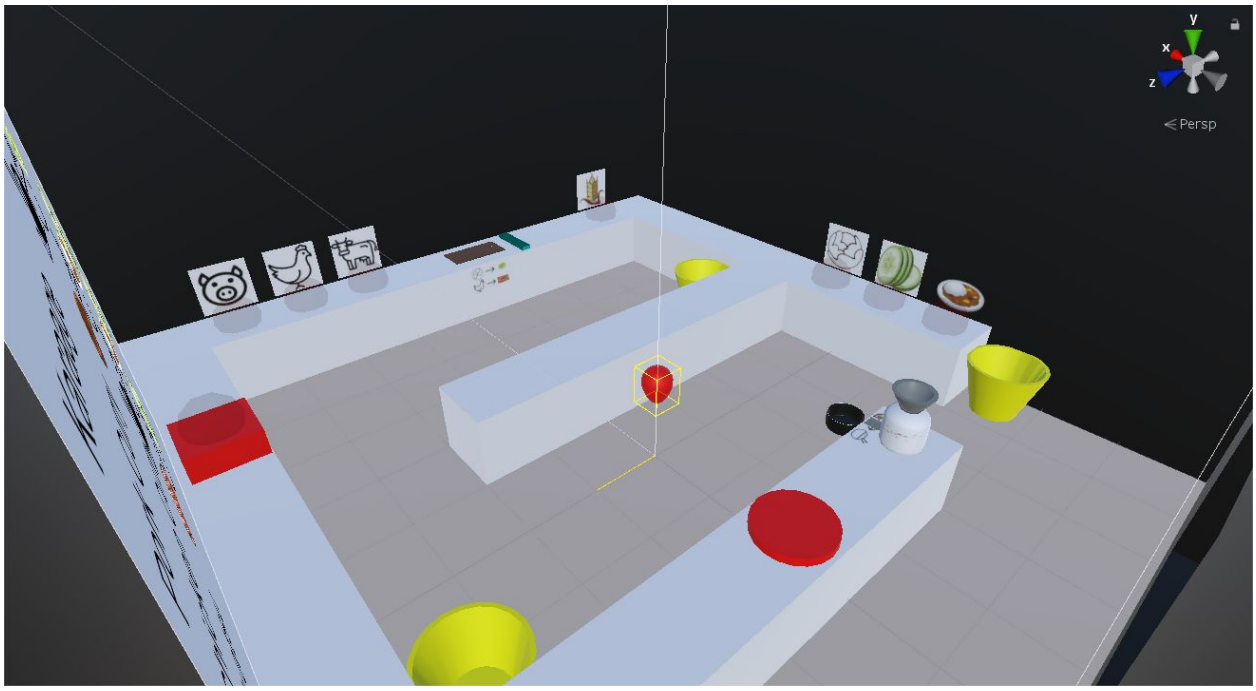
9.A.6.b. Level 1-3 Final Design

9.A.7.a. Level 1-4 Preliminary Design

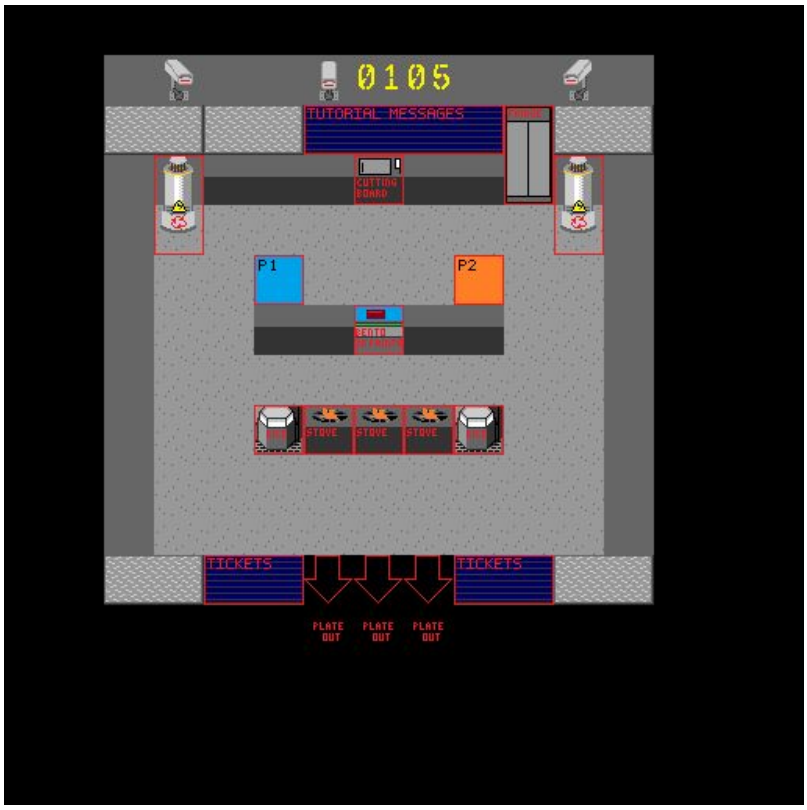


9.A.7.b. Level 1-4 Final Design

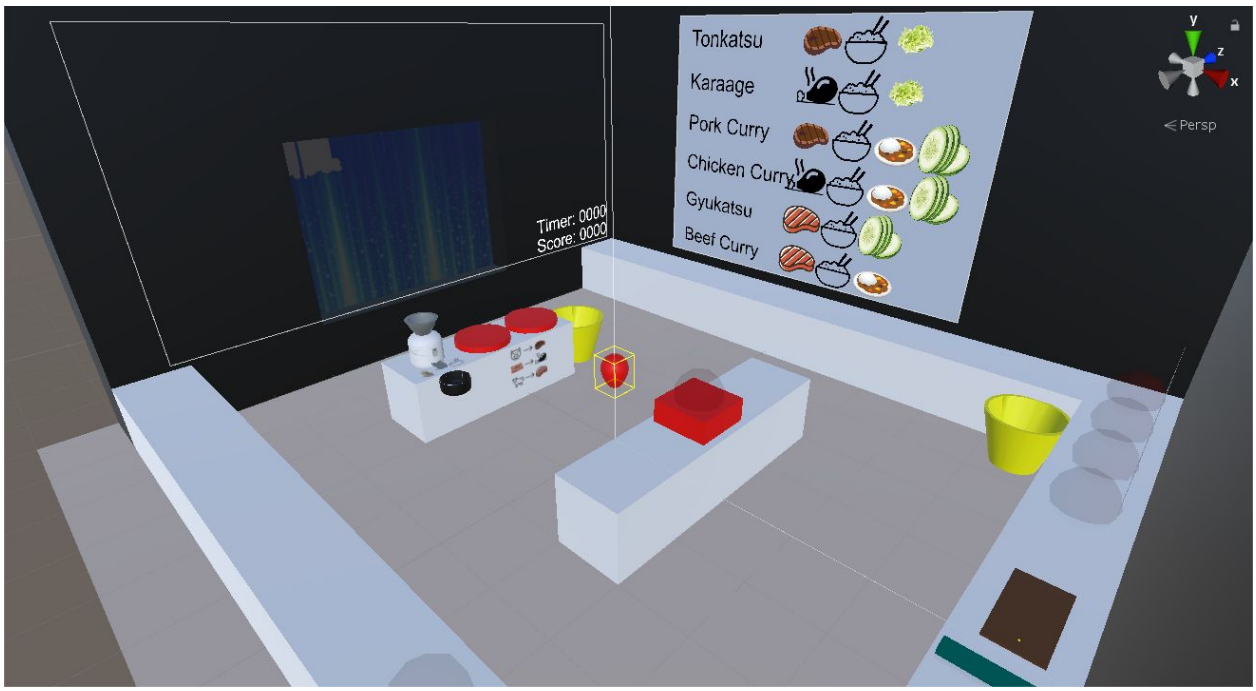




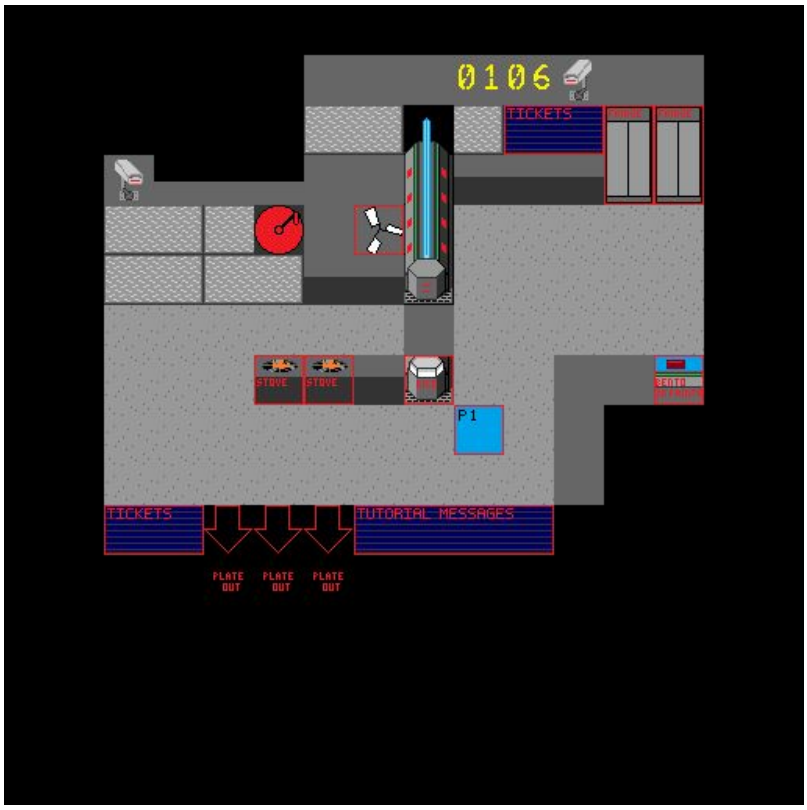
9.A.8.a Level 1-5 Preliminary Design



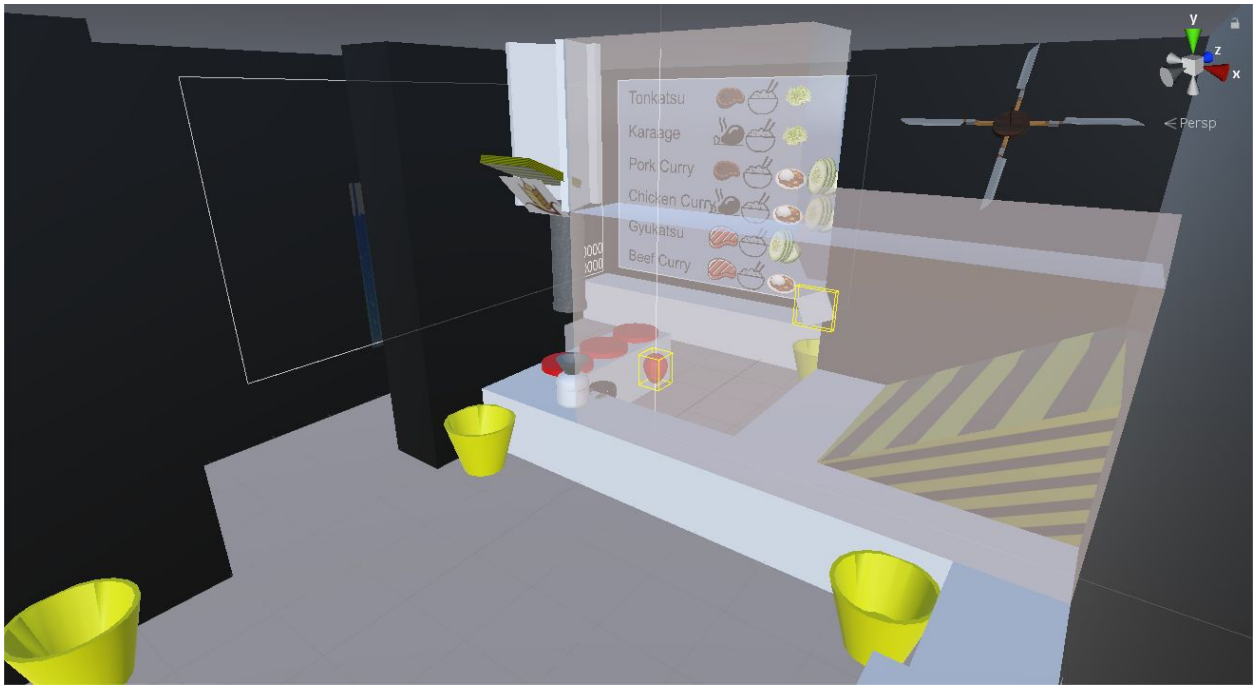
9.A.8.b. Level 1-5 Final Design

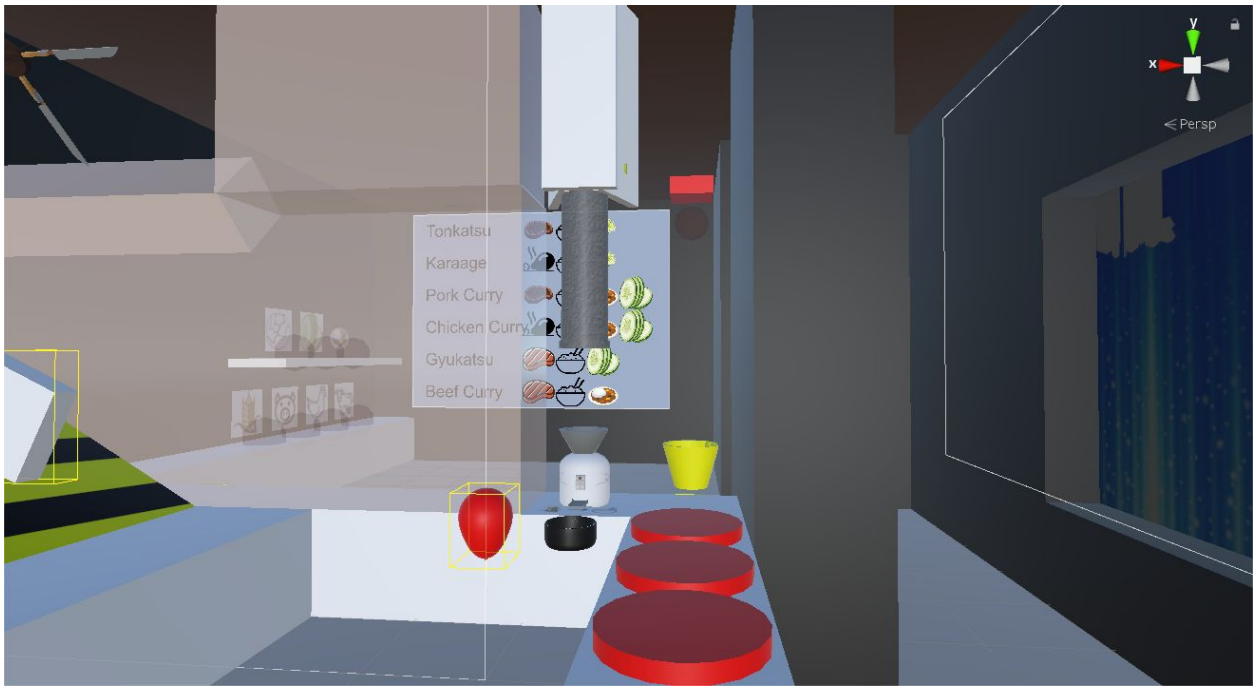


9.A.9.a. Level 1-6 Preliminary Design

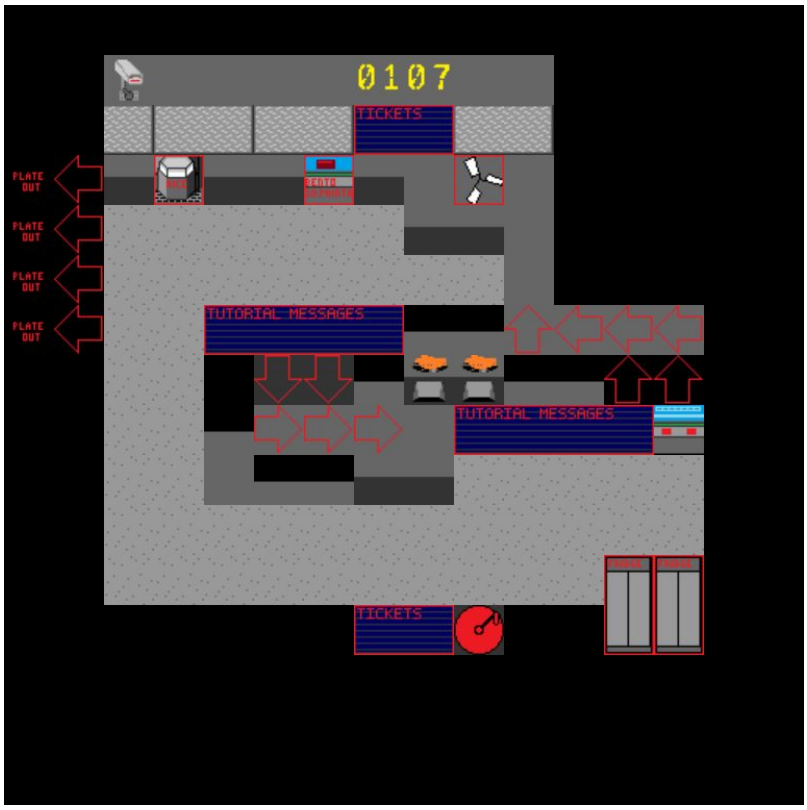


9.A.9.b. Level 1-6 Final Design



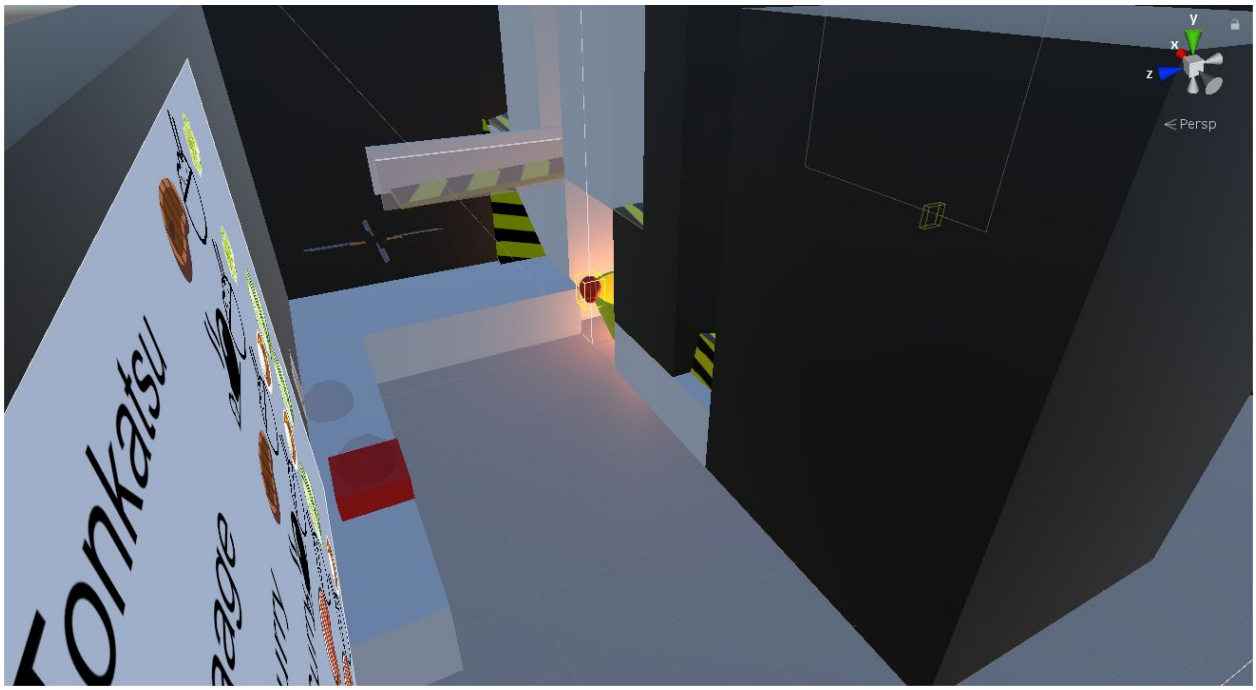


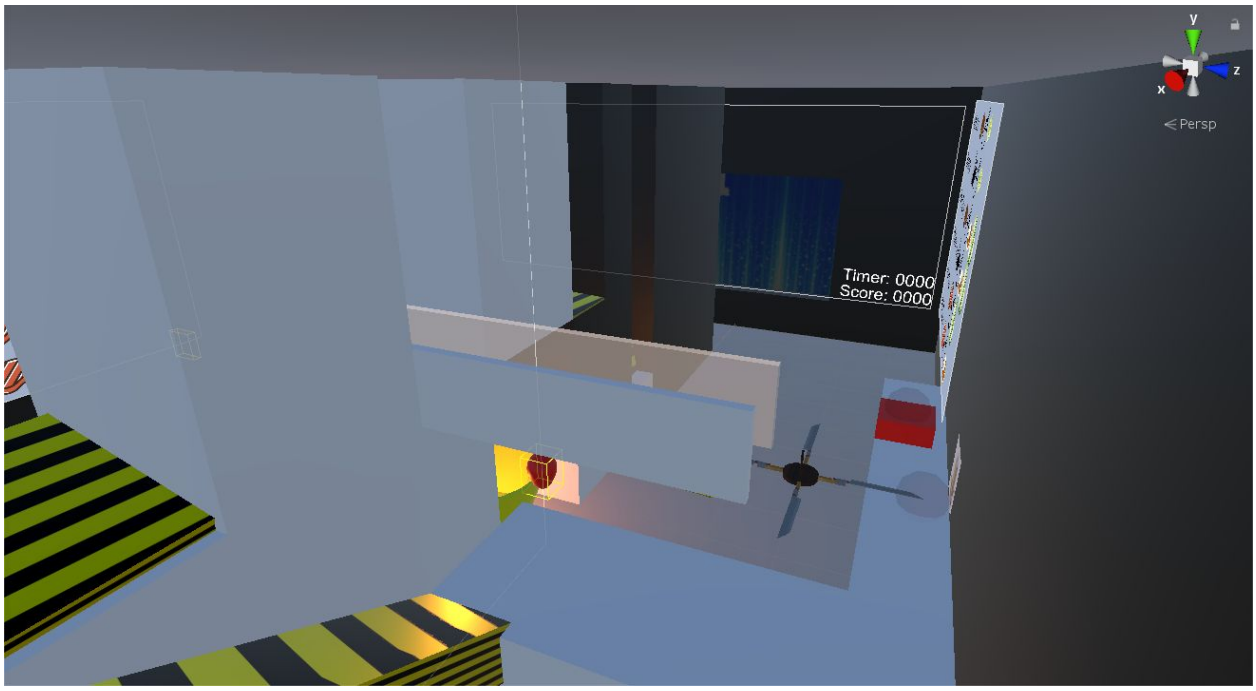
9.A.10.a. Level 1-7 Preliminary Design



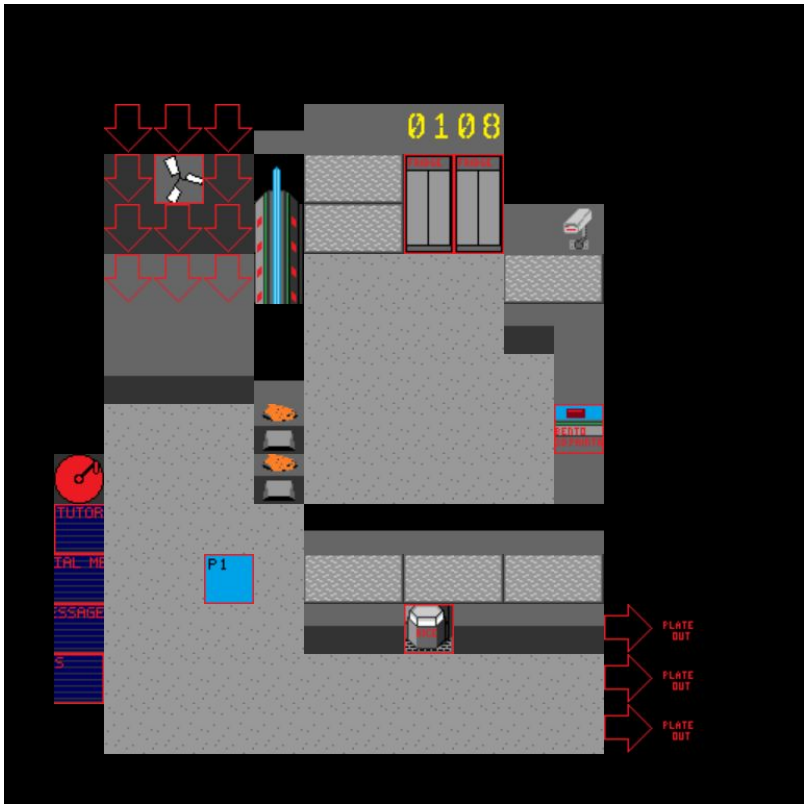
9.A.10.b. Level 1-7 Final Design


y
x
Persp


y
x
z
Persp

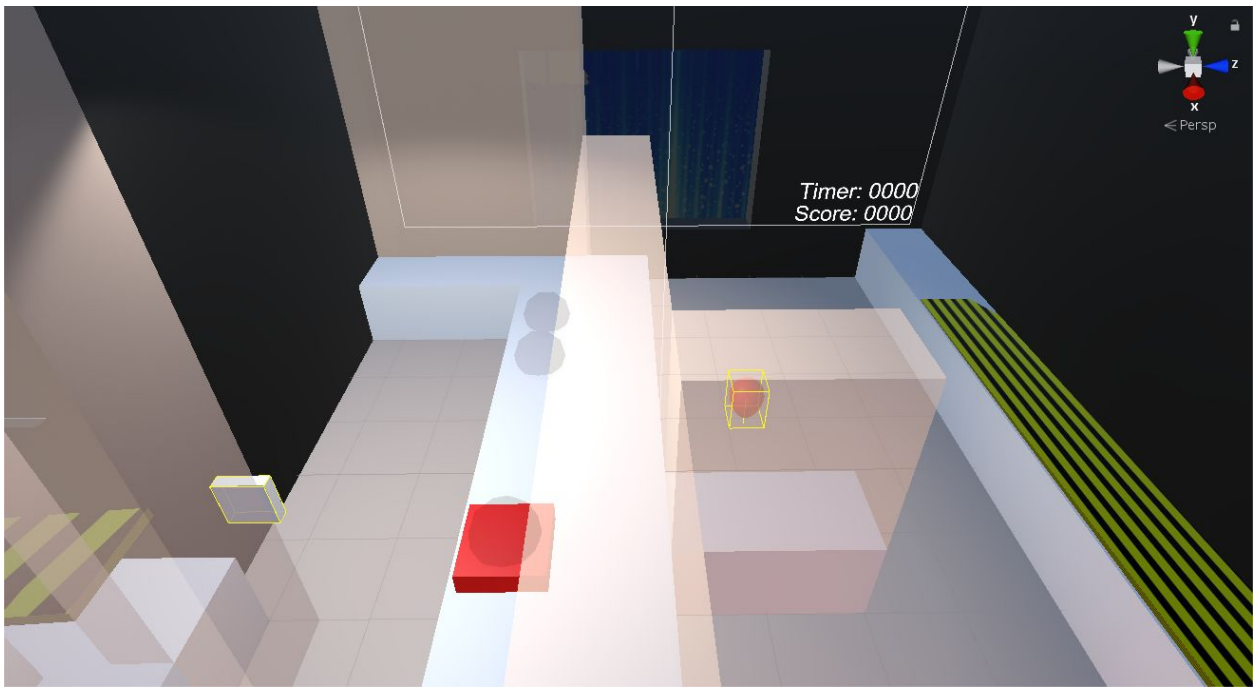


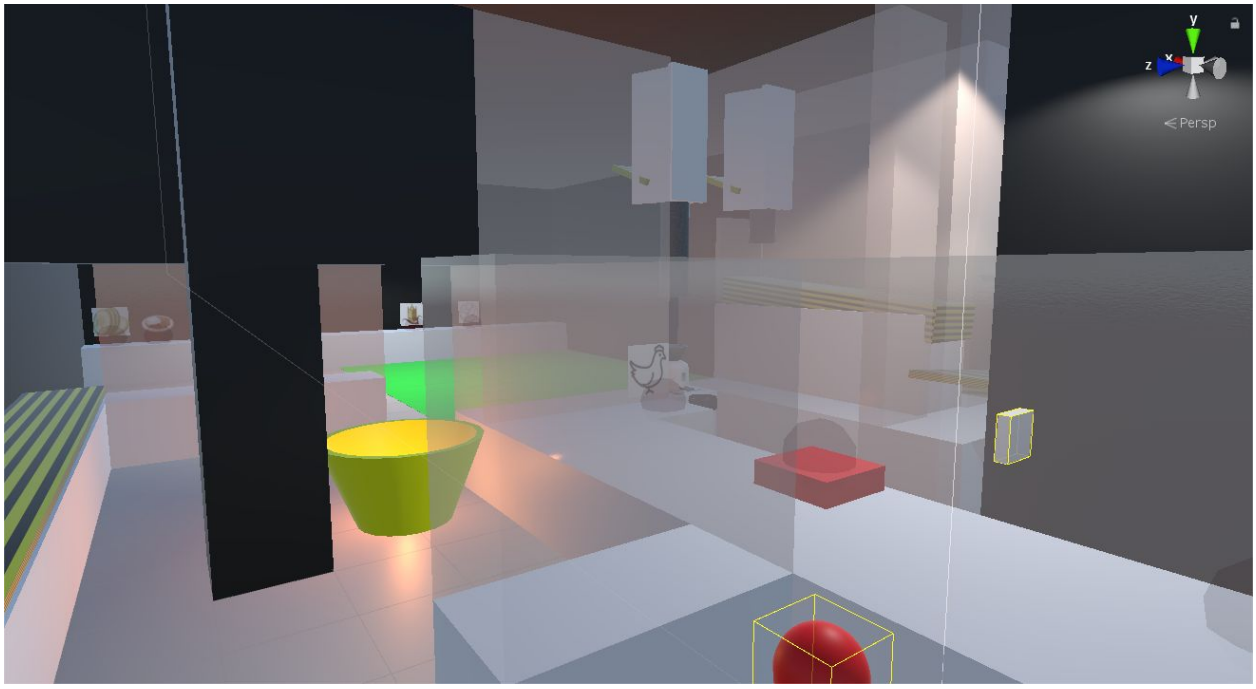
9.A.11.a. Level 1-8 Preliminary Design

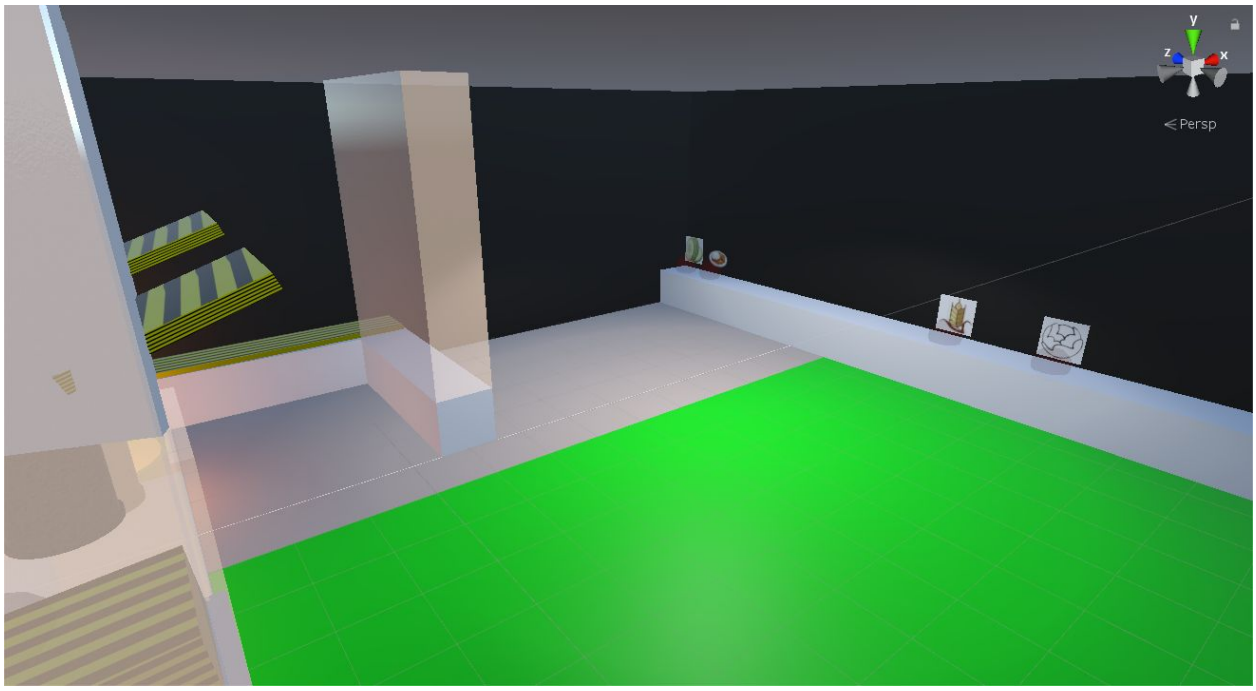


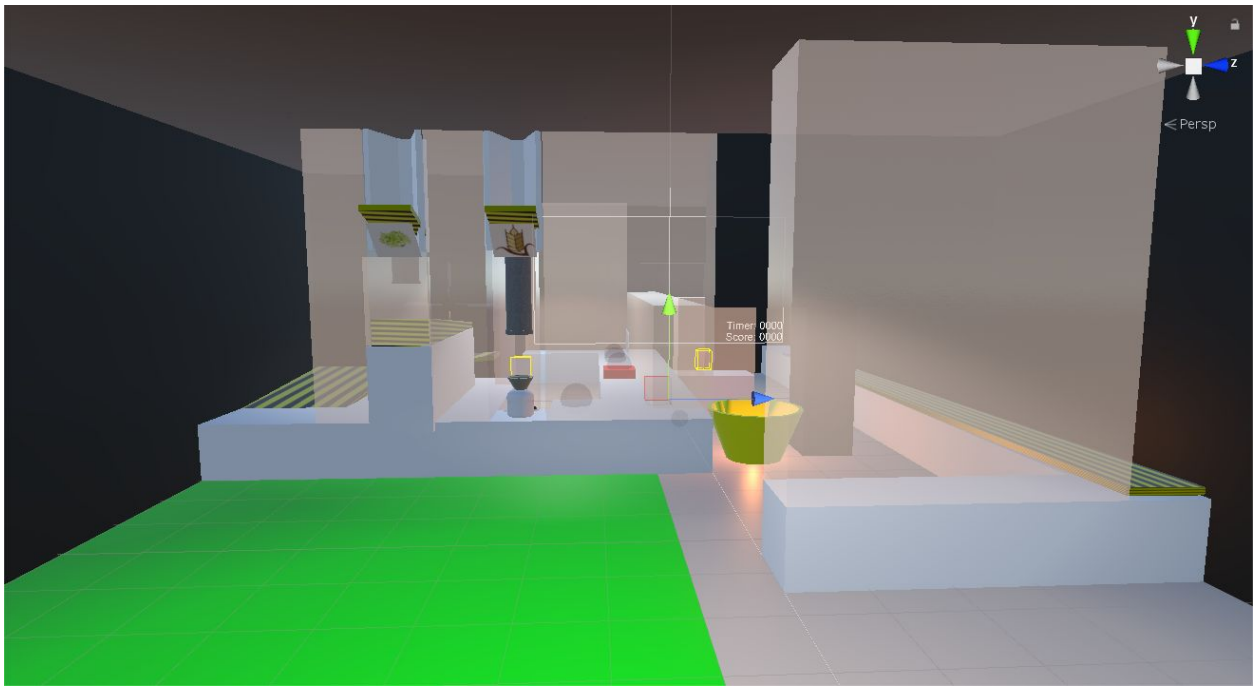
Note that the design for 1-8 was completely redone, as the original was too similar to 1-6.

9.A.11.b. Level 1-8 Final Design






y
z
x
Persp


y
z
Persp
Timer: 0000
Score: 0000

9.B. Unimplemented Wacky Machines

Wacky machine ideas, both implemented and unimplemented. Judged based on how they would meet the experience goals, or how they wouldn't. The "wackiness" of the machine was based on personal judgement, but an onomatopoeia is added to get an idea of how it would be. The descriptions below are for their original concept, and may not match what is currently implemented. They were analyzed with respect to 5 of the experience goals:

1) Variation in interaction
2) Difficulty in Flow, or opportunity for players to be able to use more efficiently with more experience
3) Requires player cooperation beyond when one person uses the machine
4) Comic Relief (should apply to most "wacky" machines)
5) Japanese style, fairly general but something that would be preferred

To determine which we should focus on, the technical, art and interface difficulty are listed. Technical refers to how difficult it would be to program the internal workings of the device, such as how to convert only certain ingredients to another. Art refers to how difficult it would be to make it look attractive. Interface refers to how difficult it would be to give players a convenient and intuitive interface for how the machine works.

9.B.1. "The Chopper"

Approved, implemented
Tech. Difficulty: 3/5
Art Difficulty: 3/5
Interface Difficulty: 4/5
Onomatopoeia: WHUBBA WHUBBA WHUBBA CHOP CHOP CHOP WHUBBA WHUBBA

Tech. Description: While spinning, items not being held by players inserted into the spinning blade collision cylinder will be transformed into the chopped component if it is choppable. Otherwise the item is destroyed. If it makes things easier items can only be inserted in one direction.

Art Description: A windmill of knife blades with a crank on the side. Depending on the kitchen type the knife type may vary.

Experience goals met: 1) Hand rotation in circle 3) Benefits from someone catching the chopped stuff on the other side. 4) Wacky 5) If the knives are suitably visible

Experience goals opposed: 5 If the knives are not suitably visible

Suggestions for improvement: None

9.B.2. Bento-box 3D Printer

Something similar implemented, no need for further implementation

Tech. Difficulty: 2/5

Art Difficulty: 2/5

Interface Difficulty: 1/5

Onomatopoeia: BEEEEEEAAAAAAANS (3D printer noises)

Tech. Description: While no bento boxes are currently in the printer and the number of boxes in the world are less than a certain amount, a new box is "printed" and spawned inside.

Art. Description: Basically a cube with the top half just being the edges with the rest solid. When "printing" it can just have a solid cube be placed to obscure the inside, spawn the object, then remove the solid cube.

Experience goals met: Not any specifically, maybe 4

Experience goals opposed: 5, if 3D printers aren't Japanese enough

Suggestions for improvement:

9.B.3. Hobo Barrel Fire

Approved, implemented

Tech. Difficulty: 2/5

Art Difficulty 3/5

Interface Difficulty 3/5

Onomatopoeia: HWKKHWAUOHWOUKK (fire noises)

Tech. Description: While an object that is not currently being held passes over, but does not land in the stove, it increments the cook time by a fixed amount. If we implement overcooking it wouldn't really be different. Objects that fail to pass (land inside) are destroyed. Objects being held over the fire (rather than thrown) are transformed into the "Burnt Crisp" item.

Art Description: Basically a normal stove, except big, with a gap in the middle large enough for items to fall into. A burnt crisp is an amorphous black object kind of like charcoal.
Experience Goals Met: 1) Passing Objects between each other 2) Players can learn advanced techniques like circling around the fire while counting to ensure it ends up on the correct side of the fire by the end. 3) Catch requires two people 4) Wacky
Experience Goals opposed: 5) Japanese don't use hobo barrel fires in their kitchens
Suggestions for Improvement:

9.B.4. Three Faucets

Tech. Difficulty: 3/5
Art. Difficulty: 4/5
Interface Difficulty: 2/5
Onomatopoeia: FSHHHHHHHHHHHH
Tech Description: Normal water does normal watery things to items thrown through or placed under it. Boiling water is for boiling food, or cleaning plates thrown through it or placed under it. Boiling oil fries food, or dirties plates thrown through it or placed under it.
Art Description: Faucets come from the roof, spilling liquid straight downward into a long rectangular basin. Space is open on either of the longer sides of the basin to allow throwing items through specific liquids. Particle effects are optional but can help distinguish the liquids.
Experience Goals Met: 1) Throwing plates/food accurately 2) Players can learn advanced techniques like proper placement of both players while still going through the right liquid 3) Enhanced by throwing 4) Wacky 5) Plates in restaurants are cleaned quickly by using high temperature/pressure boiling water.
Experience Goals Opposed: 5) people don't have falling boiling liquids for safety reasons.
Suggestions for Improvement: Normal water doesn't do much that boiling water can't, so straight up remove normal water.

9.B.5. Noodle Grate

Tech. Difficulty: 2/5
Art. Difficulty: 1/5
Interface Difficulty: 2/5
Onomatopoeia: SHWOOMP
Tech Description: Normally functions as a solid vertical plane, but when dough or another similar substance (liquid?) is thrown through it (not being held) it transforms to noodles on the other side. Should have a solid rim to provide better feedback on whether the item went through.

Art Description: Literally a grid, with a solid border.
Experience Goals Met: 1) Throwing dough through/accurately 3) Enhanced by someone catching 4) Wacky 5) Sort of like those grates we used to make Japanese sweets
Experience goals opposed: 5) Except we know how the noodles are actually made, but windmills of knives don't chop accurately if implemented
Suggestions for Improvement:

9.B.6. Vinegar/Soy Sauce Hose

No plans to implement
Tech Difficulty: 3/5
Art Difficulty: 5/5
Interface Difficulty: 4/5
Tech Description: Sprays a particle effect, but also transforms food within a short range in front of it into the appropriately soaked version.
Art Description: A hose, but hoses are hard
Experience goals met: 1) How to differentiate between picking up the hose and activating it? Whatever it is it's different. 4) Wacky
Experience goals opposed: 3) A difficult single person task reduces the time spent cooperating. 5) While certain uncultured Americans soak stuff in condiments, Japanese typically carefully apply them
Suggestions for Improvement: Too difficult really without much benefit, probably just don't even attempt implementation.

9.B.7. Mochi Hammer Time

No plans to implement
Tech Difficulty: 4/5
Art Difficulty: 3/5
Interface Difficulty: 4/5
Tech Description: One player moves the dough, the other person hammers. Repeat
Art Description: Flattened dough, an "anvil", and a hammer
Experience goals met: 1) Swinging a hammer 2) Players will enter a rhythm with the other player at a fast but controlled pace 3) Players need to adjust timing 5) How Japanese make Mochi
Experience goals opposed: 4) The longer they spend here, the less time they spend doing wacky stuff. As a result (2) might not apply. 3) Difficult to verify without punishment, which takes away from (4) 5) Mochi dough is not used for other stuff like noodles

Suggestions for Improvement: Kneading dough takes a lot of physical effort, which isn't reflected in VR. Visually it looks the same too. Maybe just skip the kneading

9.B.8. Frisbee Goal

Approved, implemented
Tech Difficulty: 1/5
Art Difficulty: 0/5
Interface Difficulty: 1/5
Onomatopoeia: DING DING DING (If valid dish) EEEEERRRRRT (If invalid) or whatever sound effect is used in Japanese game shows
Tech Description: Finished dishes can be thrown out the exit instead of carefully placed.
Art Description: Literally the same as it was before, maybe just a little bigger tho
Experience goals met: 1) More throwing is always good 4) Unconventional
Experience goals opposed: 5) Frisbee I think is more popular in North America, see "Captain America".
Suggestions for Improvement: Easy to implement, with benefits and minor downsides. Just need to design levels with it in mind

9.B.9. Conveyor Belts

Approved, didn't implement, but would have been a key point in the sushi world
Tech Difficulty: 3/5
Art Difficulty: 4/5
Interface Difficulty: 3/5
Onomatopoeia: CHUGGA CHUGGA
Tech Description: Instead of countertops there are many conveyor belts. Players can place items on them and they will keep going until it goes out the dish-out. Can be used to pass a series of items one-way between players, and discourage leaving things on tables.
Art Description: May need both a "straight" frame and a "curve" frame. The segments will proceed along a path/spline, ideally rotating appropriately.
Experience goals met: 3) Transportation of multiple items over a longer period of time than throwing encourages planning 5) Actual sushi restaurants
Experience goals opposed: None, really
Suggestions for Improvement: Can start simple and get more complex.

9.B.10. Live Chicken

Approved, partially implemented, but due to eggs not being fully fleshed out not fully implemented

Tech Difficulty: 4/5
Art Difficulty: 4/5
Interface Difficulty: 2/5
Onomatopoeia: こけこっこ (kokekotsuko, Japanese chicken noises)
Tech Description: Basically a rice cooker that makes eggs (After eating rice). Instead of producing eggs all at once like a rice cooker makes rice, the eggs would be created over a duration, ex. One egg every 10 seconds for 30 seconds. Eating more rice would reset the add more eggs into the queue up to a certain amount.
Art Description: Japanese have little regard for "animal welfare": for example 16-22 chickens weighing 2.94 kg each are placed in 1 sq meter. Chickens are killed without "stunning", and often cooked alive. Would partially reflect this by placing in a small enclosed cage, where it can only eat the rice put in, and the eggs laid would roll into a nearby container
Experience goals met: 1) Throwing more stuff is always good, in this case giving another use for rice. 2) Dividing attention takes practice, adding on the chicken to the rice cooker would help players improve this skill. 4) Unexpected use of a chicken in a practical sense could be found humorous. 5) Japanese eggs are sourced locally from very space cramped egg farms.
Experience goals opposed: none really.
Suggestions for improvement:

9.B.11. Other Less Detailed Concepts

A machine that would differ in functionality depending on the state of an egg (cooked, raw, broken, etc.)
A dog, the most popular pet in Japan
Live fish in a fish tank
A sauce shower room, which randomly sprays different types of sauces which can be caught in a bowl
Growing and harvesting rice directly